FUJITSU

FUJITSU Software
Symfoware Server V12.0.0

アプリケーション開発ガイド
(共通編)

Windows/Solaris/Linux

J2UL-1755-02Z0(00)
2013年11月

まえがき

本書の目的

本書は、アプリケーションを開発するために必要な基礎知識およびSQLの使用方法について説明しています。

本書の読者

本書は、Symfoware/RDBのデータベースを処理するアプリケーションを開発する方に読んでいただくように書かれています。本書を読むには、以下に示す技術および知識が必要です。

- Symfoware/RDBの機能およびデータベースの概要を理解している
- Symfoware/RDBを適用する業務について知識をもっている
- アプリケーションを開発する言語および開発ツールの基礎知識をもっている
- Oracle Solarisを使用できる
- Linuxシステムを使用できる
- Windows(R)システムを使用できる

なお、本書で説明しているアプリケーションは、埋込みSQLを利用することを前提に書かれています。

本書の構成

本書の構成と内容は以下のとおりです。

第1部 アプリケーション開発における基礎知識

第1章 アプリケーションの設計に必要な概念

アプリケーションの設計時に考慮すべきことを説明しています。

第2章 アプリケーションの開発

アプリケーションの動作環境および開発環境について説明しています。

第3章 アプリケーションの性能

アプリケーションの性能を向上させるための考え方を説明しています。

第2部 SQLの使用方法

第4章 SQL文の共通規則

SQL文の共通規則を説明しています。

第5章 データを検索する方法

単一行SELECT文を使用してデータを検索する方法を説明しています。

第6章 データを変更する方法

INSERT文、UPDATE文:探索、DELETE文:探索を使用してデータを変更する方法を説明しています。

第7章 カーソルを使用するデータ操作の方法

カーソルを使用してデータを操作する方法を説明しています。

第8章 複数の表を関連づけて操作する方法

複数の表のデータを関連づけて操作する方法を説明しています。

第9章 いろいろなデータ操作の方法

いろいろなバリエーションでデータを操作する方法を説明しています。

第10章 トリガを使用したデータ操作の方法

トリガを利用したデータの更新方法を説明しています。

第11章 ルーチンを使用したデータ操作の方法

プロシジャルーチンおよびファンクションルーチンの利用方法を説明しています。

第12章 XMLデータを操作する方法

XMLデータを操作する方法を説明しています。

付録A システム表

Symfoware/RDBのシステム表で出力される情報の一覧と、その意味について説明しています。

付録B 動作環境ファイルのパラメター覧

動作環境ファイルに指定できるパラメタの一覧を説明しています。

付録C 資源の見積り

ソート作業域およびメモリの見積りについて説明しています。

付録D アプリケーションおよびRDBコマンド間の排他

アプリケーションおよびSymfoware/RDBのコマンドが同時に動作した場合の排他制御について説明しています。

付録E SQLSTATE値

処理結果としてアプリケーションに通知されるSQLSTATEについて説明しています。

付録F ファンクションルーチンのパラメタのデータ型と対応するデータ形式

ファンクションルーチンのパラメタに指定されたデータベースのデータ型に対応するデータ形式について説明しています。

付録G 定量制限

Symfoware/RDBの定量制限の一覧を示します。

付録H アクセスプランの詳細

Symfoware Serverのアクセスモデルおよびエレメントの種類について説明します。

付録I トランザクションと排他制御(データベース簡単運用の場合)

データベース簡単運用時のトランザクションと排他制御について説明します。

平成25年　9月　初版 平成25年　11月　第2版

著作権

Copyright 2010-2013 FUJITSU LIMITED

目　次

第1部 アプリケーション開発における基礎知識....1

第1章 アプリケーションの設計に必要な概念....2
1.1 トランザクションと排他制御....3
1.1.1 トランザクション制御の概要....3
1.1.2 トランザクションの終了方法....4
1.1.3 トランザクションモード....6
1.1.3.1 アクセスモード....7
1.1.3.2 独立性水準....7
1.1.4 トランザクションとカーソル操作....11
1.1.5 SQL文の処理結果異常とトランザクション....13
1.1.6 アプリケーションの異常終了とトランザクション....14
1.1.7 排他制御....14
1.1.7.1 排他の属性と選択方法....14
1.1.8 資源の競合が起きた場合の制御....24
1.1.9 行単位の排他を使用する場合の注意事項....26
1.1.10 デッドロックの対処方法....29
1.1.11 トランザクション実行時間の設定....31
1.1.12 PRECEDENCE(1)を指定したSEQUENTIAL構造の特徴と注意事項....31
1.1.12.1 トランザクションの特性....32
1.1.12.2 排他に関する指定....33
1.1.12.3 対象資源が競合した場合の制御....35
1.2 クライントとサーバの接続方法....36
1.3 コネクション....37
1.3.1 複数コネクション....38
1.3.1.1 複数コネクションの接続....38
1.3.1.2 複数コネクションの切断....39
1.3.1.3 複数コネクションでSQL文を実行する場合の注意事項....40
1.3.2 マルチRDB運用のコネクション....42
1.4 リモートアクセス時の注意事項....44
1.5 通信データを暗号化する場合の設定....47
1.6 概数を使用するときの注意事項....48
1.7 SQL文のデータ操作文に関する注意事項....48
1.8 カタログについて....48

第2章 アプリケーションの開発....50
2.1 アプリケーションの動作環境....50
2.1.1 動作環境ファイル....50
2.1.2 クライアント用の動作環境ファイルの作成....54
2.1.3 初期パラメタの設定....55
2.1.3.1 アプリケーションの接続先や通信に関する実行パラメタ....55
2.1.3.2 使用する資源に関する実行パラメタ....58
2.1.3.3 デバッグおよび性能分析に関する実行パラメタ....60
2.2 アプリケーションの開発環境....64
2.2.1 利用可能な開発言語・統合開発環境....64

第3章 アプリケーションの性能....65
3.1 アクセスプラン....65
3.1.1 アクセスモデルとは....65
3.1.2 セクションとは....66
3.1.3 エレメントとは....66
3.2 アプリケーション開発時の性能チューニング....66
3.2.1 アクセスモデルの選択....67
3.3 アクセスモデルの指定....68
3.3.1 インデックスを定義する....68

3.3.2 ASSIST指定でアクセスモデルを指定する....72
3.3.2.1 ASSIST指定の使用方法....74
3.3.2.2 ASSIST指定の種類....74
3.3.2.3 制約ASSISTの使用方法....75
3.3.2.4 ジョインASSISTの使用方法....76
3.3.2.5 ASSIST指定の注意事項....79
3.4 アクセスモデルの確認....80
3.4.1 制約系アクセスモデル....81
3.4.2 ジョイン系アクセスモデル....81
3.5 その他の性能チューニング方法....84
3.5.1 ASSIST指定でSQL文の最初の1件の検索を高速化する....84
3.5.1.1 実行制御ASSISTの使用方法....84
3.5.2 SQL文の条件式には明示的に条件を記述する....87

第2部 SQLの使用方法....89

第4章 SQL文の共通規則....90
4.1 SQL文に指定する名前....90
4.2 データベースの例....91

第5章 データを検索する方法....95
5.1 列の値をそのまま取り出す....95
5.1.1 複数の列の値を取り出す....95
5.1.2 NULL値を含むデータを取り出す....96
5.1.3 複数の表からデータを取り出す....98
5.1.4 冗長な行を取り除いて取り出す....98
5.2 特定の列以外を取り出す....99
5.2.1 データ間で四則演算をする....99
5.2.2 列の値の合計値、平均値、最大値、最小値、および行数を求める....101

第6章 データを変更する方法....106
6.1 データベースにデータを追加する....106
6.1.1 追加するデータをNULL値にする....106
6.1.2 追加するデータに省略値を使用する....107
6.1.3 追加するデータに現在の日時を設定する....109
6.1.4 別の表のデータを追加する....110
6.2 データベースのデータを更新する....113
6.2.1 複数の列の値を更新する....113
6.2.2 更新するデータをNULL値にする....114
6.2.3 更新するデータに省略値を使用する....115
6.2.4 更新するデータに現在の日時を設定する....115
6.2.5 更新するデータに別の列の値を使用する....116
6.2.6 更新するデータに四則演算を行う....117
6.2.7 表のすべての行を更新する....117
6.3 データベースのデータを削除する....118
6.3.1 表のすべての行を削除する....119

第7章 カーソルを使用するデータ操作の方法....120
7.1 カーソルの概念....120
7.2 カーソルを宣言する....123
7.3 カーソルのオープンおよびクローズ....131
7.4 カーソルの位置づけおよびデータの取出し....131
7.5 カーソルを使用してデータを更新する....134
7.6 カーソルを使用してデータを削除する....136

第8章 複数の表を関連づけて操作する方法....138
8.1 複数の表から新たな表を導出する....138
8.2 表を関連づける条件を指定する....139
8.3 表に別名を用いてデータを操作する....143

8.4 同一表を関連づけて操作する....143
8.5 複数の表から行の集合和を求める....146

第9章 いろいろなデータ操作の方法....148
9.1 表をグループ化してデータを操作する....148
9.1.1 表をグループ化してデータを取り出す....148
9.1.2 グループ化した表から処理対象のグループを特定する....154
9.1.3 関連づけられた複数の表をグループ化して操作する....155
9.1.4 グループ表の集計結果を再集計する....158
9.2 いろいろな探索条件を指定する....159
9.2.1 2つの値を比較する....161
9.2.2 値が特定の範囲に含まれるかを調べる....166
9.2.3 値がいくつかの値の集合に含まれるかを調べる....167
9.2.4 値が文字列のパターンに一致するかを調べる....171
9.3 数値のデータを操作する....175
9.4 文字列のデータを操作する....179
9.5 日付のデータを操作する....182
9.6 データ型を変換してデータを操作する....189
9.7 CASE式を使用してデータを操作する....190
9.8 行識別子を指定してデータを操作する....193
9.9 並列クエリを利用してデータを操作する....193
9.10 スキーマ名を省略する....196
9.11 現行セションの利用者を変更する....197
9.12 問合せ指定の結果列に名前を付加する....197
9.13 順序を使用してデータを操作する....198
9.14 LIMITを使用して結果件数を限定する....200
9.15 ROWNUMを使用してデータを操作する....200
9.16 行値構成子を使用してデータを操作する....206

第10章 トリガを使用したデータ操作の方法....208
10.1 トリガ定義....208

第11章 ルーチンを使用したデータ操作の方法....213
11.1 プロシジャルーチンの利用方法....213
11.1.1 プロシジャルーチンとは....213
11.1.2 プロシジャルーチンを利用するアプリケーションの作成の流れ....213
11.1.3 プロシジャルーチンの定義....214
11.2 ファンクションルーチンの利用方法....216
11.2.1 ファンクションルーチンとは....216
11.2.2 ファンクションルーチンを利用するアプリケーションの作成の流れ....217
11.2.3 ファンクションルーチンの定義....217

第12章 XMLデータを操作する方法....219
12.1 XQueryの概要....219
12.1.1 XQueryの構成要素....219
12.1.2 XQueryのデータモデル....219
12.1.2.1 シーケンスおよび項目....219
12.1.2.2 原子値....220
12.1.2.3 ノード....220
12.1.2.4 データ入力源....222
12.1.3 XQueryの実行例....222
12.2 XMLデータを検索する....223
12.2.1 SQL/XMLでXMLデータを検索する....223
12.2.1.1 XMLQUERY関数....224
12.2.1.2 XMLEXISTS述語....226
12.2.1.3 XMLシリアライゼーション....226
12.2.1.4 SQL/XMLで扱えるデータ形式....227
12.3 XQueryの検索性能における留意事項....228

12.4 XMLデータの形式における留意事項....232
12.5 XQueryの指定方法....233
12.5.1 ある要素を検索する例....235
12.5.2 ある要素の内容を数値の大小比較で検索する例....235
12.5.3 ある要素の内容を検索する例....236
12.5.4 ある要素の内容をタグを除いて連結した文字列で検索する例....236
12.5.5 すべての要素の内容を検索する例....236
12.5.6 ある階層にあるすべての要素の内容を検索する....237
12.5.7 ある要素の属性の値を検索する例....237
12.5.8 ある要素の内容から現在の日時を検索する例....238
12.5.9 ある要素の内容から指定した文字列を検索する例....238
12.5.10 2つの条件を論理積を使って検索する例....239
12.5.11 ある検索結果を元に検索をする例....239
12.5.12 ある条件に応じて検索結果を分岐する....239
12.5.13 ある検索結果がある条件を満たしているかを判定する....240

付録A システム表....242
A.1 システム表の概要....242
A.2 システム表の参照....243
A.3 システム表の詳細....248
A.3.1 データベース定義情報....248
A.3.2 スキーマ定義情報....249
A.3.3 表定義情報....250
A.3.4 表定義拡張情報....251
A.3.5 列定義情報....253
A.3.6 制約定義情報....255
A.3.7 文字列を表現する表情報....257
A.3.8 一意性制約の定義情報....257
A.3.9 列一覧情報....258
A.3.10 注釈情報....259
A.3.11 インデックス定義情報....260
A.3.12 インデックス定義拡張情報....261
A.3.13 ルーチン定義情報....262
A.3.14 ルーチン定義拡張情報....264
A.3.15 ルーチンパラメタ情報....265
A.3.16 権限情報....268
A.3.17 トリガ定義情報....270
A.3.18 トリガ列一覧情報....273
A.3.19 順序情報....274
A.3.20 利用者情報....276
A.3.21 ユーザパラメタ設定情報....278
A.3.22 ロール情報....279
A.3.23 ロールに付与された権限の情報....279
A.3.24 利用者に対するロールの付与情報....282
A.3.25 SQL文の簡易実行情報....282
A.3.26 DSI情報....283
A.3.27 コネクションの動作環境のパラメタ設定情報(一括出力)....286
A.3.28 コネクションの動作環境のパラメタ設定情報(順序付き出力)....288
A.3.29 システムの動作環境のパラメタ設定情報(一括出力)....290
A.3.30 システムの動作環境のパラメタ設定情報(順序付き出力)....291
A.3.31 コネクション情報....292

付録B 動作環境ファイルのパラメター覧....295
B.1 システム用の動作環境ファイル....302
B.2 クライアント用の動作環境ファイル....306
B.3 サーバ用の動作環境ファイル....312

付録C 資源の見積り....315

C.1 ソート作業域の見積り....315
C.1.1 共用ソート作業域....315
C.1.2 固有ソート作業域....315
C.2 メモリの見積り式....316
C.2.1 アプリケーション実行時のメモリ見積り....316
C.3 ファイルディスクリプタ数の見積り....317

付録D アプリケーションおよびRDBコマンド間の排他....318

付録E SQLSTATE値....336

付録F ファンクションルーチンのパラメタのデータ型と対応するデータ形式....342

付録G 定量制限....348
G.1 Symfoware/RDBの定量制限....348
G.2 Textアダプタの定量制限(Solarisの場合)....356
G.3 XMLアダプタの定量制限....357

付録H アクセスプランの詳細....358
H.1 アクセスモデルの種類....358
H.1.1 制約系アクセスモデル....358
H.1.2 ジョイン系アクセスモデル....365
H.1.3 更新系アクセスモデル....375
H.2 エレメントの種類....381
H.2.1 SCANエレメント....381
H.2.2 MERGE JOINエレメント....388
H.2.3 FETCH JOINエレメント....389
H.2.4 NESTED LOOP JOINエレメント....390
H.2.5 INSERTエレメント....391
H.2.6 DELETEエレメント....392
H.2.7 UPDATEエレメント....393
H.2.8 OUTPUTエレメント....393
H.2.9 GROUPINGエレメント....394
H.2.10 TRIGGER EXECUTIONエレメント....394

付録I トランザクションと排他制御(データベース簡単運用の場合)....395
I.1 トランザクション制御の概要....395
I.2 トランザクションの終了方法....396
I.3 トランザクションモード....398
I.3.1 アクセスモード....399
I.3.2 独立性水準....399
I.4 トランザクションとカーソル操作....405
I.5 SQL文の処理結果異常とトランザクション....406
I.6 アプリケーションの異常終了とトランザクション....407
I.7 排他制御....407
I.7.1 排他の属性と選択方法....407
I.8 資源の競合が起きた場合の制御....416
I.9 行単位の排他を使用する場合の注意事項....418
I.10 デッドロックの対処方法....421
I.11 トランザクション実行時間の設定....423

索引....424

第1部 アプリケーション開発における基礎知識

本部では、アプリケーションを開発するうえで開発言語共通の考慮すべき事項について説明します。

第1章 アプリケーションの設計に必要な概念.....2

第2章 アプリケーションの開発.....50

第3章 アプリケーションの性能.....65

第1章 アプリケーションの設計に必要な概念

アプリケーションを利用してデータベースアクセスの設計を行う場合、基本的な概念を理解する必要があります。ここでは、データベースアクセスを行うアプリケーションの設計に必要な概念として、セション、トランザクションおよびコネクションについて説明します。

セション

Symfoware/RDBにアクセスする環境をセションと呼びます。

セションを意識する必要があるのは、埋込みSQL連携の場合のみです。

通常、アプリケーションで最初のSQL文が実行されると、Symfoware/RDBが自動的にセションを作成します。

トランザクション

データベースのアクセスにおいては、逐次的にデータベースを更新したり、いくつかのSQL文をまとめてからデータベースを更新したりすることが可能です。そのため、一連のデータ操作の途中で不慮のシステムダウンなどが発生すると、処理全体を有効にするか、無効にして再度やり直しができるようにデータベースをリカバリする必要があります。このように、一連のデータ操作の一貫性を保証する単位を“トランザクション”と呼びます。

詳細については、“1.1 トランザクションと排他制御”を参照してください。

また、データベース簡単運用の場合は、“付録I トランザクションと排他制御(データベース簡単運用の場合)”を参照してください。

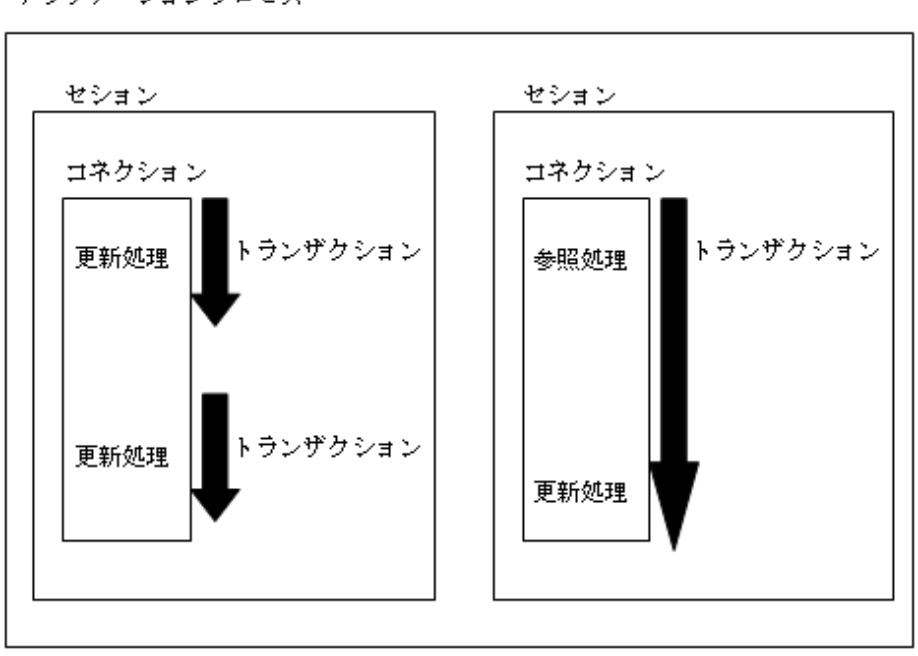
コネクション

コネクションとは、クライアントのアプリケーションがサーバのSymfoware/RDBデータベースにアクセスするために、クライアントとサーバの間を結んだ接続関係のことです。アプリケーションがデータベースシステムとコネクションをもつことによって、そのデータベースシステムの資源を操作することができます。

詳細については、“1.3 コネクション”を参照してください。

アプリケーションと、セション・トランザクション・コネクションの関係を以下に示します。

図1.1 アプリケーションと、セション・トランザクション・コネクションの関係



1つのアプリケーションプロセスには、1つまたは複数のセションを作成できます。

1つのセションでは、1つまたは複数のトランザクションを実行することができます。ただし、1つのセションで複数のトランザクションを同時に実行することはできません。

1.1 トランザクションと排他制御

Symfoware/RDBでは、トランザクション単位にデータベースの更新が行われます。複数のトランザクションが同時に同じデータベースを参照または更新すると、データの矛盾が発生してしまいます。このようなことを防ぐために、Symfoware/RDBは、排他制御を行います。ここでは、トランザクション処理と排他制御について説明します。

データベース簡単運用でのトランザクションと排他制御については、“I.1 トランザクション制御の概要”を参照してください。

なお、DSO定義でPRECEDENCE(1)が指定されたSEQUENTIAL構造の表を参照または更新する場合には、トランザクションや排他制御に違いがあります。“1.1.12 PRECEDENCE(1)を指定したSEQUENTIAL構造の特徴と注意事項”も参照してください。

1.1.1 トランザクション制御の概要

SQL文を使用してデータ操作を行った場合に、Symfoware/RDBが操作の内容をどのようにデータベースへ反映するかを説明します。

トランザクションの開始

トランザクションは、Symfoware/RDBにより自動的に開始されます。アプリケーション中のSQL文を実行する場合、トランザクションが開始された状態でなければ、SQL文の処理に先立ってトランザクションが開始されます。トランザクションが開始されていない状態とは、アプリケーションが実行開始したあと、またはトランザクション制御用のSQL文によってトランザクションが終了したあと、SQL文が一度も実行されていない状態です。

SQL文には、トランザクションを開始しないSQL文もあります。

- トランザクションを開始するSQL文
  - カーソル宣言以外のデータ操作文
  - 動的カーソル宣言以外の動的SQL文
  - CALL文
- トランザクションを開始しないSQL文
  - トランザクション制御文
  - コネクション管理文
  - セション管理文
  - 埋込み例外宣言
  - カーソル宣言
  - 動的カーソル宣言
  - 資源操作文
  - SET USER PASSWORD文
  - SET ROLE文

トランザクションの終了

トランザクションは、以下のSQL文を実行することによって、終了させることができます。また、アプリケーションが正常終了すると、自動的に終了します。

表1.1 トランザクションを制御するSQL文

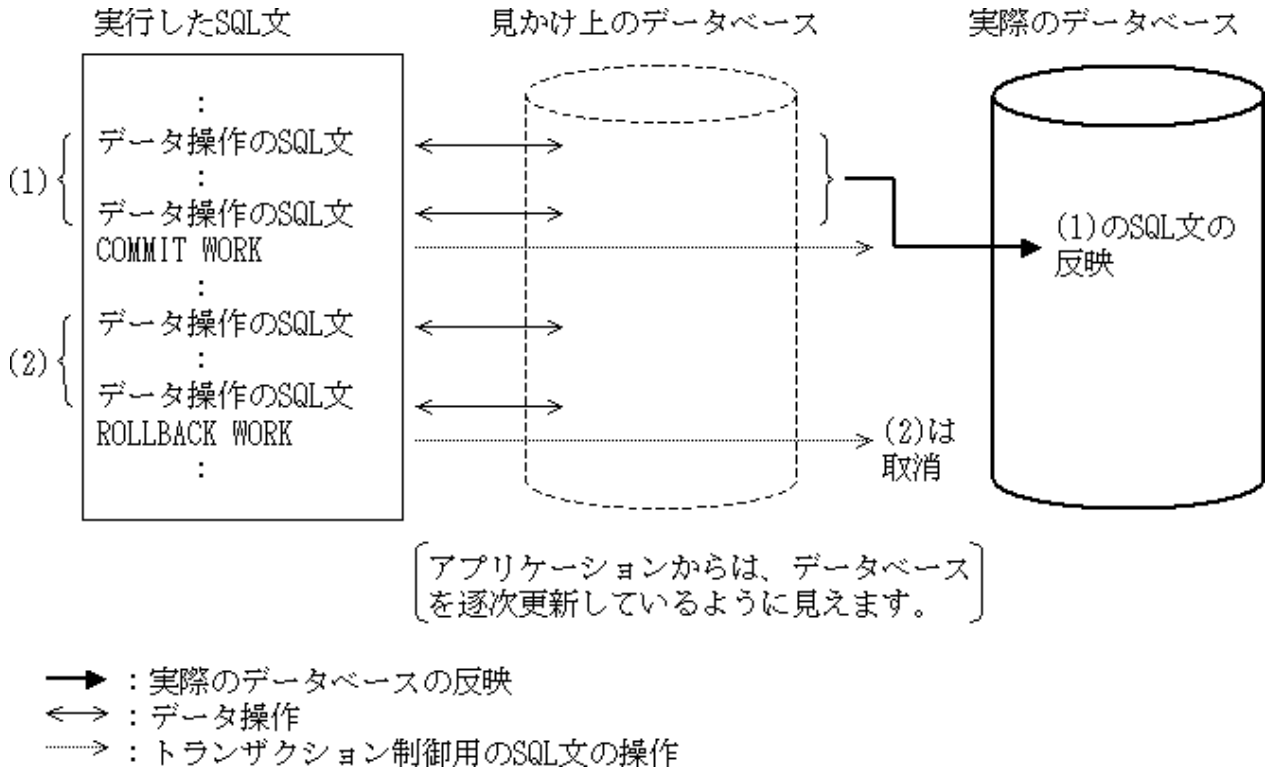
SQL文	機能
COMMIT文	現在のトランザクションを終了します。 トランザクション内でのデータ操作(注)をすべてデータベースに反映します。 オープン中のすべてのカーソルをクローズします。ただし、トランザクションを超越するカーソルはクローズしません。 トランザクション内で獲得したロックをすべて解放します。
ROLLBACK文	現在のトランザクションを終了します。 トランザクション内でのデータ操作(注)はすべて取り消します。 オープン中のすべてのカーソルをクローズします。 トランザクション内で獲得したロックをすべて解放します。

注) データ操作には、挿入、更新および削除があります。

トランザクションを制御する例

トランザクションを制御する例を以下に示します。データ操作のSQL文を実行すると、トランザクションごとに、そのトランザクション内で処理されたデータがデータベースに反映されます。

図1.2 データベースの更新処理



1.1.2 トランザクションの終了方法

トランザクションの終了方法には、以下の3つの方法があります。

- COMMIT文によるトランザクションの終了
- ROLLBACK文によるトランザクションの終了
- アプリケーションの終了によるトランザクションの終了

COMMIT文によるトランザクションの終了

COMMIT文は、トランザクションの区切りを指定するSQL文です。COMMIT文により行われる処理は、以下のとおりです。

- 処理中のトランザクションを終了し、このトランザクションで行われたデータ操作をデータベースに反映します。
- オープン中のすべてのカーソルをクローズします。ただし、トランザクションを超越するカーソルはクローズしません。
- このトランザクションで獲得した、データベース資源のロックをすべて解放します。

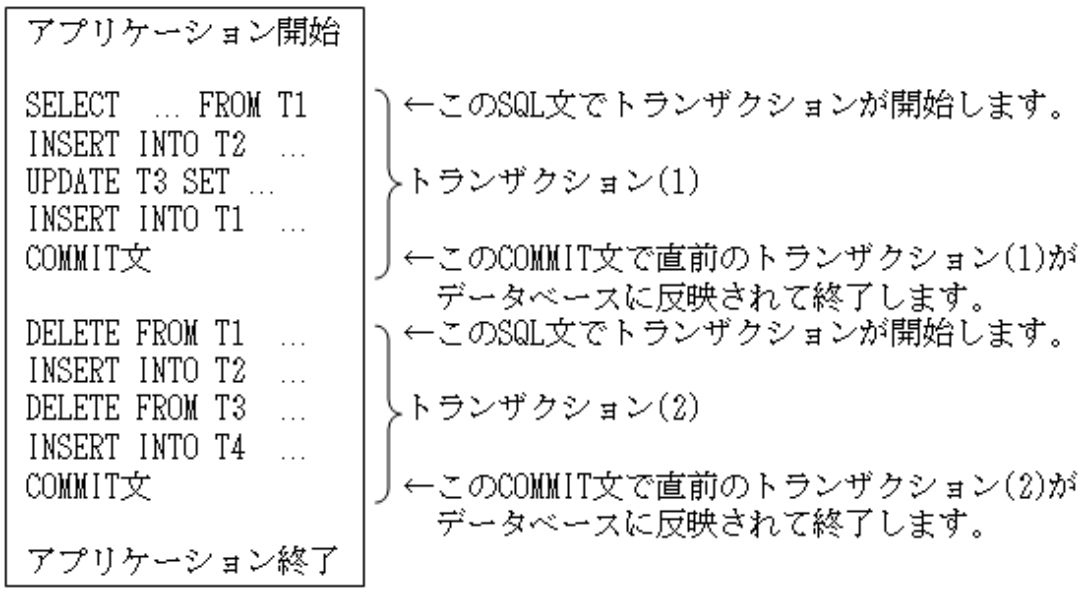
前に実行したCOMMIT文またはROLLBACK文以後、当該COMMIT文までに行ったデータベースの処理をデータベースに反映します。次に、現在のトランザクションを終了します。一度もCOMMIT文またはROLLBACK文を実行していない場合は、アプリケーションの実行開始から当該COMMIT文までのデータベースの処理をデータベースに反映します。

COMMIT文の形式を以下に示します。

```
COMMIT WORK
```

COMMIT文によるデータベース更新の例を以下に示します。

図1.3 COMMIT文によるデータベース更新の例



ROLLBACK文によるトランザクションの終了

ROLLBACK文は、トランザクション内で行ったデータベースの処理を取り消すためのSQL文です。ROLLBACK文により行われる処理を以下に示します。

- 処理中のトランザクションを終了し、このトランザクションで行われたすべての変更を取り消します。データベースは更新されません。
- オープン中のすべてのカーソルをクローズします。
- このトランザクションで獲得した、データベース資源のロックをすべて解放します。

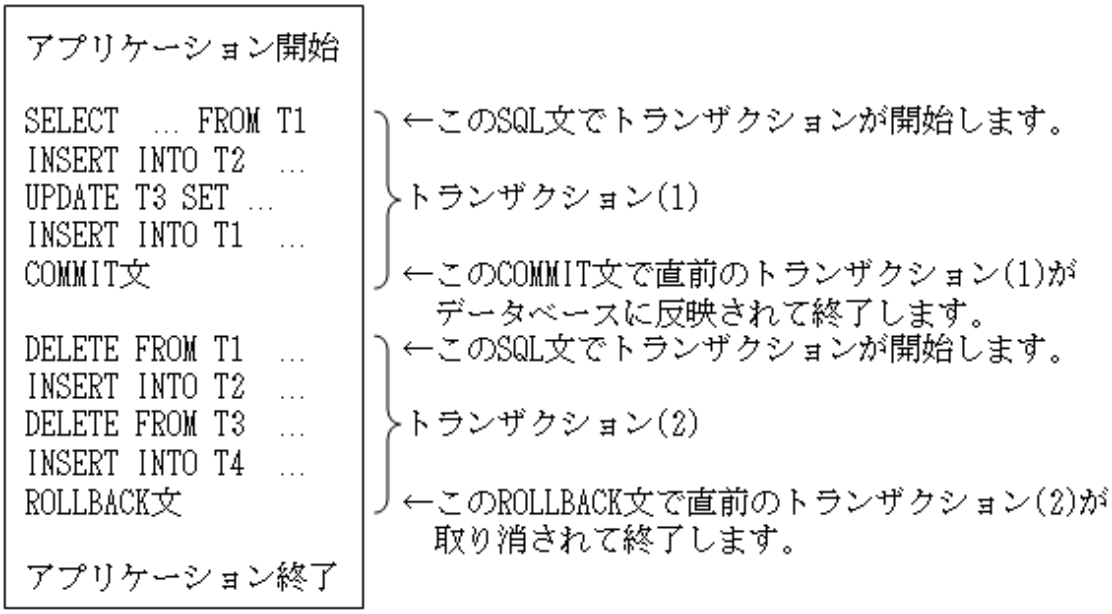
前に実行したCOMMIT文またはROLLBACK文実行以後、当該ROLLBACK文までに行ったデータベースの処理を取り消します。次に、現在のトランザクションを終了します。一度もCOMMIT文またはROLLBACK文を実行していない場合は、アプリケーションの実行開始から当該ROLLBACK文までのデータベースの処理を取り消します。

ROLLBACK文の形式を以下に示します。

```
ROLLBACK WORK
```

ROLLBACK文によるトランザクション取消しの例を以下に示します。

図1.4 ROLLBACK文によるトランザクション取消しの例

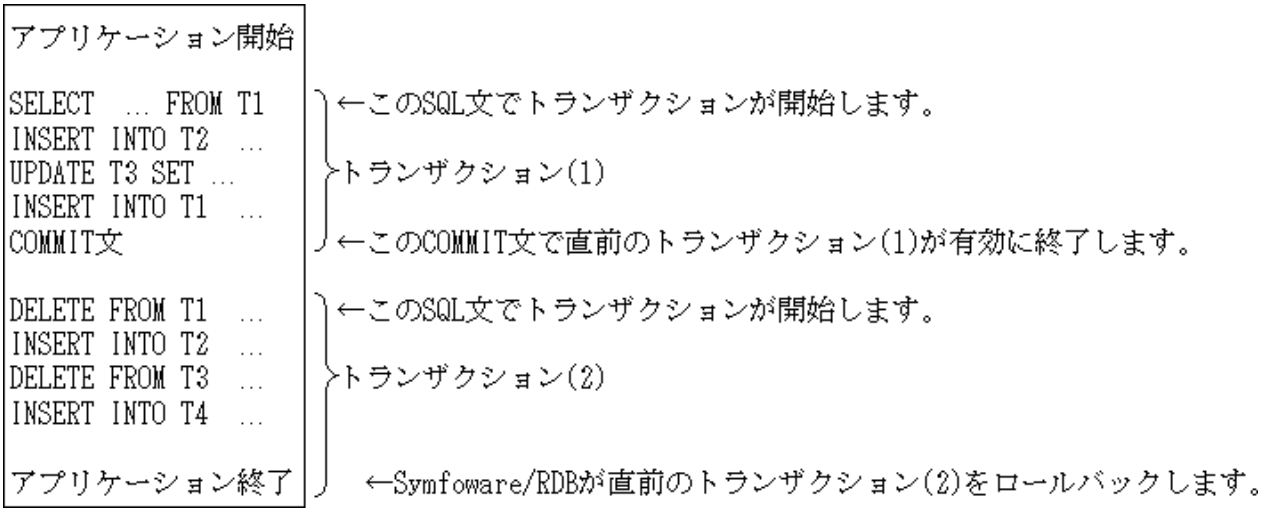


アプリケーションの終了によるトランザクションの終了

アプリケーションがトランザクション開始状態のままCOMMIT文もROLLBACK文も実行せずに終了する場合には、Symfoware/RDBがトランザクションをロールバックします。

トランザクションが開始状態のままアプリケーションが終了する例を以下に示します。

図1.5 トランザクション開始状態でのアプリケーションの終了の例



1.1.3 トランザクションモード

複数のトランザクションが同じ資源にアクセスする場合、なにも制御を行わないと、データの矛盾が発生してしまいます。このようなことが起こらないように、トランザクションモードを設定し、排他制御を行います。

トランザクションモードには、以下の2つの項目があります。

- アクセスモード
- 独立性水準

1.1.3.1 アクセスモード

トランザクションのアクセスモードでは、実行できるSQL文の種類を設定します。アクセスモードには、以下の2種類があります。

READ ONLY:

読込み専用モードです。トランザクション内で参照系のSQL文のみ、実行することができます。

READ WRITE:

更新可能モードです。トランザクション内で参照系および更新系のSQL文を実行することができます。

アクセスモードの初期値は、動作環境ファイルのDEFAULT_ACCESS_MODEの指定により決まります。

1.1.3.2 独立性水準

独立性水準では、あるトランザクションがアクセスしているデータに対し、他のトランザクションによる参照または更新を制御します。また、他のトランザクションが更新中の資源に対し、自トランザクションが参照または更新できるかを制御します。

独立性水準には以下の4種類があります。

- SERIALIZABLE
- REPEATABLE READ
- READ COMMITTED
- READ UNCOMMITTED

独立性水準の初期値は、動作環境ファイルのDEFAULT_ISOLATIONの指定により決まります。ただし、動作環境ファイルのR_LOCK指定により、独立性水準は以下のようになります。

R_LOCK	DEFAULT_ISOLATION	独立性水準
NO	REPEATABLE_READ	SERIALIZABLE
	上記以外	DEFAULT_ISOLATIONの指定に従う
YES	SERIALIZABLE	REPEATABLE READ
	上記以外	DEFAULT_ISOLATIONの指定に従う

独立性水準は、SET TRANSACTION文で変更可能です。

ただし、動作環境ファイルのR_LOCKの指定により、独立性水準は以下のようになります。

R_LOCK	SET TRANSACTION文	独立性水準
NO	REPEATABLE_READ	SERIALIZABLE
	上記以外	SET TRANSACTION文の指定に従う
YES	SERIALIZABLE	REPEATABLE READ
	上記以外	SET TRANSACTION文の指定に従う

動作環境ファイルでDSO_LOCKを指定した場合、SET TRANSACTION文は指定できません。

参考

SET TRANSACTION文の詳細は、“SQLリファレンス”を参照してください。

あるトランザクションの実行中に現れる可能性のある現象を以下に示します。

・ 現象1)

トランザクションT1が行を更新する。T1がトランザクションを終了する前に、トランザクションT2がその行を読み込む。T1がROLLBACK文を実行した場合、T2は存在しなかった行を読むことになる。

・ 現象2)

トランザクションT1が行を読み込む。トランザクションT2が、その行を更新または削除し、COMMIT文を実行する。T1がその行を再び読もうとした場合、更新した値を受け取るかまたはその行が削除されたことを知る。

・ 現象3)

トランザクションT1が、ある探索条件を満たす行の集合を読み込む。トランザクションT2が、その探索条件を満たす行を追加する。T1が、同じ探索条件で再度行の集合を読み込んだ場合、最初とは異なった行の集合を得ることになる。

独立性水準	現象1)	現象2)	現象3)
SERIALIZABLE	不可能	不可能	不可能
REPEATABLE READ	不可能	不可能	可能
READ COMMITTED	不可能	可能	可能
READ UNCOMMITTED	可能	可能	可能

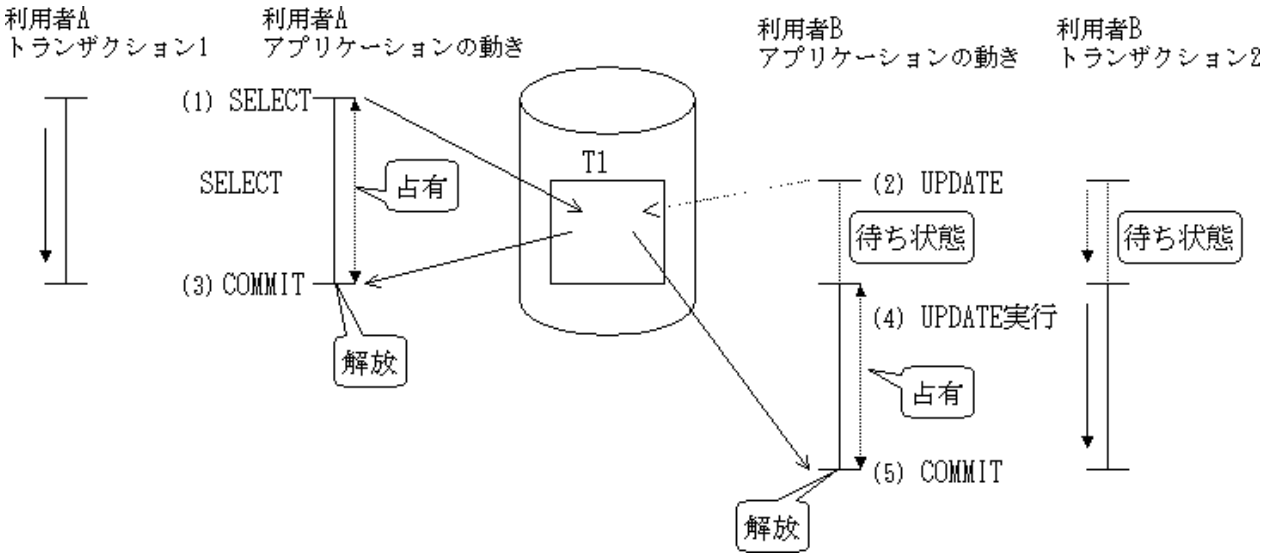
SERIALIZABLE

あるトランザクションが参照または更新した資源は、そのトランザクションが終了するまで、他のトランザクションから更新されないことが保証されます。

動作環境ファイルのパラメタR_LOCKで行単位の排他が指定されていない場合に有効です。

SERIALIZABLEの場合の排他処理を以下に示します。

図1.6 SERIALIZABLEの場合の排他



(1) 利用者AがCOMMIT(ROLLBACK)終了するまで資源を占有します。

(2) 利用者Bは、利用者Aが資源を占有しているため、待ち状態となります。

(3) 利用者AがCOMMIT(ROLLBACK)するので、利用者Aの資源は解放されます。

(4) 利用者Aが資源を解放したので、利用者Bの更新が実行されます。このとき、資源を占有します。

(5) 利用者BがCOMMIT(ROLLBACK)するので、利用者Bの資源は解放されます。

REPEATABLE READ

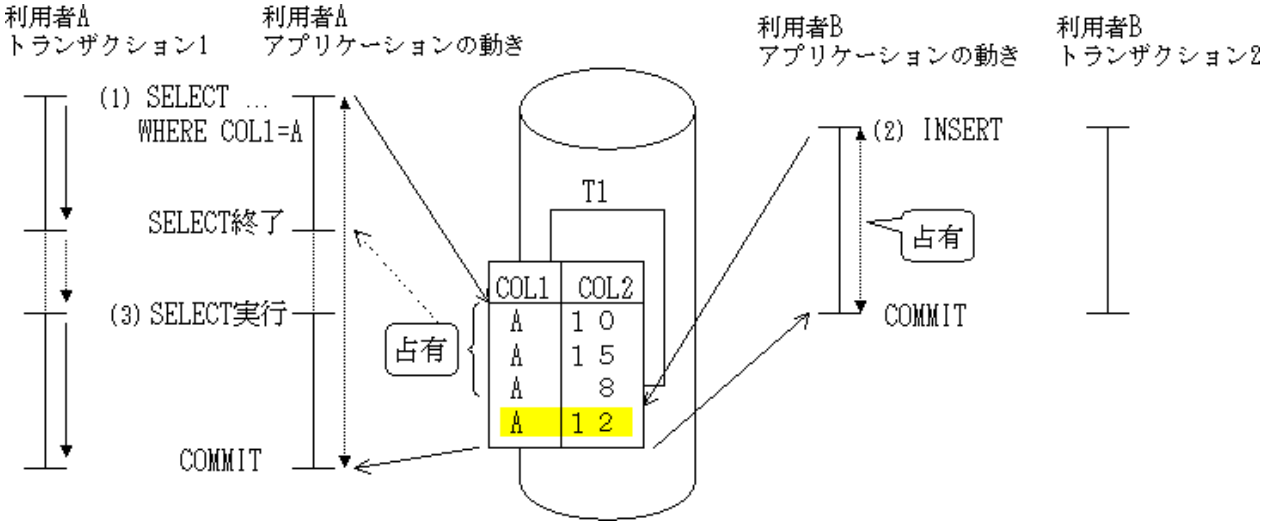
あるトランザクションが参照または更新した資源は、そのトランザクションが終了するまで、他のトランザクションからの更新がないことが保証されます。

動作環境ファイルのR_LOCKパラメタで行単位の排他が指定されている場合に有効です。

排他の単位が行単位でかつREPEATABLE READが指定された場合は、該当のトランザクションがアクセスしていない行は占有しないので、同一トランザクション内で、同一のSQL文をくり返し実行しても、異なる集合が検索される場合があります。

REPEATABLE READの場合の排他処理を以下に示します。

図1.7 REPEATABLE READの場合の排他



(1) 利用者Aは、COL1がAの行を参照します。トランザクションが終了するまでこの行の集合を占有します。

(2) 利用者Bは、利用者Aが参照した行に対しては更新することができませんが、INSERT文で行を追加することはできます。

(3) 利用者Aが、再びCOL1がAの行を参照します。このとき前回参照した行に加えて、利用者Bが追加した行も検索されます。

READ COMMITTED

あるトランザクションが更新した資源は、そのトランザクションが終了するまで、他のトランザクションから更新されないことが保証されます。しかし、以下のいずれかのSQL文で参照した資源に対しては、他のトランザクションから更新される可能性があります。

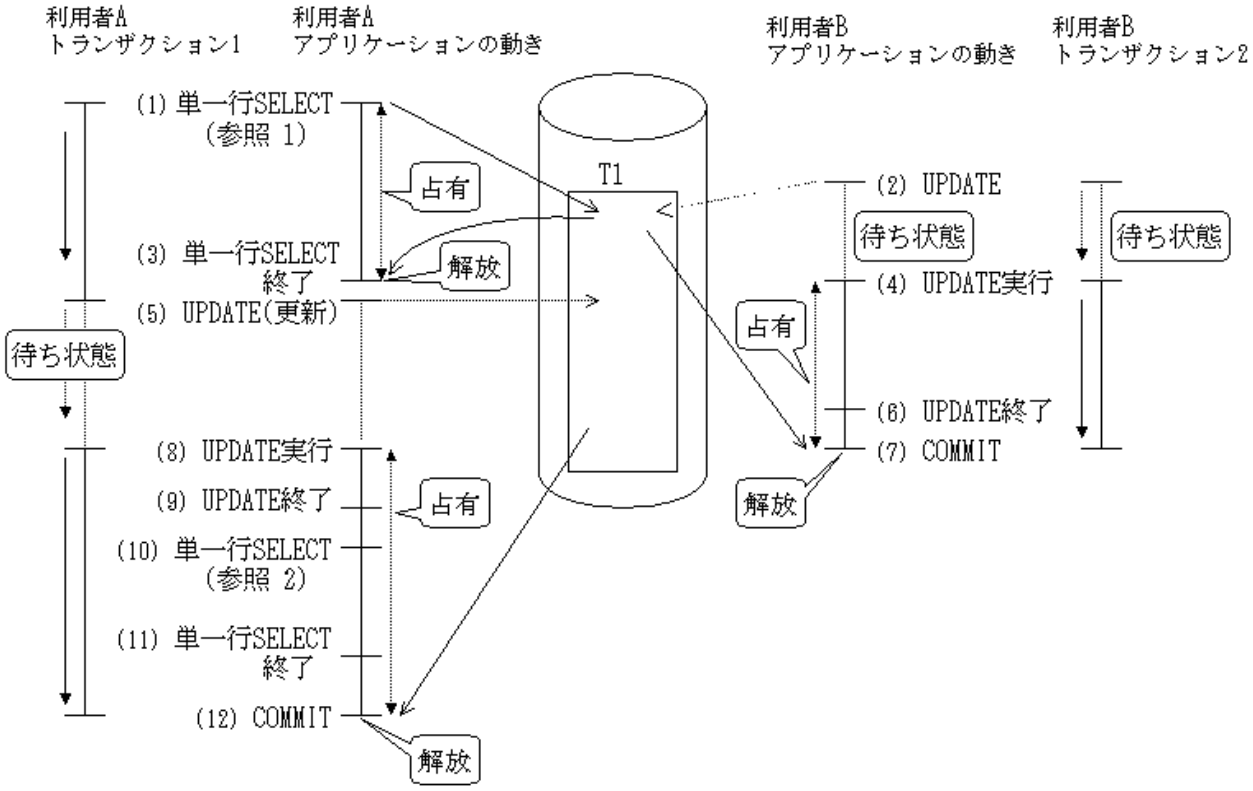
- 単一行SELECT文
- 更新可能性句に、FOR READ ONLYを指定したカーソルのOPEN文
- トランザクションアクセスモードが読込み専用モードのときのOPEN文

したがって、同一トランザクション内でも、テーブル参照ごとに内容が異なる場合があります。

DSO定義でPRECEDENCE(1)が指定されたSEQUENTIAL構造の表にアクセスするトランザクションにおける、READ COMMITTEDの場合の制御については、“1.1.12 PRECEDENCE(1)を指定したSEQUENTIAL構造の特徴と注意事項”を参照してください。

READ COMMITTEDの場合の排他処理を以下に示します。

図1.8 READ COMMITTEDの場合の排他



(1) 利用者Aが参照しているため(参照1)、この参照が終了するまで資源を占有します。

(2) 利用者Aが資源を解放するまで、利用者Bは更新を行えず、待ち状態となります。

(3) 利用者Aが参照終了(参照1)し、資源を解放します。

(4) 利用者Aが資源を解放したので、利用者Bの更新が実行されます。このとき、資源を占有します。

(5) 利用者Bが資源を解放するまで利用者Aは更新(更新)を行えず、待ち状態となります。

(6) 利用者Bの更新が終了します。

(7) 利用者Bがトランザクションを終了し、資源を解放します。

(8) 利用者Bが資源を解放したので、利用者Aの更新(更新)が実行されます。このとき、資源を占有します。

(9) 利用者Aが更新終了(更新)しますが、資源の占有は続きます。

(10) 利用者Aが参照(参照2)を行います。このとき検索される行は、利用者Bが更新した行を含みます。

(11) 利用者Aが参照(参照2)を終了しますが、資源の占有は続きます。

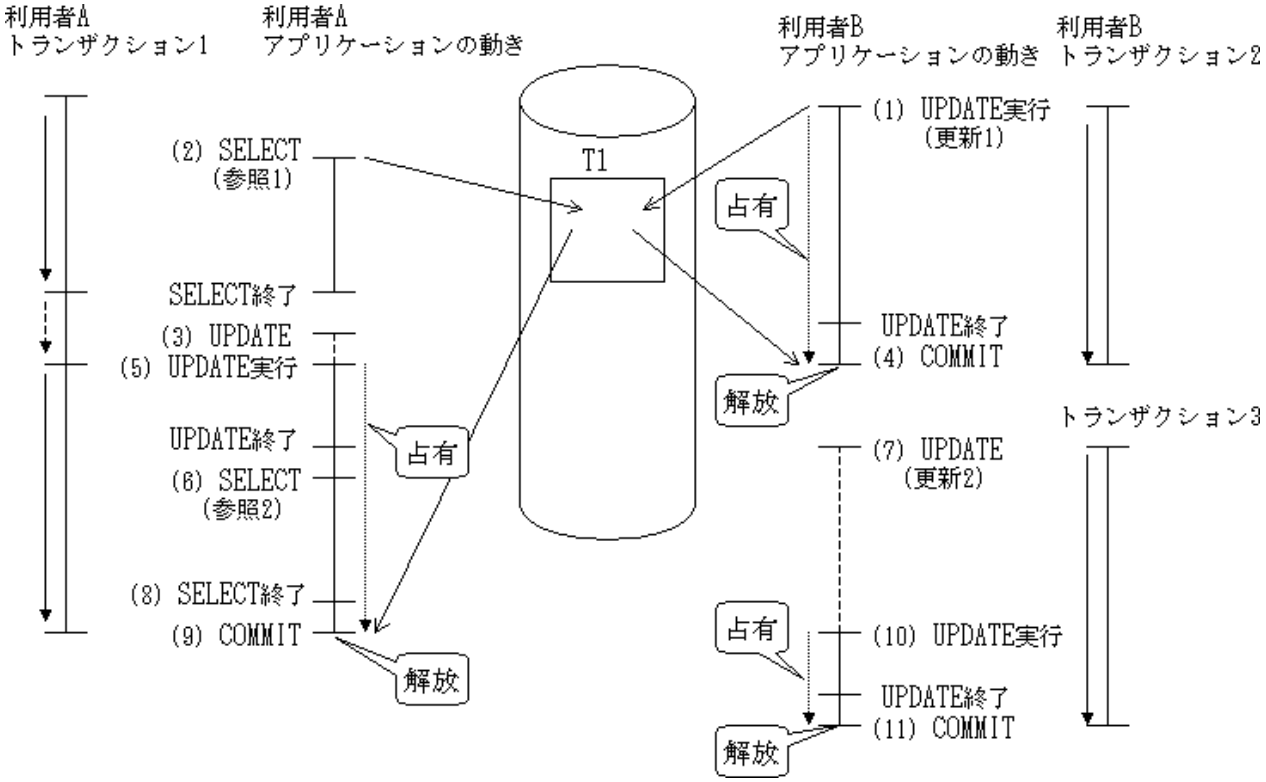
(12) 利用者Aがトランザクションを終了します。これによって資源が解放されます。

READ UNCOMMITTED

あるトランザクションが参照した資源は、他のトランザクションの参照および更新をまったく制限しません。また他のトランザクションが更新した資源も、トランザクションの終了を待たずに参照することができます。トランザクションが更新した資源は、トランザクション終了まで他のトランザクションの参照および更新を待たせます。

READ UNCOMMITTEDの場合の排他処理を以下に示します。

図1.9 READ UNCOMMITTEDの場合の排他



(1) 利用者Bは、更新処理(更新1)のため、COMMIT(ROLLBACK)終了するまで資源を占有します。

(2) 利用者Aは、資源を占有しないため、利用者Bが資源を占有していても、待ち状態にならずに参照処理(参照1)を行います。

(3) 利用者Aは利用者Bが資源を解放するまで更新を行えず、待ち状態となります。

(4) 利用者BがCOMMIT(ROLLBACK)します。利用者Bの資源は解放されます。

(5) 利用者Bが資源を解放したので、利用者Aの更新が実行されます。このとき、実際に更新した資源を占有します。

(6) 利用者Aが参照(参照2)を行います。このとき検索される行は、利用者Bが更新した行を含みます。

(7) 利用者Bは利用者Aが資源を解放するまで更新を行えず、待ち状態となります。

(8) 利用者Aが参照(参照2)を終了しますが、資源の占有は続きます。

(9) 利用者AがCOMMIT(ROLLBACK)します。

(10) 利用者Aが資源を解放したので、利用者Bの更新(更新2)が実行されます。このとき、資源を占有します。

(11) 利用者BがCOMMIT(ROLLBACK)します。

1.1.4 トランザクションとカーソル操作

データ操作において、操作の対象とする行を指示する指示子のことを“カーソル”と呼びます。

カーソルについての詳細は、“第7章 カーソルを使用するデータ操作の方法”を参照してください。

カーソルは、一般に、トランザクション終了時にクローズされます。しかし、次の場合、カーソルはクローズされません。

- トランザクションの独立性水準指定がREAD COMMITTEDで、カーソル指定の更新可能性句がFOR READ ONLYの場合
- カーソル宣言に、カーソルモード指定がある場合

以下に独立性水準指定にREAD COMMITTEDを指定し、更新可能性句にFOR READ ONLYを指定して実現する、トランザクションを超越するカーソル操作のSQL文の例を示します。

```
EXEC SQL DECLARE CSR1 CURSOR FOR                                    ……(1)
                 SELECT * FROM 取引先会社
                 WHERE  電話番号 LIKE  :hcomp_word
                 FOR  READ  ONLY;

EXEC SQL SET TRANSACTION  READ  ONLY,                               ……(2)
                          ISOLATION LEVEL  READ  COMMITTED;
EXEC SQL OPEN  CSR1;                                                ……(3)
EXEC SQL FETCH CSR1 INTO :hnumber,:hproduct,:hstock;                ……(4)
           :
EXEC SQL COMMIT WORK;                                               ……(5)


EXEC SQL FETCH CSR1 INTO :hnumber,:hproduct,:hstock;                ……(6)
           :
      （データの終了）

EXEC SQL CLOSE CSR1;                                                ……(7)
```

(1) カーソル“CSR1”を宣言します。更新可能性句に“FOR READ ONLY”を設定します。

(2) トランザクションアクセスモードに“READ ONLY”を設定します。独立性水準に“READ COMMITTED”を設定します。

(3) カーソル“CSR1”をオープンします。

(4) 探索条件が真となる行にカーソルを位置づけ、その行を読み込みます。

(5) COMMITによりトランザクションを終了します。

(6) (5)でカーソルがクローズされないため、カーソルを位置づけることができます。

(7) データ終了後、カーソル“CSR1”をクローズします。

また、以下にカーソルモード指定によって実現する、トランザクションを超越するカーソル操作のSQL文の例を示します。

```
EXEC SQL DECLARE CSR1 CURSOR FOR                                      ……(1)
         SELECT * FROM  取引先会社
         WHERE 電話番号 LIKE :hcomp_word
         WITH OPTION LOCK_MODE(FREE LOCK), CURSOR_MODE(HOLD);

EXEC SQL OPEN CSR1;                                                   ……(2)

EXEC SQL FETCH CSR1 INTO :hnumber,:hproduct,:hstock;                  ……(3)
                    :
EXEC SQL COMMIT WORK;                                                 ……(4)

EXEC SQL FETCH CSR1 INTO :hnumber,:hproduct,:hstock;                  ……(5)
                    :
               （データの終了）
EXEC SQL CLOSE CSR1;                                                  ……(6)
```

(1) カーソル“CSR1”を宣言します。カーソルモード指定に“HOLD”を設定します。

(2) カーソル“CSR1”をオープンします。

(3) 探索条件が真となる行にカーソルを位置づけ、その行を読み込みます。

(4) COMMITによりトランザクションを終了します。

(5) (4)でカーソルがクローズされないため、カーソルを位置づけることができます。

(6) データ終了後、カーソル“CSR1”をクローズします。

トランザクションを超越するカーソルを使用する場合の注意事項

トランザクションを超越するカーソルを使用する場合、カーソルのオープン中に、そのカーソルが参照する表の定義や格納構造定義が変更されると、次のトランザクションでFETCH文の実行が、SQLSTATE値が24000(カーソル不当)のエラーとなります。このエラーが発生した場合は、カーソルをクローズして、再度オープンするようにしてください。

1.1.5 SQL文の処理結果異常とトランザクション

データ操作のSQL文の処理中に異常が発生した場合には、アプリケーションにエラーで復帰します。エラーで復帰したときのトランザクションの状態は、エラーの原因によって異なります。エラーの原因を示すコードは、状態変数“SQLSTATE”に設定されます。

トランザクションがロールバックされるエラーの場合のSQLSTATE:

40xxx

トランザクションが継続されるエラーの場合のSQLSTATE:

40xxx以外

トランザクションが継続されるエラー

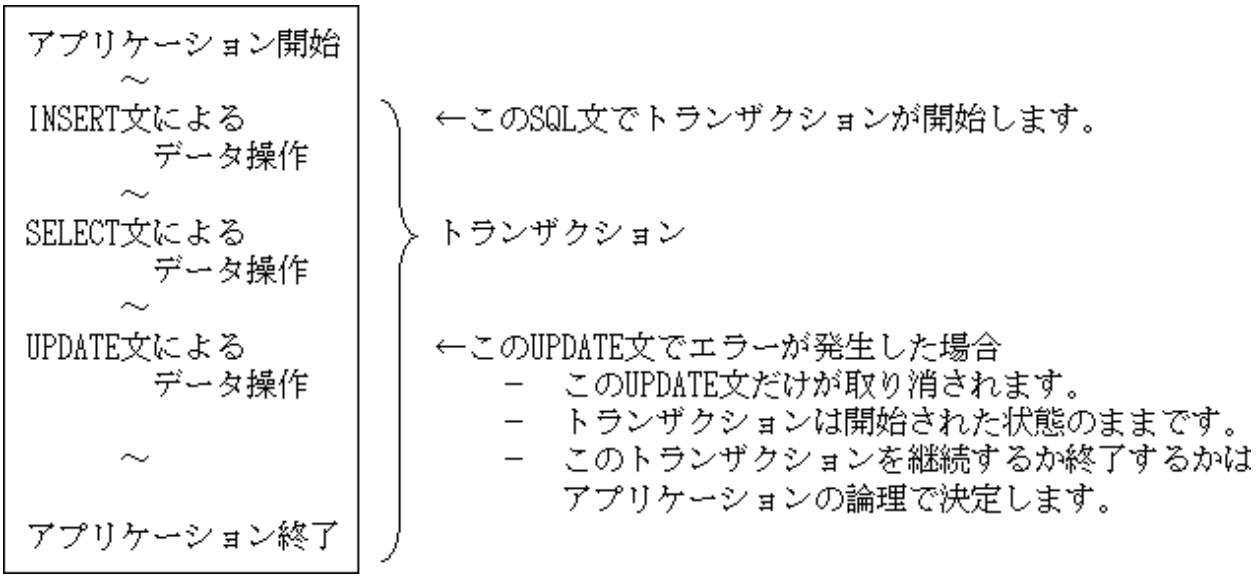
エラーが発生したSQL文による更新は、そのSQL文を実行する直前の状態に戻されてアプリケーションへ復帰します。ただし、このSQL文の実行までに獲得されたデータベース資源の占有とカーソルの位置はそのまま保持されます。このとき、トランザクションは継続されたままになっています。トランザクションをそのまま継続させるか終了させるかは、エラーの原因ごとに利用者がアプリケーションの論理に従って決定します。たとえば、INSERT文で表にデータを繰り返し挿入するような処理では、あるデータが表の一意性制約に違反するエラーとなっても、そのデータを無視して引き続きデータを挿入すればよい場合には、トランザクションは継続させて処理を続行します。また、互いに関連のある複数の表を更新するような処理では、ある表の更新時にエラーが発生した場合、表の間での整合を保つために、ROLLBACK文を実行してトランザクションをロールバックします。

なお、動作環境ファイルのパラメタ“TRAN_SPEC”の指定により、トランザクションの対処方法を変更することができます。

動作環境ファイルについては、“2.1.1 動作環境ファイル”を参照してください。

データ操作のSQL文の実行でエラーが発生した場合の例を以下に示します。

図1.10 データ操作のSQL文の実行でエラーが発生した場合の例



トランザクションがロールバックされるエラー

トランザクションはロールバックされ終了してアプリケーションへ復帰するため、このSQL文の実行までに獲得されたデータベース資源のロックは解放され、オープン中のカーソルは閉じられます。

デッドロックが発生した場合のアプリケーションの対処方法については、“1.1.10 デッドロックの対処方法”で説明します。

1.1.6 アプリケーションの異常終了とトランザクション

アプリケーションは実行中に異常終了することがあります。このとき、トランザクションが開始状態であれば、トランザクションはロールバックされて終了します。アプリケーションが異常終了するケースには以下のような場合があります。

- アプリケーションの実行環境の定義の誤り
- アプリケーションの誤りによる異常終了

1.1.7 排他制御

アプリケーションからデータベースへアクセスする場合には、トランザクション単位にSymfoware/RDBで排他制御を行います。

データベースの同一資源に対して、アプリケーションおよびRDBコマンドが同時に動作した場合のSymfoware/RDBの排他制御の詳細については、“付録D アプリケーションおよびRDBコマンド間の排他”を参照してください。

DSO定義でPRECEDENCE(1)が指定されたSEQUENTIAL構造の表の排他制御については、“1.1.12 PRECEDENCE(1)を指定したSEQUENTIAL構造の特徴と注意事項”を参照してください。

1.1.7.1 排他の属性と選択方法

排他の属性には、以下のものがあります。

- 占有の資源
- 占有の単位
- 占有の範囲
- 占有モード
- 占有待ちの有無

以下に、排他の属性の指定方法と、排他の属性の各項目の概要を説明します。

排他の属性の指定方法

排他の属性を指定するには、以下の方法があります。

- 動作環境ファイルの実行パラメタによる指定

  動作環境ファイルの実行パラメタによって設定できる属性は、以下のとおりです。

  - DSO_LOCK:

    資源の占有モードおよび占有単位について指定します。指定されたDSO配下の全DSIに対する占有モードおよび占有単位となります。

  - ISOLATION_WAIT:

    占有待ちの有無について指定します。

  - R_LOCK:

    占有単位について指定します。

  - DEFAULT_ACCESS_MODE:

    トランザクションアクセスモードの初期値を指定します。
    DEFAULT_ACCESS_MODEが指定された場合、プロセスで最初に実行されるSQL文の直前でSET TRANSACTION文が実行されたことになります。

  - DEFAULT_ISOLATION:

    独立性水準の初期値を指定します。
    DEFAULT_ISOLATIONが指定された場合、プロセスで最初に実行されるSQL文の直前でSET TRANSACTION文が実行されたことになります。

参照

各パラメタの記述形式については、“アプリケーション開発ガイド(埋込みSQL編)”の“クライアント用の動作環境ファイルによる定義”を参照してください。

- SET TRANSACTION文による指定

  アプリケーション中にSET TRANSACTION文の独立性水準指定によって、占有の強さについて設定することができます。

参照

SET TRANSACTION文の詳細は、“SQLリファレンス”を参照してください。

- Symfoware/RDBによる選択

  動作環境ファイルの実行パラメタや、SET TRANSACTION文が指定されなかった場合、Symfoware/RDBがSQL文を解釈し、その結果をもとに選択します。

- SQL文の占有モード指定による指定

  SQL文に占有モード指定をすることにより、SQL文単位に読み込んだデータベース資源(データおよびインデックス)に対して占有モードを設定することができます。この機能を、イルシデーションロックといいます。

参照

占有モード指定の詳細は、“SQLリファレンス”を参照してください。

ポイント

イルシデーションロック(elucidation lock)とは、占有モード指定により、SQL文単位に読み込んだ資源の占有方法を決めて、明示的に排他制御を行う機能です。

占有の資源

占有する対象の資源は、そのアプリケーションで使用する表のDSIおよびインデックスのDSIです。

占有の単位

占有制御によって資源を占有する単位には、行単位、ページ単位およびDSI単位があります。

行単位の占有の場合には、アプリケーションがアクセスするデータを含む行が占有の対象になります。このとき、データの参照または更新のための走査対象行も占有の範囲に含まれます。ページ単位の占有の場合には、アプリケーションがアクセスするデータを含むページが占有の対象になります。DSI単位の占有の場合には、アプリケーションがアクセスするデータを含むDSIが占有の対象になります。占有単位を指定するには、以下の方法があります。

- 動作環境ファイルの実行パラメタDSO_LOCKおよびR_LOCKでの指定
- Symfoware/RDBによる選択

動作環境ファイルの実行パラメタDSO_LOCKおよびR_LOCKでの指定

占有単位は、動作環境ファイルの実行パラメタDSO_LOCKでDSO単位に指定するかまたは、R_LOCKで指定します。

動作環境ファイルの実行パラメタの指定と占有単位について以下に示します。

表1.2 動作環境ファイルの実行パラメタによる占有単位の選択方法

動作環境ファイルの実行パラメタ	ページ指定あり	ページ指定なし
DSO_LOCK	指定したDSOの全DSIをページ単位で占有	指定したDSOの全DSIをDSI単位で占有
R_LOCK	行単位で占有	

R_LOCKの値により、DSO_LOCKを指定できない場合があります。以下にDSO_LOCKとR_LOCKの関係を示します。

表1.3 DSO_LOCKとR_LOCKの関係

クライアント用の動作環境ファイルのR_LOCK	システム用の動作環境ファイルのR_LOCK	DSO_LOCKの指定	占有の単位
NO	NO	指定可	DSO_LOCKの指定による(注1)
	YES	指定可	DSO_LOCKの指定による(注1)
YES	NO	指定不可	行単位で占有(注2)
	YES	指定不可	行単位で占有(注2)
省略	NO	指定可	DSO_LOCKの指定による(注1)
	YES	指定不可	行単位で占有(注2)

注1) DSO_LOCKの指定を省略した場合、Symfoware/RDBによって占有の単位が選択されます。

注2) DSO_LOCKを指定した場合、CONNECT文実行時にエラーとなります。

Symfoware/RDBによる選択

動作環境ファイルでDSO_LOCKを指定せず、R_LOCKにNOを指定した場合は、Symfoware/RDBによって占有の単位が選択されます。

Symfoware/RDBは、SQL文ごとに、そのSQL文でアクセスするデータの範囲(ページ数)を算定します。その範囲(ページ数)の大きさに応じて、以下に示すように占有単位を選択しています。

表1.4 Symfoware/RDBによる占有単位の選択条件

占有単位	選択条件
ページ	アクセスするデータの範囲が小さい場合
DSI	アクセスするデータの範囲が大きい場合

たとえば、表を全件処理する場合には、表のデータを格納している全DSIに対してDSI単位の占有を選択します。また、SQL文の探索条件を解析し、アクセスするページが少ないと算定した場合には、ページ単位の占有を選択します。なお、アクセスするデータの量が少なくても、その範囲(ページ数)が大きい場合には、DSI単位の占有を選択します。

占有の範囲

表やインデックスの占有範囲はその資源の検索範囲となります。表の検索範囲は、SQL文の探索条件をインデックスで評価して絞り込んだ部分です。このため、インデックスを使用しない場合は、探索条件の評価結果に関わらず、表の全範囲が占有対象となります。インデックスの検索範囲については、“H.2.1 SCANエレメント”の“4) INDEX KEY SCAN”を参照してください。

なお、DSI分割している場合は資源の検索の前にデータ操作範囲を限定します。データ操作範囲を限定するには、以下の方法があります。

- 探索条件に分割キーを指定

- アプリケーションのチューニング

> **参照**
>
> アプリケーションのチューニングによるデータ操作範囲の限定方法については、“アプリケーション開発ガイド(埋込みSQL編)”の“データ操作範囲の限定方法”を参照してください。

占有モード

データベースにアクセスする場合の占有モードには、共有モードと非共有モードの2種類があります。

共有モードの場合、ロックされたデータは、ほかのトランザクションからのデータ操作において、参照だけが許され更新操作はできません。一般に、データのロックを獲得中のトランザクションのコミットが行われるまで、ほかのトランザクションのデータの更新操作は、待ち状態となります。非共有モードの場合、ロックされたデータは、ほかのトランザクションでデータ操作することはできません。一般に、データのロックを獲得中のトランザクションのコミットが行われるまで、ほかのトランザクションの実行は、待ち状態となります。

占有モードを指定するには、以下の方法があります。

- 動作環境ファイルの実行パラメタDSO_LOCKでの指定
- Symfoware/RDBによる選択
- 動作環境ファイルの実行パラメタDEFAULT_ACCESS_MODEおよびDEFAULT_ISOLATION、またはSET TRANSACTION文での指定
- SQL文における占有モード指定での指定

動作環境ファイルの実行パラメタDSO_LOCKでの指定

アプリケーションの動作環境ファイルの実行パラメタDSO_LOCKを指定すると、データ操作のSQL文を実行したとき、データおよびインデックスに対する占有モードを指定することができます。

データに対する占有

データに対する占有モードは、以下に示すように、DSO単位に指定します。

表1.5 動作環境ファイルの実行パラメタによる占有モードの指定

DSO_LOCKでの指定	占有モード
SHを指定した場合	共有モード
EXを指定した場合、または省略した場合	非共有モード

インデックスに対する占有

インデックスに対する占有モードは、以下に示すように、インデックスのDSO単位に指定します。

表1.6 動作環境ファイルの実行パラメタによるインデックスに対する占有モードの指定

DSO_LOCKでの指定	占有モード
SHを指定した場合	共有モード
EXを指定した場合、または省略した場合	非共有モード

Symfoware/RDBによる選択

動作環境ファイルでDSO_LOCKを指定しなかった場合は、Symfoware/RDBによって占有モードが選択されます。

データに対する占有

Symfoware/RDBは、アクセスするデータに対して、以下に示す条件で、SQL文ごとにデータに対する占有モードを選択します。

表1.7 Symfoware/RDBによる占有モードの選択

占有モード	選択条件
共有モード	データを参照する処理の部分の場合
非共有モード	データを更新する処理の部分の場合

データ検索を行う以下のSQL文については、アクセスするデータに対する占有モードは共有モードになります。また、これらのSQL文については、インデックスに対する更新が行われないので、インデックスに対するモードは共有モードになります。

- OPEN文(ただし、カーソル宣言の更新可能性句にFOR UPDATEを指定したカーソルのOPEN文は、非共有モードで占有します。)
- 単一行SELECT文
- FETCH文

データの更新、削除および挿入を行う以下のSQL文については、処理(更新、削除および挿入)対象のデータに対する占有モードは、非共有モードになります。ただし、データ更新、削除および挿入を行うSQL文であっても、UPDATE文やDELETE文の更新対象のデータを位置づけるための検索やINSERT文に指定した問合せ指定の検索に関しては、共有モードになります。

- UPDATE文:探索
- UPDATE文:位置づけ
- DELETE文:探索
- DELETE文:位置づけ
- INSERT文

インデックスに対する占有

Symfoware/RDBは、SQL文の処理で使用するインデックスの更新の有無により、以下のように選択します。

表1.8 Symfoware/RDBによるインデックスに対する占有モードの選択

インデックスに対する占有モード	インデックス更新の有無
共有モード	インデックス更新なし
非共有モード	インデックス更新あり

データ更新を行う以下のSQL文については、更新対象の列に設定されているインデックスに対して更新が行われます。したがって、そのようなインデックスに対する占有モードは非共有モードになります。一方、更新対象でない列に設定されているインデックスに対しては、更新は行われないので、そのようなインデックスに対する占有モードは共有モードになります。

- UPDATE文:探索
- UPDATE文:位置づけ

データ削除および挿入を行う以下のSQL文については、更新対象の表に張られているインデックスに対して更新が行われます。したがって、そのようなインデックスに対する占有モードは非共有モードになります。

- DELETE文:探索
- DELETE文:位置づけ
- INSERT文

動作環境ファイルの実行パラメタDEFAULT_ACCESS_MODEおよびDEFAULT_ISOLATION、またはSET TRANSACTION文での指定

動作環境ファイルの実行パラメタDEFAULT_ACCESS_MODEおよびDEFAULT_ISOLATION、またはSET TRANSACTION文での指定については、“1.1.3 トランザクションモード”を参照してください。

SQL文における占有モード指定での指定

SQL文に占有モード指定を指定することにより、SQL文が読み込んだ資源(データおよびインデックス)の占有モードと占有期間を柔軟に変更することができます。占有の単位は、アプリケーションの動作環境ファイルの実行パラメタR_LOCKがYESの場合、行単位となります。R_LOCKがNOの場合は、ページ単位またはDSI単位になります。

占有モード指定がある場合、SET TRANSACTION文で指定された内容とは無関係に動作します。また、アプリケーションの動作環境ファイルの実行パラメタDSO_LOCKは、指定できません。

占有モード指定と資源の占有について以下に示します。

表1.9 占有モード指定と資源の占有

占有モード指定	占有のしかたと占有期間	他のトランザクションとの待ち関係	読込み水準
EXCLUSIVE LOCK	非共有モードで、トランザクション終了まで資源を占有する。	当該トランザクションが終了するまで、同じ資源を参照／更新する他のトランザクションを待たせる。	COMMIT済みの行を読み込む。当該SQL文で読み込んだ行は、トランザクション終了まで他のトランザクションに更新されることがないため、一度読み込んだ行は、他のトランザクションによって更新されないことが保証される。
SHARE LOCK	共有モードで、トランザクション終了まで資源を占有する。	SQL文の実行後は、当該トランザクションが終了するまで、同じ資源を更新する他のトランザクションを待たせる。	
FREE LOCK	共有モードで、SQL文終了(カーソルならば、OPEN文終了時)まで資源を占有する。	当該SQL文の実行中は、同じ資源を更新する他のトランザクションを待たせる。当該SQL文の実行が終了すると、同じ資源を更新する他のトランザクションを実行することができる。	COMMIT済みの行を読み込む。当該SQL文で読み込んだ行は、他のトランザクションに更新されることがあるため、同一トランザクションで再検索すると最新の結果を検索することができる。
NO LOCK	資源を占有しない。	当該SQL文を実行しても、他のトランザクションを待たせない。	他のトランザクションでCOMMITされていない更新中の行を読みことができる。

占有モード指定は、資源を読み込むときのモードです。読み込んだ資源の更新時には、占有モード指定での指定に関係なくEXCLUSIVE LOCKになります。

占有モード指定を指定したSQL文の待ち関係

占有モード指定を指定したSQL文が参照した資源に対する待ち関係を以下に示します。

更新した部分に関してはEXCLUSIVE LOCK指定がされたとみなして表を参照してください。

表1.10 占有モード指定と待ち関係

		先行トランザクション			
	占有モード指定	EXCLUSIVE LOCK	SHARE LOCK	FREE LOCK	NO LOCK
後続トランザクション	EXCLUSIVE LOCK	×	×	△	○
	SHARE LOCK	×	○	○	○
	FREE LOCK	×	○	○	○
	NO LOCK	○	○	○	○

○:待つことなく動作する

△:先行トランザクションのSQL文の終了を待つ

×:先行トランザクションの終了を待つ

図1.11 ○:待つことなく動作する場合

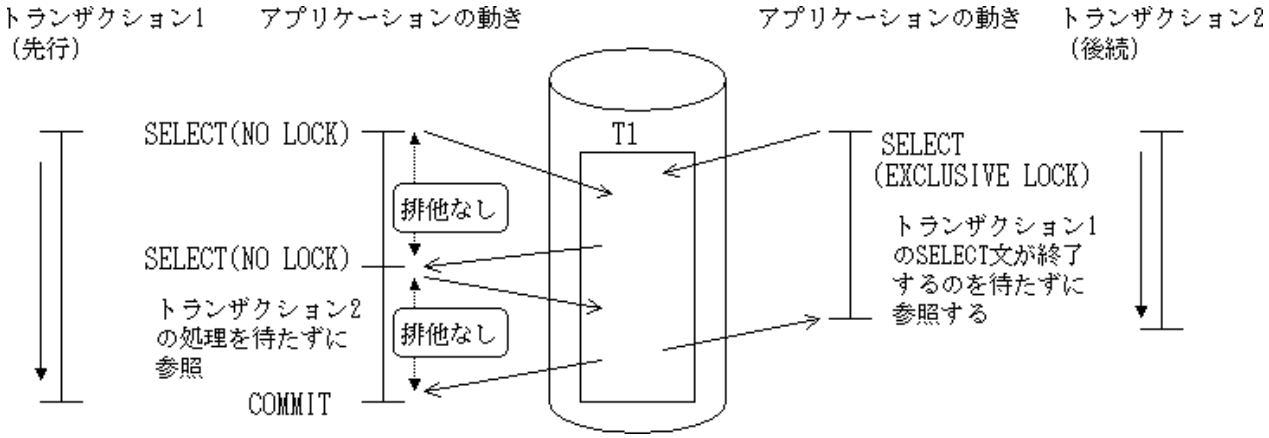


図1.12 △:先行トランザクションのSQL文の終了を待つ場合

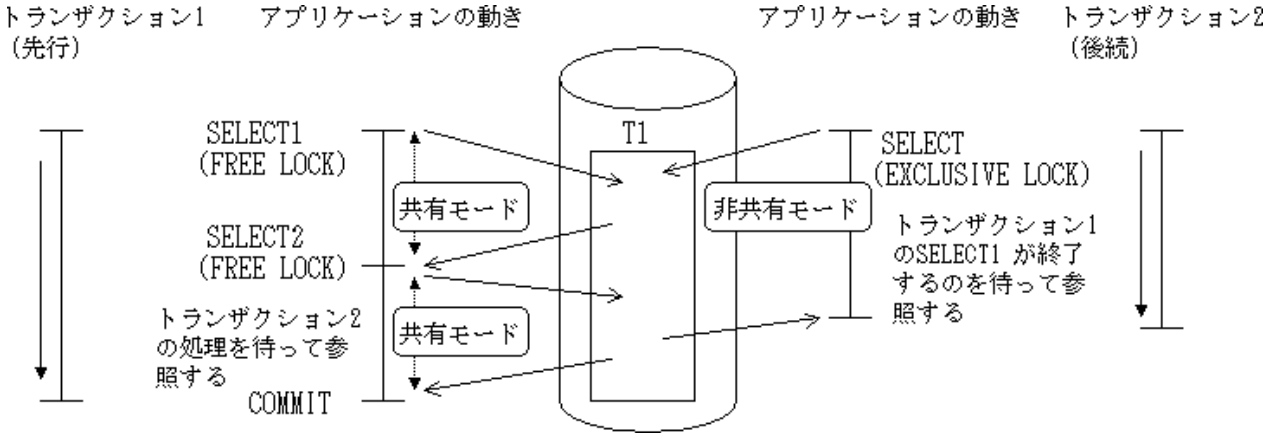
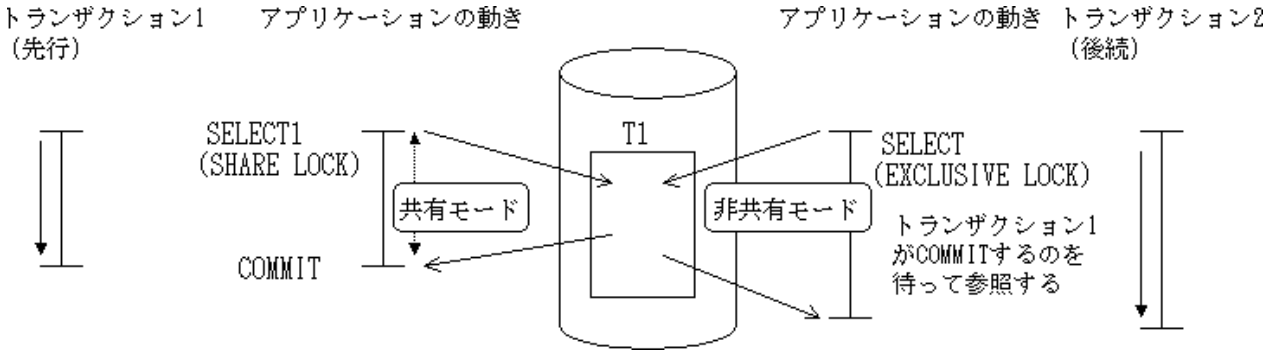


図1.13 ×:先行トランザクションの終了を待つ場合



カーソル宣言およびSET TRANSACTION文における指定とSQL文の占有モード指定における指定との対応関係

SQL文の占有モード指定を省略した場合、独立性水準、トランザクションアクセスモードおよび更新可能性句を組み合わせることによって、占有モード指定を指定したときと同等の資源の占有のしかたを設定できます。以下に、SET TRANSACTION文および更新可能性句での指定と、それに対応する占有モード指定を示します。

占有モード指定があるSQL文と、占有モード指定を省略したSQL文との待ち関係については、以下の表を利用して同等の占有モード指定を確認した後、“表1.10 占有モード指定と待ち関係”を参照してください。

- 占有モード指定を省略したSQL文がINSERT文、UPDATE文:位置づけおよびDELETE文:位置づけの場合

SET TRANSACTION文における指定		対応する占有モード指定
独立性水準	トランザクション アクセスモード	
SERIALIZABLE REPEATABLE READ READ COMMITTED READ UNCOMMITTED	READ WRITE	EXCLUSIVE LOCK

- 占有モード指定を省略したSQL文がUPDATE文:探索およびDELETE文:探索の場合

SET TRANSACTION文における指定		対応する占有モード指定
独立性水準	トランザクション アクセスモード	
SERIALIZABLE REPEATABLE READ READ COMMITTED	READ WRITE	SHARE LOCK (更新対象行を検索するとき) EXCLUSIVE LOCK (更新対象行を更新するとき)
READ UNCOMMITTED		NO LOCK (更新対象行を検索するとき) EXCLUSIVE LOCK (更新対象行を更新するとき)

ただし、クライアント用の動作環境ファイルの実行パラメタUSQL_LOCKにEXを指定した場合は、更新対象行を検索するときから、EXCLUSIVE LOCKで資源を占有します。

- 占有モード指定を省略したSQL文が、単一行SELECT文の場合

SET TRANSACTION文における指定		対応する占有モード指定
独立性水準	トランザクション アクセスモード	
SERIALIZABLE REPEATABLE READ	READ ONLY READ WRITE	SHARE LOCK
READ COMMITTED	READ ONLY READ WRITE	FREE LOCK
READ UNCOMMITTED	READ ONLY READ WRITE	NO LOCK

- 占有モード指定を省略したSQL文が、カーソル宣言の場合

SET TRANSACTION文における指定		カーソル宣言の更新可能性句	対応する占有モード指定
独立性水準	トランザクションアクセスモード		
SERIALIZABLE REPEATABLE READ	READ ONLY	FOR READ ONLY または 省略	SHARE LOCK
	READ WRITE	FOR UPDATE	EXCLUSIVE LOCK
		FOR READ ONLY または 省略	SHARE LOCK
READ COMMITTED	READ ONLY	FOR READ ONLY または 省略	FREE LOCK
	READ WRITE	FOR UPDATE	EXCLUSIVE LOCK
		FOR READ ONLY	FREE LOCK
		省略	SHARE LOCK
READ UNCOMMITTED	READ ONLY	FOR READ ONLY または 省略	NO LOCK
	READ WRITE	FOR UPDATE	EXCLUSIVE LOCK
		FOR READ ONLY または 省略	NO LOCK

SQL文単位の排他制御の例

SET TRANSACTION文によりトランザクションモードを変更する場合、一度トランザクションを終了させる必要があります。しかし、SQL文に占有モード指定を指定することにより、トランザクションを終了することなく、SQL文が読み込んだデータベース資源の占有状態を変更することができます。以下に、デッドロック抑止を目的とした占有モード指定の利用例を示します。

```
EXEC SQL SELECT * FROM  T1                                    …… (1)
         WHERE C1 = :H1
         WITH OPTION LOCK_MODE(EXCLUSIVE LOCK);

EXEC SQL UPDATE T1                                            …… (2)
     SET C2 = :H2  WHERE C1 = :H1;

EXEC SQL COMMIT WORK;                                         …… (3)
```

(1) デッドロック抑止を目的として、単一行SELECT文にEXCLUSIVE LOCKを指定します。参照時から資源を非共有モードで占有します。

(2) 単一行SELECT文によって位置づけられた行の列を更新します。

(3) COMMIT文によりトランザクションを終了します。

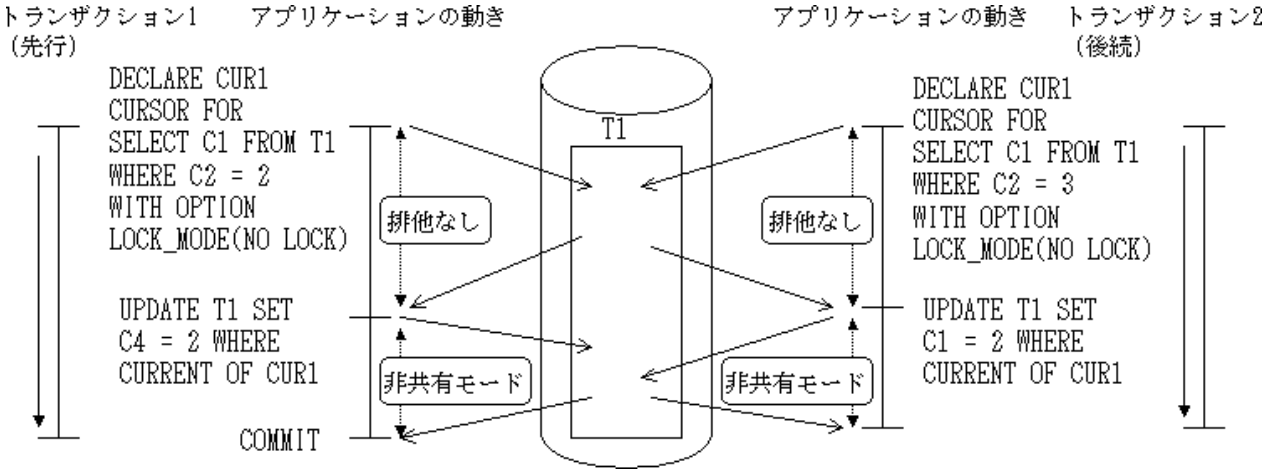
SQL文での占有モード指定の使用例

アプリケーションを多重実行する場合、更新処理があると非共有モードで資源を占有する必要があります。このため、他のアプリケーションが先に資源を占有している場合、そのアプリケーションのトランザクションが終了するまで、後続のアプリケーションは処理を待たされます。しかし、同一のレコードを同時に更新することがなければ、排

他なしで参照し、実際に更新するレコードのみ排他制御を行えばよいので、システム全体の処理スピードを向上できます。

お互い同一のレコードを同時に更新しない場合は、レコード位置づけ処理において他のアプリケーションのトランザクションを待たないように、カーソル宣言の占有モード指定にNO LOCKを指定します。

図1.14 SQL文での占有モード指定



占有待ちの有無

トランザクション間で資源(行、ページ、DSI)に対して競合が発生した場合、その時点で占有待ちに入る(資源が解放されるまで待つ)場合と、アプリケーションに復帰する場合があります。

Symfoware/RDBによる選択

占有待ちを行います。

動作環境ファイルの実行パラメタでの選択方法

アプリケーションの動作環境ファイルの実行パラメタISOLATION_WAITで、以下に示すように指定します。

表1.11 動作環境ファイルの実行パラメタでの占有待ちの有無の選択方法

占有待ちの有無	選択方法
占有待ちになる	WAITを指定した場合、または省略した場合
アプリケーションに復帰する	REJECTを指定した場合

排他制御に関するパラメタの相互関係

排他制御に関する各パラメタの指定と、占有の単位および占有のふるまいについて、以下に示します。

表1.12 排他制御に関するパラメタの相互関係

R_LOCKの値	DSO_LOCKの指定	SET TRANSACTION文の指定	占有モード指定の指定	占有のふるまい	占有の単位
YES	×	○	○	占有モード指定での指定に従う	行単位
			-	SET TRANSACTION文の独立性水準に従う。ただし、独立性水準にSERIALIZABLEを指定した場合は、REPEATABLE READになる	

<table>
<tr><th>R_LOCK
の値</th><th>DSO_LOCK
の指定</th><th>SET
TRANSACTION文の指定</th><th>占有モード
指定の指定</th><th>占有のふるまい</th><th>占有の単位</th></tr>
<tr><td rowspan="2"></td><td rowspan="2"></td><td rowspan="2">-</td><td>○</td><td>占有モード指定での指定に従う</td><td rowspan="2"></td></tr>
<tr><td>-</td><td>独立性水準は、REPEATABLE READになる</td></tr>
<tr><td rowspan="5">NO</td><td>○</td><td>×</td><td>×</td><td>独立性水準は、SERIALIZABLEになる</td><td>DSO_LOCKの指定に従う</td></tr>
<tr><td rowspan="4">-</td><td rowspan="2">○</td><td>○</td><td>占有モード指定での指定に従う</td><td rowspan="4">Symfoware/RDBによって自動的に決定される</td></tr>
<tr><td>-</td><td>SET TRANSACTION文の独立性水準に従う。ただし、独立性水準にREPEATABLE READを指定した場合は、SERIALIZABLEになる</td></tr>
<tr><td rowspan="2">-</td><td>○</td><td>占有モード指定での指定に従う</td></tr>
<tr><td>-</td><td>独立性水準は、SERIALIZABLEになる</td></tr>
</table>

○：指定可能

×：指定できません

－：指定を省略

1.1.8 資源の競合が起きた場合の制御

資源の競合が起きた場合に、どのタイミングまで占有待ちとなるのかは、トランザクションモードの指定によって異なります。以下に、トランザクションモードの指定と資源の占有について示します。

DSO定義でPRECEDENCE(1)が指定されたSEQUENTIAL構造の表の競合時の制御については、“1.1.12 PRECEDENCE(1)を指定したSEQUENTIAL構造の特徴と注意事項”を参照してください。

表1.13 占有の単位が行の場合

			先行トランザクション											
			独立性水準	SERIALIZABLE REPEATABLE READ				READ COMMITTED				READ UNCOMMITTED			
			SQL文の種類	SELECT	INSERT	UPDATE	DELETE	SELECT	INSERT	UPDATE	DELETE	SELECT	INSERT	UPDATE	DELETE
後続トランザクション	独立性水準	SQL文の種類													
	SERIALIZABLE REPEATABLE READ READ COMMITTED	SELECT		○	×	×	×	○	×	×	×	○	×	×	×
		INSERT		○	○	○	○	○	○	○	○	○	○	○	○
		UPDATE		×	×	×	×	△	×	×	×	○	×	×	×
		DELETE		×	×	×	×	△	×	×	×	○	×	×	×
	READ UNCOMMITTED	SELECT		○	○	○	○	○	○	○	○	○	○	○	○
		INSERT		○	○	○	○	○	○	○	○	○	○	○	○
		UPDATE		×	×	×	×	△	×	×	×	○	×	×	×
		DELETE		×	×	×	×	△	×	×	×	○	×	×	×

○：待つことなく動作する

△：先行トランザクションのSQL文が以下のいずれかの場合は、先行トランザクションのSQL文の終了を待つ

- 単一行SELECT文
- 更新可能性句に、FOR READ ONLYを指定したカーソルのOPEN文
- トランザクションアクセスモードが読込み専用モードのときのOPEN文

先行トランザクションのSQL文が上記以外の場合は、トランザクションの終了を待つ

×：トランザクションの終了を待つ

表1.14 占有の単位がページまたはDSIの場合

			先行トランザクション											
			独立性水準	SERIALIZABLE REPEATABLE READ				READ COMMITTED				READ UNCOMMITTED			
			SQL文の種類	SELECT	INSERT	UPDATE	DELETE	SELECT	INSERT	UPDATE	DELETE	SELECT	INSERT	UPDATE	DELETE
後続トランザクション	独立性水準	SQL文の種類													
	SERIALIZABLE REPEATABLE READ READ COMMITTED	SELECT		○	×	×	×	○	×	×	×	○	×	×	×
		INSERT		×	×	×	×	△	×	×	×	○	×	×	×
		UPDATE		×	×	×	×	△	×	×	×	○	×	×	×
		DELETE		×	×	×	×	△	×	×	×	○	×	×	×
	READ UNCOMMITTED	SELECT		○	○	○	○	○	○	○	○	○	○	○	○
		INSERT		×	×	×	×	△	×	×	×	○	×	×	×
		UPDATE		×	×	×	×	△	×	×	×	○	×	×	×
		DELETE		×	×	×	×	△	×	×	×	○	×	×	×

○：待つことなく動作する

△：先行トランザクションのSQL文が以下のいずれかの場合は、先行トランザクションのSQL文の終了を待つ

- 単一行SELECT文
- 更新可能性句に、FOR READ ONLYを指定したカーソルのOPEN文
- トランザクションアクセスモードが読込み専用モードのときのOPEN文

  先行トランザクションのSQL文が上記以外の場合は、トランザクションの終了を待つ

×：トランザクションの終了を待つ

先行トランザクションがカーソルのOPEN文を実行した場合は、カーソル宣言の更新可能性句の指定によって占有待ちの制御が異なります。

更新可能性句がREAD ONLYの場合は、SELECT文と同じになります。更新可能性句がFOR UPDATEの場合は、UPDATE文と同じになります。

1.1.9 行単位の排他を使用する場合の注意事項

行単位の排他を使用する場合に、設計時点で注意しなければならない事項について説明します。

SQL文のデータ操作文に関する注意事項

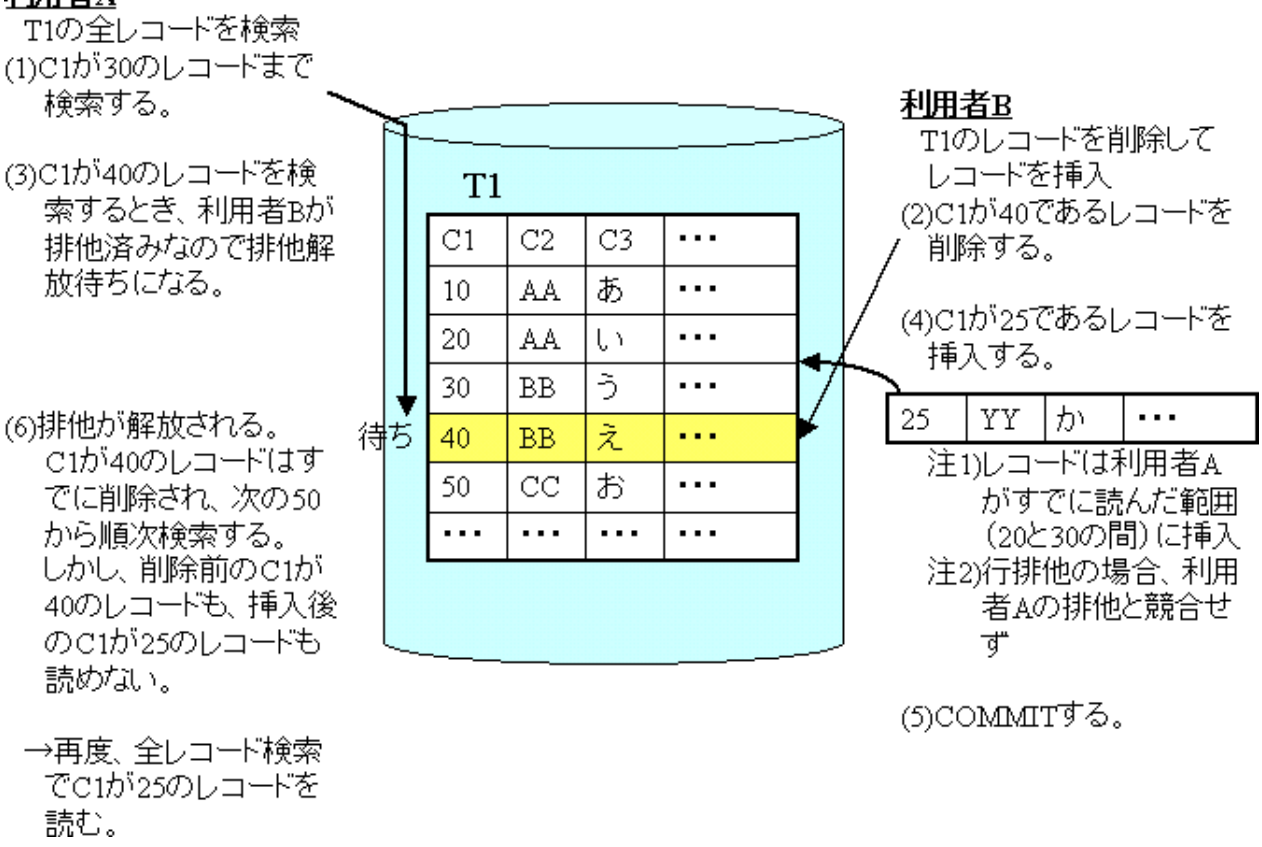
行単位の排他を使用する場合は、そのトランザクションがアクセスしていない行は占有しません。競合する別のトランザクションでは、最初のトランザクションがアクセスしていない行を更新したり、新たな行を追加したりすることができます。このため、以下のような現象が起こることがあります。構築する業務において、この現象が発生すると不都合な場合は、行単位の排他を使用しないでください。

- 同一トランザクション内で、同一のSQL文を繰り返し実行した場合に、異なる集合が検索されることがあります。例えば、
  - ー カーソルで検索する行が増えることがあります。
  - ー 一度目は成功した単一行SELECT文が、例外(基数違反)となることがあります。
  - ー 集合関数の結果が変化することがあります。
  - ー 探索条件に副問合せを指定した問合せの検索結果が変化することがあります。
- 行単位の排他では、INSERT文は他のトランザクションの終了を待たずに実行できます。このため、あるトランザクションで行を削除して、新たな行を挿入する操作を行っているとき、競合する別のトランザクションでは、まれに削除された行も挿入された行も参照できない場合があります。また、以下の条件を満たすUPDATE文の実行においても、Symfoware/RDBが内部的に行の削除と挿入を行うため、更新前の行も更新後の行も参照できない場合があります。
  - ー クラスタキー列を更新するUPDATE文を実行している。
  - ー DSI分割している場合の分割キーを更新するUPDATE文を実行している。

例

利用者AがT1の全レコードを検索します。これと同時に、利用者Bが、C1が40であるレコードを削除して、C1が25であるレコードを挿入します。

図1.15 SQL文のデータ操作文に関する注意事例



- 行単位の排他を使用して更新系の処理を行うトランザクションに対し、同じ資源を参照するSQL文を排他なしで実行した場合、以下の現象が発生する場合があります。排他なしとは、独立性水準にREAD UNCOMMITTEDを指定したトランザクション、または、占有モードにNO LOCKを指定したSQL文を指します。

  ー UPDATE文を実行中に排他なしでデータ操作のSQL文を実行した場合
  データ操作のSQL文は、更新前の行も更新後の行も参照できない場合があります。

  ー ROLLBACK文を実行中に排他なしでデータ操作のSQL文を実行した場合
  データ操作のSQL文は、更新前の行も更新後の行も参照できない場合があります。

- 更新系の処理を行う場合、内部的に更新対象行を特定するための検索処理が行われるため、占有モードが下記のとおり推移します。

  1. 更新対象行の検索時
     検索する範囲を共有モードで占有
  2. 更新対象行(検索結果)のデータの更新時
     更新対象行を非共有モードで占有

  このため、更新対象行のデータを更新する際に、更新対象行が別のトランザクションから共有モードで占有され、データ更新できないといった事象が発生する場合があります。この場合には、占有モードを以下のように指定することで、この現象を回避することが可能です。

  ー 動作環境ファイルのパラメタの設定やイルシデーションロックを用いて、内部的な検索を行うときから非共有モードでの占有とする。

  ー SELECT文によって更新する行を検索し、その後、データを更新する場合には、SELECT文にFOR UPDATE句を指定し、あらかじめ、非共有モードでの占有とする。

  占有モードを指定する方法は、“1.1.7.1 排他の属性と選択方法”を参照してください。

COMMIT文およびROLLBACK文に関する注意事項

- トランザクション中で参照された資源の排他情報は、COMMIT文またはROLLBACK文で回収されます。排他情報の数は、行排他の場合は行数、ページ排他の場合はページ数になります。COMMIT文またはROLLBACK文を実行すると以下の事象が発生します。

  ー 管理している資源数に比例して、COMMIT文またはROLLBACK文の処理時間が長くなります。また、COMMIT文やROLLBACK文実行中に、そのRDBシステム内で最初にアクセスする資源に対してSQL文やRDBコマンドを実行すると、COMMIT文やROLLBACK文が終了するまで待たされます。トランザクションを短くするか、あらかじめrdbconbfコマンドでバッファ結合を行った上で運用してください。

- 行単位の排他では、トランザクション中に行った更新内容はCOMMIT文実行時にまとめてインデックスへ反映します。このため、トランザクション中に行った挿入、インデックスキー値更新および削除の件数に比例してCOMMIT文の処理時間も長くなります。

COMMIT時のエラー処理に関する注意事項

COMMIT文実行時に、以下のエラーが発生する可能性があるため、COMMIT文もエラー処理の対象としてください。

- テンポラリログ領域不足
- インデックスのDSIの容量不足
- デッドロック

領域の再利用に関する注意事項

行単位の排他を使用する場合、DELETE文またはUPDATE文を実行することによってDSI内に発生する削除領域は、トランザクション完了後に再利用されます。そのため、同一トランザクション内で大量の件数を削除し、続いて挿入を行うような場合は、削除後に一度COMMIT文を実行してトランザクションを区切ることで、領域を有効に使用することができます。

ただし、他の更新または参照トランザクションと競合したページは、削除後に一度COMMIT文を実行してトランザクションを区切っても削除領域を再利用しない場合があります。この場合、以下の影響があります。

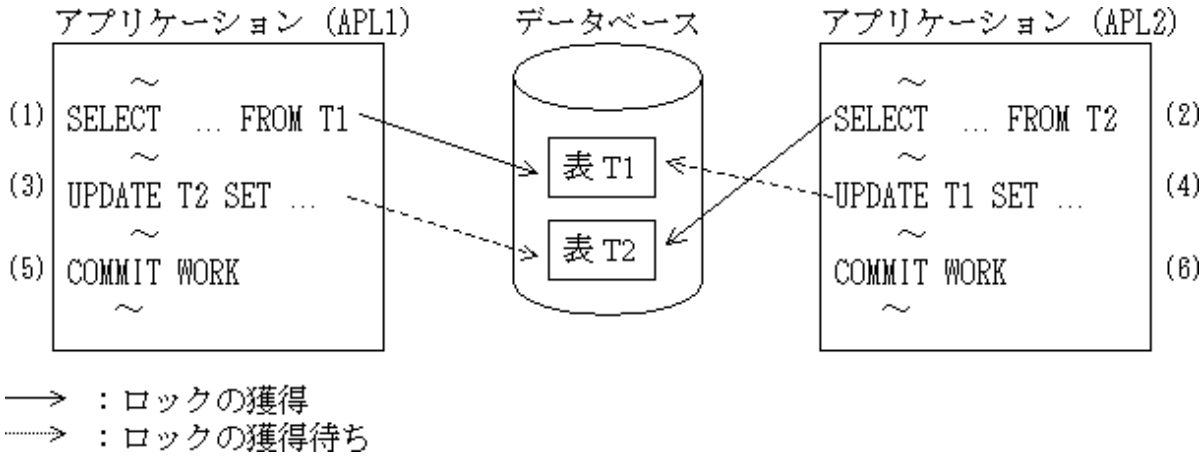
・ 自動容量拡張を指定している場合
現状の割当て領域に空きがある場合に、早めに容量拡張が動作する場合があります。

・ rdbudsiコマンドの表示結果
DSI割付けページ数とDSI割付けページに対して使用中のページの割合が増加する場合があります。また、使用中のページに対してデータが格納されている領域の割合が減少する場合があります。

1.1.10 デッドロックの対処方法

アプリケーションがデータベースをアクセスする場合の排他制御は、トランザクション単位に行われます。したがって、いくつかのアプリケーションを並行して実行した場合、データ操作の順序によっては、以下に示すようにデッドロックが発生します。このため、アプリケーションには、デッドロックの対処が必要になる場合があります。

図1.16 デッドロックの発生例



以下の順序で処理が行われるとデッドロックが発生します。

(1) APL1が表T1に対して参照ロックを獲得します（(5)のCOMMITまでロックを保持します）。

(2) APL2が表T2に対して参照ロックを獲得します（(6)のCOMMITまでロックを保持します）。

(3) APL1は表T1の参照を終え、表T2を更新するために表T2に対して更新ロックを獲得しようとしますが、表T2にはAPL2が参照ロックを獲得しているため待たされます（表T1はAPL1がロックを獲得したままとなります）。

(4) APL2は表T2の参照を終え、表T1を更新するために表T1に対して更新ロックを獲得しようとしますが、表T1にはAPL1が参照ロックを獲得しているため待たされます（表T2はAPL2がロックを獲得したままとなります）。

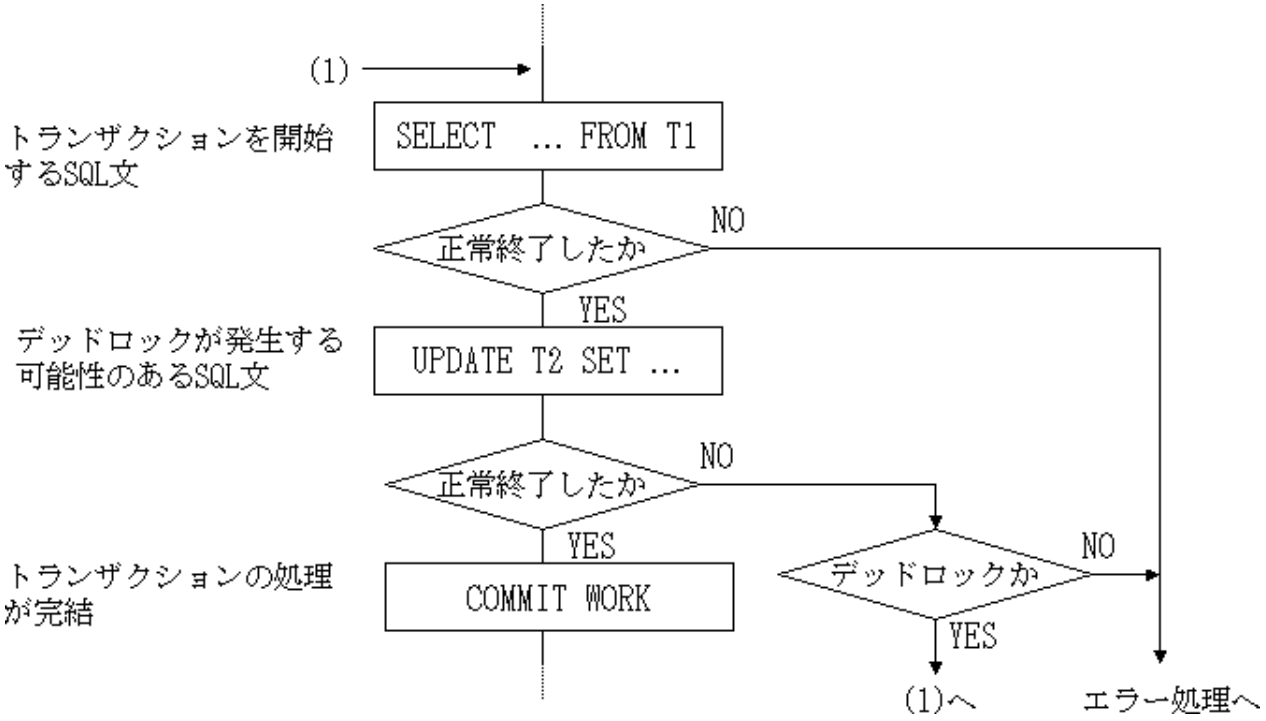
APL1とAPL2は、互いに相手方のトランザクションが終了するのを待たされる状態になります。ところが、APL1の(5)のCOMMIT、およびAPL2の(6)のCOMMITが実行されないために、どちらのアプリケーションも実行できなくなります。この状態がデッドロックです。

デッドロックはSymfoware/RDBが自動的に検出します。Symfoware/RDBがデッドロックを検出すると、複数のトランザクションのうちの1つのトランザクションを取り消して、アプリケーションにエラー復帰します。このとき、SQLSTATEには40001が設定されます。

デッドロックの発生を通知されたアプリケーションでは、一般にデッドロックが発生したトランザクションの再実行を試みます。トランザクションを再実行するには、トランザクションを開始した文の直前に制御を移します。ただし、SQL文で使用するホスト変数をトランザクションの処理途中で変更している場合には、その値をトランザクションの開始時点に戻す必要があります。

以下に、“図1.16 デッドロックの発生例”のAPL1が、トランザクションの再実行を行う場合の処理の流れを示します。

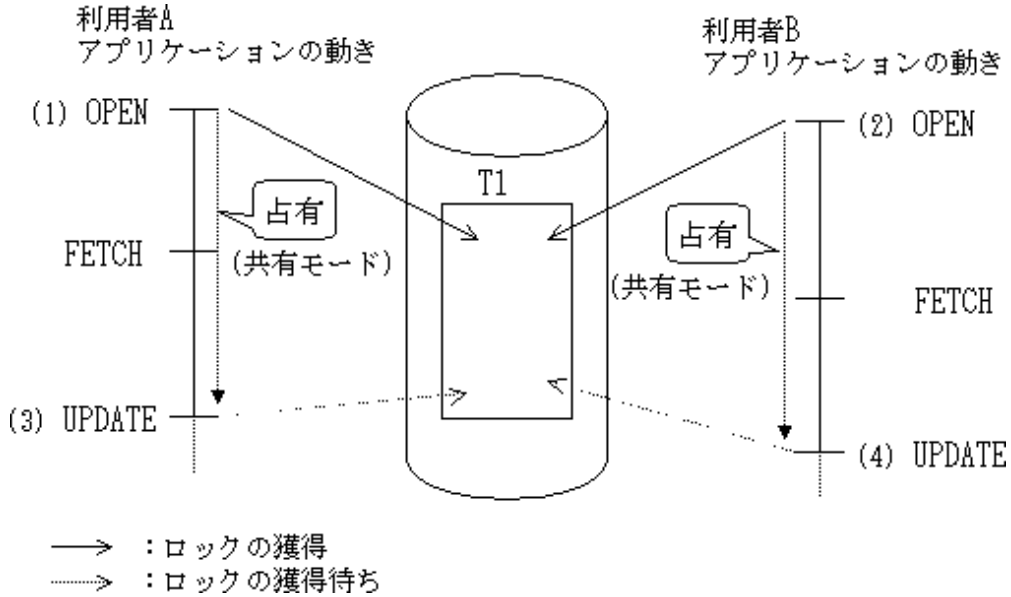
図1.17 デッドロックに対処する処理の例



カーソル操作での注意事項

カーソル宣言の更新可能性句を省略しそのカーソルが更新可能な場合、そのカーソルで更新を行うことができます。この場合、以下に示す操作を行うと、デッドロックが起きることがあります。

図1.18 カーソル操作でデッドロックが起きる例



(1) 利用者Aは、更新可能性句を省略したカーソルのOPEN文を実行します。これにより、表T1で参照した部分に対して共有モードで占有します。

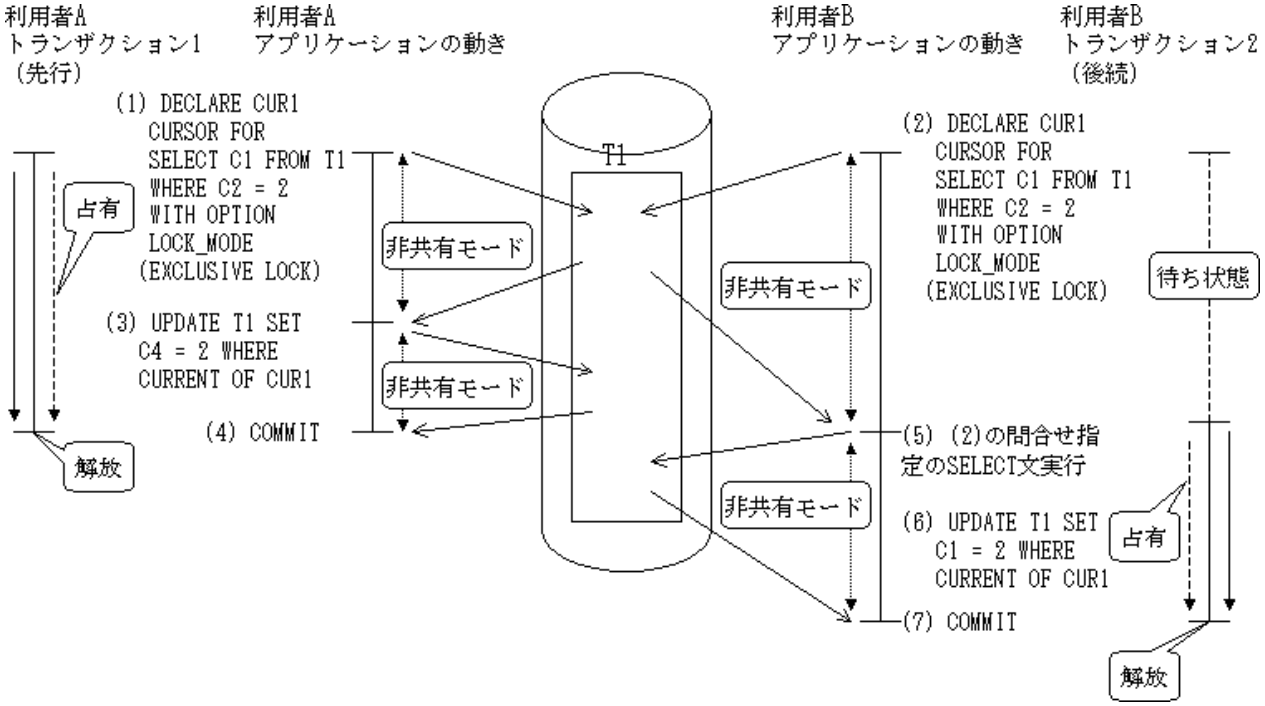
(2) 利用者Bも同じく、更新可能性句を省略したカーソルのOPEN文を実行します。これにより、表T1で参照した部分に対して共有モードで占有します。

(3) 利用者Aが表T1で参照した部分を更新するために、非共有モードで占有しようとしますが、表T1は利用者Bが共有モードで占有しているため待たされます（表T1は利用者Aが共有モードで占有したままとなります）。

(4) 利用者Bが表T1で参照した部分を更新するために、更新ロックを獲得しようとしますが、表T1は利用者Aが共有モードで占有しているため待たされます(表T1は利用者Aが共有モードで占有したままとなります)。

カーソル操作でデッドロックを起こさないため、更新系のSQL文を実行する場合には、カーソル宣言の占有モード指定に、EXCLUSIVE LOCKを指定してください。

図1.19 EXCLUSIVE LOCKを利用したデッドロックの回避例



(1) 利用者Aは、EXCLUSIVE LOCKで参照し、COMMIT(ROLLBACK)終了するまで資源を占有します。

(2) 利用者Bは、EXCLUSIVE LOCKで参照を行うため、利用者Aが資源を解放するまで参照を行えず、待ち状態となります。

(3) 利用者Aは、他のアプリケーションに待たされることなく更新処理を行います。

(4) 利用者AがCOMMIT(ROLLBACK)します。利用者Aの資源は解放されます。

(5) 利用者Aが資源を解放したので、利用者Bの参照が実行されます。この時、資源を占有します。

(6) 利用者Bの更新が実行されます。

(7) 利用者BがCOMMIT(ROLLBACK)します。

1.1.11 トランザクション実行時間の設定

アプリケーションがSQL文を実行すると、データベースサーバにメモリ資源や排他資源を獲得します。この資源を獲得したまま、アプリケーションに何らかの異常が発生した場合、獲得した資源が回収できなくなることがあります。

このような問題を防止するには、動作環境ファイルの実行パラメタTRAN_TIME_LIMITを使用します。TRAN_TIME_LIMITに1つのトランザクションの最大実行可能時間を指定すると、トランザクションが指定時間を超過した場合は、コネクションを切断し、獲得した資源を回収します。当パラメタはシステム用の動作環境ファイル、クライアント用の動作環境ファイルまたはサーバ用の動作環境ファイルに指定することができます。

動作環境ファイルの詳細については、“2.1.1 動作環境ファイル”を参照してください。

1.1.12 PRECEDENCE(1)を指定したSEQUENTIAL構造の特徴と注意事項

DSO定義でPRECEDENCE(1)を指定し、コミットデータ即時読込み機能を利用するSEQUENTIAL構造の表は、トランザクションの特性や排他の指定方法が他の格納構造の表と異なります。以下のような点に注意して利用してください。

- トランザクションの特性

・ 排他に関する指定

・ 対象資源が競合した場合の制御

1.1.12.1 トランザクションの特性

PRECEDENCE(1)が指定されたSEQUENTIAL構造の表にアクセスするトランザクションで、排他の単位に行を指定し、独立性水準にREAD COMMITTEDを指定した場合のアプリケーションの振る舞いや競合時の動作は、以下のようになります。

READ COMMITTED指定時の振る舞い

PRECEDENCE(1)が指定されていないSEQUENTIAL構造の表にアクセスするあるトランザクションが更新した行を、READ COMMITTEDを指定したトランザクションで検索すると、検索トランザクションは排他待ちになります。

しかし、PRECEDENCE(1)が指定されたSEQUENTIAL構造の表に対して以下のSQL文を実行した場合、検索する行が更新途中であっても、排他待ちすることなく、更新が発生する前のコミット済みのデータを検索することができます。そのため、更新業務と並行して1つの表に対して高頻度で検索するような業務に適しています。

・ 単一行SELECT文

・ 更新可能性句に、FOR READ ONLYを指定したカーソル、または更新可能性句を省略したカーソルのOPEN文

上記SQL文は各行の読み込み時の最新のコミット済みデータを参照します。このため、上記SQL文実行中に更新業務がコミットした場合、以下のような現象が起こることがあります。構築する業務において、この現象が発生すると不都合な場合は、PRECEDENCE(1)が指定されたSEQUENTIAL構造の表を使用しないでください。

・ SQL文の検索結果に、ある行については更新業務が更新する前の行、別の行については更新業務が更新した後の行が含まれる場合があります。

・ SQL文の検索結果に、ある行の更新前後の行がともに含まれる場合や、いずれも含まれない場合があります。

PRECEDENCE(1)を指定したSEQUENTIAL構造の表と他の格納構造の表を混在してSQL文に指定した場合、他の格納構造の表のアクセスで排他待ちになる可能性があります。そのため、排他待ちすることなく検索を実行したい場合は、SQL文中にはPRECEDENCE(1)を指定したSEQUENTIAL構造の表のみを指定してください。

また、更新トランザクションを実行する場合、ある行について検索してから更新や削除するまでの区間に、他のトランザクションが更新や削除を行ってコミットする可能性があります。このような場合、データの整合性を保証するために、更新トランザクションについて以下の対処を行ってください。

・ カーソルの更新可能性句にFOR UPDATEを指定する。

・ 占有モード指定にSHARE LOCK、またはEXCLUSIVE LOCKを指定する。

上記対処を行わないと以下の現象になります。

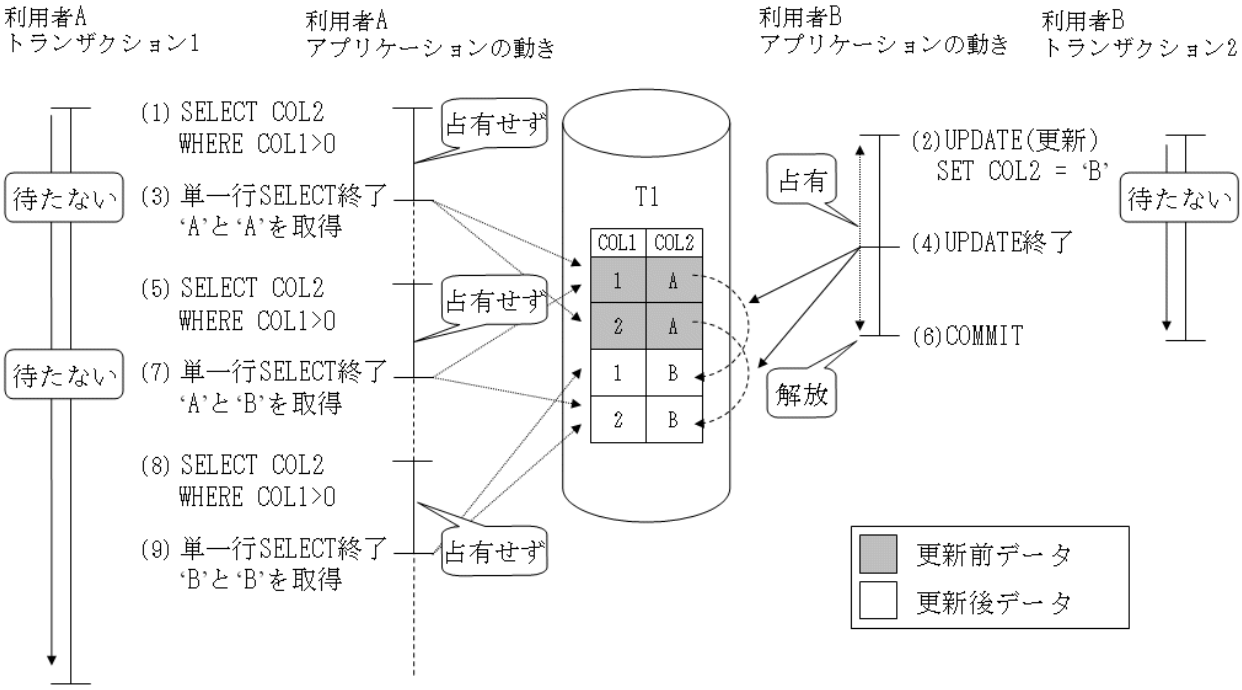
・ 他のトランザクションが更新した行を上書きにより更新してしまう場合がある。

・ 他のトランザクションが削除した行についてUPDATE文:位置づけやDELETE文:位置づけを行うと、対象行が存在しないのでエラーになる。

・ 他のトランザクションが削除した行についてUPDATE文:探索やDELETE文:探索を行うと、対象行が存在しないのでデータなしになる。

PRECEDENCE(1)を指定したSEQUENTIAL構造の表にアクセスする場合のREAD COMMITTED指定時の排他処理を以下に示します。

図1.20 PRECEDENCE(1)を指定したSEQUENTIAL構造の表にアクセスする場合のREAD COMMITTED指定時の排他



(1) 利用者AがCOL1>0の行を参照します。このとき、行は占有しません。

(2) 利用者Bは利用者Aが資源を参照中に更新を行うことができます。このとき、資源を占有します。データベースには更新前の行に加え、更新後の行が格納されます。

(3) 利用者Aは利用者Bが行を更新占有中であっても参照を完了させることができます。検索される行には、利用者Bが更新途中の結果は反映されません。

(4) 利用者Bの更新が終了しますが、行の占有は続きます。

(5) 利用者AがCOL1>0の行を参照します。このとき、利用者Bが更新占有中の行であっても参照することができます。

(6) 利用者Bがトランザクションを終了し、行を解放します。

(7) 利用者Aの参照が終了します。参照中に利用者BがCOMMITしたため、検索される行には利用者Bの更新結果が一部反映されます。

(8) 利用者AがCOL1>0の行を参照します。このとき、行は占有しません。

(9) 利用者Aの参照が終了します。このとき、検索される行には利用者Bの更新結果がすべて反映されます。

1.1.12.2 排他に関する指定

PRECEDENCE(1)が指定されたSEQUENTIAL構造の表にアクセスするアプリケーションにおいて、排他に関する指定が他の格納構造と異なる箇所について説明します。

排他の属性の指定と占有の単位

動作環境ファイルの実行パラメタによる排他の属性の指定において、他の格納構造と異なる点は以下です。

DSO_LOCK:

Pパラメタによるページ単位の占有は指定できません。

R_LOCK:

本パラメタがNOの場合、占有単位はDSIになります。

各パラメタの記述形式については、“2.1.1 動作環境ファイル”を参照してください。

占有モード

SQL文において占有モード指定にFREE LOCKを指定した場合に他の格納構造と異なる点があります。

SQL文における占有モード指定での指定

占有の単位に行を指定し、占有モード指定にFREE LOCKを指定した場合、“占有のしかたと占有期間”および“他のトランザクションとの待ち関係”が、他の格納構造と異なります。

表1.15 占有モード指定と資源の占有(PRECEDENCE(1)を指定した場合)

占有モード指定	占有のしかたと占有期間	他のトランザクションとの待ち関係	読込み水準
FREE LOCK	資源を占有しない。	当該SQL文を実行しても、他のトランザクションを待たせない。	各行の読込み時の最新のコミット済みデータを読み込む。当該SQL文で読み込んだ行は、他のトランザクションに更新されることがあるため、同一トランザクションで再検索すると最新の結果を検索することができる。

占有モード指定を指定したSQL文の待ち関係

占有の単位に行を指定し、占有モードにFREE LOCKを指定した場合、EXCLUSIVE LOCKとの待ち関係が、他の格納構造と異なります。

表1.16 占有モード指定と待ち関係(PRECEDENCE(1)を指定した場合)

		先行トランザクション			
	占有モード指定	EXCLUSIVE LOCK	SHARE LOCK	FREE LOCK	NO LOCK
後続トランザクション	EXCLUSIVE LOCK	×	×	△	○
	FREE LOCK	△	○	○	○

○:待つことなく動作する

△:FREE LOCKを指定した側SQL文が以下のいずれかの場合は、待つことなく動作する
・単一行SELECT文
・OPEN文
いずれも上記以外の場合は、先行トランザクションの終了を待つ

×:先行トランザクションの終了を待つ

カーソル宣言およびSET TRANSACTION文における指定とSQL文の占有モード指定における指定との対応関係

占有の単位に行を指定し、独立性水準がREAD COMMITTED、トランザクションアクセスモードがREAD WRITE、更新可能性句が省略された場合、対応する占有モード指定が他の格納構造と異なります。

- 占有モード指定を省略したSQL文が、カーソル宣言の場合

SET TRANSACTION文における指定		カーソル宣言の更新可能性句	対応する占有モード指定
独立性水準	トランザクションアクセスモード		
READ COMMITTED	READ WRITE	省略	FREE LOCK

排他制御に関するパラメタの相互関係

R_LOCKの値がNOの場合、占有の単位はDSI単位となり、他の格納構造と異なります。

表1.17 排他制御に関するパラメタの相互関係

R_LOCKの値	DSO_LOCKの指定	SET TRANSACTION文の指定	占有モード指定の指定	占有のふるまい	占有の単位
NO	○	×	×	独立性水準は、SERIALIZABLEになる	DSI単位
	-	○	○	占有モード指定での指定に従う	
			-	SET TRANSACTION文の独立性水準に従う。ただし、独立性水準にREPEATABLE READを指定した場合は、SERIALIZABLEになる	
		-	○	占有モード指定での指定に従う	
			-	独立性水準は、SERIALIZABLEになる	

○：指定可能

×：指定できません

－：指定を省略

1.1.12.3 対象資源が競合した場合の制御

資源の競合が起きた場合に、どのタイミングまで占有待ちとなるのかは、トランザクションモードの指定によって異なります。以下に、トランザクションモードの指定と資源の占有について示します。

なお、以下は占有の単位が行の場合についてです。占有の単位がDSIの場合については、“1.1.8 資源の競合が起きた場合の制御”を参照してください。

表1.18 トランザクションモードの指定と資源の占有

			先行トランザクション											
			独立性水準	SERIALIZABLE REPEATABLE READ				READ COMMITTED				READ UNCOMMITTED			
			SQL文の種類	SELECT	INSERT	UPDATE	DELETE	SELECT	INSERT	UPDATE	DELETE	SELECT	INSERT	UPDATE	DELETE
後続トランザクション	独立性水準	SQL文の種類													
	REPEATABLE READ	SELECT		○	×	×	×	○	×	×	×	○	×	×	×
		INSERT		○	○	○	○	○	○	○	○	○	○	○	○
		UPDATE		×	×	×	×	○	×	×	×	○	×	×	×
		DELETE		×	×	×	×	○	×	×	×	○	×	×	×
	READ COMMITTED	SELECT		○	○	○	○	○	○	○	○	○	○	○	○
		INSERT		○	○	○	○	○	○	○	○	○	○	○	○
		UPDATE		×	×	×	×	○	×	×	×	○	×	×	×
		DELETE		×	×	×	×	○	×	×	×	○	×	×	×
	READ UNCOMMITTED	SELECT		○	○	○	○	○	○	○	○	○	○	○	○
		INSERT		○	○	○	○	○	○	○	○	○	○	○	○
		UPDATE		×	×	×	×	○	×	×	×	○	×	×	×
		DELETE		×	×	×	×	○	×	×	×	○	×	×	×

○ : 待つことなく動作する

× : トランザクションの終了を待つ

先行または後続のトランザクションがカーソルのOPEN文を実行した場合は、カーソル宣言の更新可能性句の指定によって占有待ちの制御が異なります。

更新可能性句がREAD ONLYの場合、または更新可能性句を省略した場合は、SELECT文と同じになります。更新可能性句がFOR UPDATEの場合は、UPDATE文と同じになります。

1.2 クライントとサーバの接続方法

クライアントとサーバの間の接続方法には、以下があります。

- ローカルアクセス
- リモートアクセス

それぞれの接続方法と通信方法を説明します。

ローカルアクセス

クライアントと接続先のデータベースが同一のサーバ上にある場合の接続方法です。

ローカルアクセスを行う場合は、自動的にサーバと接続されます

リモートアクセス

クライアントと接続先のデータベースが異なるサーバ上にある場合の接続方法です。

リモートアクセスを行う場合には、以下の通信方法が選択できます。

通信方法	特長
RDB2_TCP連携	Symfoware/RDB専用のプロトコルを使用することで、高速にデータベースにアクセスすることができます。 TCP/IP通信を行います。
RDB2_TCPS連携	Symfoware/RDB SSL通信専用のプロトコルを使用することで、通信データを暗号化することができます。 TCP/IP(SSL)通信を行います。

通信方法の選択については、“2.1.3.1 アプリケーションの接続先や通信に関する実行パラメタ”を参照してください。

1.3 コネクション

ここでは、コネクションの概要、コネクションの接続および切断について説明します。

- コネクションとは
- コネクションの接続
- コネクションの切断

コネクションとは

コネクションとは、クライアントのアプリケーションがサーバのSymfoware/RDBデータベースにアクセスするために、クライアントとサーバの間を結んだ接続関係のことです。アプリケーションがデータベースシステムとコネクションをもつことによって、そのデータベースシステムの資源を操作することができます。

コネクションの接続

アプリケーションとサーバとのコネクションの接続は、CONNECT文で行います。また、コネクションを接続することにより、SQL文を実行することができます。

コネクションの切断

アプリケーションとサーバとのコネクションの切断は、DISCONNECT文で行います。コネクションの切断は、CONNECT文で接続したコネクションに対して実行できます。また、トランザクションは終了した状態で実行します。

コネクションのプログラミング概要を以下に示します。

［アプリケーション］

```
:
EXEC SQL  CONNECT TO  'DB01' AS 'C1';                          … (1)
                        :
EXEC SQL
 SELECT 在庫数量, 倉庫番号 INTO  :H1, :H2 FROM S.T             … (2)
                                  WHERE 倉庫番号=2 ;
                        :
EXEC SQL  COMMIT WORK ;                                        … (3)
                        :
```

```
EXEC SQL  DISCONNECT  'C1' ;                                    … (4)
                          :
              :
EXEC SQL  CONNECT TO  'SV1'  AS 'C2'  USER 'U/PW' ;             … (5)
                          :
EXEC SQL
 SELECT  在庫数量, 倉庫番号 INTO  :H1, :H2 FROM S1.T1           … (6)
                                      WHERE 倉庫番号=2 ;
                          :
EXEC SQL  COMMIT WORK ;                                         … (7)
                          :
EXEC SQL  DISCONNECT  'C2' ;                                    … (8)
                          :
```

(1)～(4)は、ローカルアクセスの場合です。(5)～(8)は、リモートアクセスの場合です。

(1) CONNECT文にデータベース名を指定することにより、ローカルのサーバ上にある“DB01”に対して“C1”のコネクションを接続します。

(2) コネクションを接続した“DB01”に対して単一行SELECT文を実行します。

(3) “DB01”に対してのトランザクションを終了します。

(4) DISCONNECT文により、“DB01”に対して“C1”のコネクションを切断します。

(5) CONNECT文にSQLサーバ名、ユーザID、パスワードを指定することにより、リモートのサーバ“SV1”に対して“C2”のコネクションを接続します。

(6) コネクションを接続した“SV1”に対して単一行SELECT文を実行します。

(7) “SV1”に対してのトランザクションを終了します。

(8) DISCONNECT文により、“SV1”に対して“C2”のコネクションを切断します。

1.3.1 複数コネクション

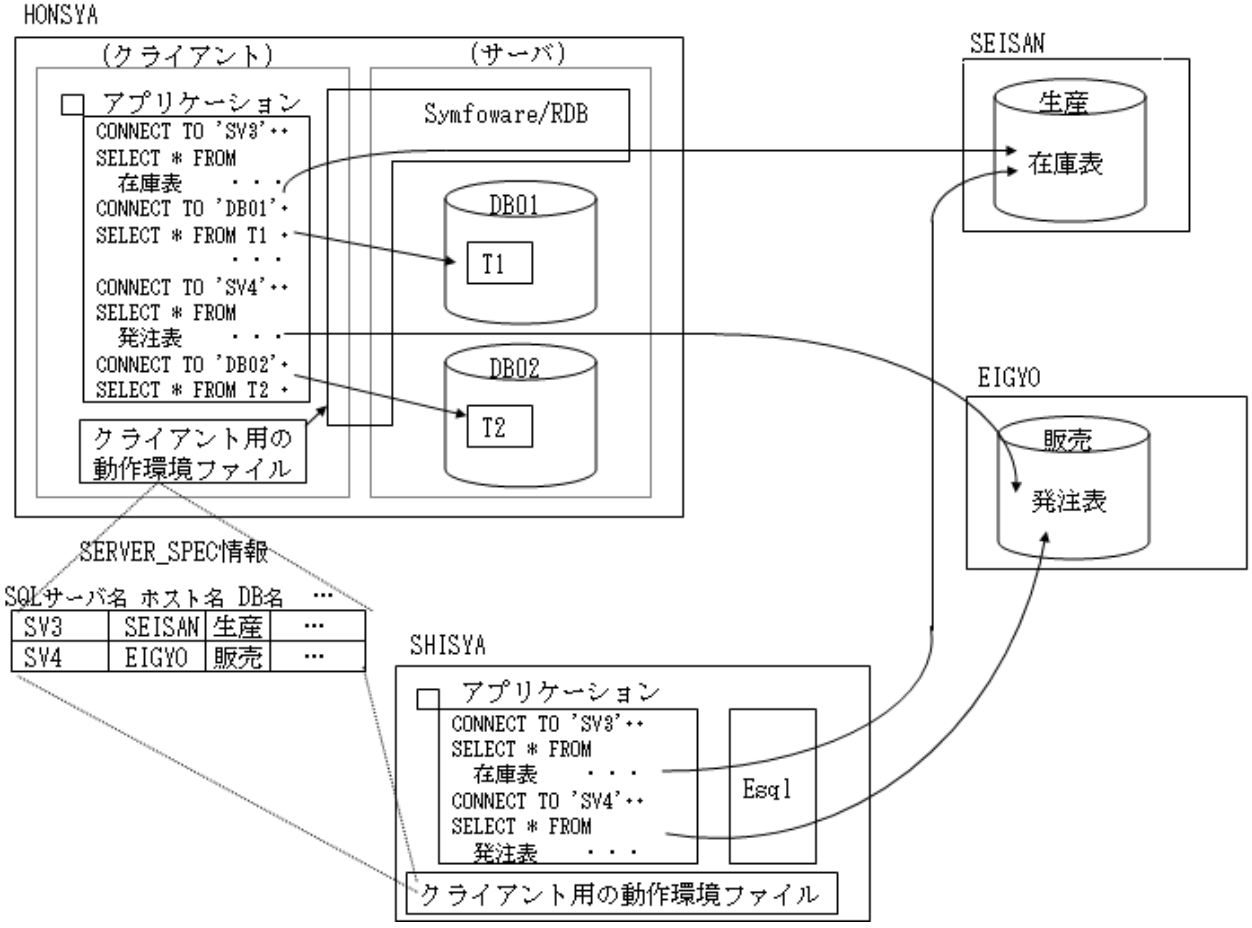
複数のCONNECT文を実行することによって、複数のコネクションを接続することができます。これによって、複数のデータベースに対し、データ操作を行うことができます。ここでは、複数コネクションの操作について、以下の順に説明します。

- 複数コネクションの接続
- 複数コネクションの切断
- 複数コネクションでSQL文を実行する場合の注意事項

1.3.1.1 複数コネクションの接続

複数のコネクションの接続は、複数のCONNECT文を実行することによって行います。以下に具体例を示します。

図1.21 複数のコネクション接続



1.3.1.2 複数コネクションの切断

複数コネクションの切断は、CONNECT文で接続した、それぞれのコネクションに対してDISCONNECT文で行います。コネクションを切断する場合は、トランザクションは終了した状態でなければなりません。コネクションを切断せずに処理を終了すると、利用されていないコネクションが残ったままとなります。

複数コネクションを切断するプログラミングの概要を以下に示します。

[アプリケーション]

```
EXEC SQL CONNECT TO  'DB01' AS  'C1' ;                                  … (1)
                          :
EXEC SQL SELECT  在庫数量,倉庫番号  INTO  :H1, :H2  FROM 在庫表      … (2)
                                    WHERE  在庫数量=300;
                          :
EXEC SQL COMMIT WORK ;                                                  … (3)
                          :
EXEC SQL CONNECT TO  'DB02' AS  'C2' ;                                  … (4)
                          :
EXEC SQL INSERT INTO  伝票(在庫カラム, 倉庫カラム)  VALUES(:H1,:H2); … (5)
                          :
EXEC SQL COMMIT WORK ;                                                  … (6)
                          :
EXEC SQL DISCONNECT 'C1' ;                                              … (7)
                          :
EXEC SQL DISCONNECT 'C2' ;                                              … (8)
             :
             :
EXEC SQL CONNECT TO  'SV3' AS  'C3'  USER  'U3/PASS3' ;                 … (9)
                          :
```

```
EXEC SQL SELECT  在庫数量,倉庫番号  INTO  :H1, :H2  FROM 在庫表１   … (10)
                                                  WHERE 在庫数量=300 ;
                        :
EXEC SQL COMMIT WORK ;                                                  … (11)
                        :
EXEC SQL CONNECT TO  'SV4' AS  'C4'  USER  'U4/PASS4' ;                 … (12)
                        :
EXEC SQL INSERT INTO 伝票１(在庫カラム,倉庫カラム) VALUES(:H1,:H2) ; … (13)
                        :
EXEC SQL COMMIT WORK ;                                                  … (14)
                        :
EXEC SQL DISCONNECT 'C3' ;                                              … (15)
                        :
EXEC SQL DISCONNECT 'C4' ;                                              … (16)
```

(1)～(8)は、ローカルアクセスの場合です。(9)～(16)は、リモートアクセスの場合です。

(1) 1回目のCONNECT文により“DB01”に対して“C1”のコネクションを接続します。

(2) “DB01”の在庫表から単一行SELECT文で在庫数量と倉庫番号を取り出します。

(3) (2)の操作に対して、トランザクションを終了します。

(4) 2回目のCONNECT文により、“DB02”に対して“C2”のコネクションを接続します。

(5) “DB02”の伝票の在庫カラムと倉庫カラムに、“DB01”から取り出した在庫数量と倉庫番号を追加します。

(6) (5)の操作に対して、トランザクションを終了します。

(7) “DB01”に対して“C1”のコネクションを切断します。

(8) “DB02”に対して“C2”のコネクションを切断します。

(9) 3回目のCONNECT文により、サーバ“SV3”に対して“C3”のコネクションを接続します。

(10) サーバ“SV3”の在庫表1から、単一行SELECT文で在庫数量と倉庫番号を取り出します。

(11) (10)の操作に対して、トランザクションを終了します。

(12) 4回目のCONNECT文により、サーバ“SV4”に対して“C4”のコネクションを接続します。

(13) サーバ“SV4”の伝票1の在庫カラムと倉庫カラムに、サーバ“SV3”から取り出した在庫数量と倉庫番号を追加します。

(14) (13)の操作に対して、トランザクションを終了します。

(15) サーバ“SV3”に対して“C3”のコネクションを切断します。

(16) サーバ“SV4”に対して“C4”のコネクションを切断します。

1.3.1.3 複数コネクションでSQL文を実行する場合の注意事項

カーソル操作での注意

カーソルをオープンしたあとに、複数のコネクションにまたがって、同じカーソルの操作を行うことはできません。

ほかのコネクションで同じカーソルを操作するには、カーソルをクローズしたあとでなければなりません。このカーソル操作の例として、誤った指定例および正しいカーソル操作の例を以下に示します。

例1

誤ったカーソル操作の場合

[アプリケーション]

```
EXEC SQL DECLARE CUR1 CURSOR FOR SELECT  在庫数量   FROM   在庫表 ;  … (1)

EXEC SQL CONNECT TO 'DB01' AS 'C1' ;                                  … (2)

EXEC SQL OPEN CUR1 ;                                                  … (3)
```

```
EXEC SQL CONNECT TO 'DB02' AS 'C2' ;                                    … (4)

EXEC SQL FETCH CUR1 INTO :STOCK  ;  →  エラー                           … (5)
```

(1) カーソル“CUR1”を宣言します。

(2) “DB01”に対して“C1”のコネクションを接続します。

(3) “DB01”に対して、カーソル“CUR1”をオープンします。

(4) “DB02”に対して“C2”のコネクションを接続します。

(5) “DB02”に対して、カーソル“CUR1”をFETCHすると、すでに“DB01”に対してカーソルがオープンしているため、エラーとなります。

例2

正しいカーソル操作の場合

[アプリケーション]

```
EXEC SQL DECLARE CUR1 CURSOR FOR SELECT  在庫数量  FROM  在庫表 ;  … (1)

EXEC SQL CONNECT TO 'DB01' AS 'C1' ;                                    … (2)
EXEC SQL OPEN CUR1 ;                                                    … (3)
           :
EXEC SQL CLOSE CUR1 ;                                                   … (4)
EXEC SQL COMMIT WORK ;                                                  … (5)

EXEC SQL CONNECT TO 'DB02' AS 'C2' ;                                    … (6)
EXEC SQL OPEN CUR1 ;
           :
EXEC SQL CLOSE CUR1 ;
```

(1) カーソル“CUR1”を宣言します。

(2) “DB01”に対して“C1”のコネクションを接続します。

(3) “DB01”に対して、カーソル“CUR1”をオープンします。

(4) カーソル“CUR1”をクローズします。

(5) (3)でオープンしたカーソルを使用して更新操作を行った場合、トランザクションを終了します。

(6) “DB02”にコネクションを接続して、カーソル“CUR1”をオープンします。

更新操作での注意

1つのトランザクションで、複数のコネクションに対してデータベースを更新するSQL文は実行できません。複数のコネクションでデータを更新する場合は、トランザクションを別々にする必要があります。

この更新操作の例として、誤った指定例および正しい更新操作の例を以下に示します。

例1

誤った更新操作の場合

[アプリケーション]

```
EXEC SQL CONNECT TO 'DB01' AS 'C1' ;                                    … (1)

EXEC SQL CONNECT TO 'DB02' AS 'C2' ;                                    … (2)

EXEC SQL UPDATE  在庫表  SET  在庫数量=300 WHERE  品番号=10;            … (3)

EXEC SQL SET CONNECTION  'C1' ;                                         … (4)

EXEC SQL UPDATE 支店表 SET 社員数=100 WHERE 支店番号=80;  →  エラー  … (5)
```

(1) “DB01”に対して“C1”のコネクションを接続します。

(2) “DB02”に対して“C2”のコネクションを接続します。この場合のコネクションが現コネクションになります。

(3) “DB02”に対して更新操作を行います。

(4) “DB01”に現コネクションを変更します。

(5) “DB02”に対する更新操作のトランザクションが終了していないため、“DB01”に対する更新操作はエラーとなります。

例2

正しい更新操作の場合

[アプリケーション]

```
EXEC SQL CONNECT TO 'DB01' AS 'C1' ;

EXEC SQL CONNECT TO 'DB02' AS 'C2' ;

EXEC SQL UPDATE  在庫表  SET  在庫数量=300 WHERE  品番号=10;

EXEC SQL COMMIT WORK ;                                          … (1)

EXEC SQL SET CONNECTION  'C1' ;                                 … (2)

EXEC SQL UPDATE  支店表  SET  社員数=100  WHERE  支店番号=80;

EXEC SQL COMMIT WORK ;
```

(1) “DB02”に対しての更新操作のトランザクションを終了します。

(2) “DB01”に現コネクションを変更して、“DB01”に対して更新操作を行います。

データベースの定義での注意

1つのトランザクションで、複数のコネクションに対して、データベースの定義変更を行うことはできません。

1.3.2 マルチRDB運用のコネクション

Symfoware/RDBでは、ローカルアクセスを行うとき、複数のSymfoware/RDBの動作環境を作成して運用することができます。

このような運用形態を、マルチRDB運用といいます。

注意

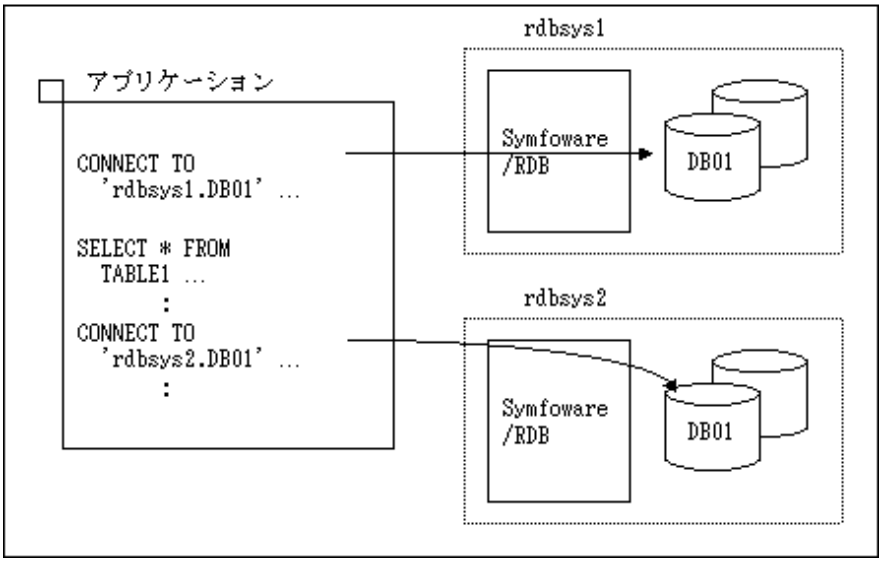
データベース簡単運用の場合は、マルチRDB運用を行うことはできません。

アプリケーションを利用して、ローカルアクセス時に異なるSymfoware/RDBに接続する場合は、データベース名にシステム名を付けて指定します。

```
EXEC SQL CONNECT TO 'rdbsys2.DB01';
```

CONNECT文を複数記述することによって、複数のSymfoware/RDB環境に接続することができます。以下に例を示します。

図1.22 マルチRDB運用のコネクション



アプリケーションをマルチRDB環境で利用する

システム名を省略した場合は、環境変数RDBNAMEで指定したものが有効となります。以下に例を示します。

環境変数での指定例

S

Solarisの場合

― Cシェルで指定する場合

```
setenv RDBNAME rdbsys1
```

― bash、Bシェル、Kシェルで指定する場合

```
RDBNAME=rdbsys1; export RDBNAME
```

L

Linuxの場合

― TCシェルで指定する場合

```
setenv RDBNAME rdbsys1
```

― bashで指定する場合

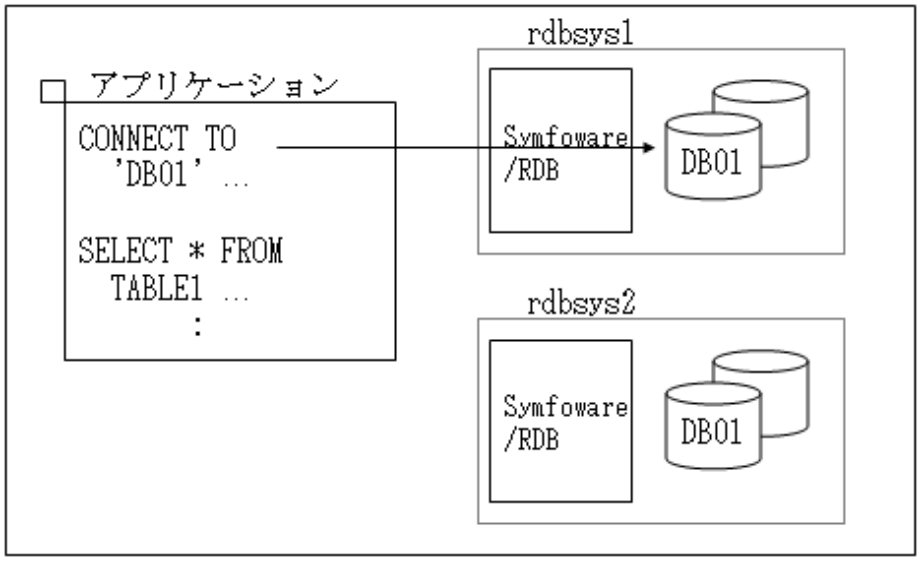
```
RDBNAME=rdbsys1; export RDBNAME
```

W

Windows(R)の場合

```
set RDBNAME = rdbsys1
```

図1.23 マルチRDB環境におけるアプリケーションの利用



1.4 リモートアクセス時の注意事項

リモートアクセスを行う場合、クライアントの電源切断によりデータベース資源が占有されたままとなることがあります。この場合、以下のいずれかの方法でデータベース資源の解放を行います。

- 接続先のシステムでrdbtermコマンドを実行する
- クライアントを再起動する

注意

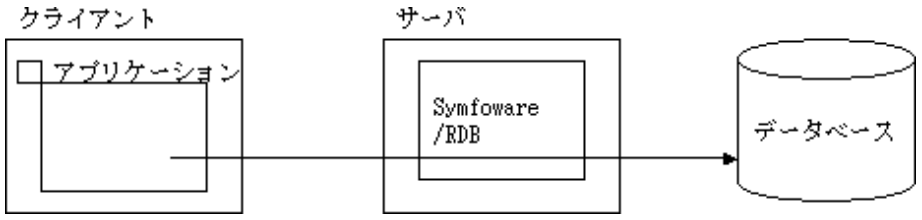
データベース簡単運用では、rdbtermコマンドによるデータベース資源の解放はできません。

なお、Symfoware/RDBでは、電源切断後に再度リモートアクセスが行われると、クライアントのシステム起動時刻を利用し、データベース資源の解放を行います。このため、クライアントのシステム時刻を業務中に変更すると、クライアントから依頼されている業務で利用しているデータベース資源が不当に解放されることがあります。クライアントのシステム時刻は、業務中は変更しないでください。

また、Connection Managerを利用している場合は、Connection Managerがクライアントのダウン検出し、コネクション自動削除機能によりデータベース資源の解放を行うため、データベース資源が不当に解放されることはありません。

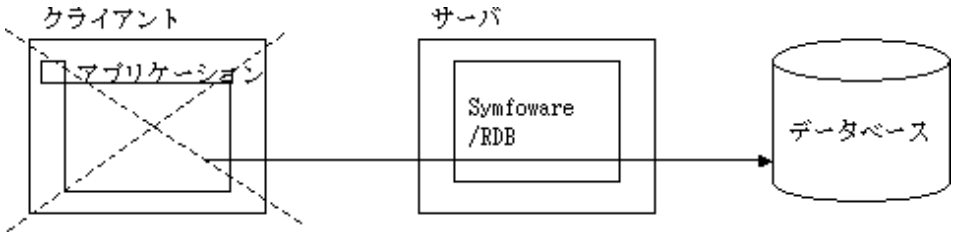
クライアントを再起動してデータベース資源を解放する手順を以下に示します。クライアントの再起動で、アプリケーションがCONNECT文を実行すると、解放されていないデータベースの資源が解放されます。

1. データベース資源アクセス中

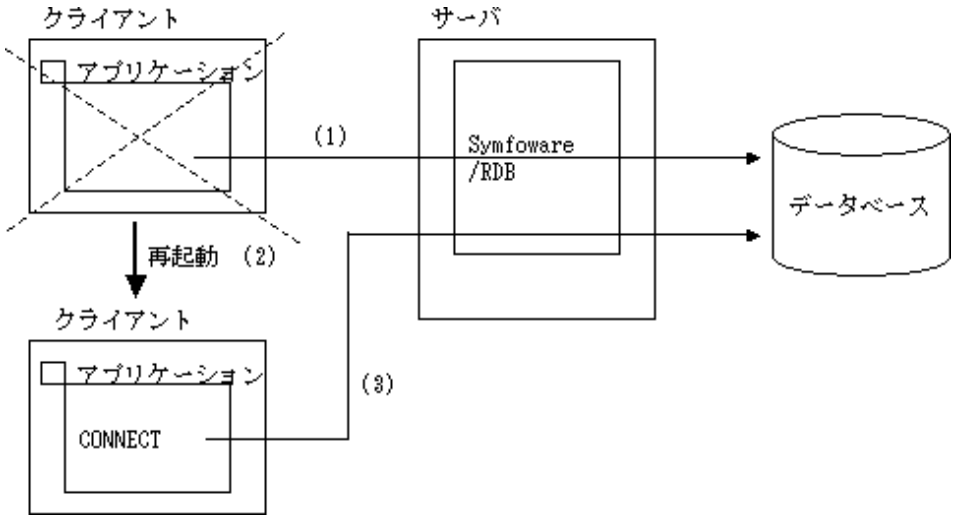


2. クライアント電源切断

クライアントの電源切断によりデータベース資源が解放されない。



3. 電源切断後に再度リモートアクセス



(1) クライアントの電源切断によりデータベース資源が解放されない場合がある。

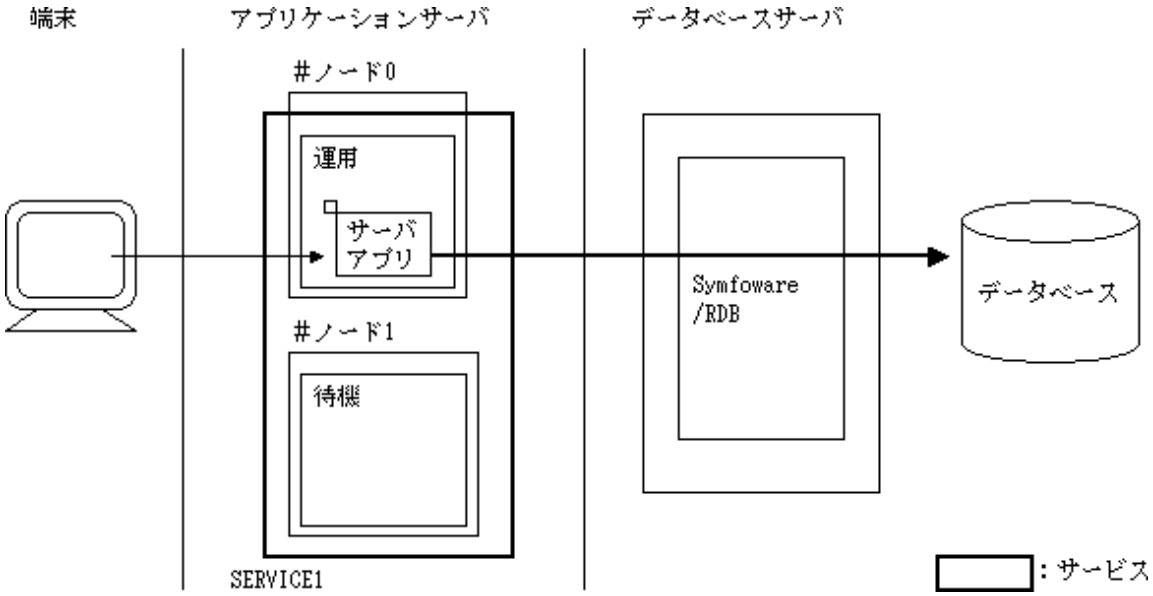
(2) クライアントが電源切断したため、再起動する。

(3) アプリケーションがCONNECT文を実行すると、解放されていないデータベースの資源の解放を行う。

また、アプリケーションサーバとデータベースサーバが分離した3階層モデルにおいて、アプリケーションサーバをフェイルオーバ運用した場合は、運用系のRDBシステムの電源切断によりデータベース資源が占有されたままとなる場合があります。このとき、運用系のRDBシステムから待機系のRDBシステムに切り替わって再度リモートアクセスが行われますが、運用系のRDBシステムと待機系のRDBシステムのIPアドレスが異なるため、待機系のRDBシステムにおいてアプリケーションがCONNECT文を実行しても、データベース資源が占有されたままとなります。フェイルオーバ運用をしている場合には、アプリケーションを実行する際に、クライアント用の動作環境ファイルにクラスタサービス名を設定して、運用系のRDBシステムと待機系のRDBシステムの関連付けを行う必要があります。クラスタサービス名の設定についての詳細は、“2.1.2 クライアント用の動作環境ファイルの作成”を参照してください。

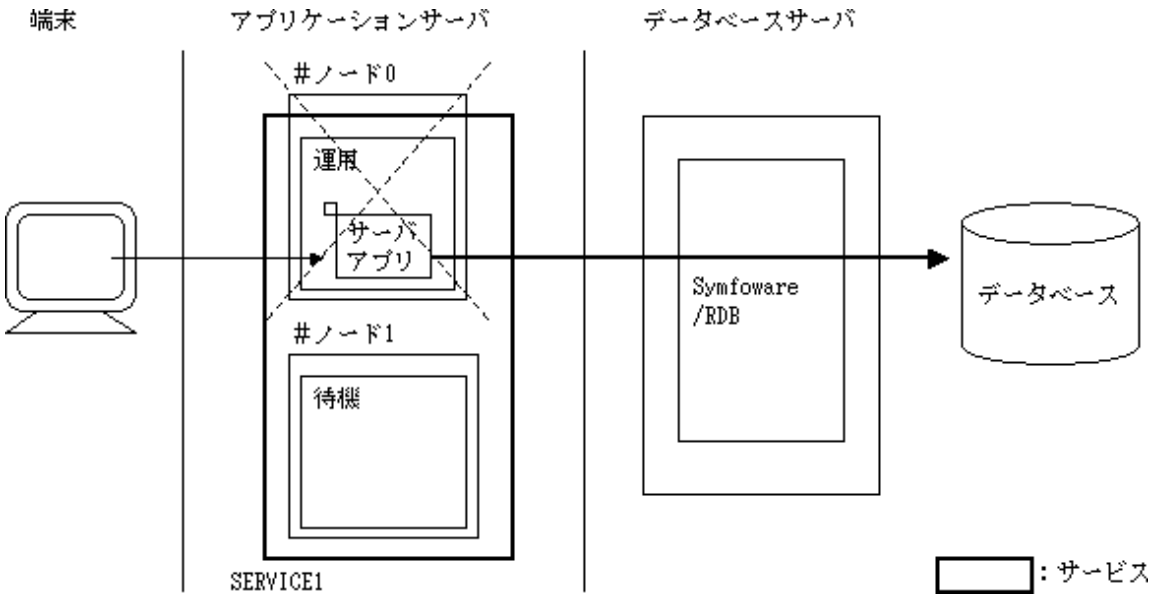
以下に、アプリケーションサーバとデータベースサーバが分離した3階層モデルにおいて、アプリケーションサーバをフェイルオーバ運用した場合のデータベース資源の解放手順を示します。

1. データベース資源アクセス中

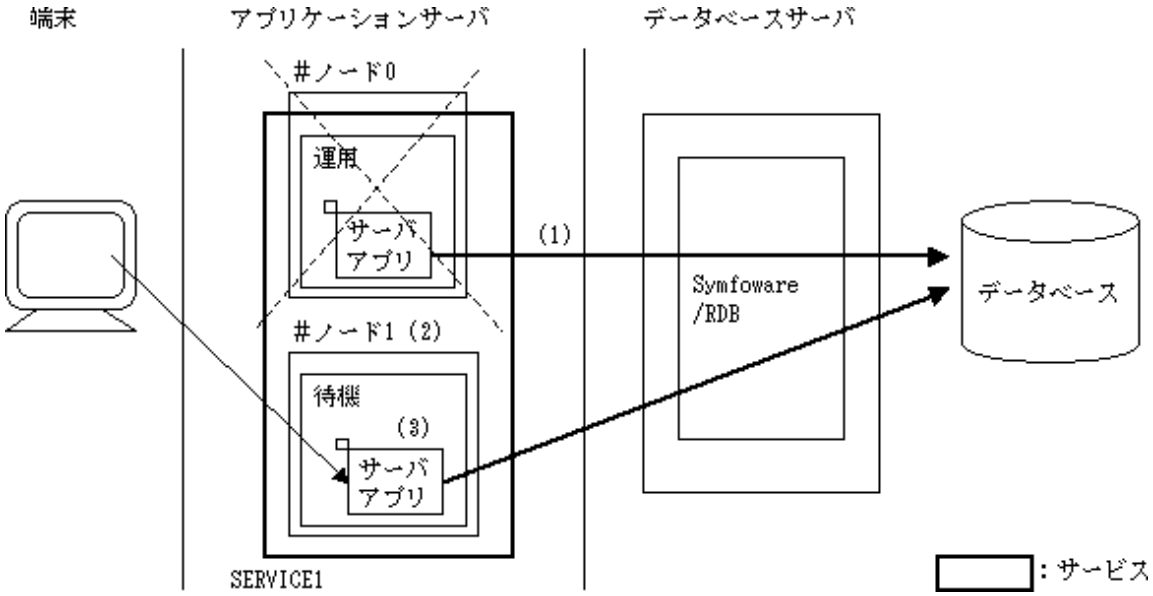


2. 運用系のRDBシステムの電源切断

運用系のRDBシステムが電源切断されたため、データベース資源が解放されない。



3. 待機系のRDBシステムに切り替わって再度リモートアクセス



運用系のクライアント用の動作環境ファイル

```
CLUSTER_SERVICE_NAME = (SERVICE1)
```

待機系のクライアント用の動作環境ファイル

```
CLUSTER_SERVICE_NAME = (SERVICE1)
```

(1) 運用系のRDBシステムからリモートでデータベースに対してアクセスしている。

(2) 運用系のRDBシステムの電源切断により待機系のRDBシステムに切り替わる。このとき、データベース資源の占有が解除されていない。

(3) 待機系のRDBシステムから再度リモートアクセスでCONNECT文を実行すると、解放されていないデータベース資源の解放を行う。ただし、この場合、アプリケーションの実行時に、クライアント用の動作環境ファイルにクラスタサービス名CLUSTER_SERVICE_NAMEを設定しておく必要がある。

1.5 通信データを暗号化する場合の設定

通信データの暗号化機能を使用してリモートアクセスを行う場合は、以下の設定を行ってください。

通信内容を暗号化してサーバに接続する設定

通信の暗号化は、システム用の動作環境ファイルのSSL_USEパラメタにONが指定されている場合にのみ使用できます。クライアント側では、クライアント用の動作環境ファイルのSERVER_SPECパラメタにRDB2_TCPSを指定します。

サーバ認証を行う場合の設定

サーバのなりすましから守るために、サーバ認証を行う場合は、データベース管理者が配布するCA証明書ファイルを格納し、格納先のパスをクライアント用の動作環境ファイルのSSL_CLI_CA_CERT_FILEパラメタに指定します。

また、埋込みSQL連携で、クライアント用の動作環境ファイルのDEFAULT_CONNECTIONパラメタを使用している場合、データベースサーバに接続するユーザのパスワードが誰にでも参照できてしまいます。そのため、サーバとの接続方法として、DEFAULT_CONNECTIONパラメタを使用せず、SQLサーバ名を使用したCONNECT文による接続を推奨します。

参照

接続方法については、“アプリケーション開発ガイド(埋込みSQL編)”を参照してください。

注意

- 通信内容を暗号化してサーバに接続する設定を行わずに、SSL_CLI_CA_CERT_FILEパラメタを指定した場合は、エラーになります。
- サーバとクライアントで暗号化の指定が矛盾している場合は、エラーになります。通信内容を暗号化してサーバに接続する設定を行った場合は、システム用の動作環境ファイルのSSL_USEパラメタにONを指定してください。
- データベース簡単運用では、通信データの暗号化はできません。

参考

ここでは、埋込みSQL連携の場合の設定を説明しています。

Java連携、ODBC連携、.NET Framework連携の場合の設定については、以下のマニュアルを参照してください。

- アプリケーション開発ガイド(JDBCドライバ編)
- アプリケーション開発ガイド(ODBCドライバ編)
- アプリケーション開発ガイド(.NET Data Provider編)

1.6 概数を使用するときの注意事項

UNIX系システムとグローバルサーバおよびWindows(R)とグローバルサーバでは概数の内部表現形式が異なるため、Symfoware/RDBでは形式の変換を行っています。

このため、アプリケーションで指定した値とグローバルサーバのデータベースに格納される値との間、およびグローバルサーバのデータベースに格納されている値とアプリケーションで取り出せる値との間には、表現形式の差に起因する誤差が生ずることがあります。

また、変換時にオーバフロー(変換元の値の絶対値が大きすぎて変換先での形式では表現できない)、またはアンダフロー(変換元の値の絶対値が小さすぎて変換先での形式では表現できない)が発生することがあります。オーバフローが発生した場合、SQL文は異常終了します。また、アンダフローが発生した場合は、値を0として処理します。

1.7 SQL文のデータ操作文に関する注意事項

表の設計変更により、列を追加したり、列の定義順序を変更したりする場合があります。このため、データベースの定義変更を考慮したアプリケーションを作成しなければなりません。以下に注意点を示します。

以下の条件を満たすSQLデータ操作文は使用しないでください。

- 選択リストに“*”や“表名.*”、“相関名.*”を指定したカーソル宣言または単一行SELECT文
- 挿入列リストを省略したINSERT文

上記のSQL文は、表の設計変更により列を追加したり、列の定義順序を変更した場合にエラーになります。アプリケーションにこのような影響を与えないためには、選択リストに“*”を指定するかわりに、すべての列をコンマで区切って指定してください。また、INSERT文には挿入列リストを指定してください。

1.8 カタログについて

Symfoware/RDBでは、カタログという概念に対して、データベースをマッピングしています。

カタログを扱うAPIには以下のものなどがあります。

- 埋込みSQL連携の場合
  - SET CATALOG文

・ Java連携の場合

　－ DatabaseMetaDataインタフェースのgetColumnsメソッド

　－ DatabaseMetaDataインタフェースのgetTablesメソッド

・ ODBC連携の場合

　－ SQLTables関数

　－ SQLColumnPrivileges関数

　－ SQLTablePrivileges関数

第2章 アプリケーションの開発

本章では、アプリケーションの動作環境および開発環境について説明します。

2.1 アプリケーションの動作環境

Symfoware/RDBでは、利用者が動作環境を指定しなくても、アプリケーションを実行できるように、あらかじめシステムの標準的な動作環境を用意しています。この動作環境は、“図2.1 動作環境のチューニングの段階”に示す単位でのチューニングが可能です。

Symfoware/RDB環境全体:

Symfoware/RDB全体の動作環境のチューニング

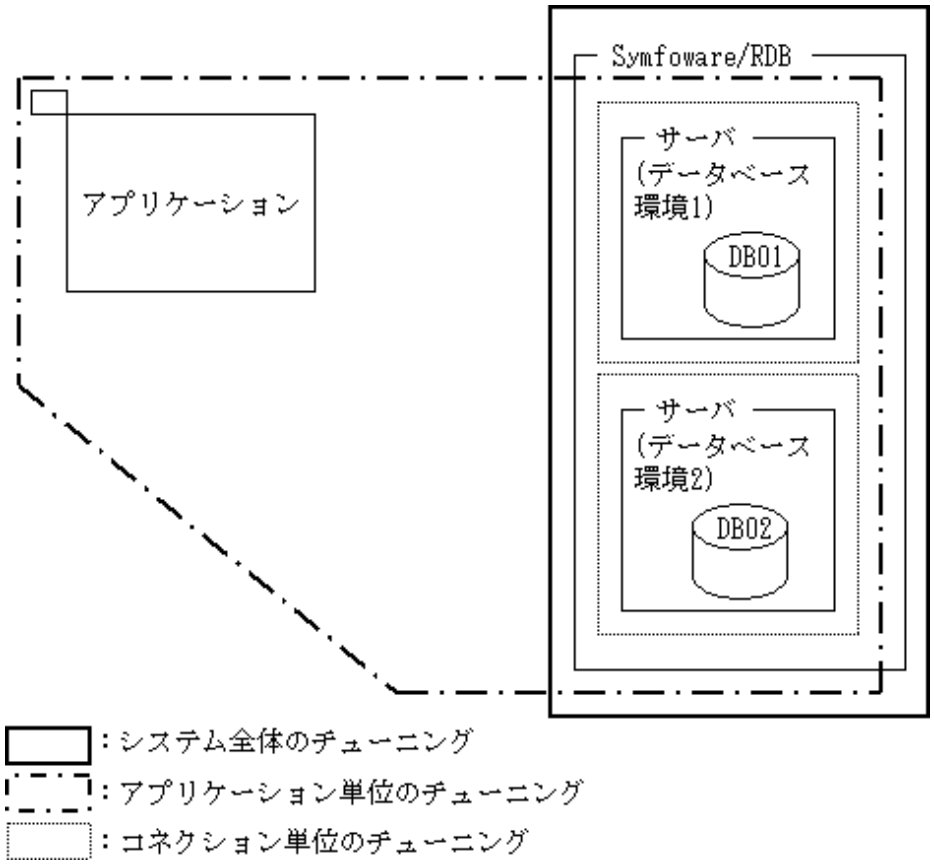
アプリケーション単位:

アプリケーションごとのチューニング

コネクション単位:

アプリケーション中のコネクションごとのチューニング

図2.1 動作環境のチューニングの段階



動作環境のチューニングは、それぞれ対応する動作環境ファイルを作成することによって行います。

2.1.1 動作環境ファイル

ここでは、動作環境ファイルの種類、優先順位、および記述形式について説明します。

動作環境ファイルの種類

動作環境ファイルは、以下の種類があります。

システム用:

1つのSymfoware/RDB環境の標準となる動作環境を設定する場合に作成します。

データベース簡単運用の場合には、Symfoware Severのインストール時に、システム用の動作環境として最適な環境が設定されるため、チューニングの必要がありません。よって、システム用の動作環境ファイルは、作成不要です。

クライアント用:

アプリケーションごとに動作環境を変更する場合に作成します。リモートアクセスのための情報もこのファイルに設定するため、リモートのデータベースにアクセスする場合は、必ず作成します。

ポイント

クライアント用の動作環境ファイルは、埋込みSQL連携およびJava連携の場合に有効となります。ただし、Java連携の場合には、指定不可能なパラメタもあります。詳細は、“B.2 クライアント用の動作環境ファイル”を参照してください。

Java連携の場合は、クライアント用の動作環境ファイルに相当する指定をctuneparamオプションで指定することができます。

ODBC連携の場合は、クライアント用の動作環境ファイルに相当する指定をODOSのオプションで指定することができます。

.NET Framework連携の場合は、クライアント用の動作環境ファイルに相当する指定をtuneparamキーワードで指定することができます。

それぞれの詳細については、以下のマニュアルを参照してください。

- アプリケーション開発ガイド(JDBCドライバ編)
- アプリケーション開発ガイド(ODBCドライバ編)
- アプリケーション開発ガイド(.NET Data Provider編)

サーバ用:

データベース環境に固有の動作環境を設定する場合に作成します。クライアント用の動作環境ファイル中で、コネクションに対応するサーバ用の動作環境ファイルを指定することによって、コネクション(データベース環境)ごとに動作環境を変更することができます。

“図2.2 動作環境ファイルの位置づけ”に、動作環境ファイルの位置づけを、“表2.1 各動作環境ファイルの特徴”に各動作環境ファイルの特徴を示します。

参照

サーバ用の動作環境ファイルについては、“B.3 サーバ用の動作環境ファイル”を参照してください。

システム用の動作環境ファイルについては、“セットアップガイド”を参照してください。

図2.2 動作環境ファイルの位置づけ

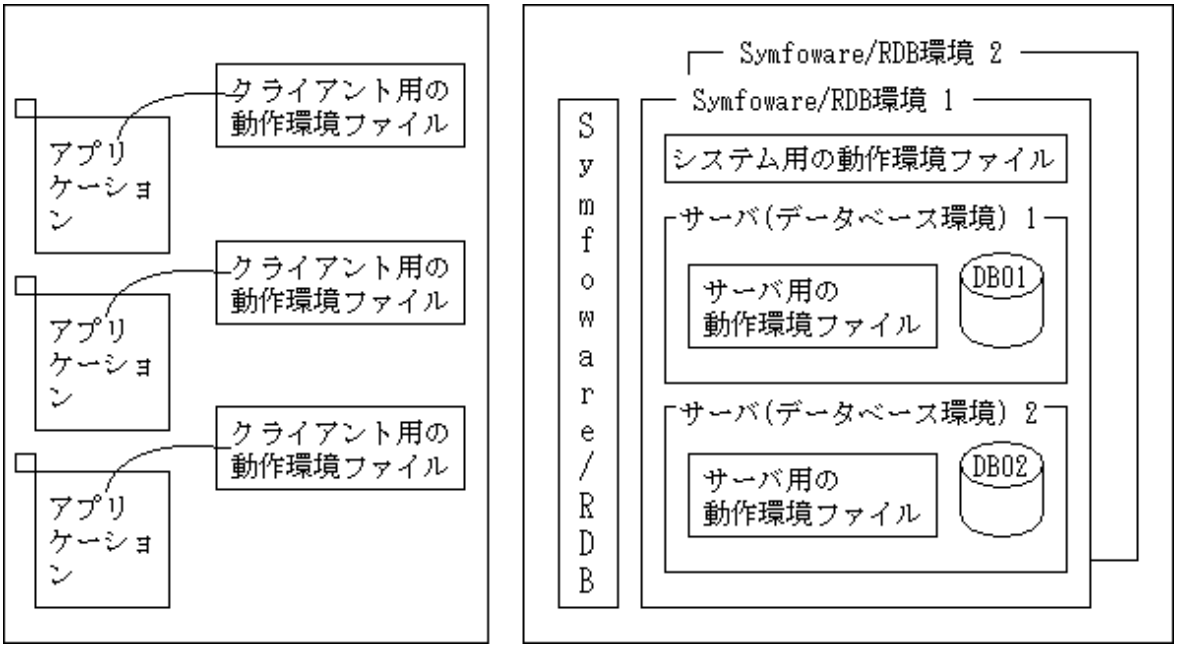


表2.1 各動作環境ファイルの特徴

種類	用途	パラメタの影響範囲	ファイル名の指定箇所	ファイルの作成
システム用	Symfoware/RDB環境ごとにアプリケーションの動作環境を変更する	Symfoware/RDBの起動から停止まで	RDB構成パラメタファイルのRDBSQLENVに絶対パスで指定	省略可
クライアント用	アプリケーションごとにアプリケーションの動作環境を変更する	アプリケーションの実行中	アプリケーション実行時のカレントディレクトリに以下のファイル名で指定 ・ S L fssqlenvc ・ W SQLRT.ENV または環境変数SQLRTENVに絶対パスで指定	リモートのデータベースにアクセスする場合は省略不可。ローカルのみの場合は省略可。
サーバ用	コネクション(データベース環境)ごとに、アプリケーションの動作環境を変更する	現コネクション中	クライアント用の動作環境ファイルのSERVER_ENV_FILEに絶対パスで指定	省略可

ポイント

・ 動作環境ファイルに指定するパラメタは、rdbuptコマンドおよびrdbunlsqlコマンドでも有効になります。

参照

・ RDB構成パラメタファイルについての詳細は、“セットアップガイド”を参照してください。

・ rdbuptコマンドおよびrdbunlsqlコマンドの詳細は、“コマンドリファレンス”を参照してください。

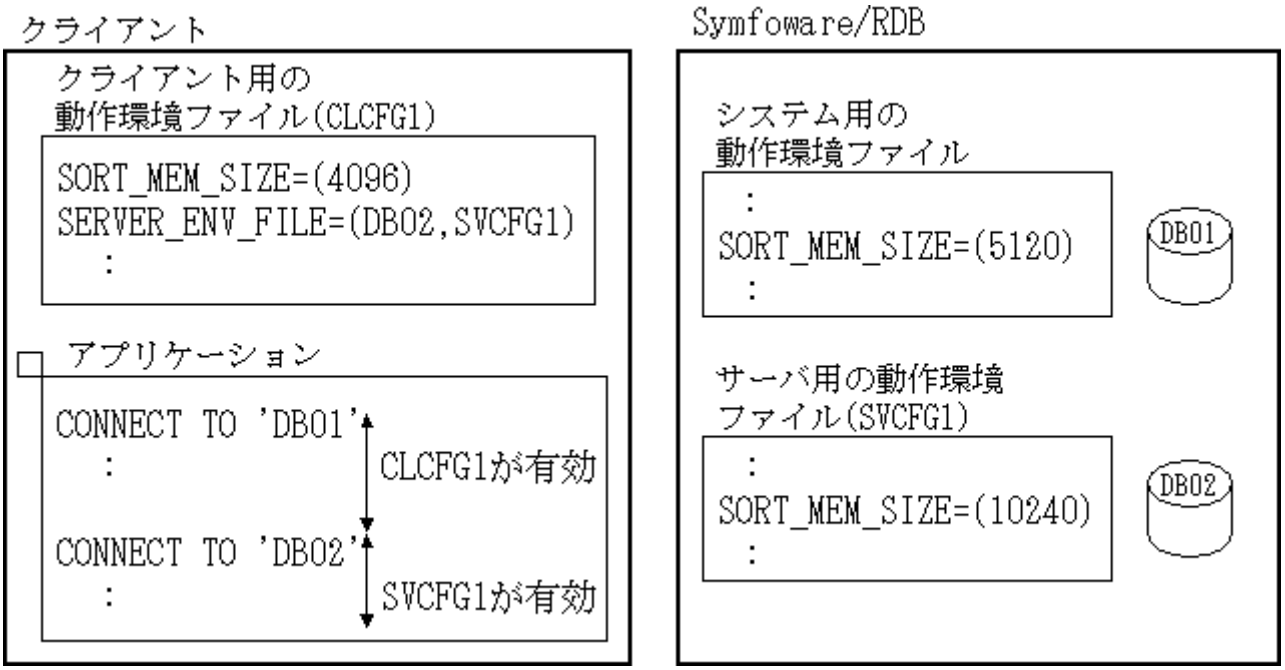
動作環境ファイルの優先順位

動作環境の設定項目の中には、各ファイルに共通して指定できるパラメタがあります。優先順位は、以下のとおりです。

1. サーバ用の動作環境ファイル
2. Java連携の場合、JDBCドライバで指定可能なオプション
   ODBC連携の場合、ODOSのオプションで設定できるパラメタ
   .NET Framework連携の場合、tuneparamキーワード
3. クライアント用の動作環境ファイル
4. システム用の動作環境ファイル

動作環境ファイルの優先順位の例を以下に示します。

図2.3 動作環境ファイルの優先順位の例



たとえば、クライアント用の動作環境ファイルで、SQLサーバ名‘DB02’に対して、サーバ用の動作環境ファイルが定義されている場合は、‘DB02’のコネクションの間は、サーバ用の動作環境ファイルでの指定が有効となります。すなわち、作業用ソート領域は、10240キロバイト獲得されます。

動作環境ファイルの記述形式

動作環境ファイルを記述する文法の一般形式を以下に示します。

```
KEYWORD=(値１, 値２,・・・, 値n)
```

詳細形式および注意事項は以下のとおりです。

・ 実行パラメタのキーワードは英大文字で記述します。

・ 1行には、実行パラメタを1つだけ記述します。

例

1行内に複数の実行パラメタを記述した誤った記述例

```
BUFFER_SIZE = 10 TRAN_SPEC = TRANSACTION_ROLLBACK
```

・ 実行パラメタのカッコ“()”は省略可能です。

・ 実行パラメタの右カッコ“)”以降の記述は、コメントとみなします。

例

実行パラメタにコメントを記述した記述例

```
WAIT_TIME = (15) 待ち時間１５秒
```

・ 行頭にセミコロン“;”のある行は、コメントとみなします。

・ キーワード、等号“=”、カッコ“()”、コンマ“,”およびセミコロン“;”の前後には、空白またはタブを記述することができます。

```
BUFFER_SIZE = (10)                    バッファサイズ10KB
TRAN_SPEC = (TRANSACTION_ROLLBACK)  暗黙のROLLBACK
```

・ 複数指定が不可能な実行パラメタを複数記述した場合には、最後に指定された記述が有効となります。

・ 1行は、227バイト以内で記述する必要があります。

・ コンマ“,”の前後で改行することができます。ただし、その場合、カッコ“()”を省略することはできません。

```
INCLUSION_DSI = (DB01.DSI01, ・・・・・DB08.DSI08,↓
     DB0n.DSI0n )↓
```

↓:改行

・ 値の並びで途中の値を省略する場合には、コンマ“,”だけを記述します。

```
BUFFER_SIZE = ,64 ← 値１を省略した例
```

・ 最後に指定した値のうしろのコンマ“,”は省略することができます。

```
DEFAULT_TABLE_SIZE = (4,512) ← 値３以降を省略した例
```

2.1.2 クライアント用の動作環境ファイルの作成

ここでは、クライアント用の動作環境ファイルの作成方法を説明します。

動作環境ファイルは、動作環境ファイルのひな型をアプリケーション実行時のカレントディレクトリなどに複写して作成します。

動作環境ファイルのひな型のファイル名と格納ディレクトリを以下に示します。

利用する製品名	OS	ディレクトリ	ファイル名
Symfoware Server	Solaris	/opt/FSUNrdb2b/demo	fssqlenvc
	Linux	/opt/FJSVrdb2b/demo	fssqlenvc
	Windows(R)	製品のインストール先ディレクトリ¥ESQL¥ETC	SQLRT.ENV
Symfoware Server クライアント機能	Solaris	/opt/FSUNrdb2b/demo	fssqlenvc
	Linux	/opt/FJSVrdb2b/demo	fssqlenvc
	Windows(R)	製品のインストール先ディレクトリ¥ESQL¥ETC	SQLRT.ENV

参照

埋込みSQL連携の場合

- 作成した動作環境ファイルを指定する方法の詳細は、“アプリケーション開発ガイド(埋込みSQL編)”の“アプリケーションの起動(UNIX系の場合)”または“アプリケーションの起動(Windows(R)の場合)”を参照してください。
- クライアント用の動作環境ファイルの詳細については、“アプリケーション開発ガイド(埋込みSQL編)”の“クライアント用の動作環境ファイルによる定義”を参照してください。

Java連携の場合

- 作成した動作環境ファイルを指定する方法およびクライアント用の動作環境ファイルの詳細は、“アプリケーション開発ガイド(JDBCドライバ編)”の“クライアント用の動作環境ファイルの指定について”を参照してください。

なお、初期設定すべきパラメタについては、“2.1.3 初期パラメタの設定”で説明します。

2.1.3 初期パラメタの設定

アプリケーションを実行するためには、以下のパラメタの設定が必要です。

- アプリケーションの接続先や通信に関する実行パラメタ
- 使用する資源に関する実行パラメタ
- デバッグおよび性能分析に関する実行パラメタ

2.1.3.1 アプリケーションの接続先や通信に関する実行パラメタ

アプリケーションの接続先のサーバ、サーバとの接続時間やサーバからのデータ待ち時間、通信データを暗号化した場合のCA証明書ファイルなど、アプリケーションの接続や通信に関するパラメタを設定します。以下の実行パラメタを設定します。

なお、CA証明書ファイルは、データベース管理者が認証局に発行手続きを行って取得後、アプリケーションを開発および実行するマシンに配布されます。

- DEFAULT_CONNECTION
- SERVER_SPEC
- SSL_CLI_CA_CERT_FILE
- TRAN_TIME_LIMIT
- WAIT_TIME

注意

DEFAULT_CONNECTIONおよびSERVER_SPECは、埋込みSQL連携の場合のみ設定することができます。

DEFAULT_CONNECTION

【指定形式】

接続するデータベースに対するアクセス方法により、指定形式が異なります。

- ローカルアクセスの場合

  DEFAULT_CONNECTION = ([RDBシステム名.]データベース名)

- リモートアクセスの場合

  DEFAULT_CONNECTION = (SQLサーバ名,認可識別子,パスワード)

【実行パラメタの意味】

以下の場合のサーバとの接続情報を指定します。

- アプリケーションにCONNECT文を記述しない場合
- CONNECT文にキーワード“DEFAULT”を指定した場合
- CONNECT文のユーザ指定を省略した場合

【パラメタの意味】

RDBシステム名:

サーバがマルチRDB運用をしている場合に結合するRDBシステム名を指定します。省略した場合は、環境変数RDBNAMEに設定されたRDBシステム名が指定されたとみなします。

データベース名:

接続するデータベース名を指定します。

SQLサーバ名:

接続するSQLサーバ名を指定します。

認可識別子:

OSのログイン名、または、CREATE USER文で登録されている利用者名を指定します。

パスワード:

OSのパスワード、または、CREATE USER文で登録されているパスワードを指定します。

以下の場合、ログイン名およびパスワードの指定は不要です。

この場合、実行時のログイン名および実行時のパスワードが使用されます。

- ローカルアクセスの場合
- 接続先ホスト名に自端末のIPアドレス、自端末のホスト名、“localhost”またはループバックアドレスを指定したリモートアクセスの場合

ログイン名およびパスワードは、各サーバでは以下のように扱われます。

ログイン名:

ログイン名

パスワード:

ログイン名のパスワード

ログイン名:

ユーザ名(ログオン名)

パスワード:

ユーザ名のパスワード

SERVER_SPEC

【指定形式】

SERVER_SPEC = (通信方法,SQLサーバ名,データ資源名,ホスト名,ポート番号[,[接続サーバ種別]])

【実行パラメタの意味】

CONNECT文でSQLサーバ名を指定した場合、サーバとの通信状態を確立するために必要な情報を記述します。このため、CONNECT文で接続するSQLサーバ名は、アプリケーションの実行時にすべてSERVER_SPECに記述しておく必要があります。ローカルのデータベースだけにアクセスする場合は、この実行パラメタは指定しません。

【パラメタの意味】

通信方法:

リモートアクセスの通信方法として、以下のどちらかを選択します。

- RDB2_TCP: TCP/IP通信を行います。通信データを暗号化しない場合に指定します。
- RDB2_TCPS:TCP/IP(SSL)通信を行います。通信データを暗号化する場合に指定します。

データベース簡単運用では、通信データの暗号化はできません。

SQLサーバ名:

CONNECT文で指定したSQLサーバ名を記述します。SQLサーバ名は、SERVER_SPECで指定されたデータ資源に対して接続する際の名前を、利用者が任意に指定します。

データ資源名:

データベース名を指定します。

ホスト名:

ホスト名管理ファイルに設定した18バイト以内のホスト名を指定します。

ポート番号:

リモートアクセスで使用するポート番号を指定します。

Symfoware Serverのバージョンレベルや対象のプラットフォームによって、デフォルトのポート番号が異なります。サーバ側で定義されているポート番号を確認の上、指定してください。

参照

ポート番号の詳細については、“アプリケーション開発ガイド(埋込みSQL編)”を参照してください。

接続サーバ種別:

以下のどちらかを選択します。ただし、通信データを暗号化する場合には、OPENのみが指定できます。また、省略した場合は、OPENが指定されたものとみなします。

- OPEN: Solaris、Linux、Windows(R)上のSymfoware/RDBと接続する場合に指定します。
- GS: グローバルサーバ上のSymfoware/RDBと接続する場合に指定します。

注意

通信データを暗号化する場合、システム用の動作環境ファイルのパラメタ指定が必要です。詳細は、“セットアップガイド”を参照してください。

SSL_CLI_CA_CERT_FILE

【指定形式】

SSL_CLI_CA_CERT_FILE = (SQLサーバ名,CA証明書ファイル名)

【実行パラメタの意味】

サーバ認証で使用する認証局(Certificate Authority: CA)証明書ファイルの配置先を指定します。

通信データを暗号化しない場合に本パラメタを設定するとエラーになります。

【パラメタの意味】

SQLサーバ名:

SERVER_SPECで指定したSQLサーバ名を指定します。

CA証明書ファイル名:

CA証明書のファイル名を絶対パスで指定します。

指定できるファイル形式は、PEM形式のみです。

TRAN_TIME_LIMIT

【指定形式】

TRAN_TIME_LIMIT = (最大トランザクション実行時間)

【実行パラメタの意味】

1つのトランザクションで使用可能な時間を指定します。

指定時間を経過した場合には、トランザクションをロールバックして、接続中のコネクションを切断します。

本実行パラメタの設定は、“表B.1 動作環境ファイルのパラメタ一覧”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

TRAN_TIME_LIMIT = (0)

【パラメタの意味】

最大トランザクション実行時間:

最大トランザクション時間を、0～32767の範囲で指定します。単位は秒です。0を指定すると無制限になります。

WAIT_TIME

【指定形式】

WAIT_TIME = (待ち時間)

【実行パラメタの意味】

サーバからのデータ受信の待ち時間を指定します。

WAIT_TIMEで指定された時間内に、サーバからのデータが受信できなかった場合には、実行中のSQL文はエラーとなり、コネクションは切断されます。

なお、本実行パラメタは、トランザクションモニタ配下では指定できません。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

WAIT_TIME = (0)

【パラメタの意味】

待ち時間:

待ち時間を0～32767の範囲で指定します。単位は秒です。0を指定した場合は、データが受信できるまで待ちます。

2.1.3.2 使用する資源に関する実行パラメタ

アプリケーションを実行するためには、ソート処理用の領域や作業用の領域など、資源に関するパラメタの設定が必要です。以下のパラメタを設定します。

- SORT_MEM_SIZE
- WORK_ALLOC_SPACESIZE
- WORK_MEM_SIZE
- WORK_PATH

SORT_MEM_SIZE

【指定形式】

SORT_MEM_SIZE = (メモリサイズ)

【実行パラメタの意味】

ソート処理のために作業用ソート領域としてサーバ側で使用するメモリの大きさを指定します。この領域は、RDBプロセスのローカルメモリにセション単位に獲得されます。

ソート処理のデータ量がSORT_MEM_SIZEに指定した値を超えると、二次記憶の作業用ソート領域にデータを書き出し、書き出したデータのソートを行います。このとき、二次記憶からのソートデータの読み込み回数はソートデータの全体量とSORT_MEM_SIZEに指定した値に依存します。このため、ソートデータの全体量に応じて、SORT_MEM_SIZEに指定する値を見積もってください。

ソート処理のデータ量がSORT_MEM_SIZEに指定した値の1万倍以上になると、ソート処理で必要なメモリが不足し、「JYP2221E 実行時の制限値を超えました. code:“4”」のエラーとなる場合があります。ただし、メモリ上の作業域の必要最低限なサイズは、作業用ソート領域へのレコードの格納順に依存するため、SORT_MEM_SIZEに指定した値の1万倍は目安となります。

本実行パラメタの設定は、“表B.1 動作環境ファイルのパラメタ一覧”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

SORT_MEM_SIZE = (2112)

【パラメタの意味】

メモリサイズ:

サーバ側で使用するメモリの大きさを64～2097150の範囲で指定します。単位はキロバイトです。

WORK_ALLOC_SPACESIZE

【指定形式】

WORK_ALLOC_SPACESIZE = ([初期量][,[増分量][,[最大量][,[保持指定]]]])

【実行パラメタの意味】

作業用ソート領域および作業用テーブルとしてサーバ側で使用するファイルサイズの初期量、増分量、最大量、保持指定を指定します。

本実行パラメタの設定は、“表B.1 動作環境ファイルのパラメタ一覧”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

WORK_ALLOC_SPACESIZE = (10000,50000,WORK_PATHで指定したパス名のディスク容量,HOLD)

初期量、増分量、最大量、保持指定のいずれかの値が省略された場合は、その値のデフォルト値が指定されたものとみなします。

以下に指定例を示します。

例1:初期量、増分量を指定する場合

WORK_ALLOC_SPACESIZE = (10000,50000)

例2:増分量、最大量を指定する場合

WORK_ALLOC_SPACESIZE = (,50000,100000)

【パラメタの意味】

初期量:

作業用ソート領域および作業用テーブルとして外部ファイルを作成する場合の初期量を5000～1000000の範囲で指定します。省略した場合は10000が指定されたものとみなします。単位はキロバイトです。

増分量:

作業用ソート領域および作業用テーブルとして作成した外部ファイルを拡張する場合の増分量を1000～1000000の範囲で指定します。省略した場合は、50000が指定されたものとみなします。単位はキロバイトです。

最大量:

作業用ソート領域および作業用テーブルとして作成する外部ファイルの最大量を5000～33553408の範囲で指定します。省略した場合は、WORK_PATHで指定したパス名のディスク容量が指定されたものとみなします。単位はキロバイトです。

保持指定:

以下の中から1つを選択します。省略した場合は、HOLDが指定されたものとみなします。

- FREE:初期量として獲得した作業用ソート領域および作業用テーブルの外部ファイルは、DISCONNECT文の実行時に解放します。
  拡張量として獲得した作業用ソート領域および作業用テーブルの外部ファイルは、その領域を使用したSQL文の実行完了時に解放します。
- HOLD:作業用ソート領域および作業用テーブルとして作成した外部ファイルは、DISCONNECT文の実行時に解放します。

WORK_MEM_SIZE

【指定形式】

WORK_MEM_SIZE = (メモリサイズ)

【実行パラメタの意味】

作業用テーブルとしてサーバ側で使用するメモリの大きさを指定します。この領域は、RDBプロセスのローカルメモリにセション単位に獲得されます。

本実行パラメタの設定は、“表B.1 動作環境ファイルのパラメタ一覧”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

WORK_MEM_SIZE = (128)

【パラメタの意味】

メモリサイズ:

サーバ側で使用するメモリの大きさを64～2097150の範囲で指定します。単位はキロバイトです。

WORK_PATH

【指定形式】

WORK_PATH = (パス名[,パス名]・・・)

【実行パラメタの意味】

サーバ側で使用するソート作業域、作業用テーブル域の獲得先ディレクトリを指定します。“Symfoware/RDBを起動するユーザID”および“RDBコマンドを実行するユーザID”には、指定するディレクトリへの書込み権が必要です。

本実行パラメタの設定は、“表B.1 動作環境ファイルのパラメタ一覧”の優先順位に従います。いずれの動作環境ファイルにも本実行パラメタが指定されなかった場合、以下の値が設定されます。

S L

WORK_PATH = (/var/tmp)

W

WORK_PATH = (Symfoware/RDBがインストールされているディレクトリ¥TMP)

作業用ソート領域および作業用テーブルの見積りについては“C.1 ソート作業域の見積り”を参照してください。

【パラメタの意味】

パス名:

獲得先ディレクトリを指定します。

2.1.3.3 デバッグおよび性能分析に関する実行パラメタ

アプリケーションのデバッグおよび性能分析に関する設定を行うために、以下の実行パラメタを設定します。

- ROUTINE_SNAP

- SQL_SNAP
- ACCESS_PLAN
- SQL_TRACE

ROUTINE_SNAP

【指定形式】

ROUTINE_SNAP=({ON|OFF},ファイル名[,出力レベル])

【実行パラメタの意味】

ROUTINE_SNAP機能を利用するかどうかを指定します。

ROUTINE_SNAP機能は、SQL手続き文の実行情報をファイルに出力する機能です。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

ROUTINE_SNAP = (OFF)

参照

ROUTINE_SNAP機能の詳細および使用方法については、“アプリケーション開発ガイド(埋込みSQL 編)”の“アプリケーションのデバッグ”を参照してください。

【パラメタの意味】

ON:

ROUTINE_SNAP機能を利用する場合に指定します。

OFF:

ROUTINE_SNAP機能を利用しない場合に指定します。

ファイル名:

SQL手続き文の実行情報の出力先のサーバ側のファイル名を、絶対パスで指定します。指定されたファイルがすでに存在する場合は、情報を追加して出力します。
複数のアプリケーションが動作する場合は、クライアント用の動作環境ファイルの実行パラメタ(DIV_TRACE_FILE)の指定により、出力ファイル名の後にプロセスIDなどの情報を付加して、個別のトレース情報を出力します。
アプリケーションがマルチスレッド環境で動作する場合は、DIV_TRACE_FILEの指定に関係なく、出力ファイル名の後にプロセスIDやセションIDなどの情報を自動的に付加して、個別のトレース情報を出力します。

出力レベル:

出力する情報のレベルとして、1または2を指定します。省略した場合は、2が指定されたものとみなします。

参照

出力レベルの指定と出力情報の対応については、“アプリケーション開発ガイド(埋込みSQL編)”の“ROUTINE_SNAP機能の利用方法”を参照してください。

SQL_SNAP

【指定形式】

SQL_SNAP = ({ON | OFF}[,[スナップファイル名][,[出力レベル][,[繰り返し幅]]]])

【実行パラメタの意味】

SQL_SNAP機能を利用するかどうかを指定します。

SQL_SNAP機能は、アプリケーションが実行したSQL文の情報をファイルに出力する機能です。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

SQL_SNAP = (OFF)

参照

SQL_SNAP機能の詳細および使用方法については、“アプリケーション開発ガイド(埋込みSQL編)”の“アプリケーションのデバッグ”を参照してください。

注意

本機能の利用は性能に影響を与えるため、デバッグ作業が終了したら、本実行パラメタの設定をOFFにしてください。

また、格納データを暗号化している場合でも、スナップファイル内の情報は暗号化されません。そのため、デバッグ作業が終了したら、本実行パラメタの設定をOFFにしてください。

【パラメタの意味】

ON:

SQL_SNAP機能を利用する場合に指定します。

OFF:

SQL_SNAP機能を利用しない場合に指定します。

スナップファイル名:

SQL_SNAP機能が出力するSQL文の実行情報の出力先ファイル名を指定します。ファイル名を省略した場合は、アプリケーションのファイル名の拡張子を“.SNP”に変更したものとなります。ファイル名にディレクトリの指定がない場合は、カレントディレクトリが指定されたものとみなします。指定されたファイルがすでに存在する場合は、情報を追加して出力します。
複数のアプリケーションが動作する場合は、クライアント用の動作環境ファイルの実行パラメタ(DIV_TRACE_FILE)の指定により、出力ファイル名の後にプロセスIDなどの情報を付加して、個別のトレース情報を出力します。
アプリケーションがマルチスレッド環境で動作する場合は、DIV_TRACE_FILEの指定に関係なく、出力ファイル名の後にプロセスIDやセションIDなどの情報を自動的に付加して、個別のトレース情報を出力します。

出力レベル:

出力する情報のレベルとして、1、2、PRC1、PRC2のいずれかを指定します。省略した場合は、2が指定されたものとみなします。

参照

出力レベルの指定と出力情報の対応については、“アプリケーション開発ガイド(埋込みSQL編)”の“SQL_SNAP機能の利用方法”を参照してください。

繰り返し幅:

出力する情報の繰り返し幅を、1から32767の範囲で指定します。1つのSQL文の出力を1とします。繰り返し幅を指定した場合は、その幅でサイクリックに情報を出力します。省略した場合は、先頭からの情報をすべて出力します。

ACCESS_PLAN

【指定形式】

ACCESS_PLAN = ({ON | OFF},ファイル名[,[出力レベル][,[SQLアドバイザ出力レベル]]])

【実行パラメタの意味】

アプリケーション単位でアクセスプランを取得するかどうかを指定します。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

ACCESS_PLAN = (OFF)

アクセスプランについては、“3.1 アクセスプラン”を参照してください。

【パラメタの意味】

ON:

アクセスプラン取得機能を利用する場合に指定します。

OFF:

アクセスプラン取得機能を利用しない場合に指定します。

ファイル名:

出力先のサーバ側のファイル名を、絶対パスで指定します。指定されたファイルがすでに存在する場合は、情報を追加して出力します。
また、指定するパスが存在することと、CONNECT文で指定したユーザIDに対する書込み権があることを確認してください。

出力レベル:

出力レベルには1または2を指定します。1を指定すると、アクセスプランのセクション情報のみを出力します。2を指定すると、セクション内の各エレメント詳細情報も出力します。省略した場合は、2が指定されたものとみなします。

SQLアドバイザ出力レベル:

SQLアドバイザ出力レベルには、“ADVICE”または“NOADVICE”を指定します。“ADVICE”を指定すると、SQL文に対するアドバイスを出力します。“NOADVICE”を指定すると、SQL文に対するアドバイスを出力しません。省略した場合は、“ADVICE”が指定されたものとみなします。

SQL_TRACE

【指定形式】

SQL_TRACE = ({ON | OFF},性能情報ファイル名[,出力レベル])

【実行パラメタの意味】

アプリケーション単位でSQL性能情報を取得するかどうかを指定します。

本実行パラメタの指定を省略した場合は、以下の値が設定されます。

SQL_TRACE = (OFF)

【パラメタの意味】

ON:

SQL性能情報取得機能を利用する場合に指定します。

OFF:

SQL性能情報取得機能を利用しない場合に指定します。

性能情報ファイル名:

出力先のサーバ側のファイル名を、絶対パスで指定します。指定されたファイルがすでに存在する場合は、情報を追加して出力します。
複数のアプリケーションが動作する場合は、クライアント用の動作環境ファイルの実行パラメタ“DIV_TRACE_FILE”の指定により、出力ファイル名の後にプロセスIDなどの情報を付加して、個別のトレース情報を出力します。
アプリケーションがマルチスレッド環境で動作する場合は、DIV_TRACE_FILEの指定に関係なく、出力ファイル名の後にプロセスIDやセションIDなどの情報を自動的に付加して、個別のトレース情報を出力します。

出力レベル:

出力レベルには1または2を指定します。1を指定すると、DSOごとに集計された性能情報を出力します。2を指定すると、DSI単位の情報までも出力します。

データベース簡単運用の場合は、表またはインデックスに関する情報が出力されます。出力レベルによる出力内容に違いはありません。

省略した場合は、2が指定されたものとみなします。

参照

出力レベルによる、出力内容の詳細については、“SQLTOOLユーザーズガイド”の“実行エレメント情報”を参照してください。

2.2 アプリケーションの開発環境

アプリケーションの統合開発環境を利用したアプリケーション開発について説明します。

2.2.1 利用可能な開発言語・統合開発環境

Symfoware Serverでは、以下の、言語、連携方法を使用してアプリケーションを開発できます。

開発言語	連携方法
C	埋込みSQL連携
C++	ODBC連携
C#	.NET Framework連携
COBOL	埋込みSQL連携
	ODBC連携
	.NET Framework連携
Java	Java連携
Visual Basic	ODBC連携
Visual Basic .NET	.NET Framework連携
	ODBC連携

以下のアプリケーション統合開発環境を利用することで、初心者でもわかりやすい操作でアプリケーションを開発することができます。

- Interstage Studio
  - COBOL(埋込みSQL連携、ODBC連携)
  - Java(Java連携)
- Visual Studio
  - C(埋込みSQL連携)
  - C++(ODBC連携)
  - C#(.NET Framework連携)
  - COBOL(.NET Framework連携)
  - Visual Basic(ODBC連携)
  - Visual Basic .NET(ODBC連携、.NET Framework連携)

注意

1つのアプリケーションプロセス内で、複数の連携方法を混在して使用することはできません。

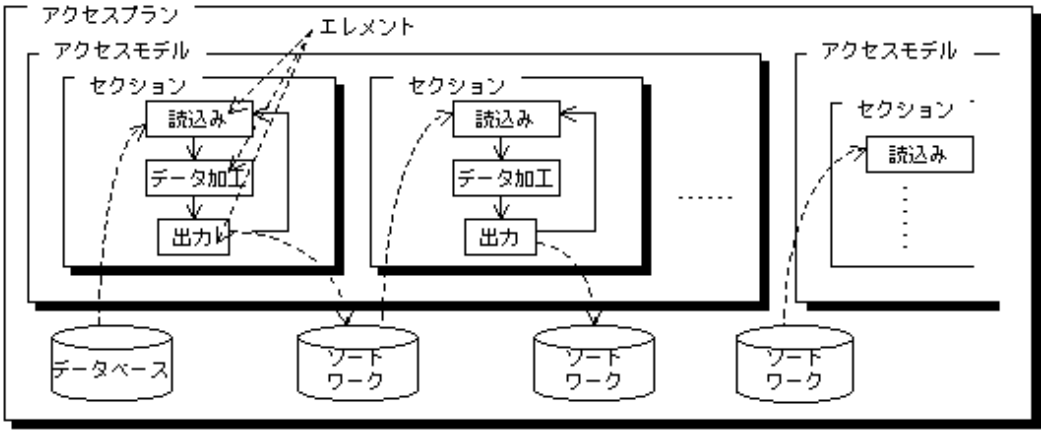
第3章 アプリケーションの性能

本章では、アプリケーションの性能を向上させるための考え方を説明しています。

3.1 アクセスプラン

Symfoware Serverでは、データをアクセスする際に、SQL文を入力として、SQL文で指定した表の格納構造やインデックス情報から、探索条件に見合った処理手順を決めています。これを最適化処理(または、オプティマイザ)と呼び、決定される処理手順をアクセスプランと呼びます。アクセスプランは、SQL文を実行するときのデータベースへのアクセスシーケンスを表現したものです。

アクセスプランの概念図を以下に示します。



3.1.1 アクセスモデルとは

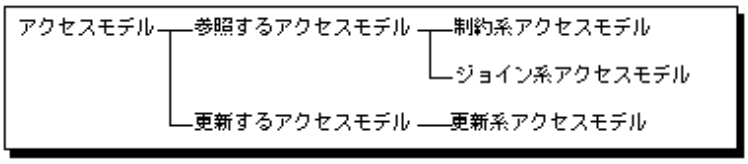
アクセスモデルとは、Symfoware/RDBがデータベースにアクセスするときに選択するアクセス手順を分類したものです。複雑なSQL文を実行するときは、アクセスモデルを複数組み合わせてアクセスプランを作成し、そのアクセスプランに従ってデータベースにアクセスすることによりSQL文を実行しています。アクセスモデルから、Symfoware Serverが持つアクセスシーケンスと特性を知ることができます。

アクセスモデルの種類

アクセスモデルは、データベースを参照するアクセスモデルと、データベースを更新するアクセスモデルに大別されます。

さらに、データベースを参照するアクセスモデルは、単一の表からデータを取り出すアクセスモデル(制約系アクセスモデル)と、制約系アクセスモデルを組み合わせて複数の表をジョインするアクセスモデル(ジョイン系アクセスモデル)からなります。

アクセスモデルの分類を以下に示します。



アクセスモデルについての詳細は、“H.1 アクセスモデルの種類”を参照してください。

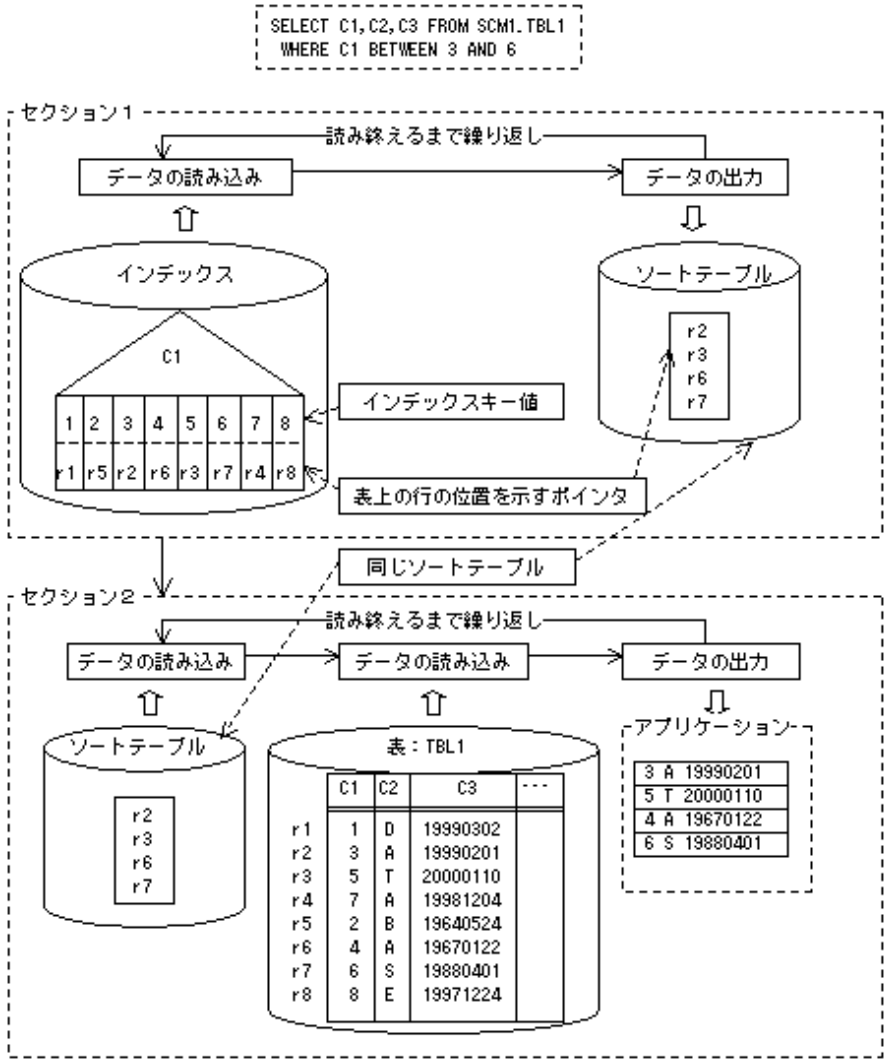
3.1.2 セクションとは

セクションとは、データベース資源からデータを読み出し、条件の判定やジョイン、データの加工を施し、中間結果(ソートテーブルやワークテーブル)やアプリケーションにデータを出力するなどといった処理の単位です。アクセスモデルは、単一あるいは複数のセクションから構成されています。

3.1.3 エレメントとは

エレメントとは、セクション内において、データを読み込む、データを加工する、データを出力するなど、セクション内の細かい処理単位です。セクションは複数のエレメントから構成されています。

各エレメントは、一番最初のデータ読み込みがデータの終了を検出するまで繰り返し処理を行います。下記の図では、まずセクション1でインデックスを検索し、その結果をソートテーブルにため込んでいます。インデックスの検索範囲のデータをすべてソートテーブルに格納するとセクション1は終了します。次にセクション2がソートテーブルからデータを読み込み、読み込んだデータごとに表から対応するデータの読み込みを行い、結果をアプリケーションに出力しています。



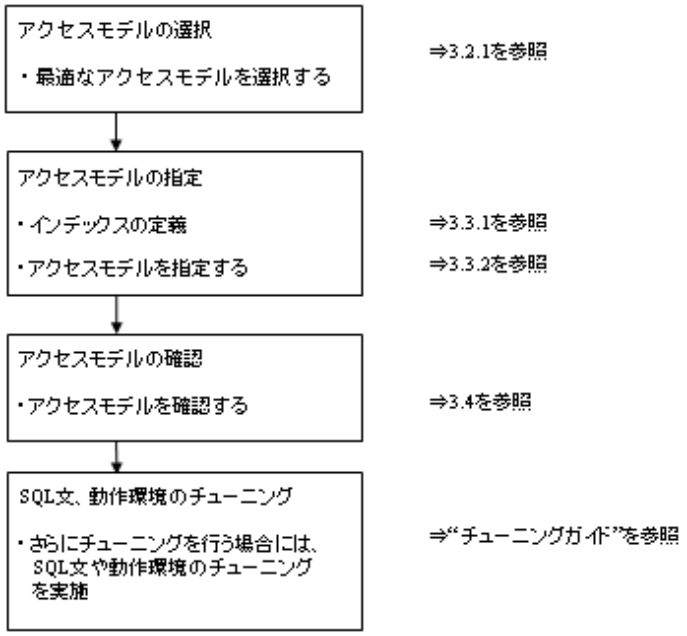
エレメントについての詳細は、“H.2 エレメントの種類”を参照してください。

3.2 アプリケーション開発時の性能チューニング

Symfoware Serverでは、最適化処理により、インデックス検索時に使用するインデックスやジョインの方法などのアクセスモデルが自動的に選択されますが、事前に、アクセスモデルを固定化する性能チューニングを実施しておくことで、最適なデータアクセスを実現させることができます。

以下に、性能チューニングの手順を説明します。

性能チューニングの手順



データベース設計の段階で、表の格納構造やインデックス定義を適切に行っておくことで、アプリケーション性能を向上させることができます。また、データベースへのアクセス手順として、検索方法やジョイン方法に関するアクセスモデルを意図的に選択することで、性能トラブルなどを未然に防ぐことが可能となります。

3.2.1 アクセスモデルの選択

データ量や処理内容に応じた、最適なアクセスモデルを選択します。

主なアクセスモデルと、アクセスモデルを選択する際の指針を示します。

分類	アクセスモデル	処理概要	選択の指針	ASSIST指定での指定方法
制約系アクセスモデル	インデックス検索	探索条件に指定した条件からインデックスの検索範囲を作成し、インデックスを検索します。	検索範囲を局所化できるため、検索効率が上がります。	制約ASSIST (ASSISTで使用するインデックスを指定する)
	表の全件検索	表の全レコードを読み込みます。	表の全データを読み込むため検索効率が下がります。	なし
ジョイン系アクセスモデル	マージジョイン(インデックスを利用してジョインする)	ジョインキーを昇順に読み込む方法として、インデックスを利用します。	大量のデータをジョインする場合に有効です。	ジョインASSIST (ASSIST指定でジョイン方法を指定する)
	ソートマージ(ソートテーブルを利用してジョインする)	ジョインする表に対し、制約系アクセスモデルで得られたレコードをジョインキーで	大量のデータをジョインする場合に有効です。	ジョインASSIST (ASSIST指定でジョイン方法を指定する)

分類	アクセスモデル	処理概要	選択の指針	ASSIST指定での指定方法
		ソートしてからジョインします。		
	フェッチジョイン	ジョインする表に対し、制約系アクセスモデルで得られたレコードをもとに、結合する表をインデックス検索し、ジョインします。	少量のデータをジョインする場合に有効です。	ジョインASSIST(ASSIST指定でジョイン方法を指定する)
	ネスティッドループ	ジョインする表を直接読み込みながらレコードの組み合わせを作成し、ジョインします。	表の全組み合わせを作成するため、ジョインに時間がかかります。	なし

アクセスモデルについての詳細は、“H.1 アクセスモデルの種類”を参照してください。

アクセスモデルは、ASSIST指定で指定します。

ASSIST指定の指定方法についての詳細は、“3.3.2 ASSIST指定でアクセスモデルを指定する”を参照してください。

3.3 アクセスモデルの指定

選択したアクセスモデルを指定する方法を説明します。

- インデックスを定義する
- ASSIST指定でアクセスモデルを指定する

3.3.1 インデックスを定義する

インデックスを付加すると、データの検索処理の効率を上げることができます。逆に、アプリケーションでの処理が、インデックスを構成する列の更新や、データの挿入、データの削除を主体としている場合は、インデックスの更新処理が行われるため、処理効率が低下することがあります。インデックスの設計は、これらを考慮し、必要なインデックスを検討しなければなりません。

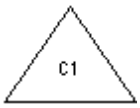
インデックスの設計手順を以下に示します。

1. 探索条件の列挙
2. インデックスの定義
3. インデックスの定義順序の決定

参考

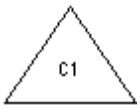
本マニュアルでは、インデックスを以下のように表現することがあります。

- 1つの列から構成されるインデックスの場合
  C1を列名リストに指定したインデックスです。以下のようなインデックスのDSO定義文で定義されたインデックスを意味します。



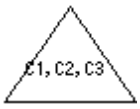
```
CREATE DSO TBL1_IX1_DSO INDEX ON SCM1.TBL1(C1)
 TYPE BTREE (PAGESIZE1(4),PAGESIZE2(4))
```

・ 1つの列から構成されるインデックスの場合(データベース簡単運用の場合)
C1を列名リストに指定したインデックスです。データベース簡単運用の場合は、以下のようなCREATE INDEX文で定義されたインデックスを意味します。



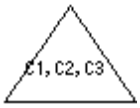

```
CREATE INDEX SCM1.TBL1.IX1 KEY(C1)
```

・ 複数の列から構成されるインデックスの場合
C1、C2、C3を列名リストに指定したインデックスです。以下のようなインデックスのDSO定義文で定義されたインデックスを意味します。マルチカラムインデックスと呼ぶこともあります。




```
CREATE DSO TBL1_IX1_DSO INDEX ON SCM1.TBL1(C1,C2,C3)
 TYPE BTREE (PAGESIZE1(4),PAGESIZE2(4))
```

・ 複数の列から構成されるインデックスの場合(データベース簡単運用の場合)
C1、C2、C3を列名リストに指定したインデックスです。データベース簡単運用の場合は、以下のようなCREATE INDEX文で定義されたインデックスを意味します。マルチカラムインデックスと呼ぶこともあります。




```
CREATE INDEX SCM1.TBL1.IX1 KEY(C1,C2,C3)
```

1.探索条件の列挙

アプリケーションで検索、更新する際の探索条件をすべて列挙します。

2.インデックスの定義

アプリケーションでのデータベースの検索、更新処理において、WHERE探索条件に指定する検索や更新のキーとなる列と、表の結合条件となる列にはインデックスを定義します。

ブール演算子“OR”で条件を指定している場合は、“OR”で指定した各々の条件ごとにインデックスの定義を検討します。

例1

```
SELECT *
 FROM  SCM1.TBL1
 WHERE C1 = 1 OR C2 BETWEEN 10 AND 20
```

この場合、TIDユニオンマージのアクセスモデルが動作します。(C1,C2)のマルチカラムインデックス(複数の列から構成されるインデックス)を定義しても、インデックスは利用されないので、C1とC2は別々のインデックスとして定義してください。

TIDユニオンマージについては、“TIDユニオンマージ”を参照してください。

例2

```
SELECT *
 FROM  SCM1.TBL1
 WHERE (C3=1 AND C4=2)
  OR   (C5=3 AND C6=4)
```

注意

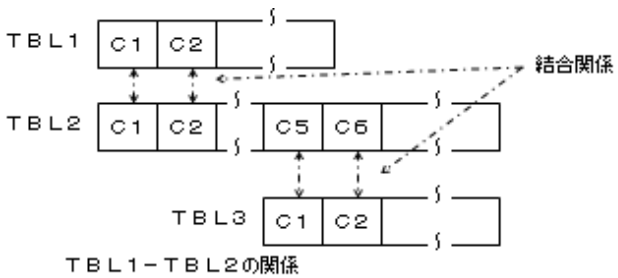
NOTと他の述語を一緒に使用した場合や、述語中に算術演算を指定した場合は、その列を先頭に持つインデックスを定義していても、インデックスは利用しません。
単独で条件として指定される列でも、たとえば性別のように、異なるキー値数が極端に少ないものにはインデックスを定義しないようにしてください。

3.定義順序の決定

マルチカラムインデックスを定義する時は、列の定義順序を決定します。

・ 表の結合関係が複数列である場合は、関係のある列の順序でインデックスを定義します。

例



ＴＢＬ１－ＴＢＬ２の関係

表名	パターン１	パターン２	パターン３
ＴＢＬ１	(Ｃ１，Ｃ２)	(Ｃ２，Ｃ１)	(Ｃ１，Ｃ２)
ＴＢＬ２	(Ｃ１，Ｃ２)	(Ｃ２，Ｃ１)	(Ｃ２，Ｃ１)
評価	○	○	×

ＴＢＬ２－ＴＢＬ３の関係

表名	パターン１	パターン２	パターン３
ＴＢＬ２	(Ｃ５，Ｃ６)	(Ｃ６，Ｃ５)	(Ｃ５，Ｃ６)
ＴＢＬ３	(Ｃ１，Ｃ２)	(Ｃ２，Ｃ１)	(Ｃ２，Ｃ１)
評価	○	○	×

・ 列の組合わせで条件を指定する場合と、単独の列で条件を指定する必要がある場合は、単独で指定する列を優先的に先頭に定義します。

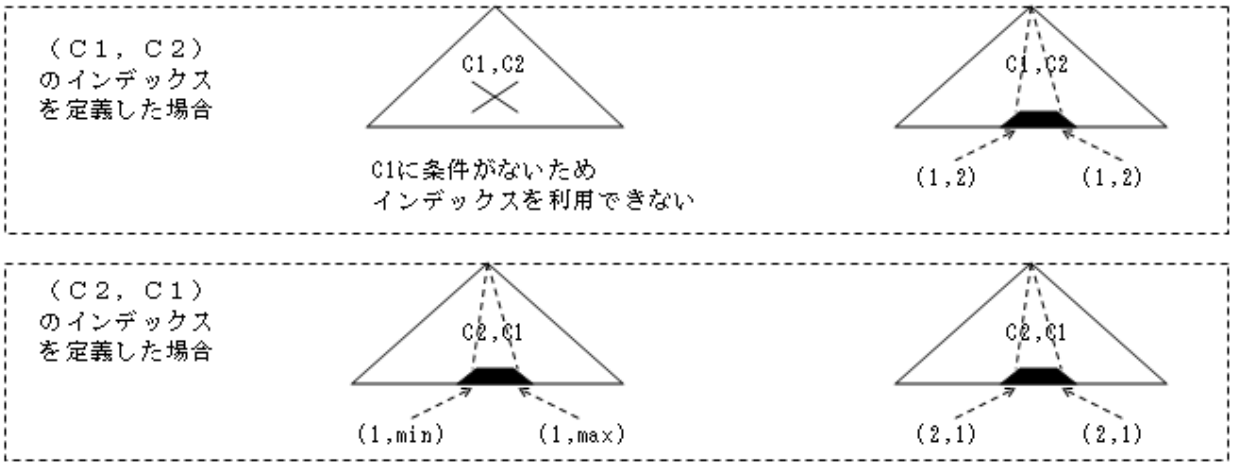
例

問い合わせ指定１

```
SELECT *
 FROM SCM1.TBL1
 WHERE C2 = 1
```

問い合わせ指定２

```
SELECT *
 FROM SCM1.TBL1
 WHERE C1 = 1 AND C2 = 2
```

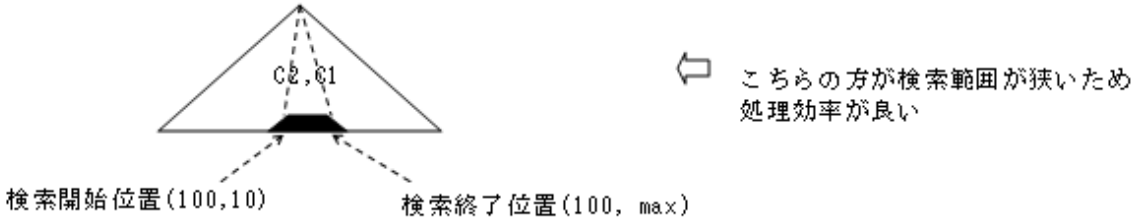


・ 検索を効率化するためのインデックス(表のレコードを絞り込むためのインデックス)は、比較述語“=”で条件を指定する列を優先的に先頭に定義します。

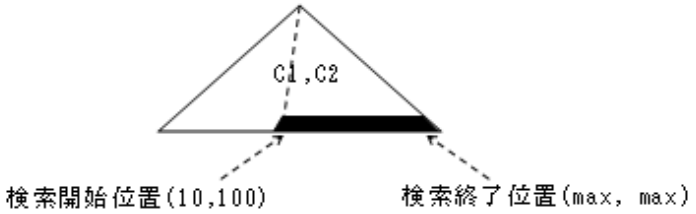
例

```
SELECT  *
 FROM  SCM1.TBL1
 WHERE C1 >=10 AND C2 = 100
```

（Ｃ２，Ｃ１）の順でインデックスを定義



（Ｃ１ ，Ｃ２）の順でインデックスを定義



・ 比較述語“=”で指定せず、BETWEEN述語などで条件を指定する場合は、実際の検索範囲が狭くなる列(検索開始位置、検索終了位置に同じ値を指定する列)を優先的に先頭に定義します。

3.3.2 ASSIST指定でアクセスモデルを指定する

アクセスモデルは、SQL文にASSIST指定を記述することで、指定できます。

本項では、ASSIST指定の使用方法について、例を使用して説明します。

以降の例で、“在庫管理.在庫表”、“在庫管理.発注表”および“在庫管理.会社表”という表を用いています。

“在庫管理.在庫表”、“在庫管理.発注表”および“在庫管理.会社表”の表定義とデータの詳細、および格納構造は次のとおりです。

表3.1 在庫管理.在庫表

製品番号	製品名	在庫数量	倉庫番号
110	テレビ	85	2
111	テレビ	90	2
123	冷蔵庫	60	1
124	冷蔵庫	75	1
140	CDプレーヤー	120	2
212	テレビ	0	2
215	ビデオ	5	2
226	冷蔵庫	8	1
227	冷蔵庫	15	1
240	CDプレーヤー	25	2
243	CDプレーヤー	14	2
351	CD	2500	2

表3.2 在庫管理.在庫表のインデックス

インデックスのDSO名またはインデックス名	構成列名
INDEX1	製品番号
INDEX2	倉庫番号

表3.3 在庫管理.発注表

取引先	取引製品	仕入価格	発注数量
61	123	48000	60
61	124	64000	40
61	140	8000	80
61	215	240000	10
61	240	80000	20
62	110	37500	120
62	226	112500	20
62	351	375	800
63	111	57400	80
63	212	205000	30
63	215	246000	10
71	140	7800	50
71	351	390	600
72	140	7000	70
72	215	210000	10
72	226	105000	20
72	243	84000	10
72	351	350	1000
74	110	39000	120
74	111	54000	120
74	226	117000	20
74	227	140400	10
74	351	390	700

表3.4 在庫管理.発注表のインデックス

インデックスのDSO名またはインデックス名	構成列名
INDEX3	取引先
INDEX4	仕入価格
INDEX5	取引製品

表3.5 在庫管理.会社表

会社番号	会社名	業種コード	電話番号	地域コード	住所
61	アダム電気	1	111－777－4444	1	東京都 小田区 新蒲田 7－8－9
62	アイデア商事	2	222－888－5555	1	神奈川県 小浜市 旭区 1－2－3
63	大月産業	3	333－999－6666	1	埼玉県 浦和町 大崎 1－1－1

会社番号	会社名	業種コード	電話番号	地域コード	住所
71	川上電気	1	444－111－7777	2	大阪府 境田市 赤坂台４－５－６
72	龍巻産業	3	555－222－8888	2	大阪府 灘町 東淀川 2－3－7
74	第一商事	2	666－333－9999	2	京都府 中市 4－16－16

表3.6 在庫管理.会社表のインデックス

インデックスのDSO名またはインデックス名	構成列名
INDEX6	業種コード
INDEX7	地域コード

3.3.2.1 ASSIST指定の使用方法

ASSIST指定は、以下のSQL文のSELECTキーワードの直後の場所にコメントとして指定します。

- カーソル指定
- 単一行SELECT文
- INSERT文の問合せ指定
- 導出表
- 副問合せ
- rdbunlsqlコマンドの問合せ式

ASSIST指定は注釈なので、ASSIST指定にエラーがあった場合でも無視して動作します。

ASSIST指定が無効となっている場合には、アドバイスとして、メッセージが以下に出力されます。メッセージに従って対処してください。

- アクセスプラン
- SQLアドバイザ
- パフォーマンスモニタ

3.3.2.2 ASSIST指定の種類

ASSIST指定には、以下の種類があります。

- 制約ASSIST

  データアクセス時に使用するインデックスを指定することで、制約系アクセスモデルを選択できます。

  詳細については、“3.3.2.3 制約ASSISTの使用方法”を参照してください。

- ジョインASSIST

  ジョイン方法を指定することで、ジョイン系アクセスモデルを選択できます。

  詳細については、“3.3.2.4 ジョインASSISTの使用方法”を参照してください。

- 実行制御ASSIST

  SQL文の最初の1件の検索を高速化できます。

  詳細については、“3.5.1.1 実行制御ASSISTの使用方法”を参照してください。

アクセスモデルは、制約ASSIST、ジョインASSISTを使用して指定します。

3.3.2.3 制約ASSISTの使用方法

ここでは、制約ASSISTの使用方法について説明します。

制約ASSISTには、USE_INDEXがあります。

参照

USE_INDEXの一般規則については、“SQLリファレンス”の“ASSIST指定”を参照してください。

USE_INDEX

ASSIST表要素に指定された表に対して、インデックスDSO名またはインデックス名で指定したインデックスを使用した、制約系のアクセスモデルを選択します。

USE_INDEXを指定しない場合は、使用するインデックスを最適化処理が決定します。

参照

最適化処理のインデックスの選択基準は、“チューニングガイド”の“アクセス方式のチューニング”を参照してください。

注意

- 非活性状態のインデックスのDSO名またはインデックス名は指定できません。
- 探索条件にインデックス構成列を指定していないなど、インデックスのDSO名またはインデックス名に指定されたインデックスの検索範囲が作成できない場合でも、指定したインデックスをINDEX ALL SCANのアクセス方式で検索します。

制約ASSISTの使用例

制約ASSISTの使用例を以下に示します。

例

以下のようなSQL文があります。

```
SELECT 取引製品,仕入価格
   FROM 在庫管理.発注表
   WHERE 取引先 = :VAL1
     AND 取引製品 BETWEEN :VAL2 AND :VAL3
```

USE_INDEXを指定しない場合、使用するインデックスに以下の候補があり、選択率の小さいINDEX3がアクセスプランに選択されます。

－ INDEX3 を「取引先 = :VAL1」の範囲で検索

－ INDEX5 を「取引製品 BETWEEN :VAL2 AND :VAL3」の範囲で検索

取引製品に対する条件は範囲で指定できますが、:VAL2と:VAL3に同じ値が設定される場合、INDEX5を使用した方が性能が良いということになります。

そこで、インデックスDSOのINDEX5を使用して検索するように、USE_INDEXを指定します。

```
SELECT /* ASSIST USE_INDEX(発注表(INDEX5)) */
     取引製品,仕入価格
    FROM 在庫管理.発注表
```

```
  WHERE 取引先 = :VAL1
    AND 取引製品 BETWEEN :VAL2 AND :VAL3
```

3.3.2.4 ジョインASSISTの使用方法

ここでは、ジョインASSISTの使用方法について説明します。

ジョインASSISTとして、LEADING_TABLEを指定する使い方、JOIN_RULEを指定する使い方、LEADING_TABLEとJOIN_RULEを同時に指定する使い方があります。

参照

ジョインASSISTの一般規則については、“SQLリファレンス”の“ASSIST指定”を参照してください。

LEADING_TABLE

ASSIST表要素に指定された表の順にジョインします。結合表中の表も指定できます。

LEADING_TABLEを指定しない場合は、ジョイン順を最適化処理が決定します。

参照

最適化処理のジョイン順の選択基準は、“チューニングガイド”の“ジョイン方法のチューニング”および“ジョイン順のチューニング”を参照してください。

注意

- ASSIST表要素に指定されていない表のジョイン順は、システムが自動的に決定します。
- LEADING_TABLEが以下の規則を満たしていない場合はASSIST指定が無効になりますが、規則を満たしているASSIST表要素に関しては、先頭から順に、指定されたジョイン順に従います。
  - 結合型がLEFTの場合、結合型の左側の表よりも後ろに右側の表を指定します。
  - 結合型がRIGHTの場合、結合型の右側の表よりも後ろに左側の表を指定します。
- 表の結合関係は、比較演算子“=”で表を結合する探索条件(例:表A.列A=表B.列B)で表現します。結合関係の探索条件は、ブール演算子“OR”でつなぎません。また、以下のいずれかの場所に指定します。
  - 結合表の結合条件
  - WHERE句

JOIN_RULE

ジョインする方法(フェッチジョインまたはマージジョイン)を指定します。

JOIN_RULEを指定しない場合は、ジョイン方法を最適化処理が決定します。

参照

最適化処理のジョイン方法の選択基準は、“チューニングガイド”の“ジョイン方法のチューニング”を参照してください。

注意

- 以下のいずれかの表をFROM句に2個以上指定した場合は、JOIN_RULEにFETCHは指定できません。
  - ジョインキーにインデックスが定義されていない、またはクラスタキーになっていない表。
  - ジョインキーに定義したインデックスDSOまたはインデックスが非活性状態である表。
  - 結合表、導出表または制約ASSISTを指定した表。
  - CAST指定をジョインキーに指定した表。
- 表の結合関係は、比較演算子“=”で表を結合する探索条件(例:表A.列A=表B.列B)で表現します。結合関係の探索条件は、ブール演算子“OR”でつなぎません。また、以下のいずれかの場所に指定します。
  - 結合表の結合条件
  - WHERE句

LEADING_TABLEとJOIN_RULEを同時に指定

LEADING_TABLEと、FETCHを指定したJOIN_RULEを同時に指定すると、ジョインキー以外に定義したインデックス、またはジョインキーになっていないクラスタキーを利用したフェッチジョインを行うことができます。これにより、以下の表をFROM句に2個以上指定することができます。

- ジョインキーにインデックスが定義されていない、またはクラスタキーになっていない表
- ジョインキーに定義したインデックスのDSO、またはインデックスが非活性状態である表
- 結合表、導出表、制約ASSISTを指定した表
- CAST指定をジョインキーに指定した表

また、USE_INDEXも同時に指定することで、ジョインで利用するインデックスを固定することができます。

なお、インデックスまたはクラスタキーが利用できない場合は表の全件検索を行います。

ジョインASSISTの使用例

ジョインASSISTの使用例を以下に示します。

例1

以下のようなSQL文があります。

```
SELECT 在庫表.倉庫番号,発注表.取引製品
    FROM 在庫管理.発注表,在庫管理.在庫表
    WHERE 発注表.取引製品 = 在庫表.製品番号
      AND 発注表.取引先 > :VAL1
```

JOIN_RULEを指定しない場合、最適化処理はマージジョインまたはフェッチジョインのいずれかを選択します。

このSQL文では、「発注表.取引先」に対する条件の絞り込みがあります。このようにデータを少量に絞り込めるような場合、フェッチジョインの方が効率的です。

そこで、フェッチジョインを行うようにJOIN_RULEを指定します。

```
SELECT /* ASSIST JOIN_RULE(FETCH) */
      在庫表.倉庫番号,発注表.取引製品
    FROM 在庫管理.発注表,在庫管理.在庫表
    WHERE 発注表.取引製品 = 在庫表.製品番号
      AND 発注表.取引先 > :VAL1
```

例2

以下のようなSQL文があります。

```
SELECT 発注表.仕入価格,発注表.取引製品
    FROM 在庫管理.在庫表,在庫管理.発注表
    WHERE 在庫表.製品番号 = 発注表.取引製品
      AND 在庫表.倉庫番号 BETWEEN :VAL1 AND :VAL2
      AND 発注表.取引先 BETWEEN :VAL3 AND :VAL4
```

このSQL文には以下の条件が指定されているため、各表のジョイン処理に先立ってインデックス検索を行うことで、ジョインする件数を少なくすることが可能です。

－ INDEX2を「在庫表.倉庫番号 BETWEEN :VAL1 AND :VAL2」の範囲で検索

－ INDEX3を「発注表.取引先 BETWEEN :VAL3 AND :VAL4」の範囲で検索

ジョイン順として、以下の2つのうち最適な方を選択します。

－ 発注表、在庫表の順

－ 在庫表、発注表の順

フェッチジョインの起点となる表は、なるべくレコード件数が絞り込まれている表である方が性能は良くなります。

取引先に対する条件の方が業務的に絞り込める場合、INDEX3を使って絞り込んだ後、フェッチジョインするというアクセスプランが良いことになります。

そこで、インデックスDSOのINDEX3を使用して検索するように、USE_INDEXを指定し、さらに発注表、在庫表の順にジョインするようにLEADING_TABLEを指定します。ジョイン方法にはフェッチジョインを行うようにJOIN_RULEを指定します。

```
SELECT /* ASSIST USE_INDEX(発注表(INDEX3))
          LEADING_TABLE(発注表,在庫表)
          JOIN_RULE(FETCH) */
        発注表.仕入価格, 発注表.取引製品
    FROM 在庫管理.在庫表,在庫管理.発注表
    WHERE 在庫表.製品番号 = 発注表.取引製品
      AND 在庫表.倉庫番号 BETWEEN :VAL1 AND :VAL2
      AND 発注表.取引先 BETWEEN :VAL3 AND :VAL4
```

例3

以下のようなSQL文があります。

```
SELECT 在庫表.製品番号,COUNT(*)
    FROM 在庫管理.在庫表,在庫管理.発注表
    WHERE 在庫表.製品番号 = 発注表.取引製品
    GROUP BY 在庫表.製品番号
```

データ分析の用途では、さまざまな方向から分析するため、多くの場合インデックスを付けていないか、または動作環境ファイルのパラメタIGNORE_INDEXをチューニングに指定してインデックスを使用しない運用にしています。

このSQL文では、IGNORE_INDEXを指定して全件をジョインという想定です。マージジョインは、ジョインするキーのデータ値に同じ値を含まない表から順にジョインする方が性能が良くなります。

そこで、このような特徴をふまえて、在庫表、発注表の順にジョインするようにLEADING_TABLEを指定します。

```
SELECT /* ASSIST LEADING_TABLE(在庫表,発注表) */
       在庫表.製品番号,COUNT(*)
    FROM 在庫管理.在庫表,在庫管理.発注表
    WHERE 在庫表.製品番号 = 発注表.取引製品
    GROUP BY 在庫表.製品番号
```

例4

発注表、会社表の順にフェッチジョインを行いたい場合、LEADING_TABLEとJOIN_RULEを同時に指定することで可能となります。

－ INDEX5を「発注表.取引製品 = :VAL1」の範囲で検索

ー 発注表、会社表の順にフェッチジョイン

```
SELECT /* ASSIST LEADING_TABLE(発注表,会社表)
          JOIN_RULE(FETCH) */
       発注表.取引製品,会社表.会社名
    FROM 在庫管理.発注表,在庫管理.会社表
    WHERE 発注表.取引先 = 会社表.会社番号
       AND 発注表.取引製品 = :VAL1
       AND 会社表.業種コード = :VAL2
       AND 会社表.地域コード = :VAL3
```

上記のSQL文の例では、会社表を検索する条件が複数あります。

ー INDEX6を「会社表.業種コード = :VAL2」の範囲で検索

ー INDEX7を「会社表.地域コード = :VAL3」の範囲で検索

どちらのインデックスを使用するかは最適化処理が決めます。使用するインデックスを細かく制御する場合は、USE_INDEXも同時に指定します。すると、以下のSQL文になります。

```
SELECT /* ASSIST LEADING_TABLE(発注表,会社表)
          JOIN_RULE(FETCH)
          USE_INDEX(会社表(INDEX6)) */
       発注表.取引製品,会社表.会社名
    FROM 在庫管理.発注表,在庫管理.会社表
    WHERE 発注表.取引先 = 会社表.会社番号
       AND 発注表.取引製品 = :VAL1
       AND 会社表.業種コード = :VAL2
       AND 会社表.地域コード = :VAL3
```

3.3.2.5 ASSIST指定の注意事項

ASSIST指定を利用する場合の注意事項を以下に示します。

ASSIST要素の優先順位

矛盾するASSIST要素を組み合わせた場合、優先順位の低い方のASSIST要素が無効になります。

ASSIST指定の優先順位は、以下のとおりです。なお、正しく指定された他のASSIST要素は有効となります。

1. USE_INDEX
2. LEADING_TABLE
3. JOIN_RULE
4. FIRST_ROWS

FIRST_ROWSについては、“3.5.1.1 実行制御ASSISTの使用方法”を参照してください。

矛盾するASSIST要素を組み合わせた例を以下に示します。

例

USE_INDEXを発注表と在庫表に指定しているので、JOIN_RULEに従ったフェッチジョインが不可能です。このような場合、優先順位に従って、JOIN_RULEが無効になります。優先順位が高いUSE_INDEXは、2つとも有効になります。

```
SELECT /* ASSIST USE_INDEX(発注表(INDEX3))
          USE_INDEX(在庫表(INDEX2))
          JOIN_RULE(FETCH) */
       発注表.仕入価格,発注表.取引製品
    FROM 在庫管理.在庫表,在庫管理.発注表
    WHERE 在庫表.製品番号 = 発注表.取引製品
```

```
    AND 在庫表.倉庫番号 = 2
    AND 発注表.取引製品
```

3.4 アクセスモデルの確認

アクセスプランを取得し、アクセスモデルを確認します。アクセスプランには、アクセスモデルを構成するセクションやエレメントの情報が出力されます。

出力されたセクション情報やエレメント情報から、意図した処理手順となっているかどうかを確認することができます。

アクセスプランの取得方法

アクセスプランの取得方法には、以下の2つがあります。

- rdbexecsqlコマンドを使用して取得する

  任意のSQL文のアクセスプランを取得します。

- アプリケーションから取得する

  アプリケーション単位にアクセスプランを取得します。

参照

アクセスプランの取得方法については、“SQLTOOLユーザーズガイド”を参照してください。

アクセスモデルの確認例

以下のアクセスプランを例に、アクセスモデルを確認する方法を説明します。

```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : SCAN        [TBL1_BASE_DSO     ][                 ][APPL              ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name        SCM1.TBL1
        scan type         TABLE ALL SCAN
        dso name          TBL1_BASE_DSO       [DSI/SH]
        condition evaluation  Yes
        scan record number    800
  [  2] OUTPUT ELEMENT
        record length         200
```

上記のアクセスプランでは、セクション情報、エレメント情報の以下の内容から、単一の表からデータを取り出す、制約系アクセスモデルであることがわかります。

- sectname(セクション名)が“SCAN”(表やインデックスなどからデータを取り出す)
- table name(アクセス対象の表名)の表が1つ

また、エレメント情報のscan type(アクセス方式)に、“TABLE ALL SCAN”と出力されていることから、制約系アクセスモデルのうちの、表の全件検索であることがわかります。

参照

アクセスプランの出力内容、出力形式についての詳細は、“SQLTOOLユーザーズガイド”を参照してください。

本節では、主なアクセスモデルについて、アクセスプランの出力例を示します。その他のアクセスモデルについては、“H.1 アクセスモデルの種類”を参照してください。

3.4.1 制約系アクセスモデル

以下のアクセスモデルについて、アクセスプランの出力例を示します。

- インデックス検索

インデックス検索

[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1==============input2============output/update======
   1 : SCAN        [TBL1_IX1_DSO      ][                ][APPL              ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        INDEX KEY SCAN(1)
        dso name         TBL1_IX1_DSO        [PAGE/SH]
        condition evaluation  No
        scan record number    10
  [  2] OUTPUT ELEMENT
        record length         8
```

3.4.2 ジョイン系アクセスモデル

以下のアクセスモデルについて、アクセスプランの出力例を示します。

- マージジョイン(インデックスを利用してジョインする)
- マージジョイン(ソートテーブルを利用してジョインする)
- フェッチジョイン
- ネスティッドループジョイン

マージジョイン(インデックスを利用してジョインする)

[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : M-JOIN      [TBL1_IX1_DSO       ][                 ][                 ]
                   [TBL2_IX1_DSO       ][                 ][APPL             ]
--------------------------------------------------------------------------------
  [  1] MERGE JOIN ELEMENT
        1) table name       SCM1.TBL1
           scan type        INDEX KEY SCAN(1)
           dso name         TBL1_IX1_DSO         [PAGE/SH]
           join key column       1
           condition evaluation  No
           scan record number    100
        2) table name       SCM1.TBL2
           scan type        INDEX KEY SCAN(1)
           dso name         TBL2_IX1_DSO         [PAGE/SH]
           join key column       1
           condition evaluation  No
           scan record number    200
  [  2] OUTPUT ELEMENT
        record length         22
```

マージジョイン(ソートテーブルを利用してジョインする)

[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1==============input2=============output/update======
   1 : SCAN        [TBL1_IX1_DSO      ][                  ][SORT0002          ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        INDEX KEY SCAN(1)
        dso name         TBL1_IX1_DSO        [PAGE/SH]
        condition evaluation  No
        scan record number    100
  [  2] INSERT ELEMENT
        table name       SORT0002
        insert record length  12
--------------------------------------------------------------------------------
   2 : SCAN        [SORT0002          ][TBL1_BASE_DSO     ][SORT0001          ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SORT0002
        scan type        TABLE ALL SCAN
        condition evaluation  No
  [  2] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        TABLE KEY SCAN
        dso name         TBL1_BASE_DSO       [PAGE/SH]
        condition evaluation  No
        scan record number    1
  [  3] INSERT ELEMENT
        table name       SORT0001
        insert record length  88
--------------------------------------------------------------------------------
   3 : M-JOIN      [SORT0001          ][                  ][                  ]
                   [TBL2_IX1_DSO      ][                  ][SORT0003          ]
--------------------------------------------------------------------------------
  [  1] MERGE JOIN ELEMENT
        1) table name       SORT0001
           scan type        TABLE ALL SCAN
           join key column       1
           condition evaluation  No
        2) table name       SCM1.TBL2
           scan type        INDEX ALL SCAN
           dso name         TBL2_IX1_DSO       [DSI/SH]
           join key column       1
           condition evaluation  No
           scan record number    1000
  [  2] INSERT ELEMENT
        table name       SORT0003
        insert record length  100
--------------------------------------------------------------------------------
   4 : SCAN        [SORT0003          ][TBL2_BASE_DSO     ][APPL              ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SORT0003
        scan type        TABLE ALL SCAN
        condition evaluation  No
  [  2] SCAN ELEMENT
        table name       SCM1.TBL2
        scan type        TABLE KEY SCAN
        dso name         TBL2_BASE_DSO       [PAGE/SH]
        condition evaluation  No
```

```
       scan record number    1
  [  3] OUTPUT ELEMENT
       record length         182
```

フェッチジョイン(インデックスを利用してジョインする)

[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1=============input2=============output/update======
   1 : SCAN        [TBL1_IX1_DSO      ][                  ][SORT0001          ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
       table name       SCM1.TBL1
       scan type        INDEX KEY SCAN(1)
       dso name         TBL1_IX1_DSO        [PAGE/SH]
       condition evaluation  No
       scan record number    5
  [  2] INSERT ELEMENT
       table name       SORT0001
       insert record length  12
--------------------------------------------------------------------------------
   2 : SCAN        [SORT0001          ][                  ][                  ]
       F-JOIN      [TBL1_BASE_DSO     ][                  ][                  ]
                   [TBL2_IX1_DSO      ][TBL2_BASE_DSO     ][APPL              ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
       table name       SORT0001
       scan type        TABLE ALL SCAN
       condition evaluation  No
  [  2] FETCH JOIN ELEMENT
       1) table name       SCM1.TBL1
          scan type        TABLE KEY SCAN
          dso name         TBL1_BASE_DSO       [PAGE/SH]
          condition evaluation  No
          scan record number    1
       2) table name       SCM1.TBL2
          scan type        INDEX TABLE SCAN(1)
          dso name         TBL2_IX1_DSO        [PAGE/SH]
                           TBL2_BASE_DSO       [PAGE/SH]
          condition evaluation  No
          scan record number    3
  [  3] OUTPUT ELEMENT
       record length         182
```

ネスティドループジョイン(実表を直接読み込んでジョインする)

[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1=============input2=============output/update======
   1 : N-JOIN      [TBL1_BASE_DSO     ][                  ][                  ]
                   [TBL2_BASE_DSO     ][                  ][APPL              ]
--------------------------------------------------------------------------------
  [  1] NESTED LOOP JOIN ELEMENT
       1) table name       SCM1.TBL1
          scan type        TABLE ALL SCAN
          dso name         TBL1_BASE_DSO        [DSI/SH]
          condition evaluation  Yes
          scan record number    1
       2) table name       SCM1.TBL2
```

```
          scan type         TABLE ALL SCAN
          dso name          TBL2_BASE_DSO        [DSI/SH]
          condition evaluation  Yes
          scan record number    1
[  2] OUTPUT ELEMENT
      record length         182
```

3.5 その他の性能チューニング方法

アクセスモデルを指定する以外にも、以下を行うことでアプリケーション性能を向上させることができます。

- ASSIST指定でSQL文の最初の1件の検索を高速化する
- SQL文の条件式には明示的に条件を記述する

3.5.1 ASSIST指定でSQL文の最初の1件の検索を高速化する

ASSIST指定の実行制御ASSISTを使用することで、SQL文の最初の1件の検索を高速化することができます。

3.5.1.1 実行制御ASSISTの使用方法

ここでは、実行制御ASSISTの使用方法について説明します。

実行制御ASSISTには、FIRST_ROWSがあります。

参照

FIRST_ROWSの一般規則については、“SQLリファレンス”の“ASSIST指定”を参照してください。

FIRST_ROWS

最初の1件の検索について最短の応答時間を目標にアクセスプランを選択します。

FIRST_ROWSを指定しない場合、探索条件を満たすすべての行を検索する場合の最短の応答時間を目標にしたアクセスプランを最適化処理が決定します。

FIRST_ROWSが有効になる条件

SQL文の形式が“FIRST_ROWSが有効となるSQL文の条件”をすべて満たし、排他の属性が“FIRST_ROWSが有効となる排他の属性の条件”のいずれかに該当する場合、FIRST_ROWSが有効になります。

FIRST_ROWSが有効となるSQL文の条件

a. 問合せ指定である。
b. DISTINCTを指定していない。
c. 集合関数を指定していない。
d. FROM句に実表を指定している。(注1)
e. 副問合せを指定していない。
f. GROUP BY句を指定していない。
g. HAVING句を指定していない。
h. ORDER BY句を指定した場合、ソート指定がすべて昇順または降順である。
i. ORDER BY句のソート指定に1つの表の列のみを指定し、各ソート指定がすべてASCまたはDESCである。
j. ORDER BY句のソート指定の列をインデックスキーの順にすべて含むインデックスが定義されている。
k. ORDER BY句のソート指定に指定した列を持つ表について、下記のいずれかを満たす。(注2)
   ・DSI分割していない。

・DSI定義で分割値の指定を1つだけ指定しており、ORDER BY句のソート指定の列の順と分割キーの列名リストの順が一致している。
・DSI定義で分割キーの列名リストで指定したすべての列に対して、WHERE句の探索条件に比較述語“=”の条件を指定している。

注1) FROM句に表を複数指定した場合はフェッチジョインが可能です。フェッチジョインについては、“3.3.2.4 ジョインASSISTの使用方法”を参照してください。

注2) データベース簡単運用の場合は、考慮する必要がありません。

FIRST_ROWSが有効となる排他の属性の条件

a. 独立性水準がSERIALIZABLEで、占有の単位がページ、またはDSIである。
b. 独立性水準がREAD COMMITTED、占有の単位が行で、コミットデータ即時読込み機能を利用している、またはデータベース簡単運用の場合(注3)。
c. 独立性水準がREAD UNCOMMITTEDである。
d. 占有モードがEXCLUSIVE LOCKで、占有の単位がページ、またはDSIである。
e. 占有モードがSHARE LOCKで、占有の単位がページ、またはDSIである。
f. 占有モードがFREE LOCK、占有の単位が行で、コミットデータ即時読込み機能を利用している、またはデータベース簡単運用の場合(注3)。
g. 占有モードがNO LOCKである。

注3) コミットデータ即時読込み機能が動作するSQL文は、以下のいずれかでなければなりません。

- 単一行SELECT文
- 更新可能性句に、FOR READ ONLYを指定したカーソル、または更新可能性句を省略したカーソルのOPEN文

参照

コミットデータ即時読込み機能については、“1.1.12 PRECEDENCE(1)を指定したSEQUENTIAL構造の特徴と注意事項”を参照してください。

また、データベース簡単運用での排他制御については、“I.7 排他制御”を参照してください。

注意

・ リザルトバッファを使用すると、1回目のFETCH文のレスポンス時間には、リザルトバッファを満杯にする時間も含まれます。このため、少量検索においては、リザルトバッファのサイズを大きくしないでください。

実行制御ASSISTの使用例

例1

FROM句に表を1個指定した場合のFIRST_ROWSの例を以下に示します。

最初の1件の検索について最短の応答時間を目標にアクセスプランを選択します。

```
SELECT /* ASSIST FIRST_ROWS */
       倉庫番号,製品番号
    FROM 在庫管理.在庫表
    WHERE 製品番号 BETWEEN 123 AND 200
    ORDER BY 倉庫番号
```

例2

FROM句に表を複数個指定した場合のFIRST_ROWSの例を以下に示します。

ORDER BY句のソート指定の列と一致するインデックスINDEX1を使用して検索しながら、在庫表、発注表の順でフェッチジョインを行います。

```
SELECT /* ASSIST FIRST_ROWS */
       在庫表.製品番号,発注表.取引先
    FROM 在庫管理.在庫表,在庫管理.発注表
    WHERE 在庫表.製品番号 = 発注表.取引製品
      AND 在庫表.倉庫番号 = 2
      AND 在庫表.製品番号 > 227
    ORDER BY在庫表.製品番号 DESC
```

ORDER BY句のソート指定の列と一致するインデックスが複数個ある場合、どちらのインデックスを使用するかは最適化処理が決定します。更新トランザクションとのインデックスキー更新の排他の関係などで、使用するインデックスを細かく制御する場合は、USE_INDEXも同時に指定します。また、FIRST_ROWSを指定するとフェッチジョインを行いますが、ジョイン順は最適化処理が決定します。ジョイン順を細かく制御する場合は、LEADING_TABLEも同時に指定します。このような場合、以下のSQL文になります。

```
SELECT /* ASSIST FIRST_ROWS
          USE_INDEX(在庫表(INDEX1))
          LEADING_TABLE(在庫表,発注表) */
       在庫表.製品番号,発注表.取引先
    FROM 在庫管理.在庫表,在庫管理.発注表
    WHERE 在庫表.製品番号 = 発注表.取引製品
      AND 在庫表.倉庫番号 = 2
      AND 在庫表.製品番号 > 227
    ORDER BY在庫表.製品番号 DESC
```

例3

ROWNUMで指定した件数分だけを在庫表から読み込むアクセスプランを選択します。

ORDER BY句のソート指定の列と一致するインデックスINDEX2を使用して検索しながら、「ROWNUM < 11」を判定し、10件読み込むと検索を中止します。

```
SELECT /* ASSIST FIRST_ROWS */
       倉庫番号,製品番号
    FROM 在庫管理.在庫表
    WHERE 製品番号 BETWEEN 123 AND 200
      AND ROWNUM < 11
    ORDER BY 倉庫番号
```

例4

導出表にFIRST_ROWSを指定し、ROWNUMで指定した件数分だけを在庫表から読み込むアクセスプランを選択します。

ORDER BY句のソート指定の列と一致するインデックスINDEX1をINDEX ALL SCANのアクセス方式で降順に検索しながら、「ROWNUM < 6」を判定し、在庫表から5件だけを読み込みます。

```
SELECT 製品番号,在庫数量
    FROM (SELECT /* ASSIST FIRST_ROWS */
                 製品番号,在庫数量
              FROM 在庫管理.在庫表
              WHERE 在庫数量 < 50
              ORDER BY 製品番号 DESC) AS D1(製品番号,在庫数量)
    WHERE ROWNUM < 6
```

例5

入れ子の導出表の最も内側にFIRST_ROWSを指定し、ROWNUMで指定した件数分だけを在庫表から読み込むアクセスプランを選択します。

ORDER BY句のソート指定の列と一致するインデックスINDEX1をINDEX ALL SCANのアクセス方式で降順に検索しながら、「ROWNUM < 11」を判定し、在庫表から10件だけを読み込みます。その後、「RN > 5」を判定し、6件目から10件目をアプリケーションに返します。

```
SELECT 製品番号,在庫数量
    FROM (SELECT ROWNUM,製品番号,在庫数量
              FROM SELECT /* ASSIST FIRST_ROWS */
                          製品番号,在庫数量
                     FROM 在庫管理.在庫表
                    WHERE 在庫数量 < 50
                    ORDER BY 製品番号 DESC) AS D1(製品番号,在庫数量)
             WHERE ROWNUM < 11) AS D2(RN,製品番号,在庫数量)
    WHERE RN > 5
```

ORDER BY句のソート指定の列と一致するインデックスが複数個ある場合、どちらのインデックスを使用するかは最適化処理が決定します。使用するインデックスを細かく制御する場合は、USE_INDEXも同時に指定します。

```
SELECT 製品番号,在庫数量
    FROM (SELECT ROWNUM,製品番号,在庫数量
              FROM SELECT /* ASSIST FIRST_ROWS
                             USE_INDEX(在庫表(INDEX1)) */
                          製品番号,在庫数量
                     FROM 在庫管理.在庫表
                    WHERE 在庫数量 < 50
                    ORDER BY 製品番号 DESC) AS D1(製品番号,在庫数量)
             WHERE ROWNUM < 11) AS D2(RN,製品番号,在庫数量)
    WHERE RN > 5
```

3.5.2 SQL文の条件式には明示的に条件を記述する

SQL文の中の条件の記述時には、明示的に条件を記述することが重要です。例えば、次の2つの条件を比較してください。

CNF形式 :C1=10 AND (C2=20 OR C2=30)

DNF形式 :(C1=10 AND C2=20)OR(C1=10 AND C2=30)

CNF形式(Conjunctive Normal Form)とは、いくつかの条件のかたまりを最終的にはANDで結合している形式です。DNF形式(Disconjunctive Normal Form)とは、いくつかの条件のかたまりを最終的にはORで結合している形式です。2つを見比べると、C1=10という条件が、全体に対して有効であることが、CNF形式の場合は明確ですが、DNFの場合はあまり明確ではありません。2つの条件、(C1=10 AND C2=20)、(C1=10 AND C2=30)の中を見比べて、C1=10という条件が両方に含まれていることに気がついてはじめて、C1=10という条件が全体に有効であることがわかります。

以下に、条件を記述する際の主な注意点を説明します。

ポイント	内容
CNF形式で記述する	AND演算子やOR演算子を複雑に絡めた条件を記述しないこと。記述は、単純なCNF形式で記述し、DNF形式は多用しないこと。 【例】 [良い例] CNF形式 :C1=10 AND (C2=20 OR C2=30) [悪い例] DNF形式 :(C1=10 AND C2=20)OR(C1=10 AND C2=30)
IN述語で記述する	ある項目がいずれかの値と一致するかの条件は、OR演算子で結合するのではなく、IN述語を使用すること。 【例】 [良い例] IN述語 :C1 IN(10, 20, 30, 40) [悪い例] OR演算子:C1=10 OR C1=20 OR C1=30 OR C1=40

ポイント	内容
ジョイン条件は列同士の=条件にする(演算式にしない)	ジョイン条件は項目同士を使用し、演算式や関数、CASTなどを使用しない。 【例】 [良い例] 項目同士 :T1.C1=T2.C1 [悪い例] 関数 :T1.C1=SUBSTRING(T2.C1 FROM 1 FOR 3) [悪い例] CAST :T1.C1=CAST(T2.C1 AS CHAR(10))
不要な条件を書かない	【例】 WHERE C1 BETWEEN :VAL1 AND :VAL2 AND C2 BETWEEN :VAL3 AND :VAL4 汎用的なアクセスルーチンなどを作成した時、ほとんどのケースはC1のみで絞り込む(C2の:VAL3、:VAL4には最小値と最大値を指定)、ごくまれにC2の値も指定するようなケースでは、C2の条件を指定しないSQLとC2の条件も指定したSQLに分ける
=で指定できるものはBETWEENで汎用的に作らない	事実上イコール条件しか指定しない(検索画面の値を指定する箱が1個など)が、将来を見越してBETWEENで指定するようなことをしない

第2部 SQLの使用方法

本部では、データ操作を行うアプリケーションの作成を目的として、データ操作のためのSQL文の使用方法について説明します。また、データ操作を行うために必要となるセション管理のためのSQL文の使用方法も説明します。

第4章 SQL文の共通規則.....90

第5章 データを検索する方法.....95

第6章 データを変更する方法.....106

第7章 カーソルを使用するデータ操作の方法.....120

第8章 複数の表を関連づけて操作する方法.....138

第9章 いろいろなデータ操作の方法.....148

第10章 トリガを使用したデータ操作の方法.....208

第11章 ルーチンを使用したデータ操作の方法.....213

第12章 XMLデータを操作する方法.....219

第4章 SQL文の共通規則

本章では、SQL文の全般的な共通規則について説明します。

4.1 SQL文に指定する名前

SQL文の中に指定する名前には、以下のものがあります。

a. スキーマ名
b. 表名
c. 列名
d. ルーチン名
e. パラメタ名
f. トリガ名
g. SQL変数名
h. 記述子名
i. コネクション名
j. SQLサーバ名
k. カーソル名
l. 相関名
m. SQL文識別子
n. 順序名

a)からc)は、データベースで定義する名前です。これらの名前は、スキーマや表の設計時に決定します。d)はプロシジャルーチン定義またはファンクションルーチン定義で定義する名前です。e)はプロシジャルーチン定義で定義する名前です。これらをSQL文の中に指定する場合には、そのとおりのつづりで指定します。また、f)はトリガ定義で定義する名前です。

参照

データベース定義、プロシジャルーチン定義およびトリガ定義については“SQLリファレンス”を参照してください。

g)は、プロシジャルーチン定義の複合文で定義する名前です。

h)からm)は、SQL文の中で付ける名前です。h)、i)およびj)のデータ型は、文字列型であることが必要です。また、k)、l)およびm)の名前を付けるときの規則を以下に示します。

S

Solarisの場合

- 英字で指定する場合
  - 先頭は、英字とします。
  - 2文字目以降は、英字、数字、または下線“_”とします。
  - 英小文字を使用した場合、対応する英大文字に変換されます。
  - 名前の長さは36文字以内とします。
- 各国語文字で指定する場合
  - EUCコード(Extended Unix Code)のコードセット1(2バイト)、シフトJISコードおよびUNICODEのUTF-8コードが使用できます。

- 名前の長さは18文字以内とします。

Linuxの場合

- 英字で指定する場合
  - 先頭は、英字とします。
  - 2文字目以降は、英字、数字、または下線“_”とします。
  - 英小文字を使用した場合、対応する英大文字に変換されます。
  - 名前の長さは36文字以内とします。
- 各国語文字で指定する場合
  - EUCコード(Extended Unix Code)のコードセット1(2バイト)およびUNICODEのUTF-8コードが使用できます。
  - 名前の長さは18文字以内とします。

Windows(R)の場合

- 英字で指定する場合
  - 先頭は、英字とします。
  - 2文字目以降は、英字、数字、または下線“_”とします。
  - 英小文字を使用した場合、対応する英大文字に変換されます。
  - 名前の長さは36文字以内とします。
- 各国語文字で指定する場合
  - シフトJISコードおよびUNICODEのUTF-8コードが使用できます。
  - 名前の長さは18文字以内とします。

なお、英字と各国語文字を混在して使用することはできません。

n)は順序定義で定義する名前です。

4.2 データベースの例

ここでは、データベースの例として、ある小売店の在庫管理データベースを示します。本書では、データ操作の説明に、このデータベースを使用します。このデータベースは、小売店の在庫管理業務を実現するもので、以下の3つの表から構成されています。

在庫表:

取り扱う製品とその在庫数量の情報を持つ表

発注表:

仕入れ先ごとの取引製品とその発注数量、および仕入価格に関する情報を持つ表

会社表:

取引会社の会社名、電話番号および住所の情報を持つ表

これらの表は“在庫管理”というスキーマに含まれます。

在庫表

在庫表の内容を“図4.1 在庫管理データベースの内容”のa)に示します。在庫表は以下の4つの列から構成されています。

製品番号:

製品につけたコード番号データの列

製品名:

製品の種類名データの列

在庫数量:

製品の在庫数データの列

倉庫番号:

製品を保管してある倉庫の番号データの列

発注表

発注表の内容を“図4.1 在庫管理データベースの内容”のb)に示します。発注表は以下の4つの列から構成されています。

取引先:

取引先の会社番号データの列

取引製品:

製品につけたコード番号データの列

仕入価格:

製品の仕入価格データの列

発注数量:

製品の発注数データの列

会社表

会社表の内容を“図4.1 在庫管理データベースの内容”のc)に示します。会社表は以下の4つの列から構成されています。

会社番号:

会社につけたコード番号データの列

会社名:

会社の名称データの列

電話番号:

会社の電話番号データの列

住 所:

会社の住所データの列

本書の中で、特に断りなくSQL文の使用例などで上記の表名および列名が使用されている場合は、“図4.1 在庫管理データベースの内容”の表および列を指定しています。必要に応じて“図4.1 在庫管理データベースの内容”を参照してください。なお、“図4.1 在庫管理データベースの内容”に示す表に含まれるデータは架空のものです。

図4.1 在庫管理データベースの内容

a） 在庫表

製品番号	製品名	在庫数量	倉庫番号
110	テレビ	85	2
111	テレビ	90	2
123	冷蔵庫	60	1
124	冷蔵庫	75	1
140	CDプレーヤー	120	2
212	テレビ	0	2
215	ビデオ	5	2
226	冷蔵庫	8	1
227	冷蔵庫	15	1
240	CDプレーヤー	25	2
243	CDプレーヤー	14	2
351	CD	2500	2

b） 発注表

取引先	取引製品	仕入価格	発注数量
61	123	48000	60
61	124	64000	40
61	140	8000	80
61	215	240000	10
61	240	80000	20
62	110	37500	120
62	226	112500	20
62	351	375	800
63	111	57400	80
63	212	205000	30
63	215	246000	10
71	140	7800	50
71	351	390	600
72	140	7000	70
72	215	210000	10
72	226	105000	20
72	243	84000	10
72	351	350	1000
74	110	39000	120
74	111	54000	120
74	226	117000	20
74	227	140400	10
74	351	390	700

c） 会社表

会社番号	会社名	電 話 番 号	住 所
61	アダム電気	111-777-4444	東京都 小田区 新蒲田 7-8-9
62	アイデア商事	222-888-5555	神奈川県 小浜市 旭区 1-2-3
63	大月産業	333-999-6666	埼玉県 浦和町 大崎 1-1-1
71	川川電気	444-111-7777	大阪府 堺田市 赤坂台 4-5-6
72	龍巻産業	555-222-8888	大阪府 灘町 東淀川 2-3-7
74	第第商事	666-333-9999	京都府 中市 4-16-16

在庫管理データベースの各表の列の属性

各表の列の属性を以下に示します。

表4.1 在庫管理データベースの各表の列属性

表名	列名	列のデータ型	列の制約	備考
在庫表	製品番号	SMALLINT	NOT NULL	製品のコード番号
	製品名	NATIONAL CHARACTER(10)	NOT NULL	製品種類名
	在庫数量	INTEGER	---	在庫する製品の個数
	倉庫番号	SMALLINT	---	保管倉庫の番号
発注表	取引先	SMALLINT	NOT NULL	取引先の会社番号
	取引製品	SMALLINT	NOT NULL	製品のコード番号
	仕入価格	INTEGER	---	製品の仕入価格
	発注数量	SMALLINT	---	発注した製品の個数
会社表	会社番号	SMALLINT	NOT NULL	会社のコード番号

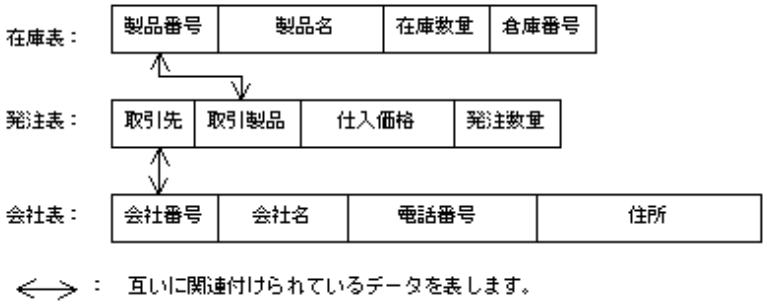
表名	列名	列のデータ型	列の制約	備考
	会社名	NATIONAL CHARACTER(10)	NOT NULL	会社の名称
	電話番号	CHARACTER(14)	---	会社の電話番号
	住所	NATIONAL CHARACTER(20)	---	会社の所在地の住所

--- : NULLを許します。

在庫表、発注表および会社表の関連

3つの表の間の関連を“図4.2 在庫表、発注表および会社表の関連”に示します。在庫表と発注表は、製品番号と取引製品で関連が付けられています。発注表と会社表は、取引先と会社番号で関連が付けられています。例えば、在庫表上で製品番号が123の製品は、製品名が“冷蔵庫”、在庫数量が“60”で、この製品が格納してある倉庫の番号が“1”です。そして、この製品の仕入価格および発注数量は、発注表の取引製品が“123”の行から、それぞれ“48000”、“60”であることが分かります。さらに、取引先の会社番号は“61”なので、会社表の会社番号が“61”の行から、その製品の取引先の会社の会社名、電話番号および住所が分かります。

図4.2 在庫表、発注表および会社表の関連



第5章 データを検索する方法

本章では、単一行SELECT文で行うことのできる各操作を説明します。

5.1 列の値をそのまま取り出す

表から1行のデータを取り出すには、単一行SELECT文を使用します。単一行SELECT文の例を以下に示します。

例

在庫表から、製品番号が“215”の行の在庫数量を求めます。

```
SELECT  在庫数量   INTO  :STOCKQTY
          (1)      (A)      (2)
        FROM  在庫管理.在庫表  WHERE  製品番号 = 215
         (B)        (3)         (C)        (4)
```

(1) 選択リスト

(A)～(2) INTO句

(2) 選択相手リスト

(B)～(3) FROM句

(3) 表名

(C)～(4) WHERE句

(4) 探索条件

この例の単一行SELECT文では、選択相手リストにホスト変数“STOCKQTY”を指定しています。ホスト変数は、アプリケーションとデータベースの間でデータの受渡しをするのに使用する変数です。すなわち、アプリケーションで参照するデータをデータベースから取り出すため、または逆に、アプリケーションで設定したデータをデータベースに格納するために、SQL文に指定する変数です。ホスト変数の宣言は埋込みSQL宣言節で行います。ホスト変数をSQL文中に指定する場合には、変数の前にコロン“:”を付けて指定します。ただし、SQL文以外のC言語およびCOBOLのステートメント中では、一般のC言語およびCOBOLの変数と同じで、コロン“:”は付けません。また、C言語ではホスト変数名に各国語文字を使用することはできません。

本節では、単一行SELECT文で行うことのできる以下の操作を説明します。

- 複数の列の値を取り出す
- NULL値を含むデータを取り出す
- 複数の表からデータを取り出す
- 冗長な行を取り除いて取り出す

5.1.1 複数の列の値を取り出す

“5.1 列の値をそのまま取り出す”で最初に示した例では、データを取り出す列は“在庫数量”だけでした。データを取り出したい列が、“在庫数量”と“倉庫番号”の2つになったとします。それには単一行SELECT文を2回実行すればよいことはすぐに分かります。しかし、これではデータを取り出したい列の数が多くなると効率が悪くなります。単一行SELECT文では、一度に複数の列の値を取り出すことができます。

複数の列のデータを取り出すには、選択リストに必要な数の列名をコンマ(,)で区切って指定します。取り出したデータを格納するホスト変数も、同じ数だけ指定します。取り出した列のデータは、左から数えて同じ順番のホスト変数に格納されます。

単一行SELECT文に複数の列を指定した例を以下に示します。

例1

在庫表から、製品番号が“215”の製品の在庫数量と倉庫番号を求めます。求めた在庫数量はホスト変数“STOCKQTY”に、倉庫番号はホスト変数“WHCODE”に格納します。

----->： データを取り出す列と、取り出したデータを格納するホスト変数の対応を表します。

図5.1 単一行SELECT文に複数の列を指定した例

在庫表

製品番号	製品名	在庫数量	倉庫番号
110	テレビ	85	2
111	テレビ	90	2
123	冷蔵庫	60	1
124	冷蔵庫	75	1
140	CDプレーヤー	120	2
212	テレビ	0	2
215	ビデオ	5	2
226	冷蔵庫	8	1
227	冷蔵庫	15	1
240	CDプレーヤー	25	2
243	CDプレーヤー	14	2
351	CD	2500	2

[ ] ： 取り出すデータを表します。

← この行のデータを取り出し、ホスト変数"STOCKQTY"および"WHCODE"に、該当する列のデータを格納します。
すなわち、以下の結果を得ます。

STOCKQTY = 5
WHCODE = 2

表に含まれるすべての列を定義された順に指定する場合は、列名の代わりにアスタリスク(*)を指定することができます。アスタリスクを指定した例を以下に示します。

例2

"*"の指定の例です。在庫表からすべての列のデータを取り出します。

```
SELECT * INTO :ITMNO, :PRODUCT, :STOCKQTY, :WHCODE

        FROM 在庫管理.在庫表 WHERE 製品番号 = 215
```

例2は、データを取り出す列に、製品番号、製品名、在庫数量、および倉庫番号の4つの列を指定したのと同じです。したがって、取り出した列のデータを格納するホスト変数は4つ指定することが必要です。

なお、列名の代わりにアスタリスクを指定すると、表の設計変更により列が追加されたり、列の定義順序が変更されたときに、アプリケーションの修正が必要になります。アプリケーションにこのような影響を与えないためには、すべての列をコンマ(,)で区切って指定するようにしてください。

5.1.2 NULL値を含むデータを取り出す

これまでに説明した例では、取り出したデータがNULL値である場合のことは考えていませんでした。ここでは、NULL値を含むデータを取り出す方法について説明します。

SQL文中でホスト変数と対にして指定し、データの内容がNULL値かどうかを表すために使用する変数を、"標識変数"と呼びます。アプリケーションで参照するデータをデータベースから取り出す場合、標識変数には、取り出したデータがNULL値かどうかを表す情報が設定されます。逆に、アプリケーションで設定したデータをデータベースに格納する場合には、標識変数には、格納するデータがNULL値かどうかを表す情報を設定しておきます。標識変数は埋込みSQL宣言節で宣言します。宣言の記述方法はホスト変数と同じです。標識変数をSQL文中に指定する場合には、変数の前にコロン":"を付けて指定します。ただし、SQL文以外のC言語およびCOBOLのステートメント中では、一般のC言語およびCOBOLの変数と同じで、コロン":"は付けません。また、C言語では標識変数名に各国語文字を使用することはできません。なお、標識変数のデータ型はSMALLINTに対応するデータ型であることが必要です。

参照

各ホスト言語でのデータ型とSQLのデータ型との対応については、“SQLリファレンス”を参照してください。

標識変数を指定した例を以下に示します。

例

在庫表から、製品番号が“215”の製品の在庫数量と倉庫番号を求めます。

```
SELECT 在庫数量, 倉庫番号 INTO :STOCKQTY INDICATOR :STOCKQTY_INDICATOR,
                              :WHCODE INDICATOR :WHCODE_INDICATOR
    FROM 在庫管理.在庫表 WHERE 製品番号 = 215
```

この例の単一行SELECT文では、“STOCKQTY_INDICATOR”および“WHCODE_INDICATOR”が標識変数です。それぞれがホスト変数“STOCKQTY”および“WHCODE”に対応します。標識変数は必要な変数にだけ指定することができます。例えば、倉庫番号にだけ標識変数を指定して、在庫数量には指定しなくてもかまいません。

取り出す列の値によって、標識変数に格納される値を以下に示します。なお、取り出す列の値の長さをL、受け取るホスト変数の長さをMとします。

表5.1 列の値と標識変数に格納される値

取り出す列のデータ型	標識変数の値		
	取り出す列の値がNULL値の場合	取り出す列の値がNULL値以外の場合	
		M＜Lの場合	M≧Lの場合
NUMERIC DECIMAL INTEGER SMALLINT FLOAT REAL DOUBLE PRECISION DATE TIME TIMESTAMP INTERVAL BLOB	-1	0	0
CHARACTER(n) NATIONAL CHARACTER(n)	-1	n	0
CHARACTER VARYING NATIONAL CHARACTER VARYING	-1	データの有効な長さ	0

備考. 列の値がNULL値の場合、データを格納するホスト変数の内容は不定になります。

標識変数を指定しないで、取り出すデータがNULL値になると、単一行SELECT文はエラーになります。列の値にNULL値が含まれるかどうかが分からない場合には、必ず標識変数を指定してください。

5.1.3 複数の表からデータを取り出す

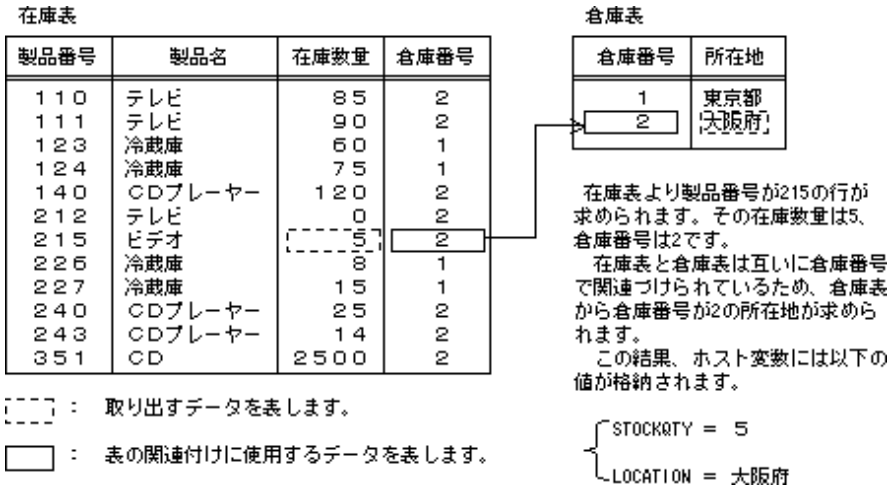
単一行SELECT文では、2つ以上の表を関連づけてデータを取り出すことができます。2つの表を関連づけた例を以下に示します。ここでは、表を関連づけた結果、どのようなデータが取り出せるかについて説明します。表を関連づけて操作するときの規則については、“第9章 いろいろなデータ操作の方法”で説明します。

例

在庫表および倉庫表から、製品番号が“215”の在庫数量と倉庫の所在地を求めます。なお、倉庫の所在地のデータは、倉庫表にあるものとします。また、倉庫表が含まれるスキーマは施設管理とします。

```
SELECT 在庫数量, 所在地 INTO :STOCKQTY, :LOCATION

        FROM 在庫管理.在庫表, 施設管理.倉庫表

        WHERE 製品番号 = 215 AND 在庫表.倉庫番号 = 倉庫表.倉庫番号
```

図5.2 2つの表を関連づけてデータを取り出す例



5.1.4 冗長な行を取り除いて取り出す

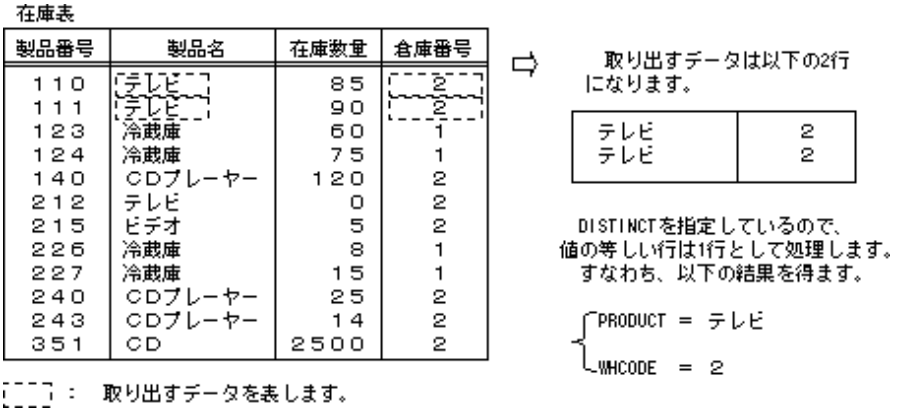
単一行SELECT文で取り出せるのは一行だけです。単一行SELECT文で探索条件の結果が真になる行が複数存在するとエラーになります。また、探索条件の指定を省略すると、表のすべての行が取り出しの対象になるため、表に2行以上のデータがあるとエラーになります。このように、取り出しの対象になる行が複数ある場合でも、取り出す列のすべてについて、それぞれの行の値が同じ場合には、結果を1行にして取り出すことができます。それには、“DISTINCT”を指定します。指定例を以下に示します。

例1

在庫表から、製品番号が“120”未満の製品名と倉庫番号を求めます。これらの値が同じ行は1行として処理します。

```
SELECT DISTINCT 製品名, 倉庫番号 INTO :PRODUCT, :WHCODE

       FROM 在庫管理.在庫表 WHERE 製品番号 < 120
```

図5.3 冗長な行を取り除いて取り出す例



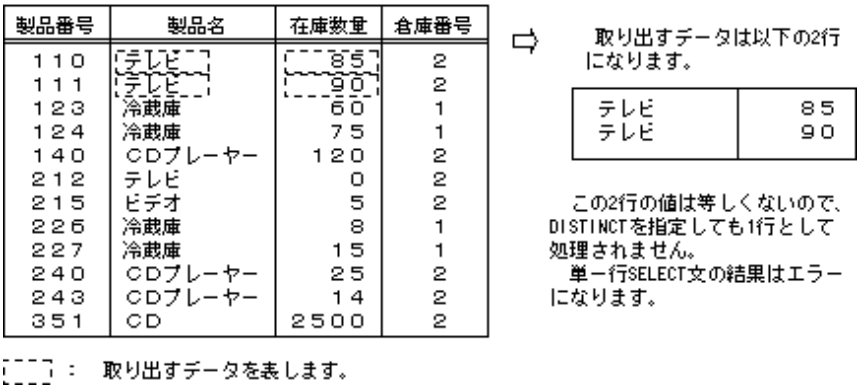
DISTINCTは、取り出す行の値が互いに等しいものを1行にするために使用します。DISTINCTの使用方法で誤りやすいのは、取り出す行が複数あってもエラーにならないと誤解した場合です。この例を以下に示します。

例2

DISTINCTを誤って使用した例です。在庫表から、製品番号が“120”未満の製品名と在庫数量を求めます。

```
SELECT DISTINCT 製品名, 在庫数量 INTO :PRODUCT, :STOCKQTY

        FROM 在庫管理.在庫表 WHERE 製品番号 < 120
```

図5.4 DISTINCTの誤った使用方法



5.2 特定の列以外を取り出す

本節では、単一行SELECT文で行うことのできる操作を説明します。

5.2.1 データ間で四則演算をする

これまでは、表から列の値をそのまま取り出す方法を説明しました。単一行SELECT文では、列の値に対して四則演算を実施した結果を取り出すことができます。それには、選択リストに演算式を指定します。指定例を以下に示します。

例

在庫表から、製品番号が“215”の製品の在庫数量に10を加算した結果を取り出します。

```
SELECT 在庫数量 + 10 INTO :STOCKQTY

        FROM 在庫管理.在庫表 WHERE 製品番号 = 215
```

“+”は加算を行う演算子です。ほかに、減算を行う“-”、乗算を行う“*”、および除算を行う“/”が演算子として使用できます。また、“10”は在庫数量に加算する値を定数で指定しています。列と定数のほかに、ホスト変数や、列どうしでの演算も可能です。

上記の例の単一行SELECT文を実行すると、ホスト変数“STOCKQTY”には“15”が格納されます。

演算式の指定方法

演算式で使用する演算子には、単項演算子と二項演算子があります。

単項演算子

単項演算子には以下の2つがあります。

+:無作用

-:符号反転

二項演算子

二項演算子には以下の4つがあります。

+:加算

-:減算

*:乗算

/:除算

以下に演算式の指定方法と意味を示します。

表5.2 演算式の指定方法と意味

種類	演算子	指定方法	意味
単項演算	+	+演算項	演算項に影響を与えません。
	-	-演算項	演算項の符号を反転します。
二項演算	+	演算項+演算項	左項と右項を加算します。
	-	演算項-演算項	左項から右項を減算します。
	*	演算項*演算項	左項と右項を乗算します。
	/	演算項/演算項	左項を右項で除算します。

演算項には、列名、ホスト変数、または定数を指定します。あるいは、演算式の結果を演算項として指定することもできます。ただし、単項演算の演算項に演算式を指定する場合には、演算式を括弧で囲って指定することが必要です。

演算の優先順序

演算の優先順序は以下のとおりです。

1. 単項演算
2. 乗算および除算
3. 加算および減算

また、括弧(“(”および“)”)を使用して優先順序を指定することができます。括弧の中の式は、ほかに先立って実行されます。

演算式の結果のデータ型

演算項と結果のデータ型の関係を説明します。

単項演算子の結果のデータ型

単項演算子の結果のデータ型は、演算項のデータ型になります。

二項演算子の結果のデータ型

二項演算子の結果のデータ型を以下に示します。なお、1つの演算式に複数の演算子を指定した場合も、結果のデータ型は演算子ごとに決まります。演算式中のすべての演算子について、演算の優先順序に従い以下の規則を適用することにより、演算式の最終的なデータ型を求めることができます。

図5.5 二項演算子の結果のデータ型

<table>
<tr><th>左項＼右項</th><th>SMALLINT</th><th>INTEGER</th><th>DECIMAL
(p2,q2)</th><th>NUMERIC
(p2,q2)</th><th>REAL</th><th>DOUBLE
PRECISION</th><th>DATE
TIME
TIMESTAMP</th><th>INTERVAL</th></tr>
<tr><td>SMALLINT</td><td colspan="2" rowspan="2">INTEGER</td><td colspan="2">(1)</td><td rowspan="3">(5)</td><td rowspan="3">(7)</td><td rowspan="5">エ
ラ
ー</td><td rowspan="5">INTERVAL
(注5)</td></tr>
<tr><td>INTEGER</td><td colspan="2">(2)</td></tr>
<tr><td>DECIMAL
(p1,q1)</td><td>(3)</td><td>(4)</td><td colspan="2">(注1)
加算および減算
DECIMAL(MIN(18,MAX(p1-q1,p2-q2)+
MAX(q1,q2)+1),MAX(q1,q2))
乗算 (注2)
DECIMAL(MIN(18,p1+p2),q1+q2)
除算
DECIMAL(MIN(18,p1-q1+q2+MAX(5,q1)),MAX(5,q1))</td></tr>
<tr><td>NUMERIC
(p1,q1)</td><td colspan="4"></td></tr>
<tr><td>REAL</td><td>(6)</td><td colspan="3">REAL (注3)</td><td></td></tr>
<tr><td>DOUBLE
PRECISION</td><td>(8)</td><td colspan="4">DOUBLE PRECISION (注4)</td><td></td></tr>
<tr><td>DATE
TIME
TIMESTAMP</td><td colspan="6">エ ラ ー</td><td>INTERVAL</td><td>DATE
TIME
TIMESTAMP</td></tr>
<tr><td>INTERVAL</td><td colspan="6">INTERVAL (注5)</td><td>DATE
TIME
TIMESTAMP</td><td>INTERVAL</td></tr>
</table>

注1) 左項がSMALLINTの場合、p1=5、q1=0として扱います。(1)

左項がINTEGERの場合、p1=10、q1=0として扱います。(2)

右項がSMALLINTの場合、p2=5、q2=0として扱います。(3)

右項がINTEGERの場合、p2=10、q2=0として扱います。(4)

注2) q1+q2>18となるとエラーになります。

注3) 左項がSMALLINT、INTEGER、DECIMAL、NUMERICの場合、REALとして扱います。(5)

右項がSMALLINT、INTEGER、DECIMAL、NUMERICの場合、REALとして扱います。(6)

注4) 左項がSMALLINT、INTEGER、DECIMAL、NUMERICの場合、DOUBLE PRECISIONとして扱います。(7)

右項がSMALLINT、INTEGER、DECIMAL、NUMERICの場合、DOUBLE PRECISIONとして扱います。(8)

注5) 乗算以外の二項演算はエラーになります。

注6) 乗算、除算以外の二項演算はエラーになります。

5.2.2 列の値の合計値、平均値、最大値、最小値、および行数を求める

これまでは、表から特定の1行について、必要な列のデータを取り出す方法について説明しました。単一行SELECT文では、複数の行のデータを集計した結果を取り出すことができます。集計により、データの合計、平均、最大、最小、および行数そのものを求めることができます。それには、選択リストに集合関数を指定します。指定例を以下に示します。

例1

発注表から発注数量の合計を取り出します。

```
SELECT SUM(発注数量) INTO :TOTALQTY  FROM 在庫管理.発注表
```

“SUM”は列の値の合計値を求めるための関数です。ほかに、平均値を求める“AVG”、最大値を求める“MAX”、最小値を求める“MIN”、および行数を求める“COUNT”が使用できます。これらの関数を“集合関数”と呼びます。

例1の単一行SELECT文では、取り出す行を特定するためのWHERE句が指定されていないことに気付かれたと思います。単一行SELECT文で取り出せるのは、1行分のデータだけですから、これまでの例では必ず取り出す行を特定するための条件を指定していました。一方この例では、表中のすべての行に対する集計結果を取り出すわけですから、行を特定するための条件は必要ありません。なお、ある条件を満たす行だけを集計の対象としたい場合には、WHERE句を指定します。この場合、WHERE句の探索条件が真になる行だけが、集計の対象となります。

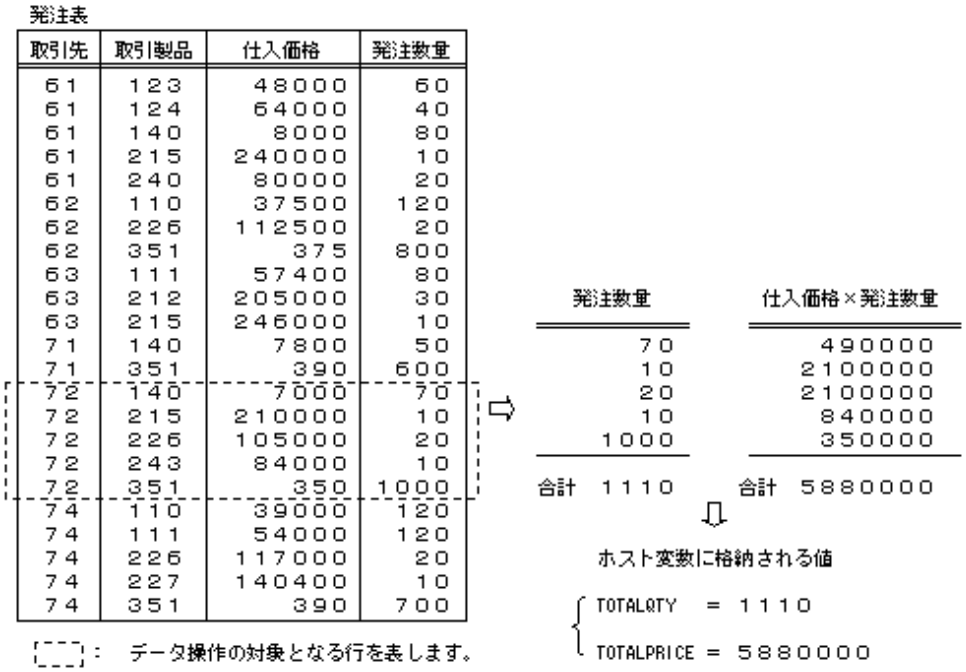
また、例1では、集合関数は列の値に適用していますが、演算式の結果に適用することもできます。この例を以下に示します。

例2

発注表から、取引先の番号が“72”の取引先について、発注数量の合計と発注価格の合計を求めます。なお、発注価格は、仕入価格×発注数量で求められます。

```
SELECT SUM(発注数量), SUM(仕入価格 * 発注数量)

       INTO :TOTALQTY, :TOTALPRICE

       FROM 在庫管理.発注表 WHERE 取引先 = 72
```

図5.6 演算式の結果に集合関数を指定した例



集合関数の種類と指定方法

集合関数には以下の種類があります。

COUNT(*)関数:

表の行数を、NULL値を含む行も含めて求めます。

AVG関数:

列の値の平均値を求めます。

MAX関数:

列の値の最大値を求めます。

MIN関数:

列の値の最小値を求めます。

SUM関数:

列の値の合計値を求めます。

COUNT関数:

列がNULL値以外の値をもつ行の数を求めます。

集合関数は、COUNT(*)関数を除き以下の2通りの指定ができます。ただし、MAX関数およびMIN関数では、どちらを指定しても結果は同じです。

ALL:

指定した列の値からNULL値を除いた行に、集合関数を適用します。

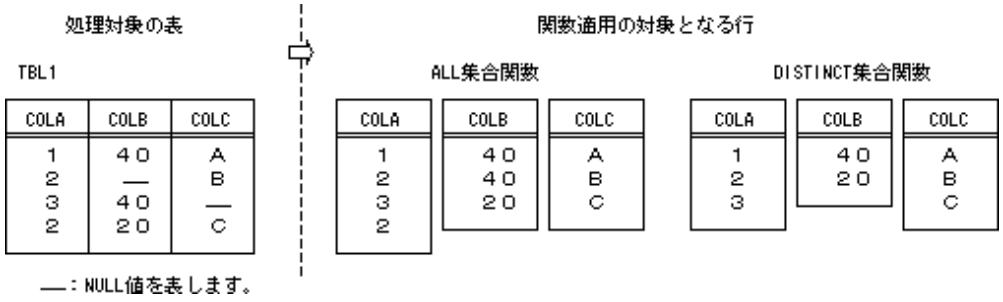
DISTINCT:

指定した列の値がNULL値の行および値の重複する行を除いて、集合関数を適用します。

ALL指定をした関数を“ALL集合関数”、DISTINCT指定をした関数を“DISTINCT集合関数”と呼びます。ALLおよびDISTINCTを省略すると、ALL集合関数になります。

以下に、ALL集合関数およびDISTINCT集合関数で適用となる行を示します。ここでは、表TBL1を使用して説明します。

図5.7 ALL集合関数およびDISTINCT集合関数の処理対象行



集合関数の指定方法を以下に示します。

表5.3 集合関数の指定方法

種 類	関数名	指定方法（注1）	意 味
COUNT(*)		COUNT(*)（注2）	表の行数
ALL集合関数	AVG	AVG(ALL 値式)	値式の平均（重複行あり）
	MAX	MAX(ALL 値式)	値式の最大（重複行あり）
	MIN	MIN(ALL 値式)	値式の最小（重複行あり）
	SUM	SUM(ALL 値式)	値式の合計（重複行あり）
	COUNT	COUNT(ALL 値式)	値式の行数（重複行あり）
DISTINCT集合関数	AVG	AVG(DISTINCT 値式)	値式の平均（重複行なし）
	MAX	MAX(DISTINCT 値式)	値式の最大（重複行なし）
	MIN	MIN(DISTINCT 値式)	値式の最小（重複行なし）

種 類	関数名	指定方法（注1）	意 味
	SUM	SUM(DISTINCT 値式)	値式の合計（重複行なし）
	COUNT	COUNT(DISTINCT 値式)	値式の行数（重複行なし）

注1) ALL集合関数を指定する場合、キーワード“ALL”は省略できます。

注2) COUNT(*)では列は指定しません。

“表5.3 集合関数の指定方法”のように、ALL集合関数もDISTINCT集合関数も、引数に値式を指定します。引数には、列名を使用した演算式も指定できます。

集合関数の結果のデータ型

集合関数の対象とする列のデータ型と、結果のデータ型の関係を以下に示します。

表5.4 集合関数の対象とする列のデータ型と関数の結果のデータ型

集合関数	対象とする列のデータ型	結果のデータ型
COUNT	---	INTEGER
SUM	INTEGER	INTEGER（注）
	SMALLINT	INTEGER（注）
	DECIMAL(p,q)	DECIMAL(m,n) m: MIN(18,p+7) n: q
	NUMERIC(p,q)	
	REAL	REAL
	DOUBLE PRECISION	DOUBLE PRECISION
	INTERVAL	INTERVAL
AVG	INTEGER	DECIMAL(12,2)（注）
	SMALLINT	DECIMAL(7,2)（注）
	DECIMAL(p,q)	DECIMAL(m,n) m: MIN(18,p+2) n: MIN(18-(p-q),q+2)
	NUMERIC(p,q)	
	REAL	REAL
	DOUBLE PRECISION	DOUBLE PRECISION
	INTERVAL	INTERVAL
MAX/MIN	INTEGER	INTEGER
	SMALLINT	SMALLINT
	DECIMAL(p,q)	DECIMAL(p,q)
	NUMERIC(p,q)	NUMERIC(p,q)
	REAL	REAL
	DOUBLE PRECISION	DOUBLE PRECISION
	CHARACTER	CHARACTER
	CHARACTER VARYING	CHARACTER VARYING
	NATIONAL CHARACTER	NATIONAL CHARACTER

集合関数	対象とする列のデータ型	結果のデータ型
	NATIONAL CHARACTER VARYING	NATIONAL CHARACTER VARYING
	DATE	DATE
	TIME	TIME
	TIMESTAMP	TIMESTAMP
	INTERVAL	INTERVAL

p: 精度 q: 位取り

注) 中間精度はINTEGERとなります。

第6章 データを変更する方法

本章では、INSERT文、UPDATE文:探索、DELETE文:探索を使用してデータの変更を行う方法を説明します。

6.1 データベースにデータを追加する

表にデータを追加する場合には、INSERT文を使用します。INSERT文の例を以下に示します。

例

発注表に取引先が“78”、取引製品が“400”、仕入価格が“125000”、そして発注数量が“50”のデータを追加します。

```
INSERT INTO 在庫管理.発注表 (取引先, 取引製品, 仕入価格, 発注数量)
                (1)                          (2)

      VALUES (78, 400, 125000, 50)
                      (3)
```

(1) 表名

(2) 挿入列リスト

(3) 挿入値リスト

本節では、INSERT文で行うことのできる以下の操作を説明します。

- 追加するデータをNULL値にする
- 追加するデータに省略値を使用する
- 追加するデータに現在の日時を設定する
- 別の表のデータを追加する

6.1.1 追加するデータをNULL値にする

表に追加するデータをNULL値にするには、挿入値リストに指定する値の代わりにキーワードの“NULL”を指定するか、標識変数を使用します。NULL値を追加する例を以下に示します。

例1

キーワードの“NULL”を指定する例です。発注表に取引先が“78”、取引製品が“400”、発注数量が“50”の行を追加します。このデータの仕入価格をNULL値とします。

```
INSERT INTO 在庫管理.発注表 (取引先, 取引製品, 仕入価格, 発注数量)

      VALUES (78, 400, NULL, 50)
```

例2

標識変数を使用する例です。追加するデータは例1と同じとします。

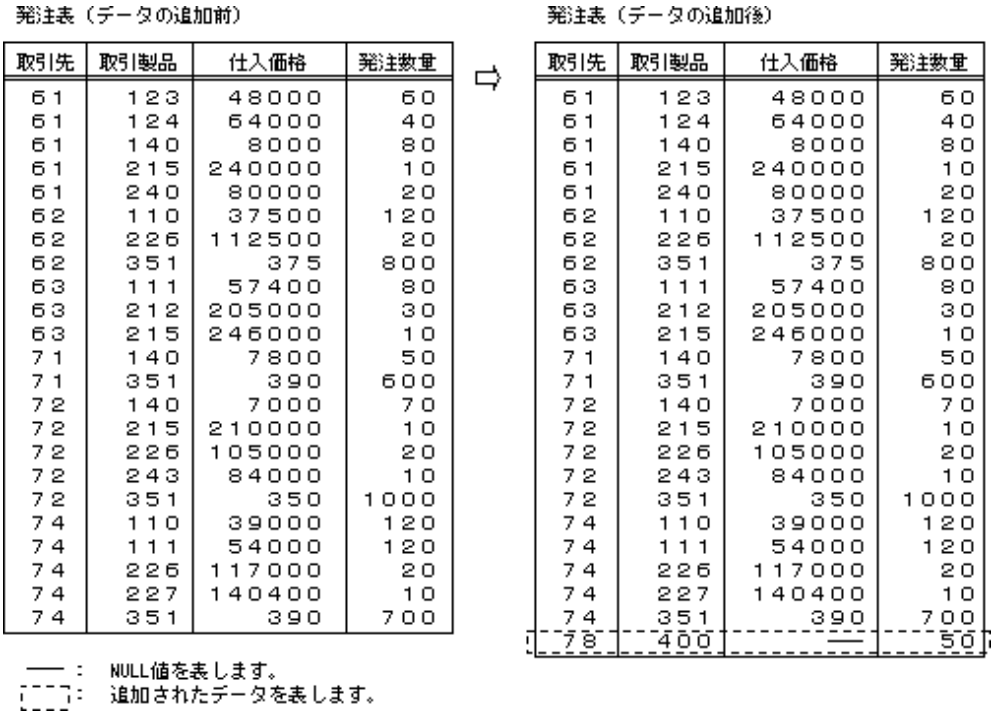
```
INSERT INTO 在庫管理.発注表 (取引先, 取引製品, 仕入価格, 発注数量)

      VALUES (78, 400, :PRICE INDICATOR :PRICE_INDICATOR, 50)
```

例2では、INSERT文を実行する前に、標識変数である“PRICE_INDICATOR”に“-1”を設定しておきます。ホスト変数“PRICE”には値を設定する必要はありません。このとき、追加されるデータはNULL値になります。なお、標識変数に“0”または正の値を設定しておくと、追加されるデータは、ホスト変数“PRICE”に設定されている値となります。

したがって、追加するデータがアプリケーションの処理によってNULL値になったりそれ以外の値となるような処理では、標識変数を使用すればINSERT文の記述は1つで済みます。一方、データを必ずNULL値とする処理では、キーワードの“NULL”を指定すればINSERT文の記述が簡単になります。

例1および例2のINSERT文の実行結果を以下に示します。

図6.1 NULL値を追加する例



表に追加するデータをNULL値にするもう1つの方法として、挿入列リストと挿入値リストから該当列の指定を省略する方法があります。その例を以下に示します。

例3

NULL値を追加する列を省略する例です。追加するデータは例1および例2と同じとします。

```
INSERT INTO 在庫管理.発注表 (取引先, 取引製品, 発注数量)

       VALUES (78, 400, 50)
```

例3では、挿入列リストの列名に仕入価格は指定していません。また、挿入値リストにも、仕入価格に対応する値は指定していません。この場合、仕入価格に省略値としてNULL値が定義されているか、省略値そのものが定義されていないと、追加されるデータはNULL値になります。省略値を使用したデータの追加は、“6.1.2 追加するデータに省略値を使用する”で説明します。

ここで注意が必要なのは、アプリケーションの処理の目的がNULL値を追加するのか、省略値を追加するのかを分けて考えることです。列の省略値は、表の設計変更などで変更されることがあります。このため、NULL値を追加するのが目的の処理であれば、例1または例2の方法を使用してください。

6.1.2 追加するデータに省略値を使用する

表を定義するとき、表に含まれるそれぞれの列には、省略値を定義しておくことができます。INSERT文では、その省略値を使用してデータを追加することができます。省略値が定義されていない列を省略した場合には、その列にNULL値が追加されますが、NOT NULL制約がある列はNULL値を設定できないのでエラーとなります。なお、省略値のことを“既定値”とも呼びます。

追加するデータの特定の列に省略値を設定する

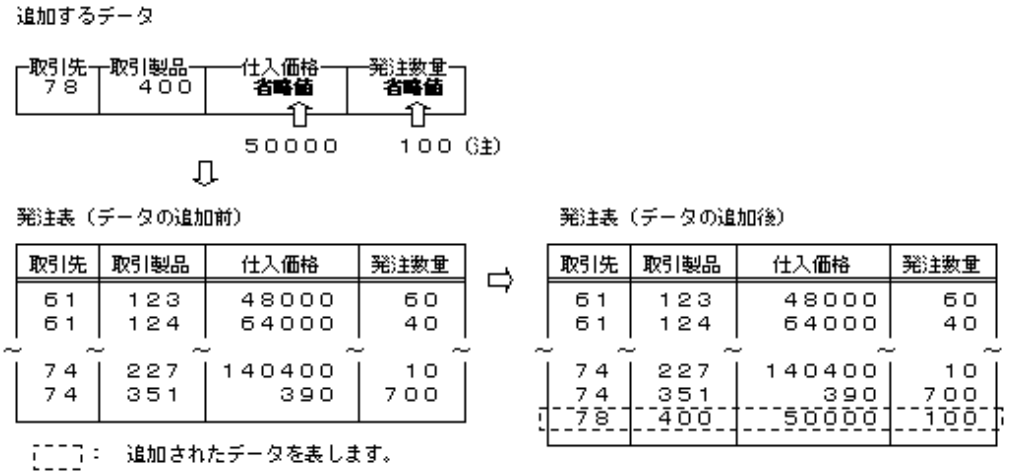
追加するデータの特定の列に省略値を設定する場合、INSERT文の挿入値リストには、キーワードの“DEFAULT”を指定します。このとき、挿入列リストは省略値を設定する列の列名を省略することはできません。2つの列に省略値を設定する例を以下に示します。

例1

発注表にデータを1行追加します。追加するデータは、取引先を“78”、取引製品を“400”とし、仕入価格と発注数量は列に定義された省略値とします。

```
INSERT INTO  在庫管理.発注表  (取引先, 取引製品, 仕入価格, 発注数量)
                                                  ↑          ↑
      VALUES (78, 400, DEFAULT, DEFAULT)        列名は省略できません。
```

図6.2 省略値を使用したデータの追加



注) 仕入価格に定義された省略値は50000、発注数量に定義された省略値は100とします。

追加するデータの特定の列に省略値を設定する場合、キーワードの“DEFAULT”を指定する代わりに、挿入値リストの値を省略する方法もあります。このとき挿入列リストも、省略値を設定する列の列名を省略します。例1のINSERT文の挿入値リストと挿入列リストを省略した場合の例を以下に示します。

例2

追加する値と追加する列を省略する例です。追加するデータは例1と同じとします。

```
INSERT INTO  在庫管理.発注表(取引先, 取引製品)
                                          ↑
      VALUES(78, 400)                     |
                 ↑            仕入価格と発注数量は省略します。
                 |
       仕入価格と発注数量の値は省略します。
```

追加するデータのすべての列に省略値を設定する

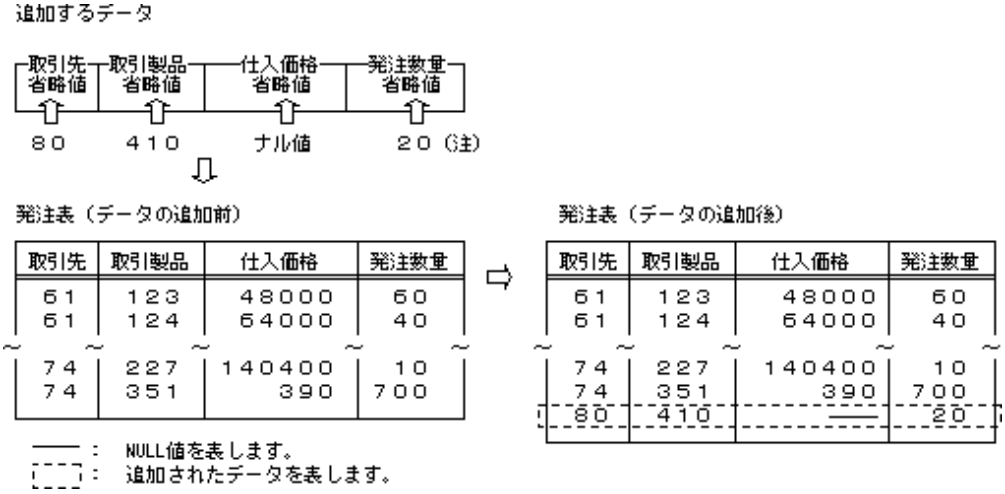
追加するデータのすべての列に省略値を設定する場合は、挿入列リストと挿入値リストの代わりにキーワード“DEFAULT VALUES”を指定します。指定例を以下に示します。

例3

発注表にデータを1行追加します。追加するデータのすべての列は、列に定義された省略値とします。

```
INSERT INTO 在庫管理.発注表 DEFAULT VALUES
```

図6.3 すべての列に省略値を使用したデータの追加



注) 取引先に定義された省略値は80、取引製品に定義された省略値は410、仕入価格に定義された省略値はNULL値、発注数量に定義された省略値は20とします。

6.1.3 追加するデータに現在の日時を設定する

追加するデータに現在の日付や時刻を設定する場合は、INSERT文の挿入値リストにキーワードを指定します。日付の設定には“CURRENT_DATE”、時間の設定には“CURRENT_TIME”、時刻印の設定には“CURRENT_TIMESTAMP”を指定します。追加するデータに現在の時間を設定する例を以下に示します。

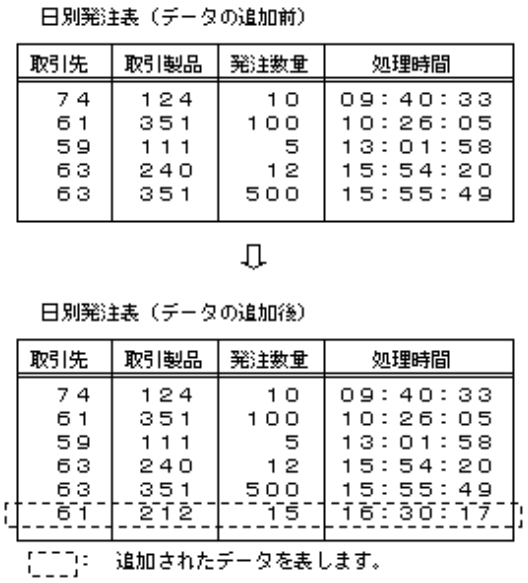
ここで、表名が日別発注表という表があるとします。この表は、一日の発注を一件ずつ格納したものです。この表は、取引先、取引製品、発注数量、および処理時間の4つの列で構成されます。

例

日別発注表にデータを1行追加します。取引先は“61”、取引製品はホスト変数“PRODNO”の値、発注数量はホスト変数“ORDERQTY”の値とし、処理時間には現在の時間(16時30分17秒)を設定します。ここでは、ホスト変数“PRODNO”には“212”、“ORDERQTY”には“15”が設定されているとします。

```
INSERT INTO 在庫管理.日別発注表(取引先, 取引製品, 発注数量, 処理時間)

       VALUES (61, :PRODNO, :ORDERQTY, CURRENT_TIME)
```

図6.4 現在の時間を使用したデータの追加



6.1.4 別の表のデータを追加する

アプリケーションの処理によっては、ある表から取り出したデータを、別の表に追加したいことがあります。この場合に、単一行SELECT文あるいはカーソルを使用してデータを取り出してから追加していたのでは効率がよくありません。INSERT文では、ある表のデータを別の表に追加することができます。追加するデータを表から取り出すには、問合せ指定を使用します。指定例を以下に示します。

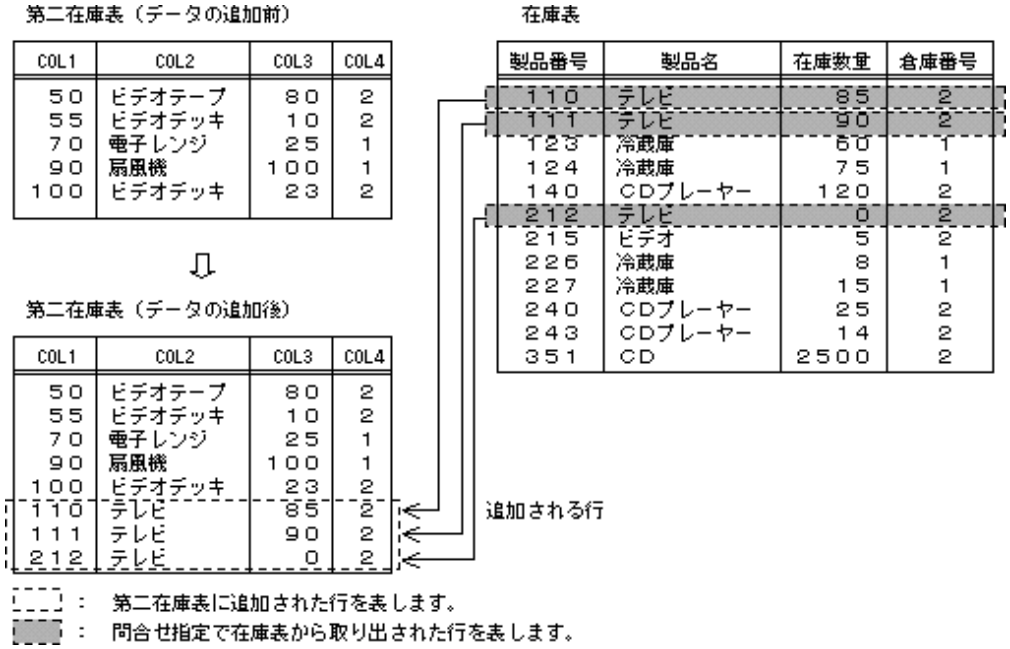
例1

在庫表から製品名が“テレビ”の行を取り出し、それを第二在庫表という名前の表に追加します。

```
INSERT INTO  在庫管理.第二在庫表 (COL1, COL2, COL3, COL4)

      SELECT 製品番号, 製品名, 在庫数量, 倉庫番号
                    (1)
      FROM 在庫管理.在庫表 WHERE 製品名 = N'テレビ'
                    (1)
```

(1) 問合せ指定

図6.5 問合せ指定によるデータの追加(列の構成が対応している場合)



備考．在庫表のデータに変化はありません。

例1では、データを追加する表と、データを取り出す表の間で、列の構成は1対1に対応していました。一般的には、両方の表の間で列の構成がこのように対応しているとは限りません。ある表の一部の列のデータを別の表に追加したり、データを追加する表の特定の列には、あらかじめアプリケーションで決めた値を追加したい場合もあります。次に、このような例を示します。

ここで、表名がCDプレーヤー管理表という表があるとします。この表は、CDプレーヤーとCDの在庫数量と発注数量を蓄積するのに使用します。この表は、製品番号、種別、数量、および区分の4つの列で構成されます。種別には、CDプレーヤーは“D”、CDは“T”が設定されます。また、数量には在庫および発注の数量、区分には、在庫は“1”、1つの取引先への発注は“2”、複数の取引先への発注は“3”が設定されます。

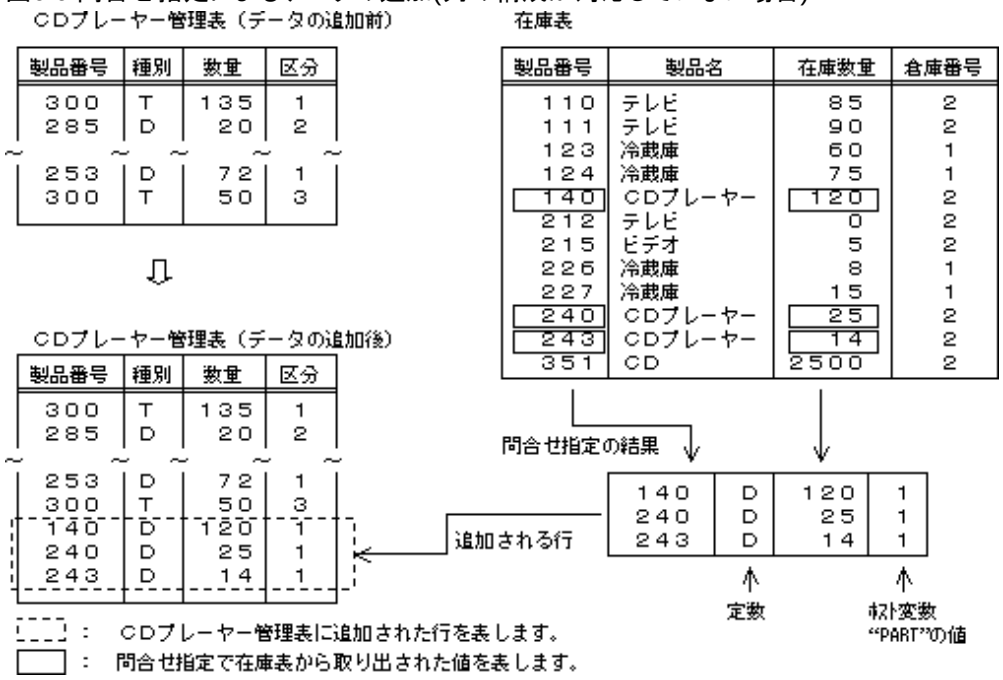
例2

CDプレーヤー管理表に、在庫表の製品番号および在庫数量を追加します。また、種別はCDプレーヤーを示す“D”とし、区分は、在庫を示す“1”がホスト変数“PART”に設定されているとします。

```
INSERT INTO 在庫管理.ＣＤプレーヤー管理表（製品番号, 種別, 数量, 区分）
```

```
    SELECT 製品番号, 'D', 在庫数量, :PART

      FROM 在庫管理.在庫表 WHERE 製品名 = N'ＣＤプレーヤー'
```

図6.6 問合せ指定によるデータの追加(列の構成が対応していない場合)



製品番号	種別	数量	区分
300	T	135	1
285	D	20	2
～	～	～	～
253	D	72	1
300	T	50	3

製品番号	製品名	在庫数量	倉庫番号
110	テレビ	85	2
111	テレビ	90	2
123	冷蔵庫	60	1
124	冷蔵庫	75	1
140	ＣＤプレーヤー	120	2
212	テレビ	0	2
215	ビデオ	5	2
226	冷蔵庫	8	1
227	冷蔵庫	15	1
240	ＣＤプレーヤー	25	2
243	ＣＤプレーヤー	14	2
351	ＣＤ	2500	2

製品番号	種別	数量	区分
300	T	135	1
285	D	20	2
～	～	～	～
253	D	72	1
300	T	50	3
140	D	120	1
240	D	25	1
243	D	14	1

140	D	120	1
240	D	25	1
243	D	14	1

備考. 在庫表のデータに変化はありません。

問合せ指定では、表から特定の値を取り出すほかに、複数の表を関連づけて操作したり、表をグループ化して値を集計することができます。その例を以下に示します。詳細については、“第9章 いろいろなデータ操作の方法”および“9.1 表をグループ化してデータを操作する”で説明しています。“図6.7 複雑な問合せ指定によるデータの追加”では内容は気にせず、問合せ指定を使用したイメージを見てください。

例3

CDプレーヤー管理表に、発注表の取引製品および取引製品ごとの発注数量の合計を追加します。追加するデータは、複数の取引先に発注しているものだけとします。また、種別はCDプレーヤーを示す“D”とし、区分は、複数の取引先への発注を示す“3”がホスト変数“PART”に設定されているとします。

```
INSERT INTO 在庫管理.ＣＤプレーヤー管理表(製品番号, 種別, 数量, 区分)

      SELECT 取引製品, 'D', SUM(発注数量), :PART

          FROM 在庫管理.発注表, 在庫管理.在庫表

          WHERE 取引製品 = 製品番号 AND 製品名 = N'ＣＤプレーヤー'

          GROUP BY 取引製品

          HAVING COUNT(取引先) >= 2
```

図6.7 複雑な問合せ指定によるデータの追加

FROM句で導出される表

取引先	取引製品	仕入価格	発注数量	製品番号	製品名	在庫数量	倉庫番号
61	123	48000	60	110	テレビ	85	2
61	123	48000	60	111	テレビ	90	2
61	123	48000	60	123	冷蔵庫	60	1
・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・

（FROM句で導出される表は、発注表と在庫表のすべての行の組合せからなる276行です）

⇩

WHERE句で導出される表

取引先	取引製品	仕入価格	発注数量	製品番号	製品名	在庫数量	倉庫番号
61	140	8000	80	140	CDプレーヤー	120	2
61	240	80000	20	240	CDプレーヤー	25	2
71	140	7800	50	140	CDプレーヤー	120	2
72	140	7000	70	140	CDプレーヤー	120	2
72	243	84000	10	243	CDプレーヤー	14	2

（WHERE句で導出される表は、FROM句の結果の表の中で取引製品と製品番号が等しく、かつ製品名が“CDプレーヤー”の5行です）

⇩

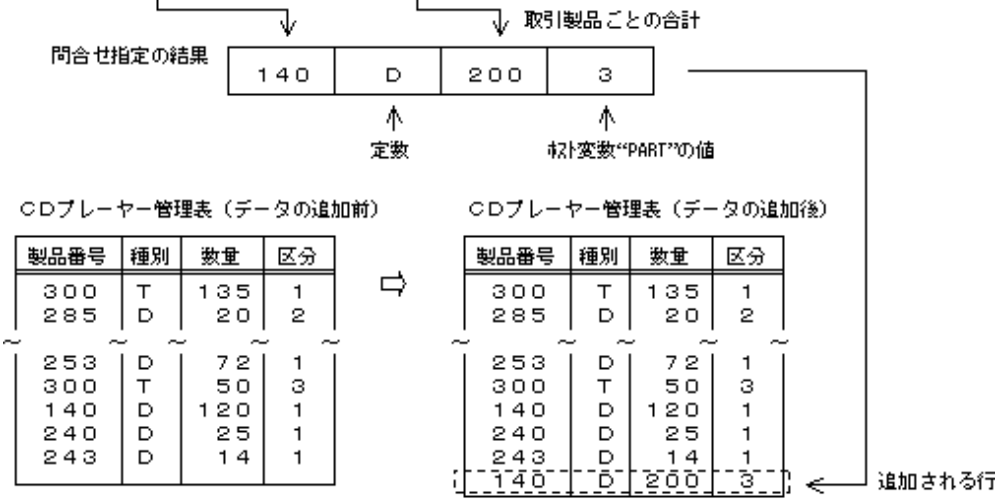
GROUP BY句で導出される表

取引先	取引製品	仕入価格	発注数量	製品番号	製品名	在庫数量	倉庫番号
61	140	8000	80	140	CDプレーヤー	120	2
71	140	7800	50	140	CDプレーヤー	120	2
72	140	7000	70	140	CDプレーヤー	120	2
61	240	80000	20	240	CDプレーヤー	25	2
72	243	84000	10	243	CDプレーヤー	14	2

⇩

HAVING句で導出される表

取引先	取引製品	仕入価格	発注数量	製品番号	製品名	在庫数量	倉庫番号
61	140	8000	80	140	CDプレーヤー	120	2
71	140	7800	50	140	CDプレーヤー	120	2
72	140	7000	70	140	CDプレーヤー	120	2



[---]： CDプレーヤー管理表に追加された行を表します。

備考. 発注表および在庫表のデータに変化はありません。

6.2 データベースのデータを更新する

表のデータを更新するには、UPDATE文を使用します。UPDATE文の例を以下に示します。

例

発注表の取引先が“61”で、かつ取引製品が“215”のデータについて、発注数量を“50”に更新します。

```
UPDATE  在庫管理.発注表  SET  発注数量 = 50
              (1)                (2)

        WHERE  取引先 = 61 AND 取引製品 = 215
                              (3)
```

(1) 表名

(2) 設定句

(3) 探索条件

本節では、UPDATE文で行うことのできる以下の操作を説明します。

- 複数の列の値を更新する
- 更新するデータをNULL値にする
- 更新するデータに省略値を使用する
- 更新するデータに現在の日時を設定する
- 更新するデータに別の列の値を使用する
- 更新するデータに四則演算を行う
- 表のすべての行を更新する

UPDATE文によるデータの更新には、ここで説明する方法のほかにカーソルを使用する方法もあります。その方法は“7.5 カーソルを使用してデータを更新する”で説明します。

なお、ここで説明するUPDATE文によるデータの更新は、探索条件により更新対象の行を特定することから、“UPDATE文:探索”と呼びます。

6.2.1 複数の列の値を更新する

“6.2 データベースのデータを更新する”で最初に示した例では、データを更新する列は“発注数量”だけでした。データを更新したい列が、“仕入価格”と“発注数量”の2つになったとします。それぞれの列を指定したUPDATE文を2回実行したのでは効率がよくありません。UPDATE文では、一度に複数の列の値を更新することができます。

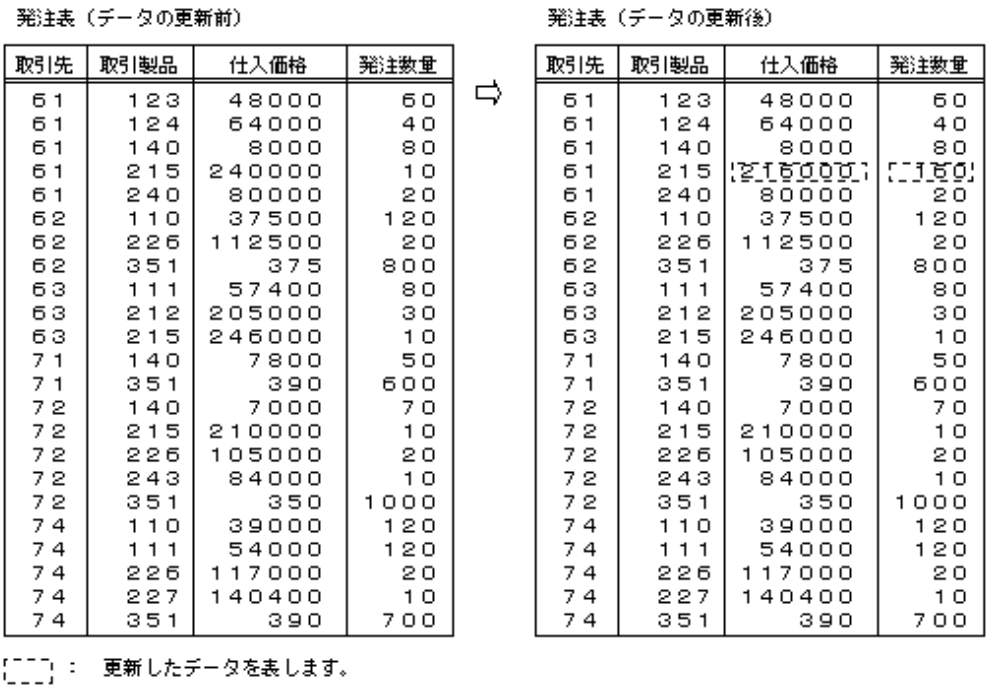
複数の列のデータを更新するには、設定句をコンマ(,)で区切って指定します。なお、設定句に指定しない列の値は変更されません。複数の列を更新する例を以下に示します。

例

発注表の取引先が“61”で、かつ取引製品が“215”のデータについて、仕入価格を“216000”に、発注数量を“160”に更新します。

```
UPDATE 在庫管理.発注表 SET 仕入価格 = 216000, 発注数量 = 160

        WHERE 取引先 = 61 AND 取引製品 = 215
```

図6.8 複数の列を更新する例



発注表（データの更新前）

取引先	取引製品	仕入価格	発注数量
61	123	48000	60
61	124	64000	40
61	140	8000	80
61	215	240000	10
61	240	80000	20
62	110	37500	120
62	226	112500	20
62	351	375	800
63	111	57400	80
63	212	205000	30
63	215	246000	10
71	140	7800	50
71	351	390	600
72	140	7000	70
72	215	210000	10
72	226	105000	20
72	243	84000	10
72	351	350	1000
74	110	39000	120
74	111	54000	120
74	226	117000	20
74	227	140400	10
74	351	390	700

発注表（データの更新後）

取引先	取引製品	仕入価格	発注数量
61	123	48000	60
61	124	64000	40
61	140	8000	80
61	215	216000	NULL
61	240	80000	20
62	110	37500	120
62	226	112500	20
62	351	375	800
63	111	57400	80
63	212	205000	30
63	215	246000	10
71	140	7800	50
71	351	390	600
72	140	7000	70
72	215	210000	10
72	226	105000	20
72	243	84000	10
72	351	350	1000
74	110	39000	120
74	111	54000	120
74	226	117000	20
74	227	140400	10
74	351	390	700

[ ] ： 更新したデータを表します。

6.2.2 更新するデータをNULL値にする

表のデータをNULL値に更新するには、設定句に指定する値の代わりにキーワードの“NULL”を指定するか、標識変数を使用します。NULL値に更新する例を以下に示します。

例1

キーワードの“NULL”を指定する例です。発注表の取引先が“61”で、かつ取引製品が“215”のデータについて、仕入価格を“216000”に、発注数量をNULL値に更新します。

```
UPDATE 在庫管理.発注表 SET 仕入価格 = 216000, 発注数量 = NULL

       WHERE 取引先 = 61 AND 取引製品 = 215
```

例2

標識変数を使用する例です。更新するデータは例1と同じとします。

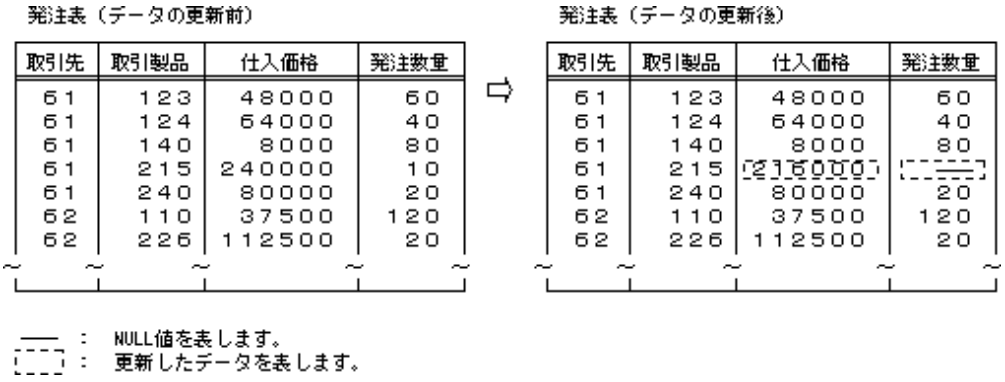
```
UPDATE 在庫管理.発注表

       SET 仕入価格 = 216000,

           発注数量 = :ORDERQTY INDICATOR :ORDERQTY_INDICATOR

       WHERE 取引先 = 61 AND 取引製品 = 215
```

例2では、UPDATE文を実行する前に、標識変数である“ORDERQTY_INDICATOR”に“-1”を設定しておきます。ホスト変数“ORDERQTY”には値を設定する必要はありません。このとき、更新されるデータはNULL値になります。なお、標識変数に“0”または正の値を設定しておくと、更新されるデータはホスト変数“ORDERQTY”に設定されている値となります。

したがって、更新するデータがアプリケーションの処理によってNULL値になったりそれ以外の値となるような処理では、標識変数を使用すればUPDATE文の記述は1つで済みます。一方、データを必ずNULL値とする処理では、キーワードの“NULL”を指定すればUPDATE文の記述が簡単になります。

例1および例2のUPDATE文の実行結果を以下に示します。

図6.9 NULL値に更新する例



6.2.3 更新するデータに省略値を使用する

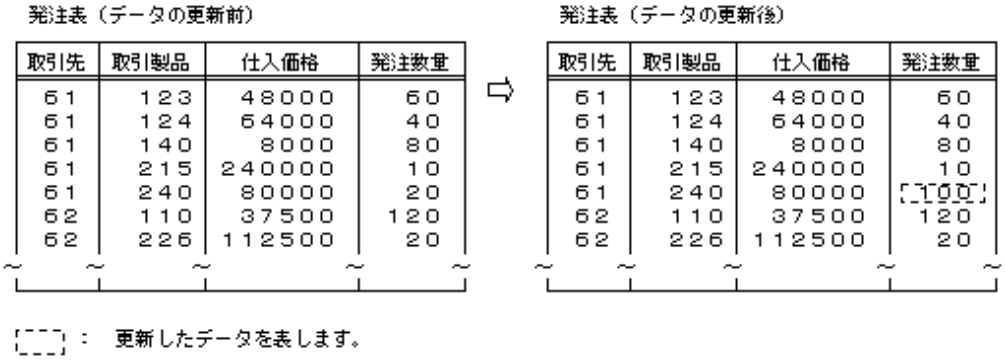
省略値を使用してデータを追加する方法は“6.1.2 追加するデータに省略値を使用する”で説明しました。UPDATE文では、省略値を使用してデータを更新することができます。更新するデータに省略値を設定するには、設定句に指定する値の代わりにキーワードの“DEFAULT”を指定します。設定句に指定した列に省略値が定義されていない場合は、その列はNULL値に更新されますが、NOT NULL制約がある列はNULL値を設定できないのでエラーとなります。指定例を以下に示します。

例

キーワードの“DEFAULT”を指定する例です。発注表の取引製品が“240”のデータについて、発注数量を省略値に更新します。

```
UPDATE 在庫管理.発注表 SET 発注数量 = DEFAULT WHERE 取引製品 = 240
```

図6.10 省略値を使用したデータの更新



備考. 発注数量に定義された省略値は100とします。

6.2.4 更新するデータに現在の日時を設定する

更新するデータに現在の日付や時刻を設定する場合は、設定句に指定する値の代わりにキーワードを指定します。日付の設定には“CURRENT_DATE”、時間の設定には“CURRENT_TIME”、時刻印の設定には“CURRENT_TIMESTAMP”を指定します。更新するデータに現在の時刻印を設定する例を以下に示します。

ここで、表名がテレビ在庫表という表があるとします。この表は、処理日時、製品番号、および在庫数量の3つの列で構成されます。在庫数量には、処理日時における在庫数量を設定します。

例

テレビ在庫表の製品番号が“212”のデータについて、処理日時を現在の日時(2007年04月14日 18時35分31秒)に、在庫数量をホスト変数“STOCKQTY”の値に更新します。ここでは、ホスト変数“STOCKQTY”には“65”が設定されているとします。

```
UPDATE 在庫管理.テレビ在庫表

        SET 処理日時 = CURRENT_TIMESTAMP, 在庫数量 = :STOCKQTY

        WHERE 製品番号 = 212
```

図6.11 現在の日時を使用したデータの更新

テレビ在庫表（データの更新前）

処理日時	製品番号	在庫数量
2007-03-03 11:07:10	110	120
2007-03-10 13:16:57	111	5
2007-04-03 15:42:09	212	80
2007-03-25 10:20:36	255	10
2007-03-19 09:55:03	280	22

⇩

テレビ在庫表（データの更新後）

処理日時	製品番号	在庫数量
2007-03-03 11:07:10	110	120
2007-04-10 13:16:57	111	5
2007-04-14 18:35:31	212	65
2007-03-25 10:20:36	255	10
2007-03-19 09:55:03	280	22

[ ]: 更新したデータを表します。

6.2.5 更新するデータに別の列の値を使用する

データを更新するときに、別の列の値を使用することができます。次に、この例を示します。

ここで、出荷表という名前の表があるとします。この表は、製品番号、出荷数量、前回出荷、および増減の4つの列から構成されます。製品を出荷すると、その数量が出荷数量に設定され、その前回に出荷した数量が前回出荷に設定されます。増減は、今回の出荷数量と、前回の出荷数量の差です。なお、増減の値は、出荷データが発生するたびに設定するのではなく、ある時点でまとめて設定することにします。

例

出荷表の製品番号が“140”の出荷数量を“120”に更新し、前回出荷を今までの出荷数量とします。増減については、ここでは更新しません。

```
UPDATE 在庫管理.出荷表 SET 出荷数量 = 120, 前回出荷 = 出荷数量

        WHERE 製品番号 = 140
```

図6.12 別の列の値を使用した更新の例

出荷表（データの更新前）

製品番号	出荷数量	前回出荷	増減
110	20	16	4
111	65	70	-5
123	40	40	0
124	23	25	-2
140	100	79	21
212	7	10	-3
215	13	7	6
226	1	3	-2
227	16	12	4
240	21	6	15
243	17	10	7
351	600	600	0

⇨

出荷表（データの更新後）

製品番号	出荷数量	前回出荷	増減
110	20	16	4
111	65	70	-5
123	40	40	0
124	23	25	-2
140	120	100	21
212	7	10	-3
215	13	7	6
226	1	3	-2
227	16	12	4
240	21	6	15
243	17	10	7
351	600	600	0

[ ]: 更新したデータを表します。

ここで注意が必要なのは、列の値を更新のデータに指定した場合、使用される値は、そのUPDATE文の実行前の値であることです。この例でも、出荷数量は定数で指定された“120”に更新されますが、前回出荷は、このUPDATE文の実行前の出荷数量の値である“100”に更新されます。このUPDATE文で更新した“120”にはなりません。

6.2.6 更新するデータに四則演算を行う

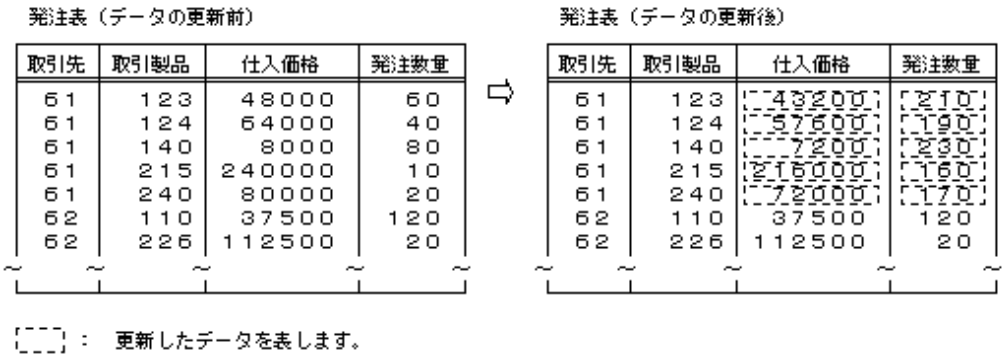
更新するデータとして、列の値などに四則演算を実施した結果を使用することができます。指定例を以下に示します。

例

発注表の取引先が“61”のデータについて、仕入価格を1割引すると同時に、発注数量に150を加算します。

```
UPDATE 在庫管理.発注表

       SET 仕入価格 = 仕入価格 * 0.9, 発注数量 = 発注数量 + 150

       WHERE 取引先 = 61
```

図6.13 更新するデータに四則演算を行う例



この例では、仕入価格を1割引するために“仕入価格×0.9”の演算をしました。目的は1割引するのですから、“仕入価格－(仕入価格×0.1)”の演算をしてもかまいません。しかし、演算子の数を多くするとSQL文の処理効率が下がることがあります。この例のように、演算式はできるだけ単純にしてください。

6.2.7 表のすべての行を更新する

表のすべての行を更新するには、探索条件を省略します。もちろん、キーワードの“WHERE”も指定しません。

“図6.12 別の列の値を使用した更新の例”で説明した例を思い出してください。この例では、出荷データが発生したときに、出荷数量と前回出荷の値を更新しました。しかし、その増減は設定していませんでした。ここで、増減の値を最新の状態に更新することにします。この例を以下に示します。

例

出荷表のすべての行について増減の値を、出荷数量から前回出荷を引いた値で更新します。

```
UPDATE 在庫管理.出荷表 SET 増減 = 出荷数量 - 前回出荷
                                                  ↑
                                                  |
                                          WHEREは指定しません。
```

図6.14 表のすべての行を更新する例

出荷表（データの更新前）

製品番号	出荷数量	前回出荷	増減
110	21	20	4
111	68	65	-5
123	40	40	0
124	21	23	-2
140	120	100	15
212	7	10	-3
215	20	13	6
226	3	3	-2
227	16	12	4
240	21	6	15
243	15	17	7
351	650	600	0

⇨

出荷表（データの更新後）

製品番号	出荷数量	前回出荷	増減
110	21	20	[1]
111	68	65	[3]
123	40	40	[0]
124	21	23	[-2]
140	120	100	[20]
212	7	10	[-3]
215	20	13	[7]
226	3	3	[0]
227	16	12	[4]
240	21	6	[15]
243	15	17	[-2]
351	650	600	[50]

[ ]： 更新したデータを表します。

備考. 値が変わらない行についても元と同じ値に更新しています。

この例では、すべての行について増減の値を更新していますが、前回の更新後に、出荷データが発生していない行については、実際の値は変わりません。この行についてもデータの更新は行われています。すなわち、元の値と同じ値に更新しているのです。表全体の行数に対して、このような行の比率が大きい場合には、なんらかの方法で更新が必要な行を管理して、必要な行だけを更新した方が効率がよくなります。

6.3 データベースのデータを削除する

表のデータを削除するには、DELETE文を使用します。DELETE文の例を以下に示します。

例

発注表の取引先が“61”で、かつ取引製品が“215”の行を削除します。

```
DELETE FROM 在庫管理.発注表 WHERE 取引先 = 61 AND 取引製品 = 215
            (1)                (2)
```

(1) 表名

(2) 探索条件

発注表（データの削除前）

取引先	取引製品	仕入価格	発注数量
61	123	48000	60
61	124	64000	40
61	140	8000	80
61	215	240000	10
61	240	80000	20
62	110	37500	120
62	226	112500	20
～	～	～	～	～

⇨

発注表（データの削除後）

取引先	取引製品	仕入価格	発注数量
61	123	48000	60
61	124	64000	40
61	140	8000	80
[ ]	[ ]	[ ]	[ ]
61	240	80000	20
62	110	37500	120
62	226	112500	20
～	～	～	～	～

[ ]： 削除したデータを表します。

本節では、DELETE文で行うことのできる以下の操作を説明します。

- 表のすべての行を削除する

DELETE文によるデータの削除には、ここで説明する方法のほかにカーソルを使用する方法もあります。その方法は“7.6 カーソルを使用してデータを削除する”で説明します。

なお、ここで説明するDELETE文によるデータの削除は、探索条件により削除対象の行を特定することから、“DELETE文:探索”と呼びます。

6.3.1 表のすべての行を削除する

表のすべての行を削除するには、探索条件を省略します。もちろん、キーワードの“WHERE”も指定しません。この例を以下に示します。

例

発注表のすべての行を削除します。

```
DELETE FROM 在庫管理.発注表
                              ↑
                              |
                    WHEREは指定しません。
```

図6.15 表のすべての行を削除する例

発注表（データの削除前）

取引先	取引製品	仕入価格	発注数量
61	123	48000	60
61	124	64000	40
61	140	8000	80
61	215	240000	10
61	240	80000	20
62	110	37500	120
62	226	112500	20
62	351	375	800
63	111	57400	80
63	212	205000	30
63	215	246000	10
71	140	7800	50
71	351	390	600
72	140	7000	70
72	215	210000	10
72	226	105000	20
72	243	84000	10
72	351	350	1000
74	110	39000	120
74	111	54000	120
74	226	117000	20
74	227	140400	10
74	351	390	700

⇨

発注表（データの削除後）

取引先	取引製品	仕入価格	発注数量

すべての行が削除されました。

第7章 カーソルを使用するデータ操作の方法

本章では、カーソルを使用してデータを操作する方法を説明します。

7.1 カーソルの概念

カーソルは、表の中の1行を特定する仮想的な道具です。カーソルを使用して処理の対象とする行を特定しておいて、その行からデータを取り出したり、その行を更新したり、または削除したりすることができます。カーソルにより行を特定することを、カーソルを位置づけるといいます。

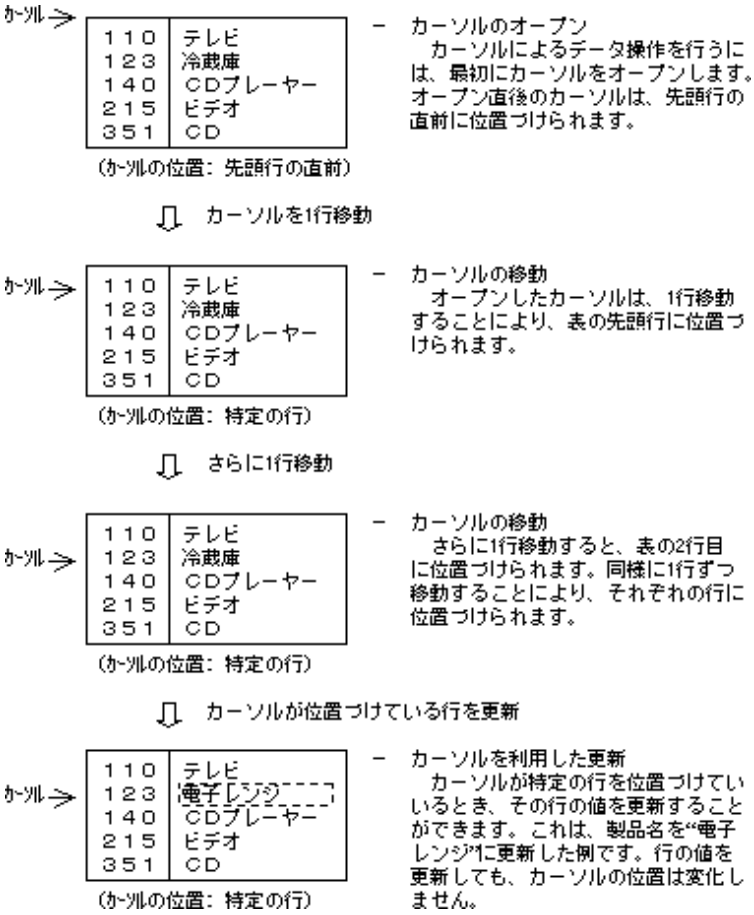
カーソルには、オープンされている状態と、クローズされている状態の2つがあります。一度もオープンされていないカーソルは、クローズされている状態と同じです。カーソルを使用してデータの操作ができるのは、オープンされている状態のときです。

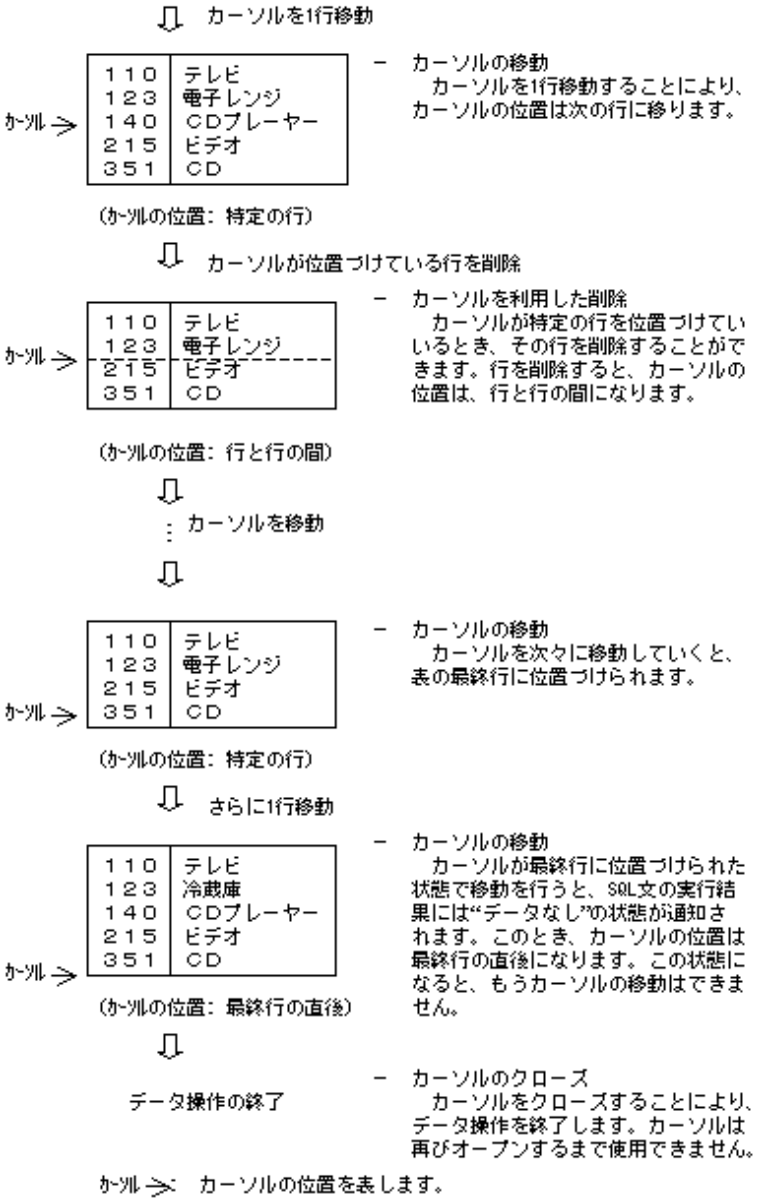
オープンされた状態のカーソルは、表の特定の行を位置づけているときと、そうでないときがあります。カーソルの概念を理解しやすくするために、カーソルの位置には以下の4通りがあると考えます。

- 先頭行の直前を位置づけている
- 特定の行を位置づけている
- 行と行の間を位置づけている
- 最終行の直後を位置づけている

カーソルの概念を以下に示します。なお、それぞれの操作に使用するSQL文については、この次に説明します。以下では説明を簡単にするために、在庫表を小さくした表を使用します。また、列名の記述は省いています。

図7.1 カーソルの概念





カーソルを使用したデータ操作の順序

カーソル系のSQL文を以下に示します。

- カーソル宣言(DECLARE CURSOR)
- OPEN文
- CLOSE文
- FETCH文
- UPDATE文
- DELETE文

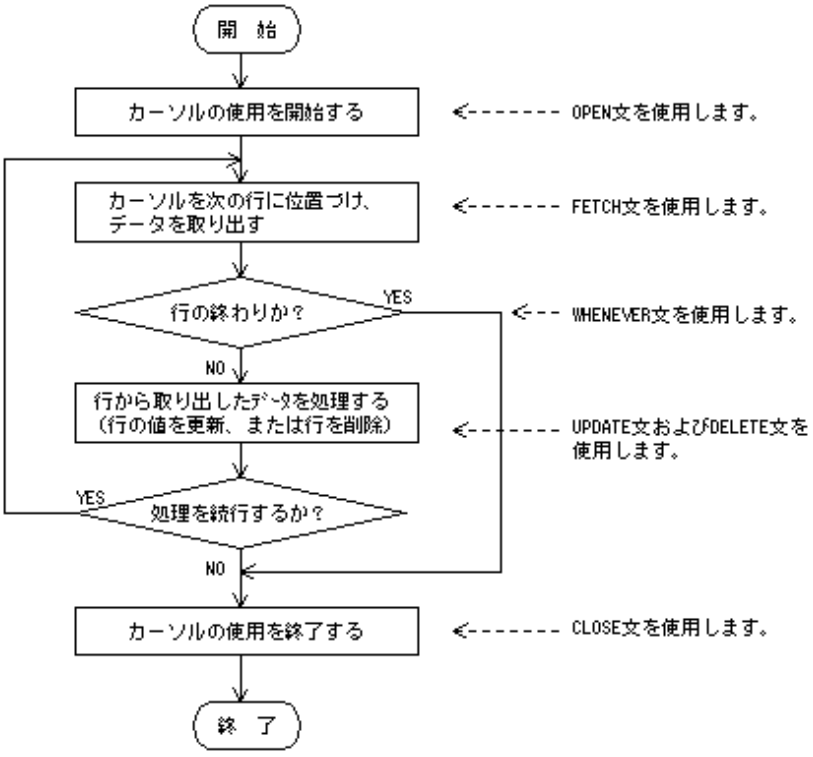
カーソルを使用してデータ操作を行うには、あらかじめカーソルを宣言しておくことが必要です。カーソルの宣言では、データ操作を行う表や、表からデータを取り出す方法を指定します。カーソルを宣言するSQL文を、“カーソル宣言”と呼びます。カーソル宣言は非実行文です。カーソル宣言は、アプリケーションの実行順序とは関係なく、そのカーソルを使用するSQL文よりも前に記述しておくことが必要です。

カーソルを使用してデータ操作を実行する順序は以下のとおりです。

1. カーソルをオープンします。OPEN文を使用します。
2. カーソルを表の特定行に位置づけ、データを取り出します。FETCH文を使用します。
3. カーソルを位置づけた行のデータを処理します。
   - 取り出したデータをアプリケーションで処理します。
   - カーソルを位置づけた行の値を更新します。UPDATE文を使用します。
   - カーソルを位置づけた行を削除します。DELETE文を使用します。
4. カーソルをクローズします。CLOSE文を使用します。

複数の行を処理する場合は、2)および3)を繰り返します。以上の操作の流れを以下に示します。

図7.2 カーソルによるデータ操作の流れ



カーソルを使用してデータを操作するアプリケーションの例を以下に示します。このアプリケーションでは、在庫表について以下の処理を行います。

- 在庫数量が“0”の行を削除します。
- 製品名が“冷蔵庫”の行の倉庫番号を“5”に変更します。
- 製品名が“冷蔵庫”の行の在庫数量の合計を計算します。合計値は変数totalに入れます。

カーソルを使用してデータ操作を行うアプリケーションの例

```
#include <stdio.h>
#include <string.h>

EXEC SQL BEGIN DECLARE SECTION;
    char    SQLSTATE[6];                                          …(1)
    char    SQLMSG[256];                                          …(1)
    char CHARACTER SET IS NCHAR   outdata1[21];                    …(1)
    long    outdata2;
EXEC SQL END DECLARE SECTION;
```

```
main()
{
   long      total;
   memset(SQLSTATE, 0x00, 6);                                          …(2)
   memset(SQLMSG, 0x00, 256);                                          …(2)
   memset(outdata1, 0x00, 21);                                         …(2)
   EXEC SQL CONNECT TO DEFAULT;                                        …(3)
   EXEC SQL DECLARE CU1 CURSOR FOR                                     …(4)
                    SELECT 製品名, 在庫数量 FROM 在庫管理.在庫表;
   EXEC SQL WHENEVER NOT FOUND GOTO :END;                              …(5)
   total = 0;
   EXEC SQL OPEN CU1;                                                  …(6)
   while(1) {
       EXEC SQL FETCH CU1 INTO :outdata1, :outdata2;                   …(7)
       if( outdata2 == 0 ) {
           EXEC SQL DELETE FROM 在庫管理.在庫表 WHERE CURRENT OF CU1;      …(8)
           continue;
       }
       if( memcmp(outdata1, "冷蔵庫              ", 20) == 0 ) {          …(9)
           total += outdata2;                                          …(10)
           EXEC SQL UPDATE 在庫管理.在庫表 SET 倉庫番号 = 5
                                          WHERE CURRENT OF CU1;
       }
    }
END:
    EXEC SQL CLOSE CU1;                                                …(11)
    EXEC SQL COMMIT WORK;                                              …(12)
    EXEC SQL DISCONNECT DEFAULT;                                       …(13)
}
```

(1) 結果データを格納するホスト変数を宣言します。

(2) ホスト変数を初期化します。

(3) コネクションを接続します。

(4) カーソルCU1の宣言文です。在庫表の製品名および在庫数量の列だけの表に対するカーソルCU1を宣言しています。

(5) (7)のカーソルの位置づけで、カーソルが最終行の次まで進んだとき、ENDへ進むことを指定しています。

(6) カーソルCU1の使用を開始します。ここで(4)の宣言のカーソル表が作成され、その先頭の行の直前にカーソルが位置づけられます。

(7) カーソルCU1を次の行（OPEN直後の場合は表の先頭行）に位置づけ、その位置づけた行からデータを取り出します。製品名のデータをoutdata1に、在庫数量のデータをoutdata2に格納します。

(8) (7)の文の実行結果をチェックします。在庫数量が“0”の場合は、その行を削除します。

(9) (7)の文の実行結果をチェックします。製品名が“冷蔵庫”の場合は、読み出した在庫数量（outdata2に格納されています）をtotalに積算します。

(10) カーソルの位置づけられている行（製品名が“冷蔵庫”）の倉庫番号を“5”に変更します。

(11) カーソルの使用を終了します。

(12) 現行のトランザクションを終了します。

(13) コネクションを切断します。

7.2 カーソルを宣言する

カーソルを使用してデータ操作を行うには、カーソルを宣言しておくことが必要です。カーソル宣言は非実行文です。カーソル宣言の指定位置は、そのカーソルを使用するSQL文よりも、アプリケーションの記述の順序で前にあることが必要です。実行順序とは関係ありません。

カーソル宣言の例を以下に示します。

例1

発注表から、取引先、取引製品、発注数量を求めるためのカーソルを宣言します。

```
DECLARE  発注表カーソル    CURSOR  FOR
              (1)

         SELECT 取引先, 取引製品, 発注数量 FROM 在庫管理.発注表
                                    (2)
```

(1) カーソル名

(2) 問合せ式

このカーソル宣言は、発注表から取引先、取引製品、および発注数量の3つの列の値を、すべての行について取り出すためのカーソルを宣言しています。

カーソルを使用して操作する表やデータの取り出し方は、カーソル宣言の問合せ式で指定します。例1の問合せ式で行うデータ操作は単純なものです。このカーソルを使用して表からデータを取り出し、アプリケーションの論理でいろいろな処理を行うことができます。一方、問合せ式では、表をグループ化して操作したり、複数の表の集合和からデータを取り出すことができます。問合せ式をうまく利用すれば、アプリケーションの処理を大幅に削減することが可能になります。

問合せ式

問合せ式には、1つの問合せ指定を指定する方法と、2つ以上の問合せ指定を“UNION”で結合して指定する方法があります。UNIONの指定方法については、“8.5 複数の表から行の集合和を求める”で説明します。

問合せ指定

問合せ指定は、単一行SELECT文からINTO句を取り除いたのと同じ形式です。問合せ指定により表を導出しますが、表の導出方法は単一行SELECT文の場合と同じです。すなわち、FROM句によって導出される表から、WHERE句の条件を満たす行について、選択リストで指定する列の値を取り出します。なお、単一行SELECT文では、取り出す行が2行以上あるとエラーになりますが、問合せ指定では、導出する行は何行あってもかまいません。

また、複数行を対象とするようなグループ処理ができます。詳しくは、“9.1 表をグループ化してデータを操作する”で説明します。ここでは、問合せ指定にGROUP BY句を使用したカーソル宣言について説明します。指定例を以下に示します。

例2

在庫表と発注表から、製品番号ごとの、製品名、発注している取引先の件数、および在庫数量と発注数量の合計を求めるためのカーソルを宣言します。

```
DECLARE 製品番号ごとに処理するカーソル CURSOR FOR

    SELECT 製品番号, 製品名, COUNT(取引先), 在庫数量 + SUM(発注数量)

        FROM 在庫管理.在庫表, 在庫管理.発注表

        WHERE 製品番号 = 取引製品

        GROUP BY 製品番号, 製品名, 在庫数量
```

図7.3 カーソル表の導出

在庫表

製品番号	製品名	在庫数量	倉庫番号
110	テレビ	85	2
111	テレビ	90	2
123	冷蔵庫	60	1
124	冷蔵庫	75	1
140	CDプレーヤー	120	2
212	テレビ	0	2
215	ビデオ	5	2
226	冷蔵庫	8	1
227	冷蔵庫	15	1
240	CDプレーヤー	25	2
243	CDプレーヤー	14	2
351	CD	2500	2

発注表

取引先	取引製品	仕入価格	発注数量
61	123	48000	60
61	124	64000	40
61	140	8000	80
61	215	240000	10
61	240	80000	20
62	110	37500	120
62	226	112500	20
62	351	375	800
63	111	57400	80
63	212	205000	30
63	215	246000	10
71	140	7800	50
71	351	390	600
72	140	7000	70
72	215	210000	10
72	226	105000	20
72	243	84000	10
72	351	350	1000
74	110	39000	120
74	111	54000	120
74	226	117000	20
74	227	140400	10
74	351	390	700

⇩

カーソル表

製品番号	製品名	(注1)	(注2)
110	テレビ	2	325
111	テレビ	2	290
123	冷蔵庫	1	120
124	冷蔵庫	1	115
140	CDプレーヤー	3	320
212	テレビ	1	30
215	ビデオ	3	35
226	冷蔵庫	3	68
227	冷蔵庫	1	25
240	CDプレーヤー	1	45
243	CDプレーヤー	1	24
351	CD	4	5600

注1) COUNT(取引先)の結果から作成される列です。

注2) 在庫数量 + SUM(発注数量)の結果から作成される列です。

備考. 演算式や集合関数の結果の列には名前がありません。

カーソルにより取り出す行の順序

カーソルによって取り出される行の順序に決まりはありません。すなわち、カーソル表に含まれる行は問合せ式によって決定しますが、それらの行の順序についてはなにも決まっていません。このことは、1つの表からすべての行を取り出すような単純なカーソルでも同じです。“図7.3 カーソル表の導出”で示したカーソル表の行は、たまたま製品番号の順番に並んでいますが、必ずしもこの順番になるとは限りません。実際にアプリケーションを実行してみると、以下のような順序になるかもしれません。

図7.4 カーソル表に含まれる行の順序は不定となる例

カーソル表

データを取り出す順序 ↓

製品番号	製品名	(注1)	(注2)
212	テレビ	1	30
123	冷蔵庫	1	120
110	テレビ	2	325
227	冷蔵庫	1	25
240	CDプレーヤー	1	45
215	ビデオ	3	35
351	CD	4	5600
140	CDプレーヤー	3	320
226	冷蔵庫	3	68
111	テレビ	2	290
243	CDプレーヤー	1	24
124	冷蔵庫	1	115

注1) COUNT(取引先)の結果から作成される列です。

注2) 在庫数量 + SUM(発注数量)の結果から作成される列です。

備考. 演算式や集合関数の結果の列には名前がありません。

行の順序を並べ替える

カーソル宣言では、カーソル表に含まれる行を並べ替えることができます。カーソル表の行を並べ替えるには、問合せ式の後にORDER BY句を指定します。なお、行を並べ替えることを“ソート”と呼びます。ORDER BY句の指定例を以下に示します。

例3

例2のカーソル宣言で、カーソルによってデータを取り出す順序を製品番号の小さい順とします。

```
DECLARE 製品番号ごとに処理するカーソル CURSOR FOR

    SELECT 製品番号, 製品名, COUNT(取引先), 在庫数量 + SUM(発注数量)

        FROM 在庫管理.在庫表, 在庫管理.発注表

        WHERE 製品番号 = 取引製品

        GROUP BY 製品番号, 製品名, 在庫数量

        ORDER BY 製品番号 ASC
          (A)          (1)
```

(1) ソート指定

(A)～(1) ORDER BY句

ORDER BY句には、ソートするための列名、集合関数または値式と、ソートの方法を指定します。ソートするための列を“ソートキー”と呼びます。ソートの方法には以下のどちらかが指定できます。なお、ソートの方法は省略することができ、その場合は“ASC”を指定したのと同じになります。

ASC:

値の小さい順に並べ替えます。“昇順ソート”といいます。

DESC:

値の大きい順に並べ替えます。“降順ソート”といいます。

NULL値は、最大の値として処理します。そのため、昇順では最後に、降順では最初に扱われます。

例3のように指定することにより、カーソル表に含まれる行の順序は“図7.3 カーソル表の導出”になることが決まります。なお、“図7.3 カーソル表の導出”では製品番号の値は行ごとに異なっているので、カーソル表の行の順序は一意に決まりました。次に、製品番号の値が同じ行が複数あった場合について説明します。

同じソートキーの値が複数行ある場合には、それらの行間での順序に決まりはありません。ソートキーの値が異なる行間では、ソート指定に従った順序となります。ソートキーの値が同じ行についても並べ替えるには、複数のソート指定をコンマ(,)で区切って指定します。ORDER BY句に指定された複数のソートキーは、左から順番に“第1ソートキー,第2ソートキー,…”と呼ばれます。カーソル表の行の順序は、まず第1ソートキーの値で並べ替えられ、その値が同じ行に対して第2ソートキーの値で並べ替えられます。すべてのソートキーについて同じ値を持つ行の順序には決まりはありません。指定例を以下に示します。

例4

在庫表から、倉庫番号の値の小さい順にデータを取り出すカーソルを宣言します。倉庫番号の値が同じ行については、在庫数量の値が大きい順とします。

```
DECLARE 在庫表カーソル CURSOR FOR

        SELECT * FROM 在庫管理.在庫表

            ORDER BY  倉庫番号  ASC,  在庫数量  DESC
              (C)        (1)    (A)      (2)    (B)
```

(1) 第1ソートキー

(2) 第2ソートキー

(1)～(A)および(2)～(B) ソート指定

(C)～(B) ORDER BY句

図7.5 ソート指定を複数指定した例

カーソル表

製品番号	製品名	在庫数量	倉庫番号
１２４	冷蔵庫	７５	１
１２３	冷蔵庫	６０	１
２２７	冷蔵庫	１５	１
２２６	冷蔵庫	８	１
３５１	ＣＤ	２５００	２
１４０	ＣＤプレーヤー	１２０	２
１１１	テレビ	９０	２
１１０	テレビ	８５	２
２４０	ＣＤプレーヤー	２５	２
２４３	ＣＤプレーヤー	１４	２
２１５	ビデオ	５	２
２１２	テレビ	０	２

第1ｿｰﾄｷｰ（昇順に並べ替えます）

第2ｿｰﾄｷｰ（第1ｿｰﾄｷｰの値が同じ行を降順に並べ替えます）

これまでは、カーソル表に含まれる列を使用して行の順序を並べ替える例について説明しましたが、カーソル表に含まれない列を使用してソートすることもできます。指定例を以下に示します。

例5

在庫表から、在庫数量が30未満の製品の製品番号と製品名を、在庫数量の値が大きい順にソートして取り出すカーソルを宣言します。

```
DECLARE 在庫数量カーソル CURSOR FOR

      SELECT 製品番号, 製品名 FROM 在庫管理.在庫表

        WHERE 在庫数量 < 30 ORDER BY 在庫数量 DESC
```

WHERE句で導出される表

製品番号	製品名	在庫数量	倉庫番号
２１２	テレビ	０	２
２１５	ビデオ	５	２
２２６	冷蔵庫	８	１
２２７	冷蔵庫	１５	１
２４０	ＣＤプレーヤー	２５	２
２４３	ＣＤプレーヤー	１４	２

⇩　　⇑ ソートキー（降順に並べ替えます）

ORDER BY句で導出される表

製品番号	製品名	在庫数量	倉庫番号
２４０	ＣＤプレーヤー	２５	２
２２７	冷蔵庫	１５	１
２４３	ＣＤプレーヤー	１４	２
２２６	冷蔵庫	８	１
２１５	ビデオ	５	２
２１２	テレビ	０	２

⇩

カーソル表

製品番号	製品名
２４０	ＣＤプレーヤー
２２７	冷蔵庫
２４３	ＣＤプレーヤー
２２６	冷蔵庫
２１５	ビデオ
２１２	テレビ

一方、問合せ指定の選択リストに演算式や、集合関数を指定した結果の列には名前がありません。名前のない列をソートキーにするには、ソート指定で列名の代わりに、列の番号を指定します。列の番号は、カーソル表の列を左から数えた順番です。指定例を以下に示します。

例6

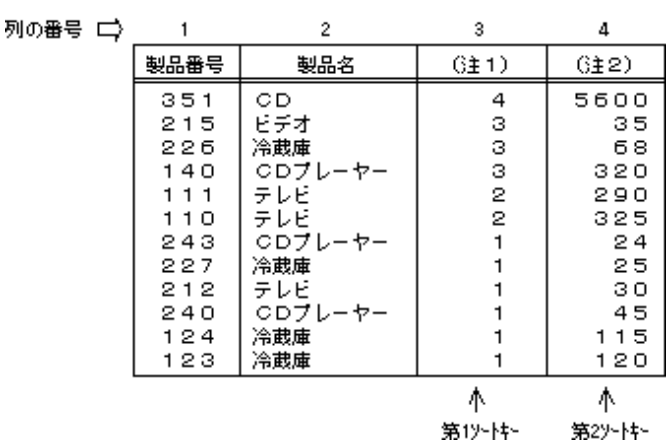
例2のカーソル宣言で、カーソルによってデータを取り出す順序を、第1ソートキーを取引先の数で降順に、第2ソートキーを在庫数量と発注数量の合計で昇順にソートします。

```
DECLARE 製品番号ごとに処理するカーソル CURSOR FOR

      SELECT 製品番号, 製品名, COUNT(取引先), 在庫数量 + SUM(発注数量)

          FROM 在庫管理.在庫表, 在庫管理.発注表

          WHERE 製品番号 = 取引製品

          GROUP BY 製品番号, 製品名, 在庫数量

           ORDER BY  3  DESC,    4  ASC
                       (1)       (2)
```

(1) 第1ソートキー

(2) 第2ソートキー

図7.6 ソート指定に列の番号を指定した例



注1) COUNT(取引先)の結果から作成される列です。

注2) 在庫数量 + SUM(発注数量)の結果から作成される列です。

備考. 演算式や集合関数の結果の列には名前がありません。

更新可能カーソルと読込み専用カーソル

カーソルには、更新可能カーソルと読込み専用カーソルの2種類があります。カーソルを使用するUPDATE文やDELETE文は更新可能カーソルだけが使用できます。FETCH文は両方が使用できます。カーソルがどちらになるかは、カーソル宣言の指定の内容によって決まります。

読込み専用カーソルとなるカーソル宣言の条件を以下に示します。この例で使用する表TBL1および表TBL2のスキーマ名はSCMとします。

読込み専用カーソルとなる カーソル宣言の条件	例
a) DISTINCTを指定している。	DECLARE CSR CURSOR FOR SELECT **DISTINCT** COLA, COLB FROM SCM.TBL1
b) 選択リストに演算式を指定している。	DECLARE CSR CURSOR FOR SELECT COLC, **COLA + COLB** FROM SCM.TBL1
c) 選択リストに集合関数を指定している。	DECLARE CSR CURSOR FOR SELECT **SUM(COLB)** FROM SCM.TBL1
d) 選択リストに同じ列を指定している。	DECLARE CSR CURSOR FOR SELECT **COLA**, COLB, **COLA**, COLC FROM SCM.TBL1
e) 選択リストに定数またはホスト変数を指定している。	DECLARE CSR CURSOR FOR SELECT COLA, **100** FROM SCM.TBL1
f) 選択リストにデータ列値関数を指定している。	DECLARE CSR CURSOR FOR SELECT **SUBSTRING(COL1 FROM 1 FOR 3)** FROM SCM.TBL1
g) 選択リストに数値関数を指定している。	DECLARE CSR CURSOR FOR SELECT **ABS(COL1)** FROM SCM.TBL1
h) 選択リストに日時値関数を指定している。	DECLARE CSR CURSOR FOR SELECT **TRUNC_DATE(COL1,'MONTH')** FROM SCM.TBL1
i) 選択リストにXMLQUERY関数を指定している。	DECLARE CSR CURSOR FOR SELECT **XMLQUERY('/a[b/text() = 1]' PASSING**

読込み専用カーソルとなる カーソル宣言の条件	例
	COLX) FROM SCM.TBL1
j) 選択リストにCASE式を指定している。	DECLARE CSR CURSOR FOR SELECT **CASE WHEN COL1=1 THEN 'ONE' END** FROM SCM.TBL1
k) 選択リストにCAST指定を指定している。	DECLARE CSR CURSOR FOR SELECT **CAST(COL1 AS CHAR(6))** FROM SCM.TBL1
l) 選択リストに連結演算子を指定している。	DECLARE CSR CURSOR FOR SELECT **COL1 || COL2** FROM SCM.TBL1
m) 選択リストにファンクションルーチンを指定している。	DECLARE CSR CURSOR FOR SELECT **USERFUNC1(C1,C2)** FROM SCM.TBL1
n) 選択リストに順序を指定している。	DECLARE CSR CURSOR FOR SELECT **SCM.順序1.NEXTVAL** FROM SCM.TBL1
o) 選択リストにROWNUMを指定している。	DECLARE CSR CURSOR FOR SELECT **ROWNUM**, COLA FROM SCM.TBL1
p) FROM句に2つ以上の表または結合表を指定している。	DECLARE CSR CURSOR FOR SELECT COLA, COLZ FROM **SCM.TBL1, SCM.TBL2**
q) FROM句に、ORDER BY句を指定した導出表を指定している。	DECLARE CSR CURSOR FOR SELECT DCOLA FROM (SELECT COLA FROM SCM.TBL1 **ORDER BY COLA**) AS D1(DCOLA)
r) FROM句にROWNUMを指定した導出表を指定している。	DECLARE CSR CURSOR FOR SELECT DCOLA FROM (SELECT COLA FROM SCM.TBL1 WHERE **ROWNUM** < 10) AS D1(DCOLA)
s) GROUP BY句またはHAVING句を指定している。	DECLARE CSR CURSOR FOR SELECT COLA FROM SCM.TBL1 **GROUP BY COLA**
t) WHERE句に副問合せを含んでいる。	DECLARE CSR CURSOR FOR SELECT * FROM SCM.TBL1 **WHERE COLA IN (SELECT COLZ FROM SCM.TBL2 WHERE COLY > 2000)**
u) UNIONを指定している。	DECLARE CSR CURSOR FOR SELECT COLC FROM SCM.TBL1 **UNION** SELECT COLX FROM SCM.TBL2
v) 更新可能性句にFOR READ ONLYを指定している。	DECLARE CSR CURSOR FOR SELECT COLA, COLB FROM SCM.TBL1 **FOR READ ONLY**
w) ORDER BY句を指定し、更新可能性句を指定していない。	DECLARE CSR CURSOR FOR SELECT COLA, COLB FROM SCM.TBL1 **ORDER BY COLA**
x) SCROLLを指定し、更新可能性句を指定していない。	DECLARE CSR **SCROLL** CURSOR FOR SELECT COLA, COLB FROM SCM.TBL1

カーソル宣言の形式が、a)からx)のいずれかに該当する場合、読込み専用カーソルになります。いずれにも当てはまらない場合は、更新可能カーソルになります。

a)からt)は、問合せ指定で指定する要素です。また、u)は問合せ式で指定する要素です。v)からx)は、カーソル宣言で指定する要素です。v)とw)は、更新可能性句にFOR UPDATEを指定したときのみ更新可能カーソルとなります。なお、t)の副問合せについては“9.2 いろいろな探索条件を指定する”を、u)のUNIONについては“8.5 複数の表から行の集合和を求める”をそれぞれ参照してください。

7.3 カーソルのオープンおよびクローズ

カーソルを使用してデータ操作を実行するには、最初にカーソルをオープンします。カーソルをオープンするには、OPEN文を使用します。そして、データ操作が終了するとカーソルをクローズします。カーソルをクローズするには、CLOSE文を使用します。

カーソルのオープン

OPEN文でカーソルの使用を開始します。カーソル宣言で指定したカーソル表は、OPEN文を実行した時点で作成されます。在庫表カーソルをオープンする例を以下に示します。

```
OPEN  在庫表カーソル
          (1)
```

(1) カーソル名

カーソル名で指定するカーソルの宣言は、アプリケーションの記述の順序でこのOPEN文よりも先に記述されていることが必要です。OPEN文で指定できるカーソルは、まだ一度もオープンされていないカーソル、またはCLOSE文によってクローズされたカーソルです。同じカーソルを二重にオープンするとエラーになります。異なるカーソルは同時にオープンすることができます。

カーソルのクローズ

CLOSE文でカーソルの使用を終了します。在庫表カーソルをクローズする例を以下に示します。

```
CLOSE  在庫表カーソル
          (1)
```

(1) カーソル名

カーソル名で指定するカーソルの宣言は、アプリケーションの記述の順序でこのCLOSE文よりも先に記述されていることが必要です。CLOSE文で指定するカーソルは、オープンされていることが必要です。オープンされていないカーソルを指定するとエラーになります。

7.4 カーソルの位置づけおよびデータの取出し

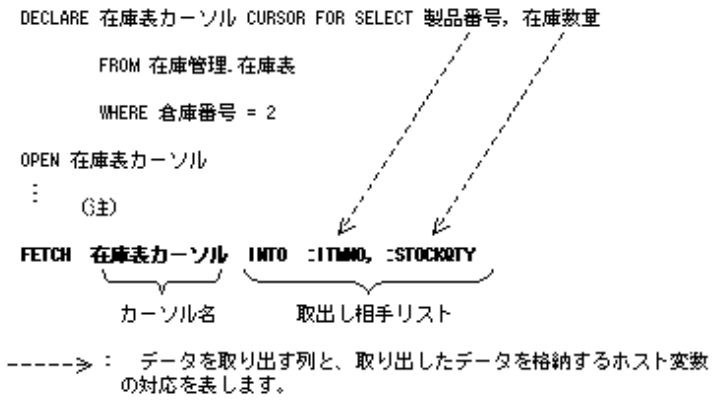
以下に示すデータ操作を行うためにFETCH文を使用します。

- カーソルの位置づけを行い、カーソルを位置づけた行からデータを取り出す。
- UPDATE文で行の更新を行うために、更新する行にカーソルを位置づける。
- DELETE文で行の削除を行うために、削除する行にカーソルを位置づける。

FETCH文の指定例を以下に示します。

例1

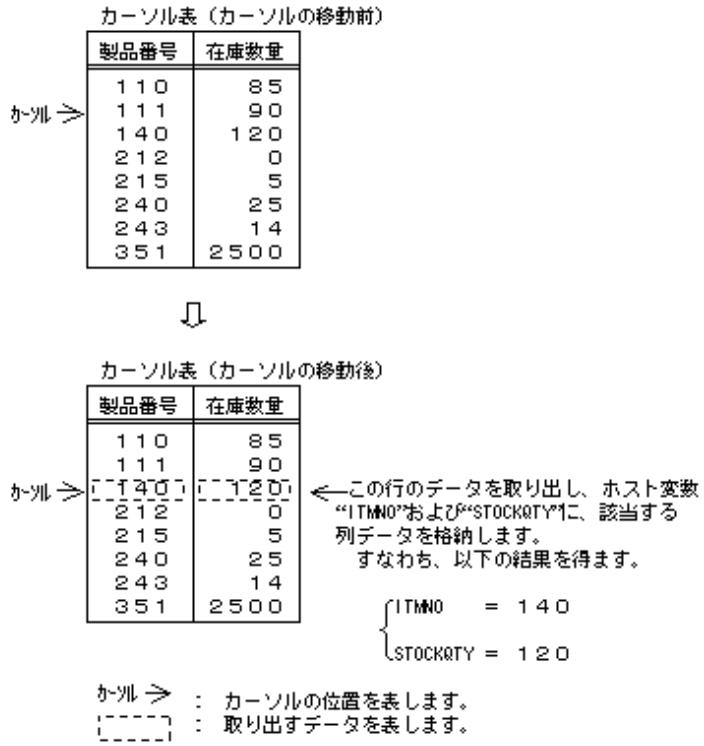
在庫表から、倉庫番号が“2”の製品の製品番号と在庫数量を取り出すカーソルが宣言されているとします。このカーソルを1行分移動し値を取り出します。



注) FETCH文は表から複数の行のデータを取り出すために繰り返し実行します。一般的には、この位置にホスト言語で繰返しを制御するための文を指定します。

例1のFETCH文を実行した結果を以下に示します。なお、すでに同じFETCH文が2回実行されており、カーソルは2行目を位置づけている状態とします。

図7.7 FETCH文によるカーソルの位置づけの例



カーソル名で指定するカーソルの宣言は、アプリケーションの記述の順序でこのFETCH文よりも先に記述されていることが必要です。FETCH文で指定するカーソルは、オープンされていることが必要です。オープンされていないカーソルを指定するとエラーになります。また、カーソルが最終行を位置づけている状態でFETCH文を実行すると、結果の状態は“データなし”になります。カーソルが最終行の直後を位置づけている状態で、FETCH文を実行した場合は“カーソル不当状態”になります。

取り出したデータを格納するホスト変数

取出し相手リストに、カーソルを位置づけた行の値を格納するホスト変数を指定します。ホスト変数は、カーソル表の列と同じ数だけ指定することが必要です。取り出した列のデータは、左から数えて同じ順番のホスト変数に格納されます。

NULL値を含むデータを取り出す

NULL値のデータを取り出すには、取り出したデータを格納するホスト変数に、標識変数を対にして指定します。指定例を以下に示します。

例2

例1のFETCH文に標識変数を指定した例です。

```
FETCH 在庫表カーソル INTO :ITMNO INDICATOR :ITMNO_INDICATOR ,
                           :STOCKQTY INDICATOR :STOCKQTY_INDICATOR
```

例2では、“ITMNO_INDICATOR”および“STOCKQTY_INDICATOR”が標識変数です。FETCH文で標識変数に格納される値は“表5.1 列の値と標識変数に格納される値”を参照してください。

標識変数を指定しないで、取り出すデータがNULL値になると、FETCH文はエラーになります。列の値にNULL値が含まれるかどうかが分からない場合には、必ず標識変数を指定してください。

いろいろな方向にカーソルを移動する

FETCH文の取出し方向の指定により、いろいろな方向にカーソルを移動することができます。取出し方向を指定するときは、カーソル宣言にSCROLLを指定します。取出し方向を省略した場合は、NEXTが指定されたものとみなします。

それぞれの指定例を以下に示します。b.からf.のカーソル宣言は、a.と同様とします。

a. 現在行の次の行の値を取り出します(NEXT指定)。カーソルは“図7.8 いろいろな方向にカーソルを移動する例”のカーソル[5]に位置づけられます。

例

```
DECLARE CSR1 SCROLL CURSOR FOR SELECT C1, C2 FROM S1.T1
      …
    OPEN  CSR1
      …
    FETCH  NEXT FROM CSR1 INTO :H1, :H2
             (1)
```

(1) 取出し方向

b. 現在行の前の行の値を取り出します(PRIOR指定)。カーソルは“図7.8 いろいろな方向にカーソルを移動する例”のカーソル[4]に位置づけられます。

例

```
FETCH PRIOR FROM CSR1 INTO :H1,:H2
```

c. 現在行に関係なく、先頭行の値を取り出します(FIRST指定)。カーソルは、“図7.8 いろいろな方向にカーソルを移動する例”のカーソル[1]に位置づけられます。

例

```
FETCH FIRST FROM CSR1 INTO :H1,:H2
```

d. 現在行に関係なく、最終行の値を取り出します(LAST指定)。カーソルは、“図7.8 いろいろな方向にカーソルを移動する例”のカーソル[8]に位置づけられます。

例

```
FETCH LAST FROM CSR1 INTO :H1,:H2
```

e. 現在行に関係なく、先頭行または最終行から数えた行番号の値を取り出します(ABSOLUTE指定)。

ー 先頭から2行目を取り出します。カーソルは、“図7.8 いろいろな方向にカーソルを移動する例”のカーソル[2]に位置づけられます。

例

```
FETCH ABSOLUTE 2 FROM CSR1 INTO :H1,:H2
```

ー 最終から2行目を取り出します。カーソルは、“図7.8 いろいろな方向にカーソルを移動する例”のカーソル[7]に位置づけられます。

例

```
FETCH ABSOLUTE -2 FROM CSR1 INTO :H1,:H2
```

f. 現在行から数えた行番号の値を取り出します(RELATIVE指定)。

ー 現在行の2行うしろを取り出します。カーソルは、“図7.8 いろいろな方向にカーソルを移動する例”のカーソル[6]に位置づけられます。

例

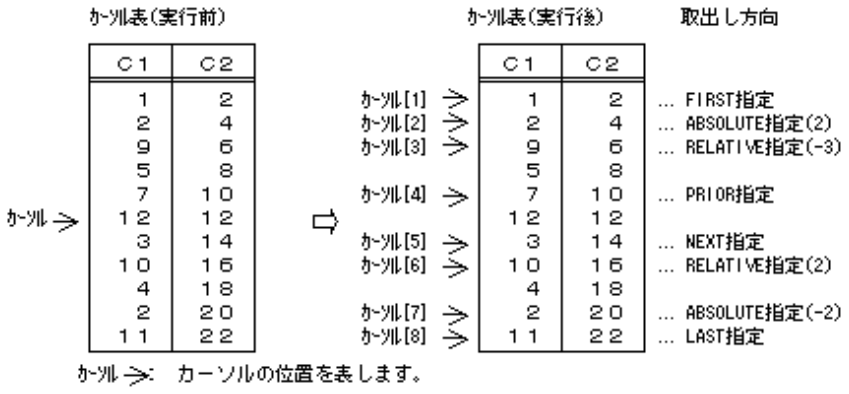
```
FETCH RELATIVE 2 FROM CSR1 INTO :H1,:H2
```

ー 現在行の3行前を取り出します。カーソルは、“図7.8 いろいろな方向にカーソルを移動する例”のカーソル[3]に位置づけられます。

例

```
FETCH RELATIVE -3 FROM CSR1 INTO :H1,:H2
```

図7.8 いろいろな方向にカーソルを移動する例



7.5 カーソルを使用してデータを更新する

カーソルを位置づけた行のデータを更新する場合は、UPDATE文を使用します。UPDATE文に使用するカーソルは更新可能カーソルであることが必要です。UPDATE文の指定例を以下に示します。

例1

“図7.7 FETCH文によるカーソルの位置づけの例”でカーソルを位置づけた行について、在庫数量を“150”に更新します。

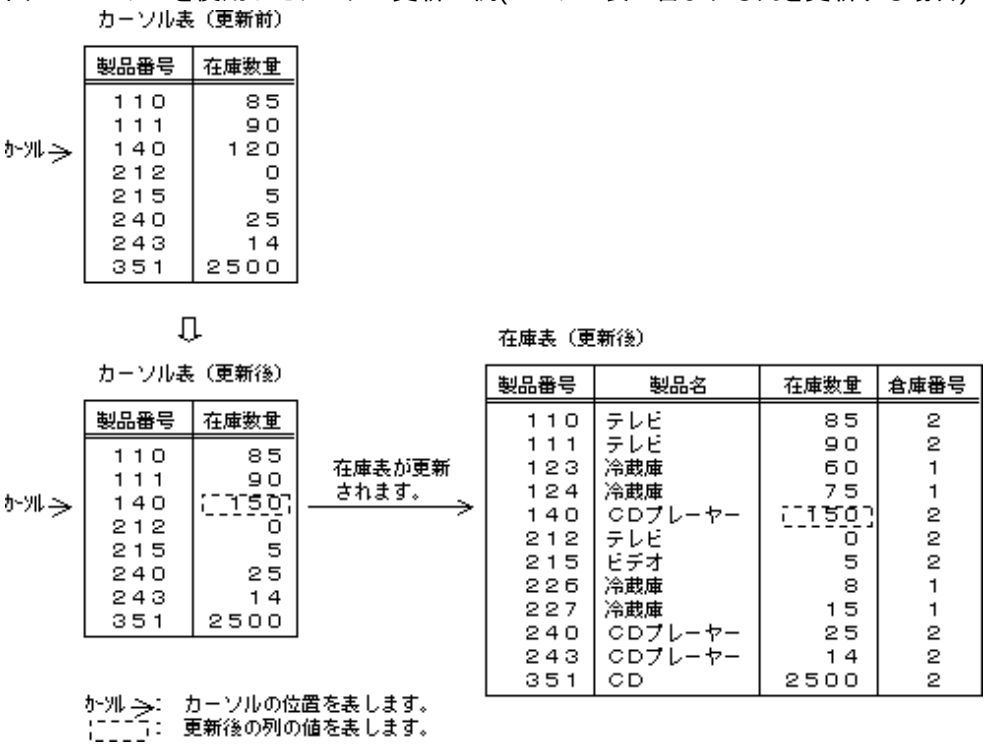
```
UPDATE  在庫管理.在庫表  SET  在庫数量 = 150
          (1)                 (2)

        WHERE CURRENT OF 在庫表カーソル
                           (3)
```

(1) 表名

(2) 設定句

(3) カーソル名

図7.9 カーソルを使用したデータの更新の例(カーソル表に含まれる列を更新する場合)



備考. 更新の前後でカーソルの位置は変化しません。

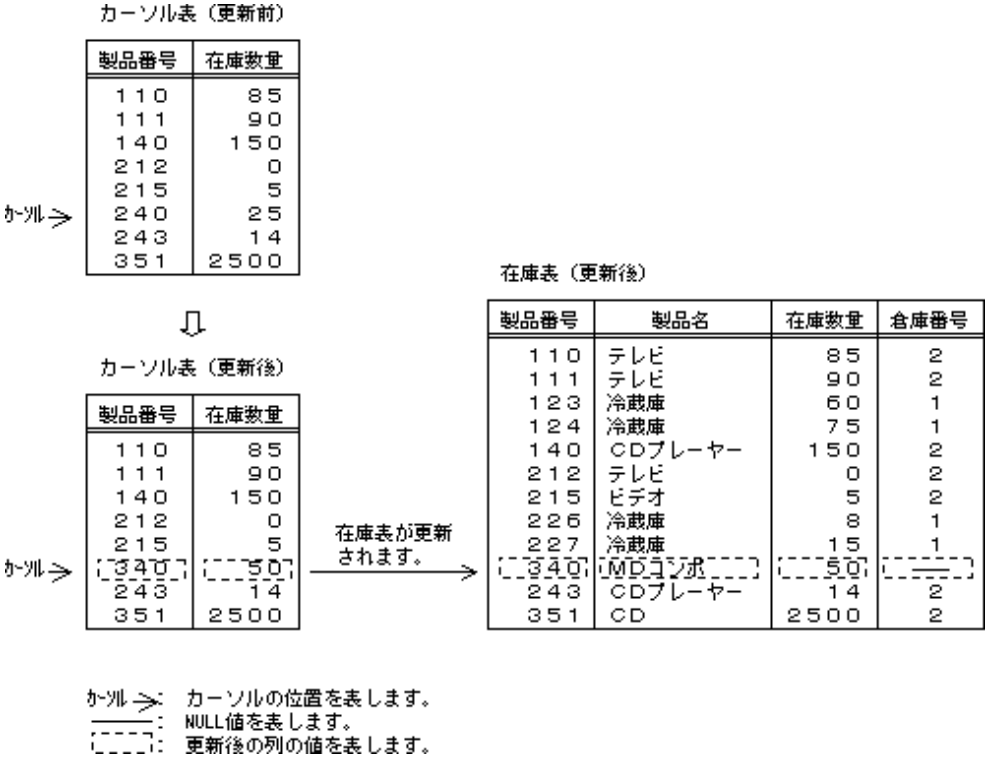
更新する列名および更新する値は、設定句で指定します。設定句の指定方法は、“6.2 データベースのデータを更新する”で説明したのと同じです。なお、更新する列はカーソル表に含まれない列でもかまいません。この例を以下に示します。

例2

“図7.7 FETCH文によるカーソルの位置づけの例”のカーソルをさらに移動した後、製品番号をホスト変数“ITMNO”の値に、製品名を“MDコンポ”に、在庫数量を元の値の2倍に、そして倉庫番号をNULL値に更新します。ここでは、ホスト変数“ITMNO”には“340”が設定されているとします。

```
UPDATE 在庫管理.在庫表
       SET 製品番号 = :ITMNO, 製品名 = N'ＭＤコンポ',
           在庫数量 = 在庫数量 * 2, 倉庫番号 = NULL
       WHERE CURRENT OF 在庫表カーソル
```

図7.10 カーソルを使用したデータの更新の例(カーソル表に含まれない列を更新する場合)



備考．更新の前後でカーソルの位置は変化しません。

UPDATE文によるデータの更新には、カーソルを使用しない方法もあります。その方法は、“6.2 データベースのデータを更新する”で説明しています。

なお、ここで説明したUPDATE文によるデータの更新は、カーソルの位置づけにより更新対象の行を特定することから、“UPDATE文:位置づけ”と呼びます。

7.6 カーソルを使用してデータを削除する

カーソルを位置づけた行のデータを削除する場合は、DELETE文を使用します。DELETE文に使用するカーソルは更新可能カーソルであることが必要です。DELETE文の指定例を以下に示します。

例

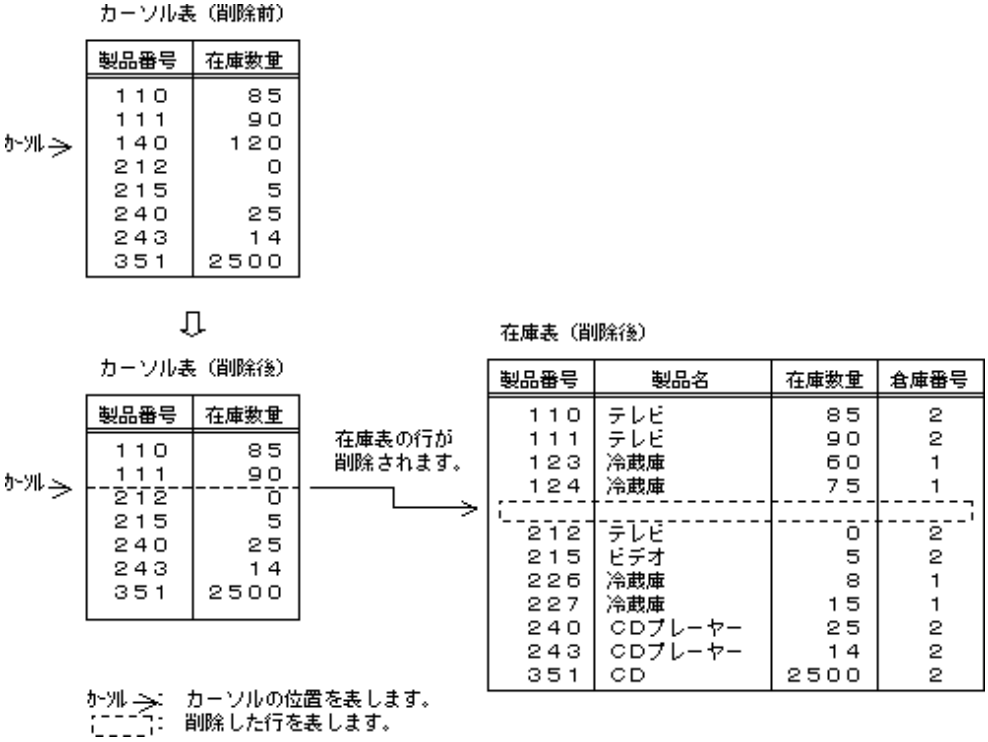
“図7.7 FETCH文によるカーソルの位置づけの例”でカーソルを位置づけた行を削除します。

```
DELETE FROM  在庫管理.在庫表  WHERE CURRENT OF 在庫表カーソル
                  (1)                          (2)
```

(1) 表名

(2) カーソル名

図7.11 カーソルを使用したデータの削除の例



備考．削除後のカーソルは行と行の間を位置づけます。

DELETE文によるデータの削除には、カーソルを使用しない方法もあります。その方法は、“6.3 データベースのデータを削除する”で説明しています。

なお、ここで説明したDELETE文によるデータの削除は、カーソルの位置づけにより削除対象の行を特定することから、“DELETE文:位置づけ”と呼びます。

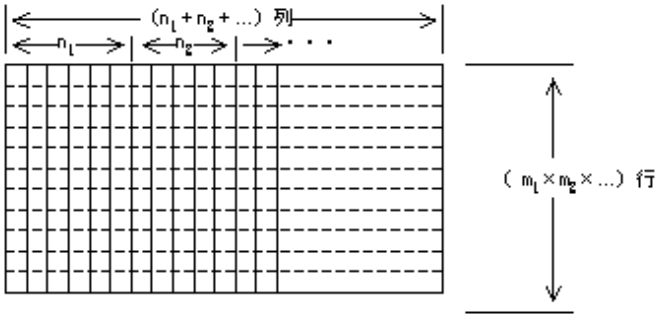
第8章 複数の表を関連づけて操作する方法

本章では、データベースのデータを取り出すときに、複数の表のデータを関連づけて操作する方法を説明します。

8.1 複数の表から新たな表を導出する

複数の表を関連づけてデータを取り出すには、表式のFROM句に表名をコンマ(,)で区切って並べて指定します。複数の表名を指定した場合、FROM句の結果で新たな表が導出されます。この導出された表からデータの取り出しが行われます。複数の表名を指定した場合に、どのような表が導出されるかを以下に示します。

FROM句に、m1行n1列の表、m2行n2列の表、…を指定した場合に導出される表は以下の図のように(m1×m2×…)行(n1＋n2＋…)列の表になります。



複数の表からデータを取り出す例を以下に示します。

例

FROM句にTBL1とTBL2の複数の表名を指定します。これらの表のスキーマ名はSCMとします。

```
SELECT  …  FROM SCM.TBL1, SCM.TBL2
```

図8.1 FROM句に2つの表名を指定した場合に導出される表

TBL1

COLA	COLB	COLC
1	40	A
2	——	B
3	40	——
2	20	C

TBL2

COLX	COLY	COLZ
A	2500	1
A	1000	2

FROM句で導出される表

COLA	COLB	COLC	COLX	COLY	COLZ
1	40	A	A	2500	1
1	40	A	A	1000	2
2	——	B	A	2500	1
2	——	B	A	1000	2
3	40	——	A	2500	1
3	40	——	A	1000	2
2	20	C	A	2500	1
2	20	C	A	1000	2

—— ：NULL値を表します。

FROM句に2つの表名を指定した場合、FROM句の結果により導出される表は、2つの表の列と行のすべてを組み合わせた結果となります。これを表の“拡張直積”と呼びます。FROM句に3つ以上の表名を指定した場合にも同じで

す。すなわち、結果の列数は、指定した表の列数の総和となります。行数は、指定した表の行数の積となります。列の順序は、表の指定順で表中の列の順序となります。

8.2 表を関連づける条件を指定する

FROM句に複数の表を指定して導出される拡張直積は、それぞれの表のすべての行を組み合わせたにすぎません。アプリケーションの処理では、拡張直積の結果のほとんどの行が無意味なデータの集まりになってしまいます。アプリケーションでは、拡張直積の結果からすべての行を取り出して、必要な行だけを処理することは可能ですが、多くの行を取り出すために処理効率が悪くなります。そこで、表を関連づけるうえで意味のあるデータだけを絞り込むことが必要になります。それには、表式のWHERE句に表を関連づけるための条件を指定します。指定例を以下に示します。

例1

“図8.1 FROM句に2つの表名を指定した場合に導出される表”の例でTBL1とTBL2の2つの表を関連づけるデータが、列COLAと列COLZの値とした場合の例です。

```
SELECT  …  FROM SCM.TBL1, SCM.TBL2  WHERE COLA = COLZ
```

なお、INNER JOINを用いて以下のように等価に指定することも可能です。

```
SELECT  …  FROM SCM.TBL1 INNER JOIN SCM.TBL2  ON COLA = COLZ
```

図8.2 WHERE句に表を関連づけるための条件を指定した例

FROM句で導出される表

COLA	COLB	COLC	COLX	COLY	COLZ	
1	40	A	A	2500	1	← COLA = COLZ が真になる行
1	40	A	A	1000	2	
2	——	B	A	2500	1	
2	——	B	A	1000	2	← COLA = COLZ が真になる行
3	40	——	A	2500	1	
3	40	——	A	1000	2	
2	20	C	A	2500	1	
2	20	C	A	1000	2	← COLA = COLZ が真になる行

⇩

WHERE句で導出される表

COLA	COLB	COLC	COLX	COLY	COLZ
1	40	A	A	2500	1
2	——	B	A	1000	2
2	20	C	A	1000	2

—— ：NULL値を表します。

例1では、列COLAと列COLZの値が同じ行を取り出すために、WHERE句に“COLA = COLZ”の条件を指定しています。

なお、FROM句に複数の表を指定した場合、これらの表の中で同じ名前の列が存在することがあります。そのような列を指定するには、列名を表名とピリオド(.)で修飾することが必要です。例えば、列名のCOLAが表TBL1と表TBL2の両方に存在する場合に、表TBL1の列COLAを指定するには、“TBL1.COLA”または“SCM.TBL1.COLA”と指定します。同様に、表TBL2の列COLAを指定するには、“TBL2.COLA”または“SCM.TBL2.COLA”と指定します。

複数の表を関連づける具体例として、本章の最初に述べた発注状況を調べる例を以下に示します。

例2

発注表と在庫表から、取引先、製品名、および発注数量を取り出します。この例では、列名には表名の修飾を付けて指定します。実際には、発注表と在庫表に同じ名前の列はないので、表名による修飾は指定しなくてもかまいません。なお、FROM句に指定する複数の表の間で同じ列名がある場合には、表名による修飾が必要です。

```
SELECT 発注表.取引先, 在庫表.製品名, 発注表.発注数量

        FROM 在庫管理.発注表, 在庫管理.在庫表
```

```
        WHERE 発注表.取引製品 = 在庫表.製品番号
```

なお、INNER JOINを用いて以下のように等価に指定することも可能です。

```
SELECT 発注表.取引先, 在庫表.製品名, 発注表.発注数量

        FROM 在庫管理.発注表 INNER JOIN 在庫管理.在庫表

        ON 発注表.取引製品 = 在庫表.製品番号
```

図8.3 複数の表を関連づけて操作する例

発注表

取引先	取引製品	仕入価格	発注数量
61	123	48000	60
61	124	64000	40
61	140	8000	80
61	215	240000	10
61	240	80000	20
62	110	37500	120
62	226	112500	20
62	351	375	800
63	111	57400	80
63	212	205000	30
63	215	246000	10
71	140	7800	50
71	351	390	600
72	140	7000	70
72	215	210000	10
72	226	105000	20
72	243	84000	10
72	351	350	1000
74	110	39000	120
74	111	54000	120
74	226	117000	20
74	227	140400	10
74	351	390	700

在庫表

製品番号	製品名	在庫数量	倉庫番号
110	テレビ	85	2
111	テレビ	90	2
123	冷蔵庫	60	1
124	冷蔵庫	75	1
140	CDプレーヤー	120	2
212	テレビ	0	2
215	ビデオ	5	2
226	冷蔵庫	8	1
227	冷蔵庫	15	1
240	CDプレーヤー	25	2
243	CDプレーヤー	14	2
351	CD	2500	2

⇩

FROM句で導出される表

取引先	取引製品	仕入価格	発注数量	製品番号	製品名	在庫数量	倉庫番号
61	123	48000	60	110	テレビ	85	2
61	123	48000	60	111	テレビ	90	2
61	123	48000	60	123	冷蔵庫	60	1
・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・

(FROM句で導出される表は、発注表と在庫表のすべての行の組み合わせからなる276行です)

⇩

WHERE句で導出される表

取引先	取引製品	仕入価格	発注数量	製品番号	製品名	在庫数量	倉庫番号
61	123	48000	60	123	冷蔵庫	60	1
61	124	64000	40	124	冷蔵庫	75	1
61	140	8000	80	140	CDプレーヤー	120	2
・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・

（WHERE句で導出される表は、FROM句の結果の表の中で取引製品と製品番号が等しい23行です）

⇩

問合せ指定で導出される表

取引先	製品名	発注数量
61	冷蔵庫	60
61	冷蔵庫	40
61	CDプレーヤー	80
61	ビデオ	10
61	CDプレーヤー	20
62	テレビ	120
62	冷蔵庫	20
62	CD	800
63	テレビ	80
63	テレビ	30
63	ビデオ	10
71	CDプレーヤー	50
71	CD	600
72	CDプレーヤー	70
72	ビデオ	10
72	冷蔵庫	20
72	CDプレーヤー	10
72	CD	1000
74	テレビ	120
74	テレビ	120
74	冷蔵庫	20
74	冷蔵庫	10
74	CD	700

結合表

例2では、発注表あるいは在庫表に存在する行は、WHERE句で導出される表の中にすべて出現していました。しかし、条件によっては出現しない行もできてしまいます。条件を満たさない行も取り出したいような場合には、結合表を利用します。結合表は、結合型で指定した側の表中のうち、結合条件の探索条件を満たさない行も、存在すればすべて取り出すことができます。外結合型にLEFTを記述した場合は結合型の左側の表の全行を取り出し、RIGHTを記述した場合は右側の表の全行を取り出します。指定例を以下に示します。

例3

会社表と日別発注表から、会社番号と取引先が等しいデータについて、会社名、取引製品、および発注数量を取り出します。このとき、会社番号と取引先が等しくない行でも日別発注表にある行はすべて取り出します。

```
SELECT 会社表.会社名, 日別発注表.取引製品, 日別発注表.発注数量

    FROM  会社表  RIGHT  OUTER  JOIN  日別発注表
            (1)    (A)    (2)          (1)
       ON  会社表.会社番号 = 日別発注表.取引先
       (B)                  (3)
```

(1) 表参照

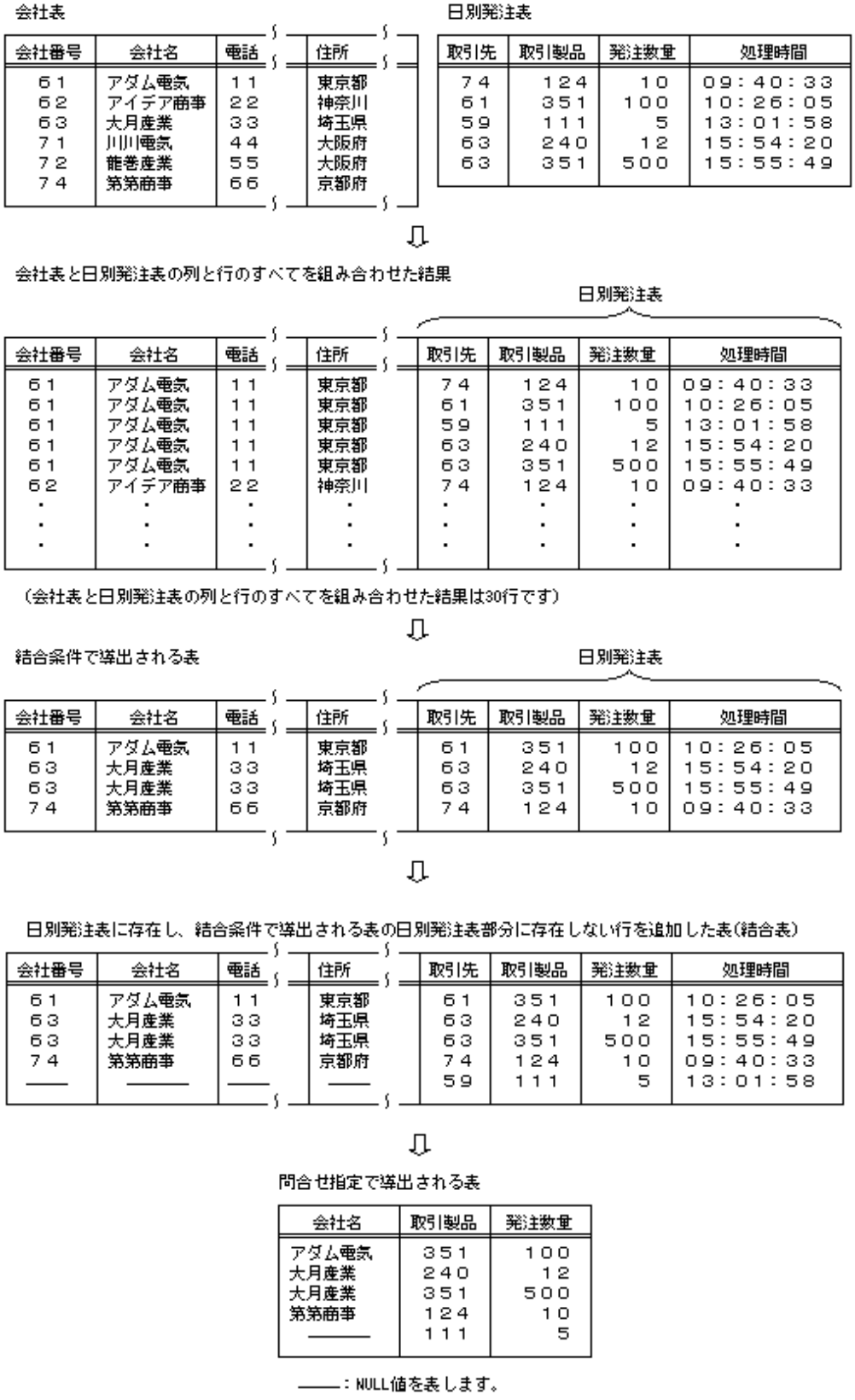
(2) 外結合型

(A)～(2) 結合型

(3) 探索条件

(B)～(3) 結合条件

図8.4 結合表を利用して複数の表を関連づける例



例3では、外結合型にRIGHTを記述した場合の例を示しましたが、同じ例文で外結合型にLEFTを記述した場合は以下のような結果になります。

```
SELECT 会社表.会社名, 日別発注表.取引製品, 日別発注表.発注数量
```

```
    FROM  会社表  LEFT  OUTER  JOIN  日別発注表

          ON  会社表.会社番号 = 日別発注表.取引先
```

会社名	取引製品	発注数量
アダム電気	351	100
大月産業	240	12
大月産業	351	500
第第商事	124	10
アイデア商事	——	——
川川電気	——	——
龍巻産業	——	——

——：NULL値を表します。

8.3 表に別名を用いてデータを操作する

列名を表名で修飾して指定するときに、表名のつづりが長いとそれぞれの列の記述が大変です。表式の指定では、表に別の名前を付けてデータ操作ができます。この名前を“相関名”と呼びます。相関名の指定例を以下に示します。

例

“図8.3 複数の表を関連づけて操作する例”の問合せ指定で、列名の修飾に相関名を使用した例です。この問合せ指定の結果は“図8.3 複数の表を関連づけて操作する例と同じです。

```
SELECT T1.取引先, T2.製品名, T1.発注数量

       FROM  在庫管理.発注表  AS  T1,  在庫管理.在庫表  AS  T2
                   (1)          (2)          (1)          (2)

       WHERE T1.取引製品 = T2.製品番号
```

(1) 表名

(2) 相関名

この例では、発注表に“T1”という相関名、在庫表に“T2”という相関名を指定しています。そして、列名の修飾には、それぞれの相関名を使用しています。相関名は、指定したSQL文の中でだけ有効です。また、表名に相関名を指定した場合、列名の修飾には相関名を使用することが必要です。列名の修飾に元の表名は使用できません。

8.4 同一表を関連づけて操作する

表のそれぞれの行について、特定の列とほかの列の値を比較したり、演算する方法はすでに説明しました。一方、1つの表の特定の行とほかの行で値を比較したり、演算することはできません。そこで、表の関連づけを利用して、1つの表をあたかも異なる表のようにして扱うことにより、これらの操作を行う方法を説明します。

そのためには、FROM句に同じ表名を指定します。そして、SQL文中でそれらの表を区別するための相関名を必ず指定します。

異なる行間での比較

1つの表の特定の行とほかの行で値を比較する例を以下に示します。

例1

在庫表から、製品名が同じである製品番号の対を求めます。取り出す列は、対になるそれぞれの製品番号とその製品名とします。この操作を行うために、FROM句に在庫表を2つ指定して、同じ列が2列ずつある表を導き出します。これらの同じ列を区別するために、一方にはT1、他方にはT2という相関名を付けます。

```
SELECT T1.製品番号, T2.製品番号, T1.製品名

       FROM 在庫管理.在庫表 AS T1,  在庫管理.在庫表 AS T2
```

```
WHERE T1.製品名 = T2.製品名 AND T1.製品番号 < T2.製品番号
```

図8.5 同じ表同士を関連づけて操作する例

FROM句で導出される表

	T1			T2			
製品番号	製品名	在庫数量	倉庫番号	製品番号	製品名	在庫数量	倉庫番号
110	テレビ	85	2	110	テレビ	85	2
110	テレビ	85	2	111	テレビ	90	2
110	テレビ	85	2	123	冷蔵庫	60	1
110	テレビ	85	2	124	冷蔵庫	75	1
110	テレビ	85	2	140	CDプレーヤー	120	2
110	テレビ	85	2	212	テレビ	0	2
110	テレビ	85	2	215	ビデオ	5	2
110	テレビ	85	2	226	冷蔵庫	8	1
110	テレビ	85	2	227	冷蔵庫	15	1
110	テレビ	85	2	240	CDプレーヤー	25	2
110	テレビ	85	2	243	CDプレーヤー	14	2
110	テレビ	85	2	351	CD	2500	2
111	テレビ	90	2	110	テレビ	85	2
111	テレビ	90	2	111	テレビ	90	2
111	テレビ	90	2	123	冷蔵庫	60	1
111	テレビ	90	2	124	冷蔵庫	75	1
111	テレビ	90	2	140	CDプレーヤー	120	2
111	テレビ	90	2	212	テレビ	0	2
111	テレビ	90	2	215	ビデオ	5	2
111	テレビ	90	2	226	冷蔵庫	8	1
111	テレビ	90	2	227	冷蔵庫	15	1
111	テレビ	90	2	240	CDプレーヤー	25	2
111	テレビ	90	2	243	CDプレーヤー	14	2
111	テレビ	90	2	351	CD	2500	2
123	冷蔵庫	60	1	110	テレビ	85	2
123	冷蔵庫	60	1	111	テレビ	90	2
123	冷蔵庫	60	1	123	冷蔵庫	60	1
123	冷蔵庫	60	1	124	冷蔵庫	75	1
123	冷蔵庫	60	1	140	CDプレーヤー	120	2
123	冷蔵庫	60	1	212	テレビ	0	2
123	冷蔵庫	60	1	215	ビデオ	5	2
123	冷蔵庫	60	1	226	冷蔵庫	8	1
123	冷蔵庫	60	1	227	冷蔵庫	15	1
123	冷蔵庫	60	1	240	CDプレーヤー	25	2
123	冷蔵庫	60	1	243	CDプレーヤー	14	2
123	冷蔵庫	60	1	351	CD	2500	2
～	～	～	～	～	～	～	～
351	CD	2500	2	227	冷蔵庫	15	1
351	CD	2500	2	240	CDプレーヤー	25	2
351	CD	2500	2	243	CDプレーヤー	14	2
351	CD	2500	2	351	CD	2500	2

⇩

WHERE句で導出される表

製品番号	製品名	在庫数量	倉庫番号	製品番号	製品名	在庫数量	倉庫番号
１１０	テレビ	８５	２	１１１	テレビ	９０	２
１１０	テレビ	８５	２	２１２	テレビ	０	２
１１１	テレビ	９０	２	２１２	テレビ	０	２
１２３	冷蔵庫	６０	１	１２４	冷蔵庫	７５	１
１２３	冷蔵庫	６０	１	２２６	冷蔵庫	８	１
１２３	冷蔵庫	６０	１	２２７	冷蔵庫	１５	１
１２４	冷蔵庫	７５	１	２２６	冷蔵庫	８	１
１２４	冷蔵庫	７５	１	２２７	冷蔵庫	１５	１
１４０	ＣＤプレーヤー	１２０	２	２４０	ＣＤプレーヤー	２５	２
１４０	ＣＤプレーヤー	１２０	２	２４３	ＣＤプレーヤー	１４	２
２２６	冷蔵庫	８	１	２２７	冷蔵庫	１５	１
２４０	ＣＤプレーヤー	２５	２	２４３	ＣＤプレーヤー	１４	２

⇩

問合せ指定で導出される表

製品番号	製品番号	製品名
１１０	１１１	テレビ
１１０	２１２	テレビ
１１１	２１２	テレビ
１２３	１２４	冷蔵庫
１２３	２２６	冷蔵庫
１２３	２２７	冷蔵庫
１２４	２２６	冷蔵庫
１２４	２２７	冷蔵庫
１４０	２４０	ＣＤプレーヤー
１４０	２４３	ＣＤプレーヤー
２２６	２２７	冷蔵庫
２４０	２４３	ＣＤプレーヤー

異なる行間での演算

1つの表の特定の行とほかの行の値で演算を行う例を以下に示します。

例2

在庫表で、1つ違いの製品番号を持つ製品の間で、在庫数量の差を求めます。なお、取り出すデータは、両方の製品番号、製品番号が大きい方の製品名、在庫数量の差とします。

```
SELECT T2.製品番号, T1.製品番号, T2.製品名, T2.在庫数量 - T1.在庫数量  AS 在庫数量差

       FROM 在庫管理.在庫表 AS T1, 在庫管理.在庫表 AS T2

       WHERE T2.製品番号 = T1.製品番号 + 1
```

図8.6 異なる行の間で演算を行う例

FROM句で導出される表は“図：同じ表同士を関連づけて操作する例”と同じです。

⇩

WHERE句で導出される表

製品番号	製品名	在庫数量	倉庫番号	製品番号	製品名	在庫数量	倉庫番号
１１０	テレビ	８５	２	１１１	テレビ	９０	２
１２３	冷蔵庫	６０	１	１２４	冷蔵庫	７５	１
２２６	冷蔵庫	８	１	２２７	冷蔵庫	１５	１

⇩

問合せ指定で導出される表

製品番号	製品番号	製品名	在庫数量差
１１１	１１０	テレビ	５
１２４	１２３	冷蔵庫	１５
２２７	２２６	冷蔵庫	７

8.5 複数の表から行の集合和を求める

複数の表に含まれる行の間で集合和を求めることができます。集合和を求めるには、2つ以上の問合せ指定を“UNION”で結合して指定します。それぞれの問合せ指定には、処理対象の表およびデータを取り出す条件を指定します。集合和を求めるときに、それぞれの列同士で値が同じ行を1行にすることができます。あるいは、元の行数のままにすることもできます。

集合和で重複する行を1行にする

問合せ指定をUNIONで結合した結果、それぞれの問合せ指定の結果の行が集められて表が導出されます。このとき、重複する行は1行になります。重複する行とは、複数の行の間で各列の値が同じ行のことです。問合せ指定をUNIONで結合した結果、どのような表が導出されるかを“図8.7 問合せ式にUNIONを指定した例”に示します。

例1

“図8.7 問合せ式にUNIONを指定した例”で示す表TBL1の列COLA、COLCと、表TBL2の列COLZ、COLXのそれぞれの行の間で集合和を求めます。表TBL1および表TBL2はスキーマSCMに含まれます。

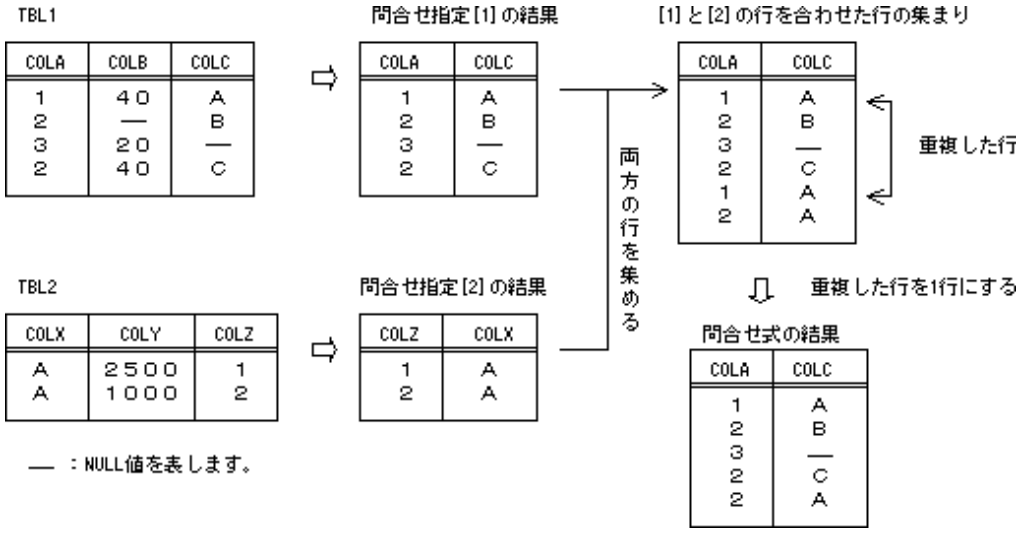
```
SELECT COLA, COLC FROM SCM.TBL1 UNION SELECT COLZ, COLX FROM SCM.TBL2
                  (1)                                 (2)
```

(1) 問合せ指定[1]

(2) 問合せ指定[2]

(1)～(2) 問合せ式

図8.7 問合せ式にUNIONを指定した例



問合せ指定をUNIONで結合すると、両方の問合せ指定の結果の列が左から順に対応し新たな表が導出されます。それぞれの問合せ指定は、以下のように指定することが必要です。

－ 問合せ指定同士で、結果の列の数は同じであることが必要です。

－ それぞれの対応する結果の列のデータ型は、比較可能であることが必要です。

比較可能なデータ型については、“表9.5 比較可能なデータ型”を参照してください。

なお、問合せ式の結果の表の列名は、左側の問合せ指定で指定した列名となります。ただし、左側の問合せ指定の選択リストが演算式や集合関数の場合は、結果の表の列は名前なしとなります。

集合和で重複する行もそのまま含める

重複する行を1行にせず、それぞれの問合せ指定の結果のすべての行から表を導出するには、UNIONとともにALLを指定します。どのような表が導出されるかを“図8.8 問合せ式にUNION ALLを指定した例”に示します。

例2

“図8.8 問合せ式にUNION ALLを指定した例”で示す表TBL1の列COLA、COLCと、表TBL2の列COLZ、COLXのそれぞれの行の間で集合和を求めます。なお、重複する行もすべて含みます。

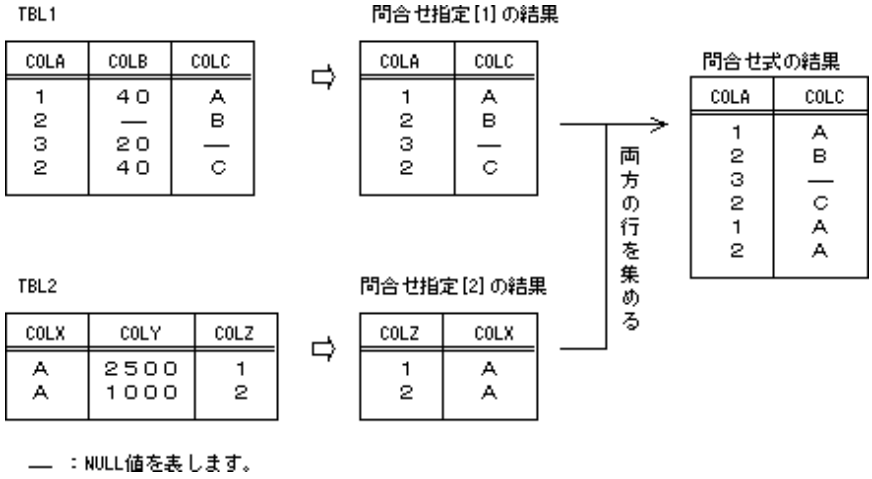
```
SELECT COLA,COLC FROM SCM.TBL1 UNION ALL SELECT COLZ,COLX FROM SCM.TBL2
                 (1)                                  (2)
```

(1) 問合せ指定[1]

(2) 問合せ指定[2]

(1)～(2) 問合せ式

図8.8 問合せ式にUNION ALLを指定した例



TBL1

COLA	COLB	COLC
1	40	A
2	—	B
3	20	—
2	40	C

問合せ指定[1]の結果

COLA	COLC
1	A
2	B
3	—
2	C

問合せ式の結果

COLA	COLC
1	A
2	B
3	—
2	C
1	A
2	A

TBL2

COLX	COLY	COLZ
A	2500	1
A	1000	2

問合せ指定[2]の結果

COLZ	COLX
1	A
2	A

UNIONによる集合和の順序

UNIONの結合は、左から順に処理されます。なお、括弧(“(”および“)”)を使用して評価順序を指定することができます。括弧の中の問合せ式は、ほかに先立って処理されます。括弧を使用した例を以下に示します。

例3

表TBL1、表TBL2、表TBL3および表TBL4の集合和を求めます。まず表TBL1と表TBL2で重複する行を取り除いて求めます([1])。次に表TBL3と表TBL4で重複する行を取り除いて求めます([2])。最後にそれらの結果同士で重複する行を含めて求めます([3])。

```
(SELECT COLA FROM SCM.TBL1 UNION SELECT COLZ FROM SCM.TBL2)
                           [1]
  UNION ALL
     [3]
  (SELECT COL1 FROM SCM.TBL3 UNION SELECT COL1 FROM SCM.TBL4)
                             [2]
```

第9章 いろいろなデータ操作の方法

本章では、SQL文によるいろいろなバリエーションのデータ操作の方法を具体的に説明します。

9.1 表をグループ化してデータを操作する

表からデータを取り出すときに、列の値によって表をグループ化してデータを操作することができます。表のグループ化とは、列または列の組の値が同じ行同士を1グループに、異なる行を別のグループになるようにグループ分けすることをいいます。このようにグループ分けした表を“グループ表”と呼びます。

表をグループ化して操作するには、GROUP BY句およびHAVING句を使用します。GROUP BY句とHAVING句は、表式の中に指定できます。表式に、FROM句、WHERE句、GROUP BY句、およびHAVING句が同時に指定された場合、処理の順序は以下のようになります。

1. FROM句
2. WHERE句
3. GROUP BY句
4. HAVING句

まず、FROM句で指定した表についてWHERE句の探索条件で行を特定します。次に、特定された行をGROUP BY句によってグループ化します。最後に、HAVING句の探索条件によりグループを特定します。

9.1.1 表をグループ化してデータを取り出す

GROUP BY句を指定して、表をグループ化します。

表をグループ化する

最初に、表をグループ化する方法について説明します。

例1

在庫表を製品名の値でグループ化します。

```
SELECT  …  FROM 在庫管理.在庫表  GROUP BY  製品名
             (B)                   (A)      (1)
```

(1) グループ化列

(A)～(1) GROUP BY句

(B)～(1) 表式

GROUP BY句によって導出される表を以下に示します。

図9.1 GROUP BY句で導出される表の例

在庫表

製品番号	製品名	在庫数量	倉庫番号
110	テレビ	85	2
111	テレビ	90	2
123	冷蔵庫	60	1
124	冷蔵庫	75	1
140	CDプレーヤー	120	2
212	テレビ	0	2
215	ビデオ	5	2
226	冷蔵庫	8	1
227	冷蔵庫	15	1
240	CDプレーヤー	25	2
243	CDプレーヤー	14	2
351	CD	2500	2

⇩

GROUP BY句で導出される表

製品番号	製品名	在庫数量	倉庫番号	
110	テレビ	85	2	1グループ
111	テレビ	90	2	
212	テレビ	0	2	
215	ビデオ	5	2	1グループ
123	冷蔵庫	60	1	1グループ
124	冷蔵庫	75	1	
226	冷蔵庫	8	1	
227	冷蔵庫	15	1	
140	CDプレーヤー	120	2	1グループ
240	CDプレーヤー	25	2	
243	CDプレーヤー	14	2	
351	CD	2500	2	1グループ

表式の中のFROM句およびWHERE句によって導出された行の集まりをグループ化します。この例では、表式のFROM句には在庫表だけを指定しているので、FROM句の結果は在庫表と同じです。また、WHERE句は省略しているので、すべての行がグループ化の対象になります。

GROUP BY句

GROUP BY句にはグループ化のキーとする列名、またはデータ列値関数や数値関数を指定します。列名を指定した場合、その列を“グループ化列”、データ列値関数または数値関数を指定した場合、その関数を“グループ化関数”と呼びます。例1でグループ化列は製品名です。GROUP BY句の結果により導出される表は、グループ化列の値が同じ行同士を1つのグループとしたグループ表です。また、複数の列をグループ化列とする場合は、列名をコンマ(,)で区切って指定します。この場合、すべてのグループ化列の値が等しい行同士が1つのグループとなります。

グループ化した表からデータを取り出す

問合せ指定では、データを操作するための表を導出します。問合せ指定でグループ表から導出する表は、それぞれのグループを1行とした表です。GROUP BY句を指定した問合せ指定では、選択リストに指定できるのは、グループ化列、グループ化関数、および集合関数です。集合関数を使用することにより、それぞれのグループ内で、列の値の合計、平均、最大値、最小値、および行の数を求めることができます。グループ化列でない列を集合関数以外で指定するとエラーになります。例えば、例1の問合せ指定で、選択リストに在庫数量を指定したとき、製品名が“テレビ”の在庫数量として“85”、“90”および“0”のうちどの値とするかが決定できないからです。一方、これらの値の合計や平均は1つの値として決定できるので、集合関数は指定できます。また、“図9.1 GROUP BY句で導出される表の例”では倉庫番号の値は、それぞれのグループ内で一意となっていますが、この列についても選択リストに直接指定することはできません。データの変動によって、いつ一意でない値が発生するかわからないからです。“図9.1 GROUP BY句で導出される表の例”のデータでは、グループ化列に製品名と倉庫番号の2つの列を指定しておけば、倉庫番号の値を取り出すことができます。なお、選択リストに定数やホスト変数を指定することは可能です。

例2

例1で示したグループ表から、製品名と在庫数量の合計を求めます。

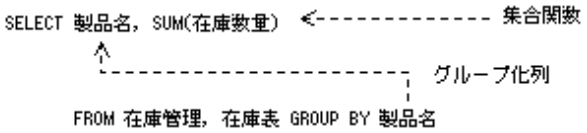


図9.2 グループ化した表からデータを取り出す例

GROUP BY句で導出される表

製品番号	製品名	在庫数量	倉庫番号	
110	テレビ	85	2	1グループ
111	テレビ	90	2	
212	テレビ	0	2	
215	ビデオ	5	2	1グループ
123	冷蔵庫	60	1	1グループ
124	冷蔵庫	75	1	
226	冷蔵庫	8	1	
227	冷蔵庫	15	1	
140	CDプレーヤー	120	2	1グループ
240	CDプレーヤー	25	2	
243	CDプレーヤー	14	2	
351	CD	2500	2	1グループ

⇩

問合せ指定で導出される表

製品名	(注)
テレビ	175
ビデオ	5
冷蔵庫	158
CDプレーヤー	159
CD	2500

注)集合関数の結果の列には名前がありません。

“図9.1 GROUP BY句で導出される表の例”から、選択リストにグループ化列だけを指定した結果が、問合せ指定で“DISTINCT”を指定したのと同じであることに気が付かれたと思います。例2で選択リストに製品名だけを指定すれば、冗長な行を取り除いた製品名を求めることができます。しかし、冗長な行を取り除くのが目的の場合には、GROUP BY句は使用せず、DISTINCTを指定した方が、SQL文の処理効率がよくなることがあります。

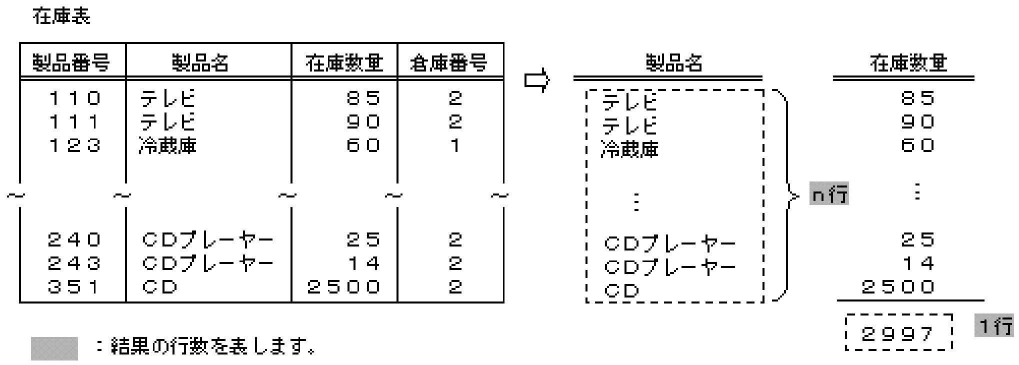
例2のように集合関数の結果と列名を取り出す場合には、表をグループ化することが必要です。選択リストに集合関数と列名を並べて指定し、かつGROUP BY句の指定がないとエラーとなります。エラーとなる例を以下に示します。

例3

集合関数の使用方法の誤りの例です。

```
SELECT 製品名, SUM(在庫数量) FROM 在庫管理.在庫表
```

図9.3 集合関数の使用方法の誤りの例



例3の結果の行数は、在庫数量の合計は1行となりますが、製品名はn行となります。したがって、この問合せ指定はエラーになります。

文字列の一部を使用して表をグループ化する

表のデータを列単位でグループ化する方法はこれまでに説明しました。列単位ではなく、文字列の一部を使用して表をグループ化するためには、GROUP BY句に文字部分列関数を指定します。指定例を以下に示します。なお、文字部分列関数の詳細については、“9.4 文字列のデータを操作する”を参照してください。

例4

顧客表の所属の都道府県名が同じ人でグループ化して、各所属の平均年齢を取り出します。

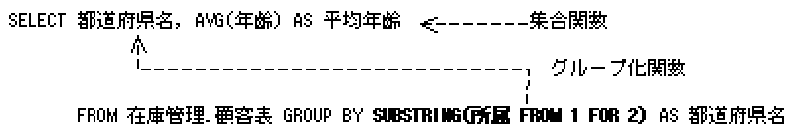
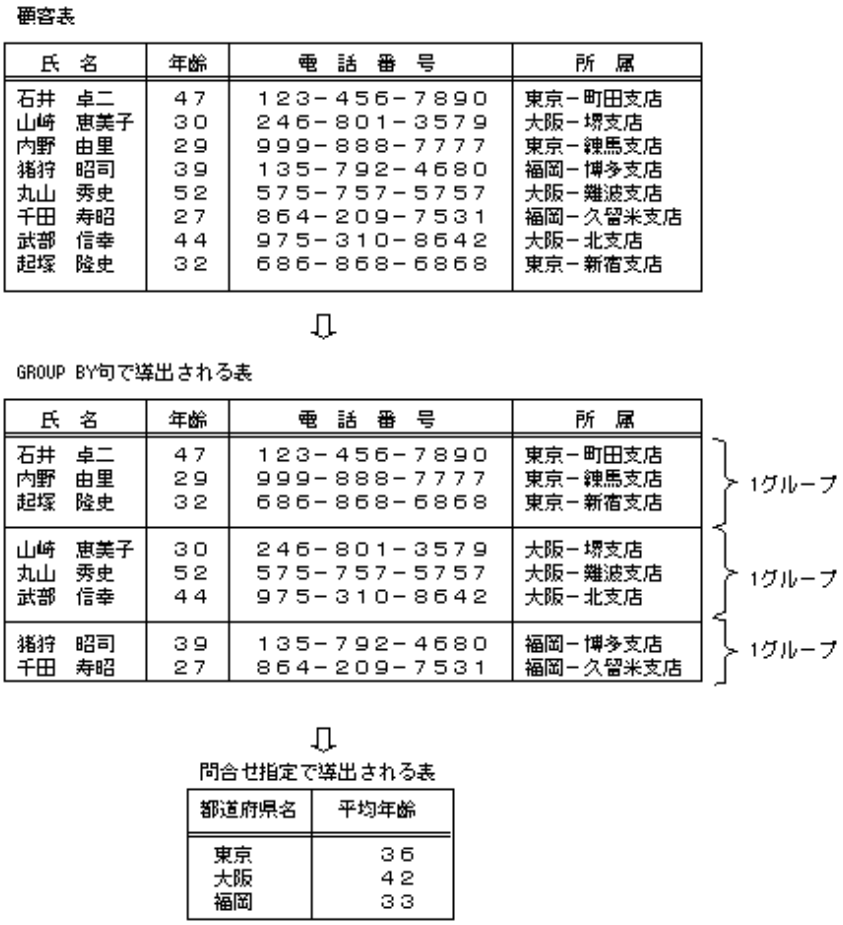


図9.4 文字列の一部を使用して表をグループ化する例



データを場合分けして表をグループ化する

列単位ではなく、列のデータを場合分けして、場合分けしたそれぞれの値ごとに表をグループ化するためには、GROUP BY句にCASE式を指定します。指定例を以下に示します。なお、CASE式の詳細については、“9.7 CASE式を使用してデータを操作する”を参照してください。

例5

小売履歴表を年代別にグループ化して、各年代の売上合計額を取り出します。

```
SELECT 年代,SUM(単価*数量) AS 売上合計額 FROM 小売履歴表

    GROUP BY CASE WHEN 年齢 < 20 THEN '10代以下'

                  WHEN 年齢 < 30 THEN '20代'

                  WHEN 年齢 < 40 THEN '30代'

                                 ELSE '40代以上' END AS 年代
```

図9.5 データを場合分けして表をグループ化する例

小売履歴表

顧客番号	年齢	販売日	製品番号	数量	単価
002	25	2007-08-10	101	1	2500
054	32	2007-08-15	102	2	1200
059	19	2007-08-22	103	1	3000
010	22	2007-08-30	104	4	1500
020	33	2007-09-07	105	2	5000
103	42	2007-09-10	103	4	3000
105	50	2007-09-10	102	3	1200
059	19	2007-09-16	104	3	1500
042	28	2007-09-25	103	1	3000
101	36	2007-09-29	106	10	500
010	22	2007-10-04	101	2	2500

⇩

GROUP BY句で導出される表

CASE式の結果 ↓

顧客番号	年齢	販売日	製品番号	数量	単価	年代	
059	19	2007-08-22	103	1	3000	10代以下	1グループ
059	19	2007-09-16	104	3	1500	10代以下	
010	22	2007-08-30	104	4	1500	20代	1グループ
010	22	2007-10-04	101	2	2500	20代	
002	25	2007-08-10	101	1	2500	20代	
042	28	2007-09-25	103	1	3000	20代	
054	32	2007-08-15	102	2	1200	30代	1グループ
020	33	2007-09-07	105	2	5000	30代	
101	36	2007-09-29	106	10	500	30代	
103	42	2007-09-10	103	4	3000	40代以上	1グループ
105	50	2007-09-10	102	3	1200	40代以上	

⇩

問合せ指定で導出される表

年代	売上合計額
10代以下	7500
20代	16500
30代	17400
40代以上	15600

月ごとに表をグループ化する

DATE型の列の年の単位や月の単位でグループ化するためには、日時値関数を指定します。指定例を以下に示します。なお、日時値関数の詳細については、“9.5 日付のデータを操作する”を参照してください。

例6

小売履歴表の販売日を月ごとにグループ化して、売上合計額を取り出します。

```
SELECT 売上月,SUM(単価*数量) AS 売上合計額 FROM  小売履歴表

    GROUP BY TRUNC_DATE(販売日,'MONTH') AS 売上月
```

図9.6 月ごとに表をグループ化する例

小売履歴表

顧客番号	年齢	販売日	製品番号	数量	単価
002	25	2007-08-10	101	1	2500
054	32	2007-08-15	102	2	1200
059	19	2007-08-22	103	1	3000
010	22	2007-08-30	104	4	1500
020	33	2007-09-07	105	2	5000
103	42	2007-09-10	103	4	3000
105	50	2007-09-10	102	3	1200
059	19	2007-09-16	104	3	1500
042	28	2007-09-25	103	1	3000
101	36	2007-09-29	106	10	500
010	22	2007-10-04	101	2	2500

⇩

GROUP BY句で導出される表

TRUNC DATE関数の結果 ⇓

顧客番号	年齢	販売日	製品番号	数量	単価	売上月	
002	25	2007-08-10	101	1	2500	2007-08-01	1グループ
054	32	2007-08-15	102	2	1200	2007-08-01	
059	19	2007-08-22	103	1	3000	2007-08-01	
010	22	2007-08-30	104	4	1500	2007-08-01	
020	33	2007-09-07	105	2	5000	2007-09-01	1グループ
103	42	2007-09-10	103	4	3000	2007-09-01	
105	50	2007-09-10	102	3	1200	2007-09-01	
059	19	2007-09-16	104	3	1500	2007-09-01	
042	28	2007-09-25	103	1	3000	2007-09-01	
101	36	2007-09-29	106	10	500	2007-09-01	
010	22	2007-10-04	101	2	2500	2007-10-01	1グループ

⇩

問合せ指定で導出される表

売上月	売上合計額
2007-08-01	13900
2007-09-01	38100
2007-10-01	5000

9.1.2 グループ化した表から処理対象のグループを特定する

表式では、FROM句に指定した表から、WHERE句を指定して処理対象の行を特定することができるように、GROUP BY句でグループ化した表から、処理対象のグループを特定することができます。それにはHAVING句を使用します。HAVING句の指定例を以下に示します。

例

“図9.1 GROUP BY句で導出される表の例”のグループ表から、製品番号が2種類以上ある製品の、製品名と在庫数量の合計を求めます。

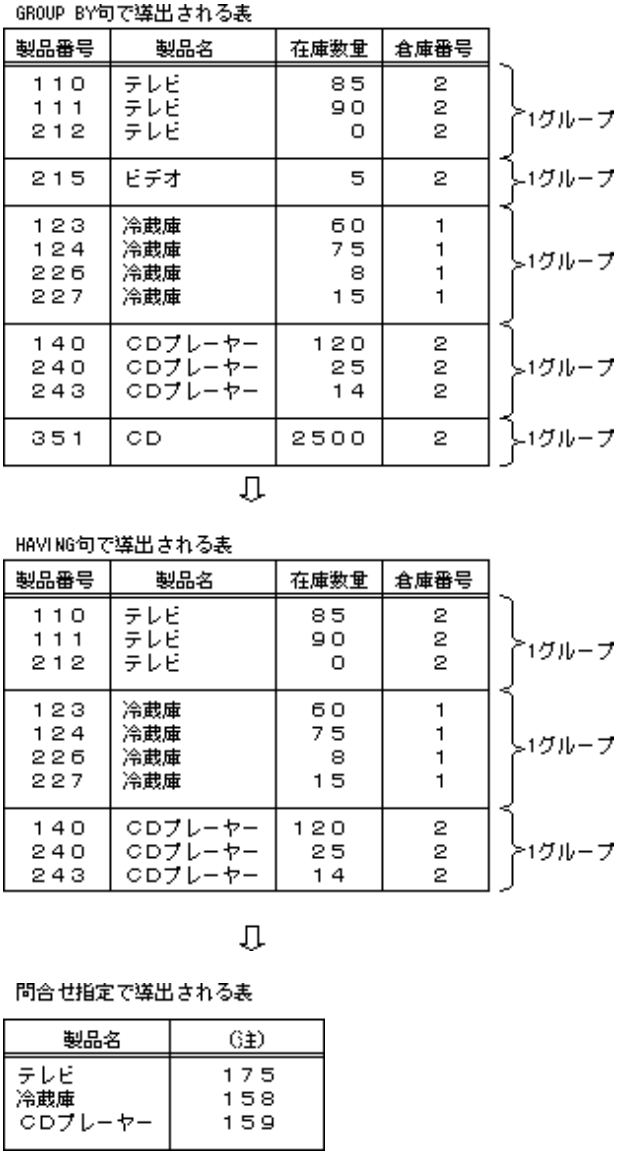
```
SELECT 製品名, SUM(在庫数量)

        FROM 在庫管理.在庫表

        GROUP BY 製品名

        HAVING    COUNT(製品名) >= 2
          (A)              (1)
```

(1) 探索条件

(A)～(1) HAVING句

HAVING句によって導出される表と、問合せ指定の結果を“図9.7 HAVING句で導出される表の例”に示します。

図9.7 HAVING句で導出される表の例



注）集合関数の結果の列には名前がありません。

HAVING句

HAVING句には、処理対象のグループを特定するための探索条件を指定します。条件とする列は、グループ化列か、集合関数で指定することが必要です。グループ化列でない列を集合関数以外で指定するとエラーになります。

なお、グループ化列を指定する条件はHAVING句ではなく、WHERE句でも指定できます。HAVING句はグループ化した表を対象にしますが、WHERE句はグループ化する前の段階で処理対象の行を特定するため、そのような条件はWHERE句に指定した方がSQL文の処理効率がよくなることがあります。例えば、上記の例の問合せ指定で、製品名が“テレビ”の行を導出する場合、“製品名 = N'テレビ'”という条件はWHERE句に指定します。

9.1.3 関連づけられた複数の表をグループ化して操作する

FROM句に複数の表を指定することにより、表を関連づけて操作できることはすでに説明しました。表式の中では、FROM句の結果は表を導出します。その表から処理対象の行を特定するWHERE句の結果も表を導出します。GROUP BY句はこの表を対象に処理します。したがって、複数の表を関連づけた結果をグループ化して操作することができます。指定例を以下に示します。

例1

在庫表と発注表から、製品番号と在庫数量ごとに、発注数量の合計が210未満でかつ複数の取引先に発注している製品について、製品番号と、在庫数量に発注数量の合計を加えた値を求めます。

```
SELECT 製品番号, 在庫数量 + SUM(発注数量)

        FROM 在庫管理.発注表, 在庫管理.在庫表
                        (1)
        WHERE 取引製品 = 製品番号
                    (1)
        GROUP BY 製品番号, 在庫数量
                    (1)
        HAVING SUM(発注数量) < 210 AND COUNT(取引先) >= 2
                        (1)
```

(1) 表式

図9.8 関連づけられた複数の表をグループ化して操作する例

FROM句で導出される表 (注1)

取引先	取引製品	仕入価格	発注数量	製品番号	製品名	在庫数量	倉庫番号
61	123	48000	60	110	テレビ	85	2
61	123	48000	60	111	テレビ	90	2
61	123	48000	60	123	冷蔵庫	60	1
・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・

(FROM句で導出される表は、発注表と在庫表のすべての行の組合せからなる276行です)

⇩

WHERE句で導出される表 (注1)

取引先	取引製品	仕入価格	発注数量	製品番号	製品名	在庫数量	倉庫番号
61	123	48000	60	123	冷蔵庫	60	1
61	124	64000	40	124	冷蔵庫	75	1
61	140	8000	80	140	CDプレーヤー	120	2
・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・

(WHERE句で導出される表は、FROM句の結果の表の中で取引製品と製品番号が等しい23行です)

⇩

GROUP BY句で導出される表

取引先	取引製品	仕入価格	発注数量	製品番号	製品名	在庫数量	倉庫番号
62	110	37500	120	110	テレビ	85	2
74	110	39000	120	110	テレビ	85	2
63	111	57400	80	111	テレビ	90	2
74	111	54000	120	111	テレビ	90	2
61	123	48000	60	123	冷蔵庫	60	1
61	124	64000	40	124	冷蔵庫	75	1
61	140	8000	80	140	CDプレーヤー	120	2
71	140	7800	50	140	CDプレーヤー	120	2
72	140	7000	70	140	CDプレーヤー	120	2
63	212	205000	30	212	テレビ	0	2
61	215	240000	10	215	ビデオ	5	2
63	215	246000	10	215	ビデオ	5	2
72	215	210000	10	215	ビデオ	5	2
62	226	112500	20	226	冷蔵庫	8	1
72	226	105000	20	226	冷蔵庫	8	1
74	226	117000	20	226	冷蔵庫	8	1
74	227	140400	10	227	冷蔵庫	15	1
61	240	80000	20	240	CDプレーヤー	25	2
72	243	84000	10	243	CDプレーヤー	14	2
62	351	375	800	351	CD	2500	2
71	351	390	600	351	CD	2500	2
72	351	350	1000	351	CD	2500	2
74	351	390	700	351	CD	2500	2

⇩

HAVING句で導出される表

取引先	取引製品	仕入価格	発注数量	製品番号	製品名	在庫数量	倉庫番号
63	111	57400	80	111	テレビ	90	2
74	111	54000	120	111	テレビ	90	2
61	140	8000	80	140	CDプレーヤー	120	2
71	140	7800	50	140	CDプレーヤー	120	2
72	140	7000	70	140	CDプレーヤー	120	2
61	215	240000	10	215	ビデオ	5	2
63	215	246000	10	215	ビデオ	5	2
72	215	210000	10	215	ビデオ	5	2
62	226	112500	20	226	冷蔵庫	8	1
72	226	105000	20	226	冷蔵庫	8	1
74	226	117000	20	226	冷蔵庫	8	1

⇩

問合せ指定で導出される表

製品番号	(注2)
111	290
140	320
215	35
226	68

注1) この例のFROM句およびWHERE句の結果の導出方法は“図8.3 複数の表を関連づけて操作する例”で説明しています。

注2) 演算式の結果の列には名前がありません。

このデータ操作で、誤りやすい例を以下に示します。データ操作が、製品番号ごとに、在庫数量と発注数量の合計を求めることから、以下のように指定しても同じと誤解すると誤りになります。

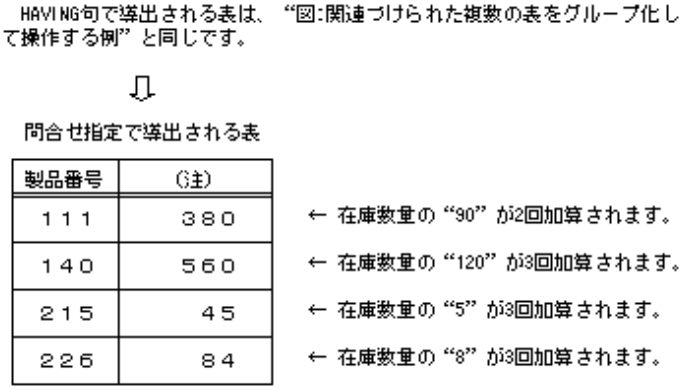
例2

例1のデータ操作を誤って指定した例です。

```
SELECT 製品番号, SUM(在庫数量) + SUM(発注数量)
                :
        GROUP BY 製品番号
```

この問合せ指定を実行すると、結果は以下のようになります。

図9.9 関連づけられた複数の表をグループ化したときに誤りやすい例



注) 演算式の結果の列には名前がありません。

上記の誤りを修正するために、MAX関数またはMIN関数を使用することは可能です。この例を以下に示します。

例3

例2の誤りを修正した例です。

```
SELECT 製品番号, MAX(在庫数量) + SUM(発注数量)
          :
        GROUP BY 製品番号
          :
```

在庫数量は、それぞれのグループごとに同じ値を複数持つため、それらの最大値または最小値はその値になり、この問合せ指定の結果は、“図9.8 関連づけられた複数の表をグループ化して操作する例”と同じになります。しかし、SQL文の処理効率をよくするには、例1のように指定してください。

9.1.4 グループ表の集計結果を再集計する

グループ表から集計した結果に対して再集計を行う場合は、導出表を利用します。導出表を利用することにより、ビュー表の定義をすることなく、再集計が可能となります。指定例を以下に示します。

例

在庫表から製品名ごとの在庫数量の合計を取り出した導出表を使って、製品の件数と在庫数量の合計を求めます。

```
SELECT COUNT(製品別在庫表.製品種類), SUM(製品別在庫表.在庫合計) FROM

    (SELECT 製品名, SUM(在庫数量) FROM 在庫管理.在庫表 GROUP BY 製品名)
                                    (1)

          AS 製品別在庫表(製品種類, 在庫合計)
                                    (2)
```

(1) 導出表

(2) 導出列リスト

導出表のGROUP BY句で導出される表

製品番号	製品名	在庫数量	倉庫番号	
110	テレビ	85	2	1グループ
111	テレビ	90	2	
212	テレビ	0	2	
215	ビデオ	5	2	1グループ
123	冷蔵庫	60	1	1グループ
124	冷蔵庫	75	1	
226	冷蔵庫	8	1	
227	冷蔵庫	15	1	
140	CDプレーヤー	120	2	1グループ
240	CDプレーヤー	25	2	
243	CDプレーヤー	14	2	
351	CD	2500	2	1グループ

⇩

導出表(製品別在庫表)

製品種類	在庫合計
テレビ	175
ビデオ	5
冷蔵庫	158
CDプレーヤー	159
CD	2500

⇨ 問合せ指定で導出される表

(注)	(注)
5	2997

注) 集合関数の結果の列には名前がありません。

9.2 いろいろな探索条件を指定する

SQL文で処理対象の行を特定するには探索条件を使用します。探索条件が成立する行が操作の対象になります。探索条件には以降で説明する述語を単独で、または論理演算子(“AND”、“OR”または“NOT”)で組み合わせて指定します。述語の結果は、“真”、“偽”、または“不定”のいずれかになります。述語または述語の組合せが“真”の場合に探索条件は成立、“偽”または“不定”の場合に探索条件は不成立となります。なお、論理演算子のことを“ブール演算子”とも呼びます。

論理演算子の指定方法

論理演算子の機能と指定方法について説明します。

例1

探索条件に論理演算子“AND”を指定した例です。在庫表から、製品番号が100以上で、かつ倉庫番号が1の行を取り出します。

```
SELECT  …  FROM  在庫管理.在庫表

           WHERE  製品番号 >= 100   AND   倉庫番号 = 1
                     (1)          (2)       (3)
```

(1) 述語

(2) 論理演算子

(3) 述語

(1)～(3) 探索条件

例2

探索条件に論理演算子“OR”を指定した例です。在庫表から、製品番号が100以上か、または倉庫番号が1の行を取り出します。

```
SELECT  …  FROM  在庫管理.在庫表

           WHERE  製品番号 >= 100   OR   倉庫番号 = 1
                     (1)          (2)       (3)
```

(1) 述語

(2) 論理演算子

(3) 述語

(1)～(3) 探索条件

例3

探索条件に論理演算子“NOT”を指定した例です。在庫表から、製品名が“テレビ”以外の行を取り出します。

```
SELECT  …  FROM  在庫管理.在庫表

           WHERE  NOT  製品名 = N'テレビ'
                   (1)        (2)
```

(1) 論理演算子

(2) 述語

(1)～(2) 探索条件

述語と述語のANDを指定した結果を“表9.1 ANDの真理表”に、ORを指定した結果を“表9.2 ORの真理表”に示します。述語の前にNOTを指定した結果を“表9.3 NOTの真理表”に示します。

表9.1 ANDの真理表

述語の結果	真	偽	不定
真	真	偽	不定
偽	偽	偽	偽
不定	不定	偽	不定

表9.2 ORの真理表

述語の結果	真	偽	不定
真	真	真	真
偽	真	偽	不定
不定	真	不定	不定

表9.3 NOTの真理表

述語の結果	NOTの結果
真	偽

述語の結果	NOTの結果
偽	真
不定	不定

論理演算子の評価順序

探索条件には、複数の論理演算子を組み合わせて使用することができます。このとき、論理演算子の評価順序は以下のとおりです。

1. NOT
2. AND
3. OR

また、括弧(“(”および“)”)を使用して評価順序を指定することができます。括弧の中の条件は、ほかに先立って評価されます。

9.2.1 2つの値を比較する

2つの値を比較するには、比較述語またはNULL述語を使用します。

比較演算子を用いて比較する

2つの値を比較するには比較述語を使用します。比較述語で指定する比較演算子を以下に示します。

表9.4 比較演算子

比較演算子	機 能（注）
＝	“＝”の左側の値式と右側の値式が等しい場合に真となり、等しくない場合に偽となります。
＜	“＜”の左側の値式が右側の値式より小さい場合に真となり、大きいまたは等しい場合に偽となります。
＜＝	“＜＝”の左側の値式が右側の値式より小さいまたは等しい場合に真となり、大きい場合に偽となります。
＞	“＞”の左側の値式が右側の値式より大きい場合に真となり、小さいまたは等しい場合に偽となります。
＞＝	“＞＝”の左側の値式が右側の値式より大きいまたは等しい場合に真となり、小さい場合に偽となります。
＜＞	“＜＞”の左側の値式と右側の値式が等しくない場合に真となり、等しい場合に述語は偽となります。

注) 比較演算子の左側または右側の値式がNULL値の場合、比較述語の結果は不定になります。

比較演算子の左右の値式の対は、比較可能であることが必要です。比較可能なデータ型を以下に示します。

参照

なお、各データ型については、“SQLリファレンス”を参照してください。

表9.5 比較可能なデータ型

		右辺							
		文字列型	各国語文字列型	真数型・概数型	DATE型	TIME型	TIMESTAMP型	INTERVAL型(年月)	INTERVAL型(日時)
左辺	文字列型	○	×	×	○(注1)	○(注2)	○(注3)	○(注4)	○(注5)
	各国語文字列型	×	○	×	×	×	×	×	×
	真数型・概数型	×	×	○	×	×	×	○(注6)	○(注7)
	DATE型	○(注1)	×	×	○	×	×	×	×
	TIME型	○(注2)	×	×	×	○	×	×	×
	TIMESTAMP型	○(注3)	×	×	×	×	○	×	×
	INTERVAL型(年月)	○(注4)	×	○(注6)	×	×	×	○	×
	INTERVAL型(日時)	○(注5)	×	○(注7)	×	×	×	×	○

○:比較可能

×:比較不可

注1) 比較相手が変数指定の場合のみ可能です。このとき変数の内容は日付列であることが必要です。

注2) 比較相手が変数指定の場合のみ可能です。このとき変数の内容は時刻列であることが必要です。

注3) 比較相手が変数指定の場合のみ可能です。このとき変数の内容は時刻印列であることが必要です。

注4) 比較相手が変数指定の場合のみ可能です。このとき変数の内容は比較元と同じ精度を持つ年月クラスの時間隔定数であることが必要です。

注5) 比較相手が変数指定の場合のみ可能です。このとき変数の内容は比較元と同じ精度を持つ日時クラスの時間隔定数であることが必要です。

注6) 比較相手が真数(位取り0)の変数指定の場合のみ可能です。このとき変数の内容は比較元と同じ単一日時フィールドからなる年月クラスの時間隔定数であることが必要です。

注7) 比較相手が真数(位取り0)の変数指定の場合のみ可能です。このとき変数の内容は比較元と同じ単一日時フィールドからなる日時クラスの時間隔定数であることが必要です。

比較述語の指定例を以下に示します。

例1

在庫表から製品番号が123の行を取り出します。

```
SELECT  …  FROM  在庫管理.在庫表

           WHERE  製品番号  =  123
                    (1)     (2) (3)
```

(1) 値式

(2) 比較演算子

(3) 値式

(1)～(3) 比較述語

例2

在庫表から在庫数量が50未満の行を取り出します。

```
SELECT  …  FROM 在庫管理.在庫表 WHERE 在庫数量 < 50
```

例3

発注表から“仕入価格×発注数量”が100万以下の行を取り出します。

```
SELECT  …  FROM 在庫管理.発注表
           WHERE 仕入価格 * 発注数量 <= 1000000
```

例4

在庫表から在庫数量が500より大きい行を取り出します。

```
SELECT  …  FROM 在庫管理.在庫表 WHERE 在庫数量 > 500
```

例5

在庫表から在庫数量が500以上の行を取り出します。

```
SELECT  …  FROM 在庫管理.在庫表 WHERE 在庫数量 >= 500
```

例6

在庫表から“テレビ”以外の製品名の行を取り出します。

```
SELECT  …  FROM 在庫管理.在庫表 WHERE 製品名 <> N'テレビ'
```

論理演算子を使用して、複数の比較述語を組み合わせて指定する例を以下に示します。

例7

在庫表から製品名が“テレビ”で、かつ在庫数量が90以上の行を取り出します。

```
SELECT  …  FROM 在庫管理.在庫表
           WHERE 製品名 = N'テレビ' AND 在庫数量 >= 90
```

例8

在庫表から在庫数量が10以上で製品名がビデオ、または在庫数量が200以上で製品名がテレビの行を取り出します。

```
SELECT  …  FROM 在庫管理.在庫表
           WHERE 在庫数量 >= 10 AND 製品名 = N'ビデオ' OR
                 在庫数量 >= 200 AND 製品名 = N'テレビ'
```

例9

在庫表から“テレビ”または“ビデオ”以外の製品名の行を取り出します。

```
SELECT  …  FROM 在庫管理.在庫表

           WHERE NOT 製品名 = N'テレビ' AND

                 NOT 製品名 = N'ビデオ'
```

副問合せを用いた比較述語

探索条件中の比較述語に指定する値が、関数によって導き出される値であるような場合に、探索条件の中に副問合せを記述することができます。

副問合せの括弧の中は問合せ指定と同じ形式です。ただし、副問合せの場合、導出される表は、列数が1つであることが必要です。したがって、探索条件の左辺には1個の値式(列名、関数など)を指定することを意味しています。副問合せを用いた比較述語の例を以下に示します。

例10

発注表から仕入価格の平均をとり、それよりも高い仕入価格の取引先を発注表から取り出します。

```
SELECT  取引先  FROM  在庫管理.発注表

  WHERE  仕入価格  >  (SELECT AVG(仕入価格) FROM 在庫管理.発注表)
           (1)     (2)          ^^^^^^^^^^^^^ (3)
                                  ↓
                                 列数は1つ
```

(1) 値式(1個)

(2) 比較演算子

(3) 副問合せ

(1)～(3) 比較述語

図9.10 副問合せを用いた比較述語を指定してデータを検索する例

発注表

取引先	取引製品	仕入価格	発注数量
61	123	48000	60
61	124	64000	40
61	140	8000	80
61	215	240000	10
61	240	80000	20
62	110	37500	120
62	226	112500	20
62	351	375	800
63	111	57400	80
63	212	205000	30
63	215	246000	10
71	140	7800	50
71	351	390	600
72	140	7000	70
72	215	210000	10
72	226	105000	20
72	243	84000	10
72	351	350	1000
74	110	39000	120
74	111	54000	120
74	226	117000	20
74	227	140400	10
74	351	390	700

⇨

副問合せの結果

(注)
81048

⇩

WHERE句の結果

取引先	取引製品	仕入価格	発注数量
61	215	240000	10
62	226	112500	20
63	212	205000	30
63	215	246000	10
72	215	210000	10
72	226	105000	20
72	243	84000	10
74	226	117000	20
74	227	140400	10

⇩

結果

取引先
61
62
63
63
72
72
72
74
74

注) AVG(仕入価格)の結果から作成される列です。

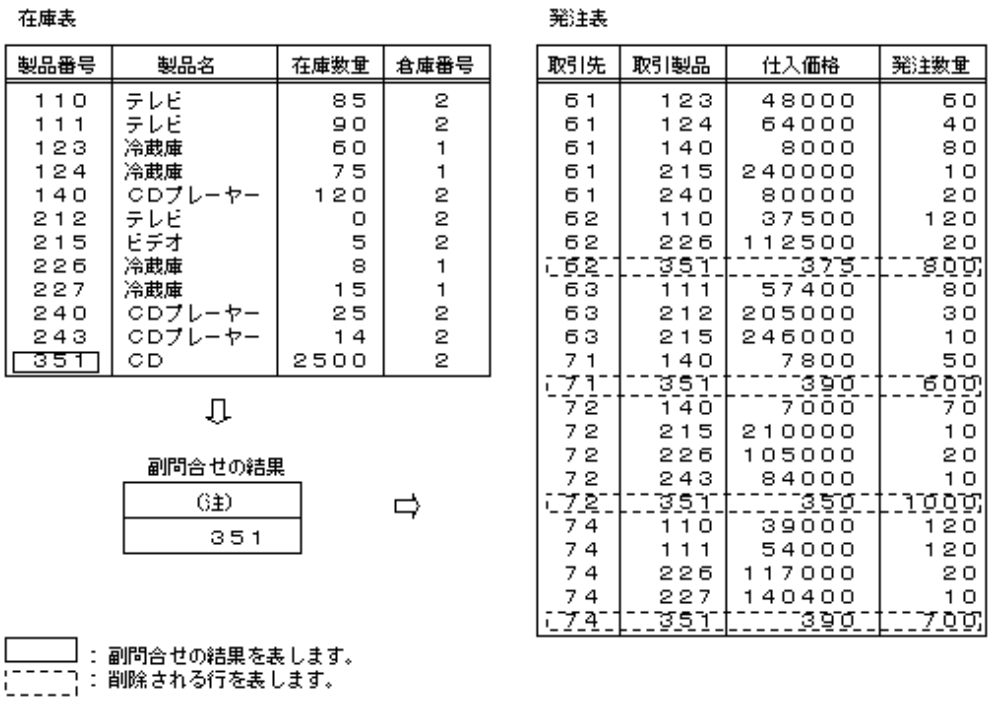
備考. 集合関数の結果の列には名前がありません。

例11

在庫表から製品番号の最大値をとり、その値の取引製品の行を発注表から削除します。

```
DELETE FROM 在庫管理.発注表

        WHERE 取引製品 = (SELECT MAX(製品番号) FROM 在庫管理.在庫表)
```

図9.11 副問合せを用いた比較述語を指定してデータを削除する例



注) MAX(製品番号)の結果から作成される列です。

備考. 集合関数の結果の列には名前がありません。

NULL値かどうかを調べる

列の値がNULL値かどうかを調べるにはNULL述語を使用します。指定例を以下に示します。

例12

在庫表から倉庫番号がNULL値の行を取り出します。

```
SELECT  …  FROM 在庫管理.在庫表

           WHERE 倉庫番号  IS  NULL
                  (1)          (A)
```

(1) 列名

(1)～(A) NULL述語

例13

在庫表から倉庫番号がNULL値でない行を取り出します。

```
SELECT  …  FROM 在庫管理.在庫表

           WHERE 倉庫番号  IS  NOT NULL
                          (1)
```

(1) NULL述語

9.2.2 値が特定の範囲に含まれるかを調べる

値が特定の範囲に含まれるかを調べるためには、BETWEEN述語を使用します。指定例を以下に示します。

例1

在庫表から在庫数量が50以上で100以下の行を取り出します。

```
SELECT  … FROM 在庫管理.在庫表

          WHERE  在庫数量 BETWEEN  50  AND  100
                   (1)              (2)      (3)
```

(1) 値式1

(2) 値式2

(3) 値式3

(1)～(3) BETWEEN述語

BETWEEN述語の結果は、3つの値式の値により以下の条件が満たされる場合、真となります。

```
値式2 ＜＝ 値式1 ＜＝ 値式3
```

値式1が、値式2より小さいか値式3より大きい場合、結果は偽となります。

値式1がNULL値、値式1が値式2以上で値式3がNULL値、または値式1が値式3以下で値式2がNULL値の場合、結果は不定になります。

例2

在庫表の在庫数量が10以上で50以下、または200以上で300以下の行を取り出します。

```
SELECT  …  FROM 在庫管理.在庫表

           WHERE 在庫数量 BETWEEN 10 AND 50 OR

                 在庫数量 BETWEEN 200 AND 300
```

例3

発注表から“仕入価格×発注数量”が100万未満または200万を超える行を取り出します。

```
SELECT  …  FROM 在庫管理.発注表

   WHERE 仕入価格 * 発注数量 NOT BETWEEN 1000000 AND 2000000
                              ↑
                            NOTの指定
```

この指定の結果は以下と同じになります。

```
SELECT  …  FROM 在庫管理.発注表

   WHERE NOT 仕入価格 * 発注数量 BETWEEN 1000000 AND 2000000
```

値が特定の範囲に含まれるかを調べるためには、比較述語も使用できます。BETWEEN述語で指定した例1を比較述語で指定すると、以下のようになります。

例4

在庫表から在庫数量が50以上で100以下の行を取り出します。

```
SELECT  …  FROM 在庫管理.在庫表  WHERE 在庫数量  BETWEEN 50 AND 100

                        ↓

SELECT  …  FROM 在庫管理.在庫表  WHERE 在庫数量  >= 50 AND 在庫数量  <= 100
```

9.2.3 値がいくつかの値の集合に含まれるかを調べる

値がいくつかの値の集合に含まれるかを調べるには、IN述語、限定述語またはEXISTS述語を使用します。

値の集合に含まれるかを調べる

特定の値が、限定された値の集合の中に含まれるかを調べるには、IN述語を使用します。比較する値の集合は、限定値リストまたは副問合せで指定します。限定値リストを指定した例を以下に示します。

例1

在庫表から、製品番号が“123”、“124”、“226”または“227”のどれかである行を特定します。指定の結果を“図9.12 IN述語の指定例”に示します。

```
SELECT 製品番号, 製品名, 在庫数量, 倉庫番号 FROM 在庫管理.在庫表

      WHERE  製品番号  IN  (123, 124, 226, 227)
                (1)              (2)
```

(1) 値式

(2) 限定値リスト

(1)～(2) IN述語

図9.12 IN述語の指定例

在庫表

製品番号	製品名	在庫数量	倉庫番号	
110	テレビ	85	2	
111	テレビ	90	2	
123	冷蔵庫	60	1	← IN述語が真となる行
124	冷蔵庫	75	1	← IN述語が真となる行
140	CDプレーヤー	120	2	
212	テレビ	0	2	
215	ビデオ	5	2	
226	冷蔵庫	8	1	← IN述語が真となる行
227	冷蔵庫	15	1	← IN述語が真となる行
240	CDプレーヤー	25	2	
243	CDプレーヤー	14	2	
351	CD	2500	2	

⇩

IN述語の結果特定される行の集まり

製品番号	製品名	在庫数量	倉庫番号
123	冷蔵庫	60	1
124	冷蔵庫	75	1
226	冷蔵庫	8	1
227	冷蔵庫	15	1

限定値リストには、値式と比較する値を括弧でくくって指定します。限定値リストの値は、ホスト変数または定数で指定します。列名などは指定できません。

IN述語の結果は、値式の値が括弧内の値のどれかと等しい場合、真となります。すべてと等しくない場合、結果は偽となります。値式がNULL値か、括弧内のどれかの値がNULL値の場合、結果は不定となります。

参照

IN述語の結果については“SQLリファレンス”を参照してください。

副問合せを用いたIN述語

値式と比較する値が、ほかの表から導き出される値であるような場合に、限定値リストの代わりに副問合せを記述します。

副問合せの括弧の中は問合せ指定と同じ形式です。ただし、副問合せの場合、導出される表は、列数が1つであることが必要です。したがって、探索条件の左辺には1個の値式(列名、関数など)を指定することを意味しています。副問合せを用いたIN述語の例を以下に示します。

例2

取引製品が“351”の製品を発注している会社名を取り出します。

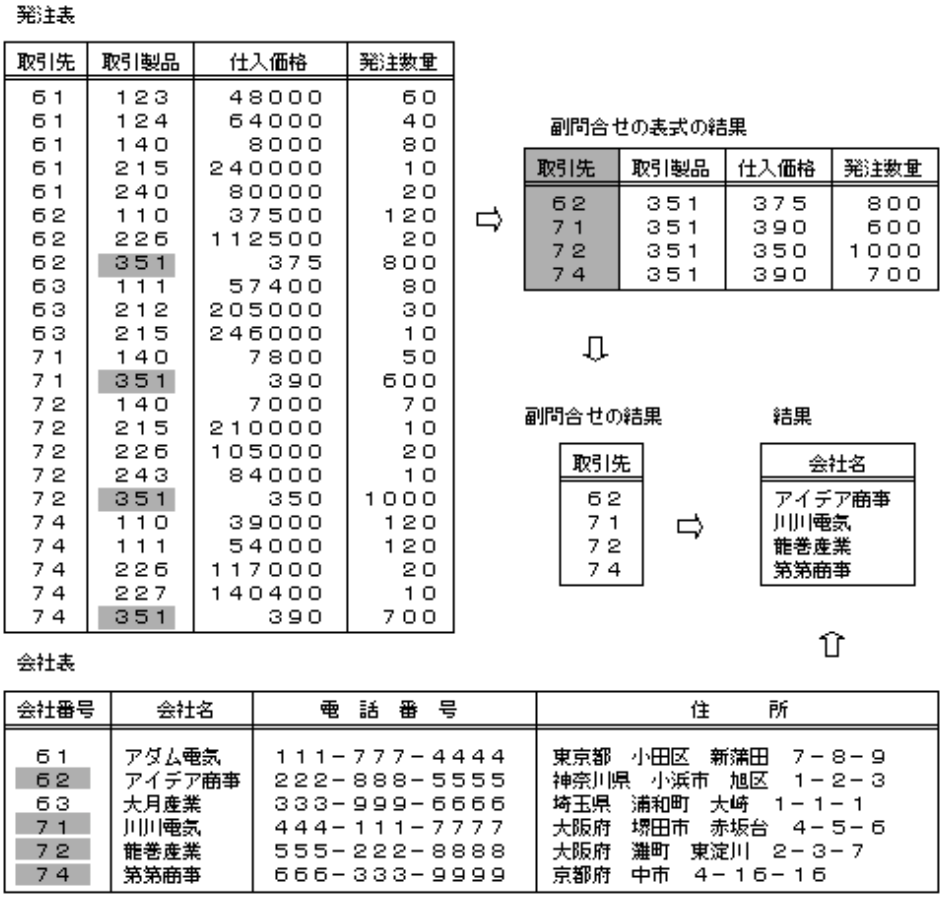
```
SELECT 会社名  FROM  在庫管理.会社表  WHERE

   会社番号 IN (SELECT 取引先 FROM 在庫管理.発注表 WHERE 取引製品 = 351)
     (1)              ↓              (2)
                  列数は1つ
```

(1) 値式(1個)

(2) 副問合せ

(1)～(2) IN述語

図9.13 副問合せを用いたIN述語の例



値の集合と比較する

探索条件として、限定された値の集合との比較を指定する場合には、限定述語を使用します。限定述語は、比較演算子と限定子(ALL、SOMEまたはANY)を使用して指定します。

副問合せの括弧の中は問合せ指定と同じ形式です。ただし、副問合せの場合、導出される表は、列数が1つであることが必要です。したがって、探索条件の左辺には1個の値式(列名、関数など)を指定することを意味しています。

また、副問合せに指定された表に含まれない列、すなわち外側の問合せに指定された表に含まれる列を副問合せの探索条件に指定している場合、その副問合せは“外への参照がある副問合せ”と呼びます。限定述語に外への参照がある副問合せを指定した例を以下に示します。

例3

すべての取引先について、その取引先からの発注数量が在庫数量を上回っている製品番号を検索します。副問合せの表(発注表)に含まれない列、すなわち外側の問合せの表(在庫表)に含まれる列(製品番号)を探索条件に指定しています。検索の結果を“図9.14 限定述語の指定例”に示します。

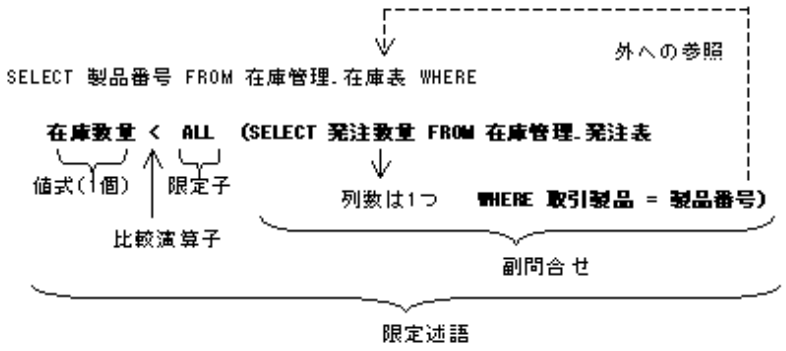


図9.14 限定述語の指定例

在庫表

製品番号	製品名	在庫数量	倉庫番号
110	テレビ	85	2
111	テレビ	90	2
123	冷蔵庫	60	1
124	冷蔵庫	75	1
140	CDプレーヤー	120	2
212	テレビ	0	2
215	ビデオ	5	2
226	冷蔵庫	8	1
227	冷蔵庫	15	1
240	CDプレーヤー	25	2
243	CDプレーヤー	14	2
351	CD	2500	2

発注表

取引先	取引製品	仕入価格	発注数量
62	110	37500	★120
74	110	39000	★120
63	111	57400	80
74	111	54000	★120
61	123	48000	60
61	124	64000	40
61	140	8000	80
71	140	7800	50
72	140	7000	70
63	212	205000	★ 30
61	215	240000	★ 10
63	215	246000	★ 10
72	215	210000	★ 10
62	226	112500	★ 20
72	226	105000	★ 20
74	226	117000	★ 20
74	227	140400	10
61	240	80000	20
72	243	84000	10
62	351	375	800
71	351	390	600
72	351	350	1000
74	351	390	700

★： 発注数量が在庫数量を上回っている製品

限定述語の結果、
特定される行の集まり

製品番号
110
212
215
226

製品番号
110
111
212
215
226

備考. 限定子に"SOME"または"ANY"を指定した場合
★のすべての行が真になります。

特定の条件の行が存在するかを調べる

副問合せを指定し、その副問合せの結果が空(すなわち、副問合せの結果、導出される表の行数が0)となるか否かで行を特定する場合に、EXISTS述語を使用します。"EXISTS"のあとに指定した副問合せの結果が空でない場合、このEXISTS述語は真となります。副問合せの結果が空の場合は、EXISTS述語は偽となります。

副問合せの括弧の中は問合せ指定と同じ形式です。副問合せの結果が2列以上の表になってもかまいません。EXISTS述語の指定例を以下に示します。

例4

発注価格が500万を超えている製品の製品番号と製品名を取り出します。発注価格は、仕入価格×発注数量で求められます。

```
SELECT 製品番号, 製品名 FROM 在庫管理.在庫表 WHERE

  EXISTS (SELECT ＊  FROM 在庫管理.発注表 WHERE 取引製品 = 製品番号
   (1)          ↓                          (2)
       複数の列も可能        AND (仕入価格 ＊ 発注数量) > 5000000)
                                                  (2)
```

(1)～(2) EXISTS述語

(2) 副問合せ

図9.15 EXISTS述語の指定例

発注表

取引先	取引製品	仕入価格	発注数量	仕入価格×発注数量
61	123	48000	60	2880000
61	124	64000	40	2560000
61	140	8000	80	640000
61	215	240000	10	2400000
61	240	80000	20	1600000
62	110	37500	120	4500000
62	226	112500	20	2250000
62	351	375	800	300000
63	111	57400	80	4592000
63	212	205000	30	6150000
63	215	246000	10	2460000
71	140	7800	50	390000
71	351	390	600	234000
72	140	7000	70	490000
72	215	210000	10	2100000
72	226	105000	20	2100000
72	243	84000	10	840000
72	351	350	1000	350000
74	110	39000	120	4680000
74	111	54000	120	6480000
74	226	117000	20	2340000
74	227	140400	10	1404000
74	351	390	700	273000

在庫表

製品番号	製品名	在庫数量	倉庫番号
110	テレビ	85	2
111	テレビ	90	2
123	冷蔵庫	60	1
124	冷蔵庫	75	1
140	CDプレーヤー	120	2
212	テレビ	0	2
215	ビデオ	5	2
226	冷蔵庫	8	1
227	冷蔵庫	15	1
240	CDプレーヤー	25	2
243	CDプレーヤー	14	2
351	CD	2500	2

⇨

EXISTS述語の結果、
特定される行の集まり

製品番号	製品名
111	テレビ
212	テレビ

9.2.4 値が文字列のパターンに一致するかを調べる

文字列型および各国語文字列型のデータについて、あるパターンの文字列に一致するかどうかを調べる場合に、LIKE述語を使用します。LIKE述語の結果は、列に、指定したパターンを含む場合、真となります。含まない場合、偽となります。指定した列またはパターンがNULL値の場合、結果は不定となります。なお、文字列型データ、各国語文字列型データ、およびパターン内の空白文字は、英字および各国語文字と同じように1文字として扱います。

また、LIKE述語に指定した列あるいはパターンが可変長の文字列の場合には、データの有効長の範囲で評価されます。LIKE述語の指定例を以下に示します。

例

会社表の会社名の中に“電気”を含む行を特定します。指定した結果を“図9.16 LIKE述語による行の特定”に示します。

```
SELECT 会社番号, 会社名, 電話番号, 住所 FROM 在庫管理.会社表

       WHERE  会社名   LIKE   N'%電気%'
                (1)                (2)
```

(1) 列名

(2) パターン

(1)～(2) LIKE述語

図9.16 LIKE述語による行の特定

会社表

会社番号	会社名	電 話 番 号	住 所
６１	アダム電気	１１１－７７７－４４４４	東京都 小田区 新蒲田 ７－８－９
６２	アイデア商事	２２２－８８８－５５５５	神奈川県 小浜市 旭区 １－２－３
６３	大月産業	３３３－９９９－６６６６	埼玉県 浦和町 大崎 １－１－１
７１	川川電気	４４４－１１１－７７７７	大阪府 堺田市 赤坂台 ４－５－６
７２	能著産業	５５５－２２２－８８８８	大阪府 灘町 東淀川 ２－３－７
７４	第第商事	６６６－３３３－９９９９	京都府 中市 ４－１６－１６

⇩

LIKE述語の結果特定される行の集まり

会社番号	会社名	電 話 番 号	住 所
６１	アダム電気	１１１－７７７－４４４４	東京都 小田区 新蒲田 ７－８－９
７１	川川電気	４４４－１１１－７７７７	大阪府 堺田市 赤坂台 ４－５－６

パターンは、文字列および任意文字列指示子と任意文字指示子で表します。任意文字列指示子と任意文字指示子の意味を以下に示します。パターンが文字列の場合は、英字で指定します。また、パターンが各国語の場合は、各国語文字で指定します。

表9.6 任意文字列指示子と任意文字指示子の意味

指示子	記 号	記 号 の 意 味
任意文字列指示子	% (英字) ％(各国語文字)	パターンの中のこの文字が位置する部分の文字は、任意の文字列(0個以上の文字列)であることを表します。
任意文字指示子	_(英字) ＿(各国語文字)	パターンの中のこの文字が位置する部分の文字は、任意の文字(1文字)であることを表します。

任意文字列指示子“%”の使用例

任意文字列指示子“%”の使用例を以下に示します。“図9.17 LIKE述語の指定例(任意文字列指示子を使用した場合)”にどの行が特定されるかを示します。ここで使用する表TBL5はスキーマSCMに含まれます。

例1

表TBL5の列COL1が“ABC”の行を特定します。

```
SELECT ・・・ FROM SCM.TBL5 WHERE COL1 LIKE 'ABC'
```

例2

表TBL5の列COL1の先頭が“ABC”の行を特定します。

```
SELECT ・・・ FROM SCM.TBL5 WHERE COL1 LIKE 'ABC%'
```

例3

表TBL5の列COL1の最後が“ABC”の行を特定します。

```
SELECT ・・・ FROM SCM.TBL5 WHERE COL1 LIKE '%ABC'
```

例4

表TBL5の列COL1の任意の位置に“ABC”を含む行を特定します。

```
SELECT ・・・ FROM SCM.TBL5 WHERE COL1 LIKE '%ABC%'
```

例5

表TBL5の列COL1の先頭が“AB”で最後が“C”の行を特定します。

```
SELECT ・・・ FROM SCM.TBL5 WHERE COL1 LIKE 'AB%C'
```

例6

表TBL5の列COL1の先頭が“ABC”で最後が空白の行を特定します。

```
SELECT ・・・ FROM SCM.TBL5 WHERE COL1 LIKE 'ABC% '
```

例7

表TBL5の列COL1の先頭が“ABC”の行を特定します。パターンを6バイトの固定長のホスト変数“HOST1”に設定する場合には、'ABC%'を指定します。後方の空白を外すためにTRIM関数で指定します。

```
SELECT ・・・ FROM SCM.TBL5 WHERE COL1 LIKE TRIM(TRAILING :HOST1)
```

例8

表TBL5の列COL1の先頭が“ABC”の行を特定します。パターンを可変長のホスト変数“HOST1”に設定する場合には、'ABC%'を指定します。

```
SELECT ・・・ FROM SCM.TBL5 WHERE COL1 LIKE :HOST1
```

図9.17 LIKE述語の指定例(任意文字列指示子を使用した場合)

TBL5

COL1	
ＡＢＣ	← 例１、例２、例３、例４、例５、例７、例８のLIKE述語が真となる行
ＡＢＣ□□□	← 例２、例４、例６、例７、例８のLIKE述語が真となる行
ＡＢＣＸ	← 例２、例４、例７、例８のLIKE述語が真となる行
ＡＢＣＸＹＺ	← 例２、例４、例７、例８のLIKE述語が真となる行
ＡＢＢＣＡＢ	
ＡＸＢＣ	
ＡＸＢＣＹ	
ＡＢＸＹＺＣ	← 例５のLIKE述語が真となる行
ＸＡＢＣ	← 例３、例４のLIKE述語が真となる行
ＸＡＢＣＹＺ	← 例４のLIKE述語が真となる行
ＸＹＡＢＣ	← 例３、例４のLIKE述語が真となる行
ＸＹＺＡＢＣ	← 例３、例４のLIKE述語が真となる行

□： 空白文字を表します。

備考. 表TBL5の列COL1は可変長です。そして、第1行の値は3文字です。第2行の値は右側に空白を含む6文字です。したがって、第1行と第2行は異なる値です。第3行以降についても第1行と同様に、アルファベットの文字の数が値の文字数です。値には空白は含みません。

任意文字指示子“_”の例

任意文字指示子“_”の使用例を以下に示します。“図9.18 LIKE述語の指定例(任意文字指示子を使用した場合)”にどの行が特定されるかを示します。

例1

表TBL5の列COL1の先頭が“ABC”で次の1文字が任意である4文字構成の行を特定します。

```
SELECT ・・・ FROM SCM.TBL5 WHERE COL1 LIKE 'ABC_'
```

例2

表TBL5の列COL1の先頭が“ABC”で次の3文字が任意である6文字構成の行を特定します。

```
SELECT ・・・ FROM SCM.TBL5 WHERE COL1 LIKE 'ABC___'
```

例3

表TBL5の列COL1の先頭1文字が任意でうしろが“ABC”である4文字構成の行を特定します。

```
SELECT ・・・ FROM SCM.TBL5 WHERE COL1 LIKE '_ABC'
```

例4

表TBL5の列COL1の先頭2文字が任意でうしろが“ABC”である5文字構成の行を特定します。

```
SELECT ・・・ FROM SCM.TBL5 WHERE COL1 LIKE '__ABC'
```

例5

表TBL5の列COL1の先頭が“A”で次の1文字が任意、そして最後が“BC”である4文字構成の行を特定します。

```
SELECT ・・・ FROM SCM.TBL5 WHERE COL1 LIKE 'A_BC'
```

図9.18 LIKE述語の指定例(任意文字指示子を使用した場合)

TBL5

COL1	
ＡＢＣ	
ＡＢＣ□□□	← 例２のLIKE述語が真となる行
ＡＢＣＸ	← 例１のLIKE述語が真となる行
ＡＢＣＸＹＺ	← 例２のLIKE述語が真となる行
ＡＢＢＣＡＢ	
ＡＸＢＣ	← 例５のLIKE述語が真となる行
ＡＸＢＣＹ	
ＡＢＸＹＺＣ	
ＸＡＢＣ	← 例３のLIKE述語が真となる行
ＸＡＢＣＹＺ	
ＸＹＡＢＣ	← 例４のLIKE述語が真となる行
ＸＹＺＡＢＣ	

□： 空白文字を表します。

備考. 表TBL5の列COL1は可変長です。そして、第1行の値は3文字です。第2行の値は、右側に空白を含む6文字です。したがって、第1行と第2行は異なる値です。
第3行以降についても第1行と同様に、アルファベットの文字の数が値の文字数です。
値には空白は含みません。

エスケープ文字

LIKE述語のパターンの指定において、“%”および“_”は任意文字位置の指定に使用しますが、任意文字位置の指定としてではなく、“%”および“_”自身を検索の対象とする場合に、エスケープ文字の指定を行います。指定例を以下に示します。“図9.19 LIKE述語の指定例(エスケープ文字を使用した場合)”にどの行が特定されるかを示します。

例1

表TBL5の列COL2が“ABC%”の行を特定します。

```
SELECT  …  FROM SCM.TBL5 WHERE COL2 LIKE  'ABC#%'  ESCAPE  '#'
                                                          (1)
```

(1) エスケープ文字

“ESCAPE”のうしろに指定した文字(1文字だけ指定します)がエスケープ文字となります。このエスケープ文字を、文字パターン上、検索対象とする“%”または“_”の前に指定します。エスケープ文字には任意の文字が指定できます。

例2

表TBL5の列COL2の先頭が“ABC%%”の行を特定します。

```
SELECT  …  FROM SCM.TBL5 WHERE COL2 LIKE  'ABC#%#%%'  ESCAPE '#'
```

例3

表TBL5の列COL2の先頭2文字が任意で残りが“__ABC__”である9文字構成の行を特定します。

```
SELECT  …  FROM SCM.TBL5 WHERE COL2 LIKE  '__$_$_ABC$_$_'  ESCAPE '$'
```

図9.19 LIKE述語の指定例(エスケープ文字を使用した場合)

TBL5

COL2	
ＡＢＣ％	← 例6のLIKE述語が真となる行
ＡＢＣ％％	← 例7のLIKE述語が真となる行
ＡＢＣ％％ＸＹＺ	← 例7のLIKE述語が真となる行
ＡＢＣ＃％	
ＡＢＣ＃％＃％％	
ＡＢＣ％％＃	← 例7のLIKE述語が真となる行
ＡＢＣ	
ＸＹ＿ ＿ＡＢＣ＿ ＿	← 例8のLIKE述語が真となる行
＿ ＿ ＿ ＿ＡＢＣ＿ ＿	← 例8のLIKE述語が真となる行
Ｘ＿ ＿ＡＢＣ	
＿ ＿＄＿＄＿ＡＢＣ＄＿＄＿	
＄＄＿ ＿ＡＢＣ＿ ＿	← 例8のLIKE述語が真となる行

備考. 表TBL5の列COL2は可変長です。そして、第1行の値は4文字です。第2行以降についても第1行と同様に、アルファベット、%、＃、＄および下線の数が値の文字数です。値には空白は含みません。

9.3 数値のデータを操作する

表のデータを操作するときに、数値のデータに対して絶対値、最小整数値、最大整数値などを求めることができます。数値のデータを操作する処理には以下のものがあります。

表9.7 数値を操作する処理

処理名	機能
ABS式	ABS式は、指定した数値式の絶対値を求めます。
CEIL式	CEIL式は、指定した数値式以上の最小整数値を求めます。

処理名	機能
FLOOR式	FLOOR式は、指定した数値式以下の最大整数値を求めます。
ROUND式	ROUND式は、数値式を丸め位置で四捨五入します。
TRUNC式	TRUNC式は、数値式を切捨て位置で切り捨てます。
ACOS関数	ACOS関数は、数値式のアークコサイン(逆余弦)を求めます。
ASIN関数	ASIN関数は、数値式のアークサイン(逆正弦)を求めます。
ATAN関数	ATAN関数は、数値式のアークタンジェント(逆正接)を求めます。
ATAN2関数	ATAN2関数は、数値式1、数値式2のアークタンジェント(逆正接)を求めます。
COS関数	COS関数は、数値式のコサイン(余弦)を求めます。
EXP関数	EXP関数は、数値式の指数関数値を求めます。
LN関数	LN関数は、数値式の自然対数を求めます。
POWER関数	POWER関数は、数値式の指定された累乗値を求めます。
SIGN関数	SIGN関数は、数値式の符号の標識を返却します。
SIN関数	SIN関数は、数値式のサイン(正弦)を求めます。
SQRT関数	SQRT関数は、数値式の平方根を求めます。
TAN関数	TAN関数は、数値式のタンジェント(正接)を求めます。
CHR関数	CHR関数は、数値式を文字コードとして該当するASCII文字を返却します。

例1

出荷表から、出荷数量の変動の大きい(前回と今回の出荷数量の差が10以上とします)製品番号を求めます。

```
SELECT 製品番号,ABS(出荷数量 - 前回出荷数量) AS 出荷変動
                (A)          (1)
  FROM 在庫管理.出荷表

  WHERE ABS(出荷数量 - 前回出荷数量) >= 10
```

(1) 数値式

(A)～(1) ABS式

出荷表

製品番号	出荷数量	前回出荷数量
110	21	20
111	68	40
123	40	42
124	21	20
140	120	140
212	7	2
215	20	30

⇩

結果

製品番号	出荷変動
111	28
140	20
215	10

例2

すべての地区の最高気温、最低気温をその値以下の最大整数値にします。

```
SELECT 地区名, FLOOR(最高気温) AS 最高気温, FLOOR(最低気温) AS 最低気温
              (A)     (1)
  FROM 気象情報管理表
```

(1) 数値式

(A)～(1) FLOOR式

気象情報管理表

地区ｺｰﾄﾞ	地区名	最高気温	最高気温日付	最低気温	最低気温日付
1111	札幌□□□	27.255	1978-07-14	-21.235	1994-01-11
2222	仙台□□□	30.000	1999-08-01	-8.500	1983-02-02
3333	東京□□□	35.554	1984-07-23	-0.456	1988-12-23
4444	横浜□□□	34.123	1988-08-01	0.789	1967-01-04
5555	大阪□□□	32.987	1995-08-11	6.357	1979-02-15
6666	広島□□□	31.567	1997-09-01	11.258	1990-02-06
7777	博多□□□	37.525	1985-06-30	17.369	1995-12-27

□：全角ブランクを示しています

⇩

結果

地区名	最高気温	最低気温
札幌□□□	27.000	-22.000
仙台□□□	30.000	-9.000
東京□□□	35.000	-1.000
横浜□□□	34.000	0.000
大阪□□□	32.000	6.000
広島□□□	31.000	11.000
博多□□□	37.000	17.000

例3

すべての地区の最高気温を小数第2位で四捨五入して、小数第1位までの数字にします。

```
SELECT 地区名, ROUND(最高気温,  1)  AS  最高気温
              (A)     (1)      (2)
  FROM 気象情報管理表
```

(1) 数値式

(2) 丸め位置

(A)～(2) ROUND式

気象情報管理表

地区ｺｰﾄﾞ	地区名	最高気温	最高気温日付	最低気温	最低気温日付
1111	札幌□□□	27.255	1978-07-14	-21.235	1994-01-11
2222	仙台□□□	30.000	1999-08-01	-8.500	1983-02-02
3333	東京□□□	35.554	1984-07-23	-0.456	1988-12-23
4444	横浜□□□	34.123	1988-08-01	0.789	1967-01-04
5555	大阪□□□	32.987	1995-08-11	6.357	1979-02-15
6666	広島□□□	31.567	1997-09-01	11.258	1990-02-06
7777	博多□□□	37.525	1985-06-30	17.369	1995-12-27

□：全角ブランクを示しています

⇩

結果

地区名	最高気温
札幌□□□	27.300
仙台□□□	30.000
東京□□□	35.600
横浜□□□	34.100
大阪□□□	33.000
広島□□□	31.600
博多□□□	37.500

例4

算術関数の使用方法例を以下に記載します。例では表Tの列C1に対して各関数を実行しています。

```
C1の値が1.0である場合、
SELECT ACOS( C1 ) FROM T  →0.000000E+00が返されます。
SELECT ASIN( C1 ) FROM T  →1.570796E+00が返されます。
SELECT ATAN( C1 ) FROM T  →7.853982E-01が返されます。
SELECT COS( C1 ) FROM T   →5.403023E-01が返されます。
SELECT SIN( C1 ) FROM T   →8.414710E-01が返されます。
SELECT TAN( C1 ) FROM T   →1.557408E+00が返されます。
SELECT LN( C1 ) FROM T    →0.000000E+00が返されます。
SELECT EXP( C1 ) FROM T   →2.718282E+00が返されます。
SELECT SIGN( C1 ) FROM T  →+1.が返されます。
SELECT SQRT( C1 ) FROM T  →1.000000E+00が返されます。
       (A)  (1)
```

(1) 数値式

(A)～(1) 算術関数

```
C1の値が2、C2の値が3である場合、
SELECT POWER( C1, C2 ) FROM T   →8.000000E+00が返されます。
       (A)   (1) (2)
```

(1) 数値式1

(2) 数値式2

(A)～(2) 算術関数

例5

複数の文字列を改行コードで連結します。例では表Tの列C1、C2、C3(いずれもデータ型は文字列型)を改行コードで連結します。

```
SELECT C1 || CHR(10) || C2 || CHR(10) || C3 FROM T
           (A) (1)
```

(1) 数値式

(A)～(1) CHR関数

9.4 文字列のデータを操作する

表のデータを操作するときに、文字列を操作したり、文字列と文字列を結合させたりすることができます。文字列の一部を操作する処理には以下のものがあります。

表9.8 文字列の一部を操作する処理

処理名	機能
文字部分列関数	文字部分列関数は、SUBSTRINGまたはOCTET_SUBSTRINGを指定して、文字値式の開始位置から、文字列長分の部分列を求めます。
POSITION式	POSITION式は、POSITIONまたはOCTET_POSITIONを指定して、1番目の文字値式の文字列を持つ2番目の文字値式中の文字の開始位置を求めます。
LENGTH式	LENGTH式は、データ列値式の文字数またはバイト数を求めます。
TRIM関数	TRIM関数は、指定された文字を取り除きます。
LTRIM関数	LTRIM関数は、TRIM元の前端から連続する空白を削除します。
RTRIM関数	RTRIM関数は、TRIM元の後端から連続する空白を削除します。
大文字小文字変換	大文字小文字変換は、UPPERまたはLOWERを指定して、文字を大文字または小文字に変換します。
連結	連結は、両項の文字列を結合します。
LPAD式	LPAD式は、文字値式の左側に、全体の文字列数が結果長になるまで、埋込み文字列をサイクリックに埋め込んで、結果長数分の文字列を返します。
RPAD式	RPAD式は、文字値式の右側に、全体の文字列数が結果長になるまで、埋込み文字列をサイクリックに埋め込んで、結果長数分の文字列を返します。
REPLACE式	REPLACE式はREPLACE元の文字列中に含まれる検索文字列のすべてを置換文字列に置き換えた結果を返却します。
REPLICATE式	REPLICATE式は、指定された結果文字数の範囲内で、指定された回数だけ、値式を繰り返します。
REVERSE式	REVERSE式は、値式を逆に並び替えたものを返します。
ASCII関数	ASCII関数は、文字値式の左端の1バイトのASCIIコードを返却します。

例1

会社表から、会社名および住所を取り出します。会社名にはうしろの空白文字を取り除いて“株式会社”を結合させ、住所は都道府県のみを取り出します。

特定の文字列の一部を取り出したり、特定の列の値と文字列を結合させて取り出すことができます。TRIM関数は、TRIM指定にLEADINGを指定した場合は値式の前端から、TRAILINGを指定した場合は後端から、BOTHを指定した場合は両端から文字を取り除きます。

連結は、複数の文字列を連結演算子(“||”)でつないで記述します。このとき、列のデータ型は文字列型または各国語文字列型であることが必要です。また、連結の両項のデータ型は文字列型または各国語文字列型であり、それらは比較可能であることが必要です。比較可能なデータ型については、“表9.5 比較可能なデータ型”を参照してください。

文字部分列関数は、SUBSTRINGを指定して、値式の開始位置から文字列長分の部分列を求めます。

```
SELECT  TRIM( TRAILING  FROM  会社名 ) || N'株式会社',
          (1)     (2)                    (A)

        SUBSTRING( 住所  FROM  1  FOR  4 )
           (B)        (3)          (4)     (5)

      FROM 在庫管理.会社表
```

(1) TRIM指定

(2) 値式

(1)～(2) TRIM関数

(1)～(A) 連結

(3) 値式

(4) 開始位置

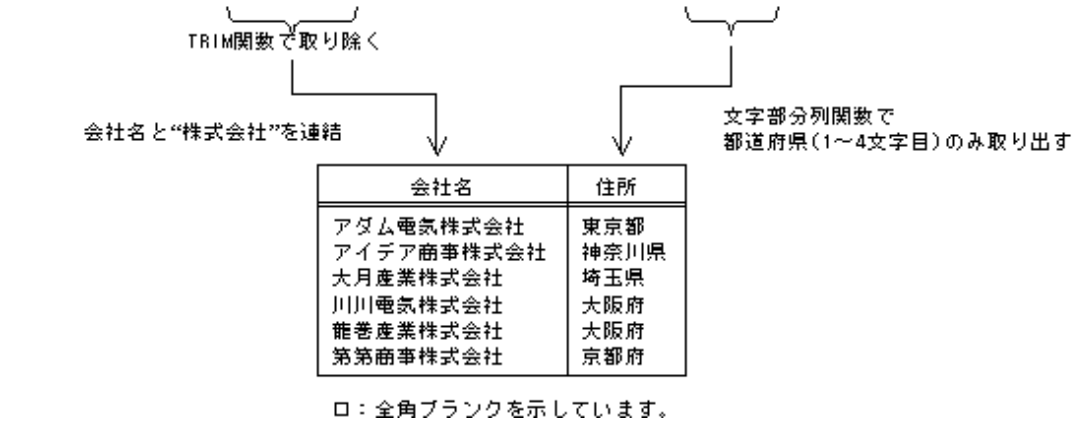
(5) 文字列長

(B)～(5) 文字部分列関数

図9.20 文字値関数を組み合わせてデータを取り出す例

会社表

会社番号	会社名	電　話　番　号	住　　所
６１	アダム電気□□□□□	１１１－７７７－４４４４	東京都　小田区　新蒲田　７－８－９
６２	アイデア商事□□□□	２２２－８８８－５５５５	神奈川県　小浜市　旭区　１－２－３
６３	大月産業□□□□□□	３３３－９９９－６６６６	埼玉県　浦和町　大崎　１－１－１
７１	川川電気□□□□□□	４４４－１１１－７７７７	大阪府　堺田市　赤坂台　４－５－６
７２	龍巻産業□□□□□□	５５５－２２２－８８８８	大阪府　灘町　東淀川　２－３－７
７４	第第商事□□□□□□	６６６－３３３－９９９９	京都府　中市　４－１６－１６



□：全角ブランクを示しています。

例2

営業所一覧表という表から、コードおよび営業所名を取り出します。このとき、営業所名は"営業所"という文字列を取り除いて取り出します。

```
SELECT コード,

    SUBSTRING(営業所名 FROM 1 FOR POSITION(N'営業所' IN 営業所名)-1)
                                    (A)        (1)            (2)

    FROM 在庫管理.営業所一覧表
```

(1) 値式

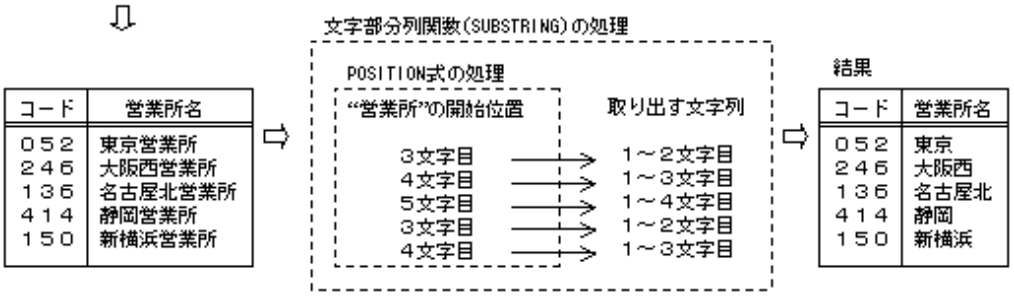
(2) 値式

(A)～(2) POSITION式

図9.21 文字部分列関数と数値関数を組み合わせてデータを取り出す例

営業所一覧表

コード	営業所名	処理日
０５２	東京営業所	２００７－０２－２８
２４６	大阪西営業所	２００７－０６－１０
１３６	名古屋北営業所	２００７－０９－１９
４１４	静岡営業所	２００７－１０－０１
１５０	新横浜営業所	２００７－１２－２０



例3

第二在庫表の製品名の文字列の最終3文字を取り出します。

```
SELECT コード, SUBSTRING(COL2 FROM CHAR_LENGTH(TRIM(COL2))-2) AS 製品略称,在庫数量
                                   (A)          (1)
          FROM  在庫管理.第二在庫表
```

(1) TRIM関数

(A)～(1) LENGTH式

図9.22 文字部分列関数と数値関数を組み合わせてデータを更新する例

第二在庫表

コード	製品名	在庫数量	倉庫番号
５０	ビデオテープロロロロ	８０	２
５５	ビデオデッキロロロロ	１０	２
７０	電子レンジロロロロロ	２５	１
９０	扇風機ロロロロロロロ	１００	１
１００	ビデオデッキロロロロ	２３	２

コード	製品略称	在庫数量
５０	テープ	８０
５５	デッキ	１０
７０	レンジ	２５
９０	扇風機	１００
１００	デッキ	２３

ロ：全角ブランクを示しています。

[ ]：　更新したデータを表します。

例4

支店名の左側に"*"を埋め込みます。全体の長さは、5文字とします。

```
SELECT 支店コード, LPAD(TRIM(TRAILING N'□' FROM   支店名), 5, N'*')
                   (B)   (A)     (1)                  (2)   (3)   (4)
  FROM 支店コード表
```

(1) TRIM指定

(2) 値式

(3) 結果長

(4) 埋込み文字列

(A)～(2) TRIM関数

(B)～(4) LPAD式

支店コード表

支店ｺｰﾄﾞ	支店名	支店所在地	電話番号
１１１	札幌□□□□□	札幌市中央区大通り西１－２□□□□□	０１１－９９９－９９９９
２２２	仙台□□□□□	仙台市青葉区一番町１－２－３□□□□	０２２－１１１－１１１１
３３３	六本木□□□□	東京都港区六本木７－８－９□□□□□	０３－３１２３－９８７６
４４４	新横浜□□□□	横浜市港北区新横浜１－２３－４５□□	０４５－９８７－６５４３
５５５	大阪□□□□□	大阪市中央区北浜８－９□□□□□□□	０６－６１２３－４５６７
６６６	博多□□□□□	福岡市博多区博多駅前１－２－３□□□	０９２－９８７－１２３４

□：全角ブランクを示しています

⇩

結果

支店ｺｰﾄﾞ	(注)
１１１	＊＊＊札幌
２２２	＊＊＊仙台
３３３	＊＊六本木
４４４	＊＊新横浜
５５５	＊＊＊大阪
６６６	＊＊＊博多

注) LPAD(TRIM(TRAILING N'□' FROM 支店名), 5, N'＊')の結果から作成される列です

例5

第二在庫表という表のCOL2の“ビデオ”という文字を“VIDEO”に置換えます。

```
UPDATE 在庫管理.第二在庫表
  SET  COL2 =  REPLACE(TRIM(COL2),  N'ビデオ',N'ＶＩＤＥＯ')
                 (A)      (1)              (2)          (3)
```

(1) REPLACE元

(2) 検索文字列

(3) 置換文字列

(A)～(3) REPLACE式

第二在庫表

COL1	COL2	COL3	COL4
５０	ビデオテープ□□□□	８０	２
５５	ビデオデッキ□□□□	１０	２
７０	電子レンジ□□□□□	２５	１
９０	扇風機□□□□□□□	１００	１
１００	ビデオデッキ□□□□	２３	２

□：全角ブランクを示しています

⇩

結果

COL1	COL2	COL3	COL4
５０	ＶＩＤＥＯテープ□□	８０	２
５５	ＶＩＤＥＯデッキ□□	１０	２
７０	電子レンジ□□□□□	２５	１
９０	扇風機□□□□□□□	１００	１
１００	ＶＩＤＥＯデッキ□□	２３	２

9.5 日付のデータを操作する

表のデータを操作するときに、日付や時間などの演算を行ったり、日時値式のデータを文字に変換したり、文字型のデータをDATE型に変換することができます。日付のデータを操作する処理には以下のものがあります。

表9.9 日付のデータを操作する処理

処理名	機能
CURRENT DATE値関数	現在の日付を返却します。
CURRENT TIME値関数	現在の時刻を返却します。
CURRENT TIMESTAMP値関数	現在の時刻印を返却します。
ROUND_DATE関数	日付を指定した日時フィールドで丸めます。
TRUNC_DATE関数	日付を指定した日時フィールドで切捨てます。
SPAN_DATE関数	2つの日付間の時間隔を、指定した時間隔の単位で返却します。
ADD_DATE関数	DATE型に、数値属性で表現した時間隔を加算した結果の日時を返却します。
LAST_DAY関数	当該月の最終日付を返却します。
EXTRACT式	抜き出し元の日時値式および時間隔値式から、抜き出しフィールドで指定された日時フィールドに相当する部分を抽出します。
CNV_CHAR関数	日時値式のデータを指定された書式に従って文字型に変換します。
CNV_DATE関数	指定された日付書式で記述された文字型データをDATE型に変換します。
CNV_TIME関数	指定された時刻書式で記述された文字型データをTIME型に変換します。
CNV_TIMESTAMP関数	指定された時刻印書式で記述された文字型データをTIMESTAMP型に変換します。

例1

発注表から四半期ごとの発注数量の平均を求めます。

TRUNC_DATE関数により、日付を切り捨てることができます。切り捨ては、日付を年で切り捨て(当年の1月1日)、日付を月で切り捨て(当月の1日)や日付を四半期で切り捨てなどの単位で行うことができます。

```
SELECT 四半期,AVG(発注数量) AS 平均発注数量
  FROM 発注表
  GROUP BY TRUNC_DATE(発注日,'QUARTER') AS 四半期
             (A)       (1)      (2)
```

(1) 日付

(2) 切り捨ての単位

(A)～(2) TRUNC_DATE関数

発注表

取引先	発注番号	発注数量	発注日	納品日
61	123	60	2007-07-14	2007-10-20
62	124	40	2007-08-01	2007-11-20
63	140	80	2007-07-10	2007-07-20
71	215	10	2007-08-01	2007-11-20
72	240	20	2007-05-11	2007-08-20
74	110	120	2007-09-01	2007-12-20
74	226	20	2007-06-30	2007-09-20

⇩

GROUP BY句で導出される表

取引先	発注番号	発注数量	発注日	納品日	
72	240	20	2007-05-11	2007-08-20	1グループ：当年の4月1日～6月30日
74	226	20	2007-06-30	2007-09-20	
61	123	60	2007-07-14	2007-10-20	1グループ：当年の7月1日～9月30日
62	124	40	2007-08-01	2007-11-20	
63	140	80	2007-07-10	2007-07-20	
71	215	10	2007-08-01	2007-11-20	
74	110	120	2007-09-01	2007-12-20	

⇩

結果

四半期	平均発注数量
2007-04-01	20
2007-07-01	62

例2

発注表から現在の日付までの経過日数を求めます。現在の日付は、2007-08-10とします。

SPAN_DATE関数により、日付間の差(時間隔)を求めることができます。時間隔の単位には、時間隔の単位としてYEAR、MONTHまたはDAYを指定することができ、日付間の年、月または日の変わる回数が求められます。

```
SELECT 発注番号, SPAN_DATE(発注日,CURRENT_DATE,'DAY') AS 経過日数
                 (A)      (1)       (2)         (3)        (B)
  FROM 発注表
  WHERE 納品日 IS NULL
```

(1) 日付

(2) 日付

(3) 時間隔の単位

(A)～(B) SPAN_DATE関数

発注表

取引先	発注番号	発注数量	発注日	納品日	
61	123	60	2007-07-14	2007-10-20	
62	124	40	2007-08-01		← NULL
63	140	80	2007-07-10	2007-07-20	
71	215	10	2007-08-01	2007-11-20	
72	240	20	2007-05-11		← NULL
74	110	120	2007-09-01	2007-12-20	
74	226	20	2007-06-30	2007-09-20	

⇩

結果

発注番号	経過日数
124	9
240	91

例3

発注表の発注日から1ヶ月以上経過している発注番号を求めます。現在の日付は、2007-08-01とします。

ADD_DATE関数により、日付を加算することができます。時間隔の単位には、加算する単位としてYEAR、MONTHまたはDAYを指定することができ、加算数を年数、月数、日数として加算することができます。

```
SELECT 発注番号
  FROM 発注表
  WHERE 納品日 IS NULL
    AND ADD_DATE(発注日, 1,'MONTH') < CURRENT_DATE
          (A)       (1)  (2)  (3)
```

(1) 日付

(2) 加算数

(3) 時間隔の単位

(A)～(3) ADD_DATE関数

発注表

取引先	発注番号	発注数量	発注日	納品日	
61	123	60	2007-07-14	2007-10-20	
62	124	40	2007-08-01		← NULL
63	140	80	2007-07-10	2007-07-20	
71	215	10	2007-08-01	2007-11-20	
72	240	20	2007-05-11		← NULL
74	110	120	2007-09-01	2007-12-20	
74	226	20	2007-06-30	2007-09-20	

⇩

結果

発注番号
240

例4

発注日から月末までの日数を求めます。

LAST_DAY関数により、当該月の最終日付を求めることができます。

```
SELECT 発注番号,SPAN_DATE(発注日, LAST_DAY(発注日), 'DAY')  AS 月末までの残日数
  FROM 発注表                          (A)      (1)
  WHERE 納品日 IS NULL
```

(1) 日付

(A)～(1) LAST_DAY関数

発注表

取引先	発注番号	発注数量	発注日	納品日
61	123	60	2007-07-14	2007-10-20
62	124	40	2007-08-01	← NULL
63	140	80	2007-07-10	2007-07-20
71	215	10	2007-08-01	2007-11-20
72	240	20	2007-05-11	← NULL
74	110	120	2007-09-01	2007-12-20
74	226	20	2007-06-30	2007-09-20

⇩

結果

発注番号	月末までの残日数
124	30
240	20

例5

気象情報管理表の最高気温日付のデータを'YYYY.MM.DD DY'の文字列に変換します。CNV_CHAR関数により、日付型を文字型に変換することができます。変換は、日付書式に従って行われます。

```
SELECT 地区コード , CNV_CHAR(最高気温日付, 'YYYY.MM.DD DY') AS 最高気温日
                    (A)          (1)          (2)
  FROM 気象情報管理表
```

(1) 変換元

(2) 日付書式

(A)～(2) CNV_CHAR関数

気象情報管理表

地区ｺｰﾄﾞ	地区名	最高気温	最高気温日付	最低気温	最低気温日付
1111	札幌□□□	27.255	1978-07-14	-21.235	1994-01-11
2222	仙台□□□	30.000	1999-08-01	-8.500	1983-02-02
3333	東京□□□	35.554	1984-07-23	-0.456	1988-12-23
4444	横浜□□□	34.123	1988-08-01	0.789	1967-01-04
5555	大阪□□□	32.987	1995-08-11	6.357	1979-02-15
6666	広島□□□	31.567	1997-09-01	11.258	1990-02-06
7777	博多□□□	37.525	1985-06-30	17.369	1995-12-27

□：全角ブランクを示しています

⇩

結果

地区ｺｰﾄﾞ	最高気温日
1111	1978.07.14 FRI
2222	1999.08.01 SUN
3333	1984.07.23 MON
4444	1988.08.01 MON
5555	1995.08.11 FRI
6666	1997.09.01 MON
7777	1985.06.30 SUN

例6

気象情報管理表の最低気温日付を指定した日付書式にしたがった文字型に変換します。なお、CHAR型のホスト変数“:DAYFMT”には、日付書式“DAY MONTH DD,YYYY”が設定されているとします。

```
SELECT 地区コード , CNV_CHAR(最低気温日付, :DAYFMT) AS 最低気温日
  FROM 気象情報管理表
```

気象情報管理表

地区ｺｰﾄﾞ	地区名	最高気温	最高気温日付	最低気温	最低気温日付
1111	札幌□□□	27.255	1978-07-14	-21.235	1994-01-11
2222	仙台□□□	30.000	1999-08-01	-8.500	1983-02-02
3333	東京□□□	35.554	1984-07-23	-0.456	1988-12-23
4444	横浜□□□	34.123	1988-08-01	0.789	1967-01-04
5555	大阪□□□	32.987	1995-08-11	6.357	1979-02-15
6666	広島□□□	31.567	1997-09-01	11.258	1990-02-06
7777	博多□□□	37.525	1985-06-30	17.369	1995-12-27

□：全角ブランクを示しています

⇩

結果

地区ｺｰﾄﾞ	最低気温日
1111	Tuesday January 11, 1994
2222	Wednesday February 02, 1983
3333	Friday December 23, 1988
4444	Wednesday January 04, 1967
5555	Thursday February 15, 1979
6666	Tuesday February 06, 1990
7777	Wednesday December 27, 1995

例7

最低気温日付文字表の日付文字を日付に変換します。

CNV_DATE関数により、日付書式で記述された文字型のデータを日時型にすることができます。

```
SELECT 地区コード , CNV_DATE(最低気温日, 'DAY MONTH DD,YYYY') AS 日付
                    (A)       (1)          (2)
  FROM 最低気温日付文字表
```

(1) 変換元

(2) 日付書式

(A)～(2) CNV_DATE関数

最低気温日付文字表

地区ｺｰﾄﾞ	最低気温日
1111	Tuesday January 11, 1994
2222	Wednesday February 02, 1983
3333	Friday December 23, 1988
4444	Wednesday January 04, 1967
5555	Thursday February 15, 1979
6666	Tuesday February 06, 1990
7777	Wednesday December 27, 1995

⇩

結果

地区ｺｰﾄﾞ	日付
1111	1994-01-11
2222	1983-02-02
3333	1988-12-23
4444	1967-01-04
5555	1979-02-15
6666	1990-02-06
7777	1995-12-27

例8

在庫管理表の更新時刻文字を時刻に変換します。

CNV_TIME関数により、時刻書式で記述された文字型のデータを日時型にすることができます。

```
SELECT 商品コード , CNV_TIME(更新時刻文字, 'ss:mm:hh24') AS 更新時刻
                    (A)       (1)          (2)
  FROM 在庫管理表
```

(1) 変換元

(2) 時刻書式

(A)～(2) CNV_TIME関数

在庫管理表

商品ｺｰﾄﾞ	更新時刻文字
1111	14:02:05
2222	42:38:08
3333	40:12:12
4444	32:38:15
5555	21:08:18
6666	50:42:21
7777	44:22:23

⇩

結果

商品ｺｰﾄﾞ	更新時刻
1111	05:02:14
2222	08:38:42
3333	12:12:40
4444	15:38:32
5555	18:08:21
6666	21:42:50
7777	23:22:44

例9

会議室予約表の使用日時を時刻印に変換します。

CNV_TIMESTAMP関数により、時刻印書式で記述された文字型のデータを日時型にすることができます。

```
SELECT 予約番号 , CNV_TIMESTAMP(使用日時文字, 'MM-DD-YYYY hh24:mm:ss') AS 使用日時
                    (A)        (1)          (2)
  FROM 会議室予約表
```

(1) 変換元

(2) 時刻印書式

(A)～(2) CNV_TIMESTAMP関数

会議室予約表

予約番号	使用日時文字
1111	11-24-2007 08:00:00
2222	11-25-2007 09:15:10
3333	11-26-2007 10:30:25
4444	11-27-2007 11:45:55
5555	11-28-2007 13:01:40
6666	11-29-2007 14:16:41
7777	11-30-2007 15:31:57

⇩

結果

予約番号	使用日時
1111	2007-11-24 08:00:00
2222	2007-11-25 09:15:10
3333	2007-11-26 10:30:25
4444	2007-11-27 11:45:55
5555	2007-11-28 13:01:40
6666	2007-11-29 14:16:41
7777	2007-11-30 15:31:57

9.6 データ型を変換してデータを操作する

CAST指定を使用して、列のデータ型を変換することにより、文字列を用いて日付や時間などの演算を行ったり、日時型や時間隔型の列にデータの追加や更新を行うことができます。

文字列のデータ型を時間隔型(年月)に変換して、日付の演算を行う例を以下に示します。

例1

営業所一覧表から、コード、および処理日に6ヵ月加算した日付を取り出します。なお、文字列型のホスト変数“CDATE”には“6”が設定されているとします。

```
SELECT コード, 処理日 + CAST(:CDATE AS INTERVAL MONTH)  AS 加算後日付
                        (A)   (1)          (2)
      FROM 在庫管理.営業所一覧表
```

(1) 値式

(2) データ型

(A)～(2) CAST指定

図9.23 文字列型から時間隔型(年月)へのデータ変換

営業所一覧表

コード	営業所名	処理日
052	東京営業所	2006-02-28
246	大阪西営業所	2006-06-10
136	名古屋北営業所	2006-09-19
414	静岡営業所	2006-10-01
150	新横浜営業所	2006-12-20

⇨

コード	加算後日付
052	2006-08-28
246	2006-12-10
136	2007-03-19
414	2007-04-01
150	2007-06-20

文字列のデータ型を時間隔型(日時)に変換して、時間の演算を行う例を以下に示します。

例2

日別発注表という表の発注数量が“10”未満のデータについて、処理時間を1時間30分減算した値に更新します。なお、文字列型のホスト変数“OTIME”には、“1:30”が設定されているとします。

```
UPDATE 在庫管理.日別発注表

      SET 処理時間 = 処理時間 - CAST(:OTIME AS INTERVAL HOUR TO MINUTE)

      WHERE 発注数量 < 10
```

図9.24 文字列型から時間隔型(日時)へのデータ変換

日別発注表（データの更新前）

取引先	取引製品	発注数量	処理時間
74	124	10	09:40:33
61	351	100	10:26:05
59	111	5	13:01:58
63	240	12	15:54:20
63	351	500	15:55:49

日別発注表（データの更新後）

取引先	取引製品	発注数量	処理時間
74	124	10	09:40:33
61	351	100	10:26:05
59	111	5	11:31:58
63	240	12	15:54:20
63	351	500	15:55:49

：更新したデータを表します。

数値のデータ型を文字列型に変換して、文字列と結合する例を以下に示します。

例3

発注表から、取引先が“61”のデータについて、取引製品および仕入価格を取り出します。このとき、仕入価格の先頭には“¥”を結合させます。

```
SELECT 取引製品, '¥' || CAST(仕入価格  AS  CHARACTER(6))

       FROM 在庫管理.発注表 WHERE 取引先 = 61
```

図9.25 数値型から文字列型へのデータ変換

発注表

取引先	取引製品	仕入価格	発注数量
61	123	48000	60
61	124	64000	40
61	140	8000	80
61	215	240000	10
61	240	80000	20
62	110	37500	120
62	226	112500	20
～	～	～	～
74	226	117000	20
74	227	140400	10
74	351	390	700

⇨

取引製品	仕入価格
123	¥48000□
124	¥64000□
140	¥8000□□
215	¥240000
240	¥80000□

□：全角ブランクを示しています。

9.7 CASE式を使用してデータを操作する

CASE式は、列やホスト変数などの値に応じて結果列の値の変更、取り出す列の優先付けをすることができます。また、場合分けをして、それぞれに対応した値に設定することもできます。

CASE式の使用例を以下に示します。

例1

履歴表から、利用者ごとに利用したサービスの回数を求めます。

```
SELECT 利用者, COUNT(CASE WHEN サービス    = 1
                          THEN サービス    ELSE NULL END) AS サービス1
             , COUNT(CASE WHEN サービス    = 2
                          THEN サービス    ELSE NULL END) AS サービス2
             , COUNT(CASE WHEN サービス    = 3
                          THEN サービス    ELSE NULL END) AS サービス3
      FROM 履歴表
      GROUP BY 利用者
```

履歴表

履歴番号	利用者	サービス
1	USER-A	1
2	USER-B	2
3	USER-A	2
4	USER-A	1
5	USER-C	3

⇨

利用者	サービス1	サービス2	サービス3
USER-A	2	1	0
USER-B	0	1	0
USER-C	0	0	1

例2

ポイント表から、ポイントに応じて記号(PLATINUM、GOLDまたはSILVER)を付加した利用者と合計ポイントを求めます。

```
SELECT 利用者, 合計ポイント,
  CASE WHEN 合計ポイント >= 50000 THEN 'PLATINUM'
         WHEN 合計ポイント >= 10000 THEN 'GOLD'
         WHEN 合計ポイント >= 3000  THEN 'SILVER'
    ELSE        NULL END AS ランク
```

```
  FROM ポイント表
```

ポイント表

利用者	合計ポイント
USER-A	3240
USER-B	10720
USER-C	2310
USER-D	54590

⇨

利用者	合計ポイント	ランク
USER-A	3240	SILVER
USER-B	10720	GOLD
USER-C	2310	-
USER-D	54590	PLATINUM

-：NULL値

例3

利用者表から、メールアドレス、TEL番号、住所の優先順で最初の設定のある項目を求めます。

```
SELECT 利用者, CASE WHEN メール IS NOT NULL THEN メール
                    WHEN TEL    IS NOT NULL THEN TEL
                    ELSE                         住所 END AS 連絡先
  FROM 利用者表
```

利用者表

利用者	名前	愛称	メール	TEL	住所
USER-1	TANAKA	TANAKA	E-MAIL-A	-	OSAKA
USER-2	SUZUKI	NAME-B	-	TEL-B	TOKYO
USER-3	KATOU	NAME-C	E-MAIL-C	TEL-C	KYUSYU

-：NULL値

⇩

利用者	連絡先
USER-1	E-MAIL-A
USER-2	TEL-B
USER-3	E-MAIL-C

例4

ポイント表から、次のランクのポイントにあと500ポイントで到達する利用者と合計ポイントを求めます。それぞれのランクは、SILVERが3000ポイント、GOLDが10000ポイント、PLATINUMが50000ポイントとします。

```
SELECT 利用者, 合計ポイント FROM  ポイント表
  WHERE 合計ポイント >= CASE WHEN 合計ポイント <= 3000 THEN 2500
                             WHEN 合計ポイント <=10000 THEN 9500
                             WHEN 合計ポイント <=50000 THEN 49500 END
```

ポイント表

利用者	合計ポイント
USER-A	2780
USER-B	10720
USER-C	2310
USER-D	54590
USER-E	9830

⇨

利用者	合計ポイント
USER-A	2780
USER-E	9830

例5

履歴表から、利用したサービスをグループ化してグループごとの総利用回数を求めます。

```
SELECT サービスグループ, COUNT(*)  AS 総利用回数
FROM 履歴表
GROUP BY   CASE WHEN サービス    IN (1,4,7) THEN 'サービスグループ１'
                WHEN サービス    IN (2,5,8) THEN 'サービスグループ２'
                ELSE 'サービスグループ３'   END  AS サービスグループ
```

履歴表

履歴番号	利用者	サービス
1	USER-A	1
2	USER-B	2
3	USER-A	5
4	USER-A	7
5	USER-C	3

⇨

サービスグループ	総利用回数
サービスグループ１	2
サービスグループ２	2
サービスグループ３	1

例6

履歴表に記述されたサービス記号を変更します。

```
UPDATE 履歴表
  SET サービス =   CASE WHEN サービス  =  1  THEN  111
                        WHEN サービス  =  2  THEN  222
                        WHEN サービス  =  3  THEN  333
                        ELSE 999                      END
```

履歴表

履歴番号	利用者	サービス
1	USER-A	1
2	USER-B	2
3	USER-A	5
4	USER-A	7
5	USER-C	3

⇨

履歴番号	利用者	サービス
1	USER-A	111
2	USER-B	222
3	USER-A	999
4	USER-A	999
5	USER-C	333

例7

履歴表にデータを1行追加します。履歴番号はホスト変数“:RIREKI_NO”の値、利用者はホスト変数“RIYOUSYA”の値、サービスはホスト変数“SERVICE”に対応するコードを割り振るCASE式の結果の値とします。ここでは、ホスト変数“RIREKI_NO”には“6”、“RIYOUSYA”には“USER-C”、“SERVICE”には“宿泊”が設定されているとします。CASE式により、“SERVICE”の値である“宿泊”は4に変換してから履歴表に追加します。

```
INSERT INTO 履歴表(履歴番号,利用者,サービス)
  VALUES(:RIREKI_NO,:RIYOUSYA,
          CASE WHEN :SERVICE = N'飲食' THEN 1
                WHEN :SERVICE = N'ショッピング' THEN 2
                WHEN :SERVICE = N'公共料金' THEN 3
                WHEN :SERVICE = N'宿泊' THEN 4
                WHEN :SERVICE = N'交通' THEN 5
                WHEN :SERVICE = N'給油' THEN 6
                WHEN :SERVICE = N'通販' THEN 7
                WHEN :SERVICE = N'移行' THEN 8
                ELSE 9                    END)
```

履歴表

履歴番号	利用者	サービス
1	USER－A	1
2	USER－B	2
3	USER－A	5
4	USER－A	7
5	USER－C	3

⇨

履歴番号	利用者	サービス
1	USER－A	1
2	USER－B	2
3	USER－A	5
4	USER－A	7
5	USER－C	3
6	USER－C	4

[ ] ： 追加したデータを表します。

9.8 行識別子を指定してデータを操作する

行識別子“ROW_ID”は、データベース中の任意の表の行を一意に認識します。単一行SELECT文またはFETCH文により、行識別子を取り出します。行識別子を受け取るホスト変数はROW_IDと対応する変数であることが必要です。

参照

データ型と対応する変数定義については、“SQLリファレンス”を参照してください。

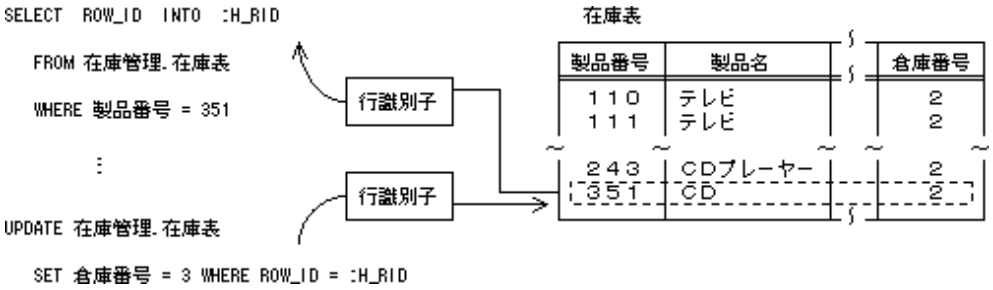
取り出した行識別子は、以下の探索条件に指定して任意の行を操作することができます。

- UPDATE文:探索の探索条件
- DELETE文:探索の探索条件
- 問合せ指定のWHERE句の探索条件

行識別子の指定例を以下に示します。

例

行識別子を使用してデータの更新を行います。



9.9 並列クエリを利用してデータを操作する

定型的な業務のようにデータ検索のキーとなる項目が明確である場合、インデックスを設定するなどの手段を使って検索効率の向上を図ることができます。一方、定型業務で使用するデータベースを夜間バッチ処理で帳票出力する場合、検索時間が長くかかることがあります。そのようなバッチ処理を効率的に行うためには、データベースを複数のDSIに分割し、それぞれを並列に検索するようにします。これを並列クエリと呼びます。

注意

データベース簡単運用では、並列クエリを利用できません。

並列クエリを実行するためには、以下の指定を行います。

- 並列クエリを利用する範囲によって以下のいずれかを指定します。

  ー カーソル宣言に並列指定“PARALLEL”を指定します。

  ー 動作環境ファイルの“PARALLEL_SCAN”にYESを指定します。

  並列クエリを利用する範囲と指定方法の関係を以下に示します。

並列クエリ実行の範囲	指定方法
SQL文単位	カーソル宣言に並列指定“PARALLEL”を指定します。
コネクション単位	サーバ用の動作環境ファイルの“PARALLEL_SCAN”にYESを指定します。
アプリケーション単位	クライアント用の動作環境ファイルの“PARALLEL_SCAN”にYESを指定します。

  優先順位は、SQL文単位、コネクション単位、アプリケーション単位の順になります。

- 動作環境ファイルの“SQL_LEVEL”にSQL96、SQL2000またはSQL2007を指定します。
- 動作環境ファイルの“MAX_PARALLEL”に並列の多重度を指定します。

参照

データベースを複数のDSIに分割する方法については“RDB運用ガイド(データベース定義編)”を、動作環境ファイルについては“付録B 動作環境ファイルのパラメタ一覧”をそれぞれ参照してください。

並列指定の指定例を以下に示します。

例

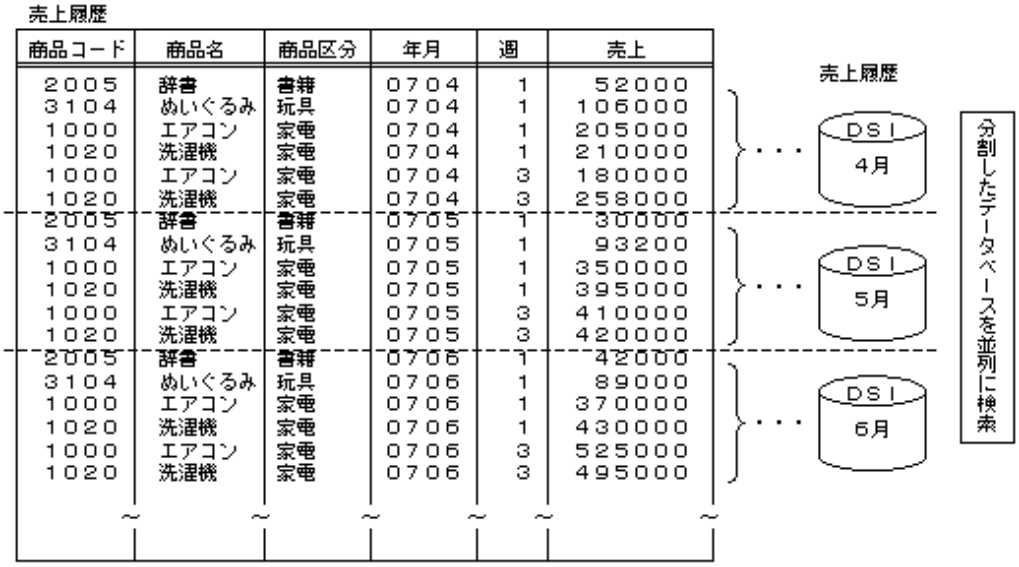
売上履歴から、商品区分が家電の商品の、商品コードと年月ごとの売上の合計を求めます。

アプリケーション

```
DECLARE CSR1 CURSOR FOR
  SELECT 商品コード, 年月, SUM(売上) FROM 売上履歴
    WHERE 商品区分 IN(N'家電') GROUP BY 商品コード, 年月 PARALLEL
      :                                                  並列指定
OPEN  CSR1
      :
FETCH CSR1 INTO :H1, :H2, H3
      :
```

動作環境ファイル

```
SQL_LEVEL = SQL96
MAX_PARALLEL = 3
```



⇩

WHERE句で導出される表

商品コード	商品名	商品区分	年月	週	売上
1000	エアコン	家電	0704	1	205000
1020	洗濯機	家電	0704	1	210000
1000	エアコン	家電	0704	3	180000
1020	洗濯機	家電	0704	3	258000
1000	エアコン	家電	0705	1	350000
1020	洗濯機	家電	0705	1	395000
1000	エアコン	家電	0705	3	410000
1020	洗濯機	家電	0705	3	420000
1000	エアコン	家電	0706	1	370000
1020	洗濯機	家電	0706	1	430000
1000	エアコン	家電	0706	3	525000
1020	洗濯機	家電	0706	3	495000

⇩

GROUP BY句で導出される表

商品コード	商品名	商品区分	年月	週	売上
1000	エアコン	家電	0704	1	205000
1000	エアコン	家電	0704	3	180000
1000	エアコン	家電	0705	1	350000
1000	エアコン	家電	0705	3	410000
1000	エアコン	家電	0706	1	370000
1000	エアコン	家電	0706	3	525000
1020	洗濯機	家電	0704	1	210000
1020	洗濯機	家電	0704	3	258000
1020	洗濯機	家電	0705	1	395000
1020	洗濯機	家電	0705	3	420000
1020	洗濯機	家電	0706	1	430000
1020	洗濯機	家電	0706	3	495000

⇩

問合せ指定で導出される表

商品コード	年月	(注)
1000	0704	385000
1000	0705	760000
1000	0706	895000
1020	0704	468000
1020	0705	815000
1020	0706	925000

注) 集合関数の結果の列には名前がありません。

9.10 スキーマ名を省略する

これまでの例文では、SQL文中に表名を指定する場合に“在庫管理.在庫表”のように表名をスキーマ名で修飾していました。本節では、表宣言を使用することにより、SQL文中で表名を指定するときにスキーマ名による修飾を省略する方法について説明します。

表宣言では、アプリケーション中で使用する表名と、その表が含まれるスキーマ名を指定します。表宣言を指定しておくと、それ以降で記述するSQL文では、表名を指定するときにスキーマ名修飾を省略することができます。表宣言の例を以下に示します。

例

表名の指定におけるスキーマ名の修飾を省略できるようにするため、在庫表、発注表および会社表を表宣言します。

```
DECLARE 在庫表,  発注表,  会社表  TABLE ON  在庫管理
          (1)      (2)      (3)                (4)

                :
SELECT 在庫数量 INTO :STOCKQTY FROM 在庫表 WHERE 製品番号 = 215
                :
```

(1) 表名

(2) 表名

(3) 表名

(4) スキーマ名

9.11 現行セションの利用者を変更する

アプリケーションでデータベースをアクセスする場合に、現行セションの利用者はデータベースにコネクトした人となります。アプリケーション中でSET SESSION AUTHORIZATION文を指定することにより、この利用者を変更することができます。

例

現行セションの利用者を“USER2”に変更します。

```
CONNECT TO 'RDBDB1' USER 'USER1/PASS1'
          :
SELECT * FROM SCM.TBL
          :
COMMIT WORK

SET SESSION AUTHORIZATION 'USER2/PASS2'
          :
SELECT * FROM SCM.TBL
          :
```

9.12 問合せ指定の結果列に名前を付加する

問合せ指定で選択リストに関数を指定した場合、導出される列には名前が付加されません。名前を付加したい場合は、AS句で列名を付加することができます。指定例を以下に示します。

例

在庫表から製品名ごとの在庫数量の合計を取り出します。このとき、問合せ指定で導出される表で、在庫数量の合計の列に名前を付加します。

```
SELECT 製品名, SUM(在庫数量)  AS  在庫合計

    FROM 在庫管理.在庫表 GROUP BY 製品名
```

GROUP BY句で導出される表

製品番号	製品名	在庫数量	倉庫番号	
110	テレビ	85	2	1グループ
111	テレビ	90	2	
212	テレビ	0	2	
215	ビデオ	5	2	1グループ
123	冷蔵庫	60	1	1グループ
124	冷蔵庫	75	1	
226	冷蔵庫	8	1	
227	冷蔵庫	15	1	
140	CDプレーヤー	120	2	1グループ
240	CDプレーヤー	25	2	
243	CDプレーヤー	14	2	
351	CD	2500	2	1グループ

⇩

問合せ指定で導出される表

製品名	在庫合計(注)
テレビ	175
ビデオ	5
冷蔵庫	158
CDプレーヤー	159
CD	2500

注) 集合関数の結果の列には名前が付加されます。

9.13 順序を使用してデータを操作する

作成した順序に対して、CURRVALおよびNEXTVALを使用して順序番号を取り出すことができます。CURRVALは、アプリケーション内の現在の順序番号を取り出します。NEXTVALは、RDBシステム内で取り出した最後の順序番号の次の値を取り出します。

CURRVALおよびNEXTVALはアプリケーションおよびストアドプロシジャの以下のSQL文の箇所に指定できます。

- 単一行SELECT文の選択リスト
- 問合せ指定の選択リスト:カーソル指定の問合せ式のみ
- UPDATE文のSET句
- INSERT文に指定した問合せ指定の選択リスト
- INSERT文のVALUES句
- ストアドプロシジャのSET文
- 被トリガSQL文

なお、CURRVALおよびNEXTVALは、以下のSQL文の箇所には指定できません。

- WHERE句の探索条件
- 副問合せの選択リスト
- ALTER TABLE ADD COLUMNのDEFAULT句

SELECT文の場合、順序番号は検索結果の行に対して採番されます。

UPDATE文の場合は、更新行に対して採番されます。また、INSERT文の場合は、挿入先の挿入行に対して採番されます。いずれの場合も、結果が複数件ある場合は、結果の各行に対して採番されます。

セション内で最初に順序番号を取り出す場合は、NEXTVALで順序番号を取り出してください。

1つのSQL文中で、同じ順序のNEXTVALを複数個指定している場合は、すべて同じ順序番号が返却されます。また、1つのSQL文中で、同じ順序のNEXTVALとCURRVALの両方を指定している場合は、それらが文中で出現する順番にかかわらず、NEXTVALとCURRVALの両方に対して、NEXTVALとして採番された同じ順序番号が返却されます。

例1

アプリケーションのSQL文中に順序を指定する方法を示します。従業員番号を増加させ、新しい従業員のためにその値を割り振ります。

順序の定義

```
CREATE SEQUENCE 会社管理.順序１
       INCREMENT BY 1
       START WITH 1
```

アプリケーションのSQL文中の順序の指定

```
INSERT INTO  会社管理.従業員表

       VALUES （ 会社管理.順序１.NEXTVAL ， "山田太郎",200000, "企画部" ）
                          (1)
```

(1) 順序

従業員表

従業員番号	従業員名	給与	部署	
1	青木　一郎	300000	勤労課	
2	富士　花子	200000	総務課	
3	山田　太郎	200000	企画部	⇦ 挿入行

例2

順序を使用して挿入行に自動的に順序番号を入れる方法を示します。受注表の受注番号を増加させ、新しい受注番号を割り振ります。

挿入行に自動的に順序番号を入れるには、表定義の列にDEFAULT句で順序を指定します。アプリケーションのINSERT文では、DEFAULTに順序を指定した列を省略するか、挿入値にDEFAULTと指定します。

```
順序の定義
CREATE SEQUENCE 在庫管理.順序１
      INCREMENT BY 1
      START WITH 1
      MAXVALUE 10000

表の定義

CREATE TABLE  在庫管理.受注表(
      受注番号 INTEGER DEFAULT 在庫管理.順序１.NEXTVAL  NOT NULL ,
                                        (1)
      顧客コード DECIMAL(5,0) NOT NULL,
      商品番号   CHAR(4) NOT NULL,
      数量       INTEGER)


アプリケーションのSQL文の指定

INSERT INTO  在庫管理.受注表(顧客コード, 商品番号,数量)

      VALUES ( 12345,'A002',5)
```

(1) 順序

受注表

受注番号	顧客コード	商品番号	数量	
1	12345	A002	5	⇦ 追加の行

DEFAULT句の順序番号

例3

在庫管理.順序1の順序番号を、CURRVALおよびNEXTVALを指定して参照する例を示します。

```
CURRVALを指定して現在の順序番号を参照します。

SELECT 在庫管理.順序１.CURRVAL
               (1)
      FROM RDBII_SYSTEM.RDBII_ASSISTTABLE   (注)

NEXTVALを指定して次の順序番号を参照します。

SELECT 在庫管理.順序１.NEXTVAL
               (2)
      FROM RDBII_SYSTEM.RDBII_ASSISTTABLE   (注)
```

(1) 順序

(2) 順序

注) RDBII_ASSISTTABLEは、SQL文を簡易的に実行するための表です。この表には1の値を持つデータが1件格納されています。選択リストに日時値関数、順序などを記述した単一行SELECT文を実行して、現在の時刻、CURRVAL値などを取得する場合に利用できます。

9.14 LIMITを使用して結果件数を限定する

LIMITを使用して、結果として取り出す件数を限定することができます。

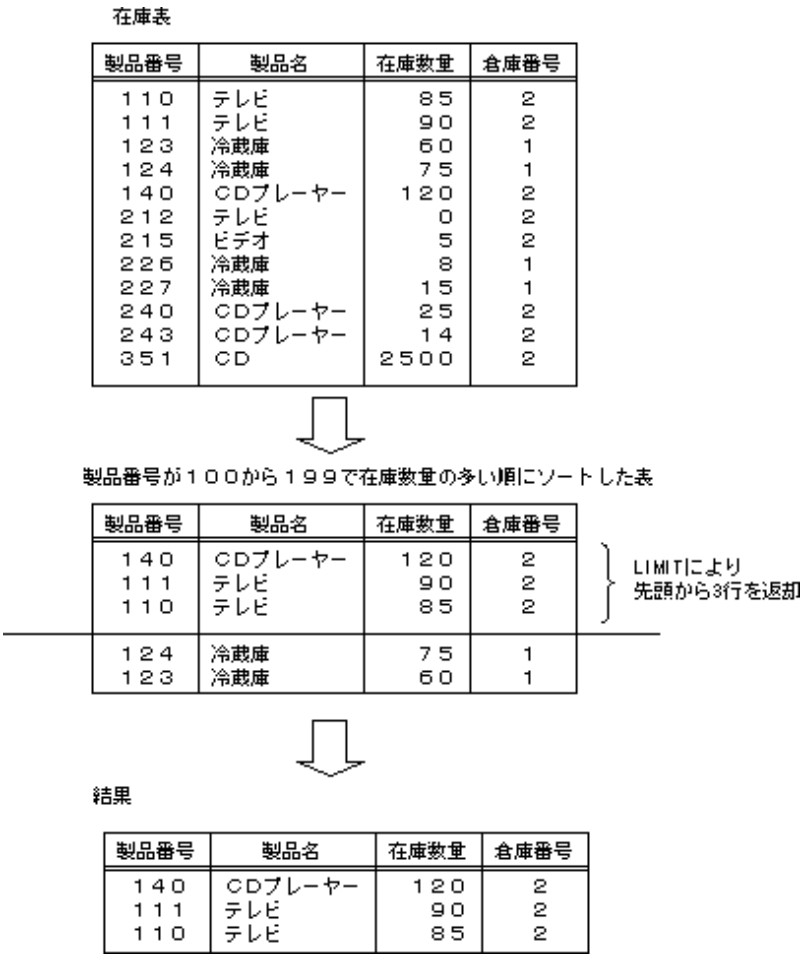
LIMITは、カーソル宣言または単一行SELECT文に指定します。

カーソル宣言にORDER BY句を指定すると、ソートされたカーソル表の先頭から、LIMITで指定した件数分を結果として取り出すことができます。

例

在庫表から、製品番号が100から199で在庫数量が多い順に3行取り出します。

```
SELECT * FROM 在庫管理.在庫表
        WHERE 製品番号 BETWEEN 100 AND 199 ORDER BY 在庫数量 DESC

 WITH OPTION LIMIT (3)
```



製品番号	製品名	在庫数量	倉庫番号
１１０	テレビ	８５	２
１１１	テレビ	９０	２
１２３	冷蔵庫	６０	１
１２４	冷蔵庫	７５	１
１４０	ＣＤプレーヤー	１２０	２
２１２	テレビ	０	２
２１５	ビデオ	５	２
２２６	冷蔵庫	８	１
２２７	冷蔵庫	１５	１
２４０	ＣＤプレーヤー	２５	２
２４３	ＣＤプレーヤー	１４	２
３５１	ＣＤ	２５００	２

製品番号	製品名	在庫数量	倉庫番号
１４０	ＣＤプレーヤー	１２０	２
１１１	テレビ	９０	２
１１０	テレビ	８５	２
１２４	冷蔵庫	７５	１
１２３	冷蔵庫	６０	１

製品番号	製品名	在庫数量	倉庫番号
１４０	ＣＤプレーヤー	１２０	２
１１１	テレビ	９０	２
１１０	テレビ	８５	２

9.15 ROWNUMを使用してデータを操作する

ROWNUMを使用すると、以下のことができます。

- 結果として取り出す件数を限定することができます(例1、例2、例3)。
- 結果として取り出す行に1から順に番号を割り振ることができます(例4)。
- 結果として取り出す行を途中から読み込むことができます(例5、例6)。

例1

在庫表から在庫数量が100以下の行を、先頭から5行分取り出します。

```
SELECT 製品番号,製品名,在庫数量
  FROM 在庫管理.在庫表 WHERE 在庫数量 <= 100 AND ROWNUM < 6
```

在庫表

製品番号	製品名	在庫数量	倉庫番号
１１０	テレビ	８５	２
１１１	テレビ	９０	２
１２３	冷蔵庫	６０	１
１２４	冷蔵庫	７５	１
１４０	ＣＤプレーヤー	１２０	２
２１２	テレビ	０	２
２１５	ビデオ	５	２
２２６	冷蔵庫	８	１
２２７	冷蔵庫	１５	１
２４０	ＣＤプレーヤー	２５	２
２４３	ＣＤプレーヤー	１４	２
３５１	ＣＤ	２５００	２

在庫数量が100以下の行で、ROWNUMにより先頭から5行を返却

結果

製品番号	製品名	在庫数量
１１０	テレビ	８５
１１１	テレビ	９０
１２３	冷蔵庫	６０
１２４	冷蔵庫	７５
２１２	テレビ	０

例2

在庫表から、製品番号が100から300の製品の数を求めます。ただし、50行を超える検索はしません。

```
SELECT COUNT(*) FROM 在庫管理.在庫表
 WHERE 製品番号 BETWEEN 100 AND 300 AND ROWNUM < 51
```

在庫表

製品番号	製品名	在庫数量	倉庫番号
１１０	テレビ	８５	２
１１１	テレビ	９０	２
１２３	冷蔵庫	６０	１
１２４	冷蔵庫	７５	１
１４０	ＣＤプレーヤー	１２０	２
⋮	⋮	⋮	⋮
２１２	テレビ	０	２
２１５	ビデオ	５	２
２２６	冷蔵庫	８	１
２２７	冷蔵庫	１５	１
２４０	ＣＤプレーヤー	２５	２
２４３	ＣＤプレーヤー	１４	２

製品番号が100から300の製品の数を求める。ただし。検索は、50行以内とする。

結果

COUNT(*)
５０

例3

在庫表から、製品番号が100から199で在庫数量が多い順に3行取り出します。

```
SELECT 製品番号,製品名,在庫数量,倉庫番号
  FROM (SELECT 製品番号,製品名,在庫数量,倉庫番号 FROM 在庫管理.在庫表
                    WHERE 製品番号 BETWEEN 100 AND 199
```

```
                    ORDER BY 在庫数量 DESC) AS D1(製品番号,製品名,在庫数量,倉庫番号)
                          (1)
WHERE ROWNUM < 4
      (2)
```

(1) 在庫数量の降順にソートします。

(2) 在庫数量が多い順に3行取り出します。

在庫表

製品番号	製品名	在庫数量	倉庫番号
１１０	テレビ	８５	２
１１１	テレビ	９０	２
１２３	冷蔵庫	６０	１
１２４	冷蔵庫	７５	１
１４０	ＣＤプレーヤー	１２０	２
２１２	テレビ	０	２
２１５	ビデオ	５	２
２２６	冷蔵庫	８	１
２２７	冷蔵庫	１５	１
２４０	ＣＤプレーヤー	２５	２
２４３	ＣＤプレーヤー	１４	２
３５１	ＣＤ	２５００	２

製品番号が１００から１９９で在庫数量の多い順にソートした表

製品番号	製品名	在庫数量	倉庫番号	
１４０	ＣＤプレーヤー	１２０	２	ROWNUMにより
１１１	テレビ	９０	２	先頭から3行を返却
１１０	テレビ	８５	２	
１２４	冷蔵庫	７５	１	
１２３	冷蔵庫	６０	１	

結果

製品番号	製品名	在庫数量	倉庫番号
１４０	ＣＤプレーヤー	１２０	２
１１１	テレビ	９０	２
１１０	テレビ	８５	２

例4

在庫表から、製品番号が100から200の行を取り出し、取り出した行に1から順に番号を振ります。

```
SELECT ROWNUM AS "ROWNUM",製品番号,製品名,在庫数量 FROM 在庫管理.在庫表
 WHERE 製品番号 BETWEEN 100 AND 200
```

在庫表

製品番号	製品名	在庫数量	倉庫番号
１１０	テレビ	８５	２
１１１	テレビ	９０	２
１２３	冷蔵庫	６０	１
１２４	冷蔵庫	７５	１
１４０	ＣＤプレーヤー	１２０	２
２１２	テレビ	０	２
２１５	ビデオ	５	２
２２６	冷蔵庫	８	１
２２７	冷蔵庫	１５	１
２４０	ＣＤプレーヤー	２５	２
２４３	ＣＤプレーヤー	１４	２
３５１	ＣＤ	２５００	２

製品番号が100以上200以下の行を取り出す

結果

ROWNUM	製品番号	製品名	在庫数量
１	１１０	テレビ	８５
２	１１１	テレビ	９０
３	１２３	冷蔵庫	６０
４	１２４	冷蔵庫	７５
５	１４０	ＣＤプレーヤー	１２０

例5

在庫表から、製品番号が100から300の行の、先頭から11件目以降を取り出します。

```
SELECT 製品番号,行番号
  FROM (SELECT 製品番号,ROWNUM FROM 在庫管理.在庫表
                          (1)
        WHERE 製品番号 BETWEEN 100 AND 300) AS D1(製品番号,行番号)
 WHERE 行番号 > 10
          (2)
```

(1) 問合せの結果に対し、1から順に行番号を振ります。

(2) ROWNUMで振った行番号を判定します。

在庫表

製品番号	製品名	在庫数量	倉庫番号
110	テレビ	85	2
111	テレビ	90	2
123	冷蔵庫	60	1
124	冷蔵庫	75	1
140	CDプレーヤー	120	2
212	テレビ	0	2
215	ビデオ	5	2
226	冷蔵庫	8	1
227	冷蔵庫	15	1
240	CDプレーヤー	25	2
243	CDプレーヤー	14	2
351	CD	2500	2

製品番号が100以上300以下の行を返却

製品番号が１００から３００以下で、ＲＯＷＮＵＭで行番号を振った導出表

製品番号	行番号
110	1
111	2
123	3
124	4
140	5
212	6
215	7
226	8
227	9
240	10
243	11

行番号11以降を返却

結果

製品番号	行番号
243	11

誤った使い方の例

```
SELECT 製品番号
  FROM 在庫管理.在庫表 WHERE 製品番号 BETWEEN 100 AND 200 AND ROWNUM > 10
                                                              (1)
```

(1) 結果集合の行にROWNUMの値を1から順に振りますが、探索条件は常に偽になるため、2以上の値が振られることがありません。そのため、結果集合の行はすべて偽になり、検索結果は0件になります。

例6

取引先などあるソートキーの昇順に画面を次々にめくっていくような場合、前回の続きから取り出します。

```
SELECT 行番号,取引先,取引製品,仕入価格
  FROM (SELECT ROWNUM,取引先,取引製品,仕入価格
                (3)
          FROM (SELECT 取引先,取引製品,仕入価格
                  FROM 在庫管理.発注表
                 WHERE 取引先 > 61
                 ORDER BY 取引先) AS D1(取引先,取引製品,仕入価格)
                       (1)
          WHERE ROWNUM <= 10)
                  (2)
       AS D2(行番号, 取引先,取引製品,仕入価格)
 WHERE 行番号 >= 6
         (4)
```

(1) 取引先の昇順にソートします。

(2) 取引先が小さい順に10行取り出します。

(3) 取引先でソートした結果に1から順番号を振ります。

(4) 順番号について、6以上の行を取り出します。

発注表

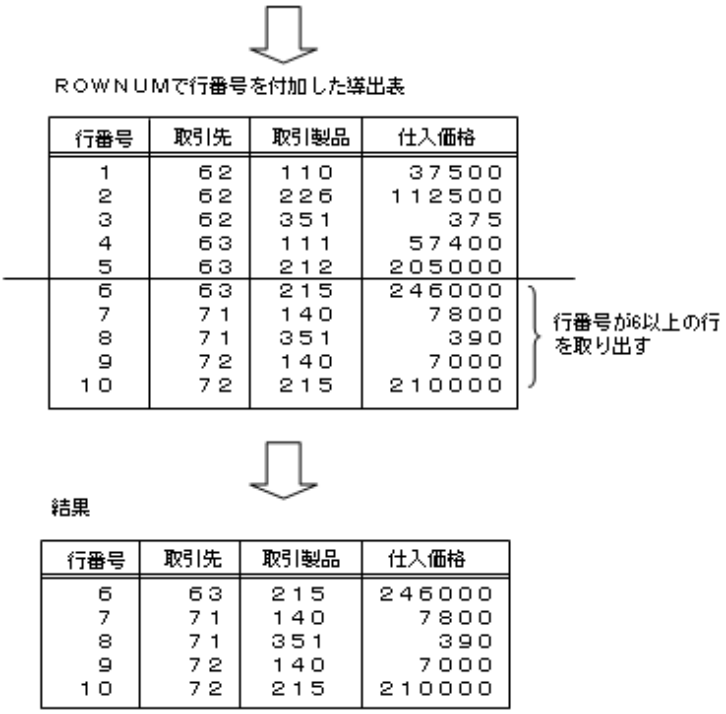
取引先	取引製品	仕入価格	発注数量
61	123	48000	60
61	124	64000	40
61	140	8000	80
61	215	240000	10
61	240	80000	20
62	110	37500	120
62	226	112500	20
62	351	375	800
63	111	57400	80
63	212	205000	30
63	215	246000	10
71	140	7800	50
71	351	390	600
72	140	7000	70
72	215	210000	10
72	226	105000	20
72	243	84000	10
72	351	350	1000
74	110	39000	120
74	111	54000	120
74	226	117000	20
74	227	140400	10
74	351	390	700

取引先が61より大きい行を取り出す

取引先が61より大きくて取引先の小さい順にソートした導出表

取引先	取引製品	仕入価格
62	110	37500
62	226	112500
62	351	375
63	111	57400
63	212	205000
63	215	246000
71	140	7800
71	351	390
72	140	7000
72	215	210000
72	226	105000
72	243	84000
72	351	350
74	110	39000
74	111	54000
74	226	117000
74	227	140400
74	351	390

取引先が小さい順に10行を取り出す



行番号	取引先	取引製品	仕入価格
1	62	110	37500
2	62	226	112500
3	62	351	375
4	63	111	57400
5	63	212	205000
6	63	215	246000
7	71	140	7800
8	71	351	390
9	72	140	7000
10	72	215	210000

行番号	取引先	取引製品	仕入価格
6	63	215	246000
7	71	140	7800
8	71	351	390
9	72	140	7000
10	72	215	210000

9.16 行値構成子を使用してデータを操作する

行値構成子を使用して、複数の列の並びを比較することができます。行値構成子はWHERE句に指定した以下の述語に指定可能です。

- 比較述語
- BETWEEN述語

例1

小売履歴表から2010年4月1日以降に販売した製品の販売日と製品番号を求めます。

```
SELECT 販売年,販売月,販売日,製品番号
  FROM 小売履歴表 WHERE (販売年,販売月,販売日) >= (2010,4,1)
```

小売履歴表

顧客番号	年齢	販売年	販売月	販売日	製品番号	数量	単価
002	25	2007	7	10	101	1	2500
054	32	2007	10	30	102	2	1200
059	19	2008	1	22	103	1	3000
010	22	2008	10	30	104	4	1500
020	33	2009	2	07	105	2	5000
103	42	2009	5	10	103	4	3000
105	50	2010	4	10	102	3	1200
059	19	2010	12	16	104	3	1500
042	28	2011	3	25	103	1	3000
101	36	2011	5	29	106	10	500
010	22	2011	8	04	101	2	2500

⇩

：比較述語が真となる行を表します。

結果

販売年	販売月	販売日	製品番号
2010	4	10	102
2010	12	16	104
2011	3	25	103
2011	5	29	106
2011	8	04	101

例2

小売履歴表から2008年4月と2010年3月の間に販売した顧客の番号を求めます。

```
SELECT 顧客番号 FROM 小売履歴表
 WHERE (販売年,販売月) BETWEEN (2008,4) AND (2010,3)
 GROUP BY 顧客番号
```

小売履歴表

顧客番号	年齢	販売年	販売年	販売日	製品番号	数量	単価
002	25	2007	7	10	101	1	2500
054	32	2007	10	30	102	2	1200
059	19	2008	1	22	103	1	3000
010	22	2008	10	30	104	4	1500
020	33	2009	2	07	105	2	5000
103	42	2009	5	10	103	4	3000
105	50	2010	4	10	102	3	1200
059	19	2010	12	16	104	3	1500
042	28	2011	3	25	103	1	3000
101	36	2011	5	29	106	10	500
010	22	2011	8	04	101	2	2500

結果 ⇩

：BETWEEN述語が真となる行を表します。

顧客番号
010
020
103

第10章　トリガを使用したデータ操作の方法

本章では、トリガを利用したデータの更新方法を説明します。

10.1 トリガ定義

トリガ定義は、CREATE TRIGGER文で行います。トリガ定義では、トリガ事象と、起動する手続きを指定します。トリガ事象は、トリガを起動するきっかけとなる表に対するデータの更新方法を指定します。また、起動する手続きは、処理したい手続きを被トリガSQL文で指定します。

ここでは、トリガの利用目的によって、以下のように分類します。

- 単純なトリガ
- プロシジャルーチンを呼び出すトリガ

単純なトリガ

表に対する更新を契機にして、別の表にデータを挿入するINSERT文を定義して自動的に実行することができます。

プロシジャルーチンを呼び出すトリガ

トリガの被トリガSQL文にCALL文を指定して、プロシジャルーチンを呼び出すことができます。

プロシジャルーチンを使用することによって、複数のSQL文で構成する処理手続きを被トリガSQL文として定義することができます。また、業務要件に応じたデータの整合性や一貫性を自由にカスタマイズすることができます。

たとえば、データの整合性や一貫性を保証する手続きを定義したプロシジャルーチンをトリガにより呼び出し、表間のデータの整合性を保証したり、更新するデータの値をチェックしたりすることができます。

トリガを使用すると、これらの情報の設定をシステムで自動的に実行することができるため、アプリケーションを簡素化でき、信頼性の高いシステムを構築することができます。

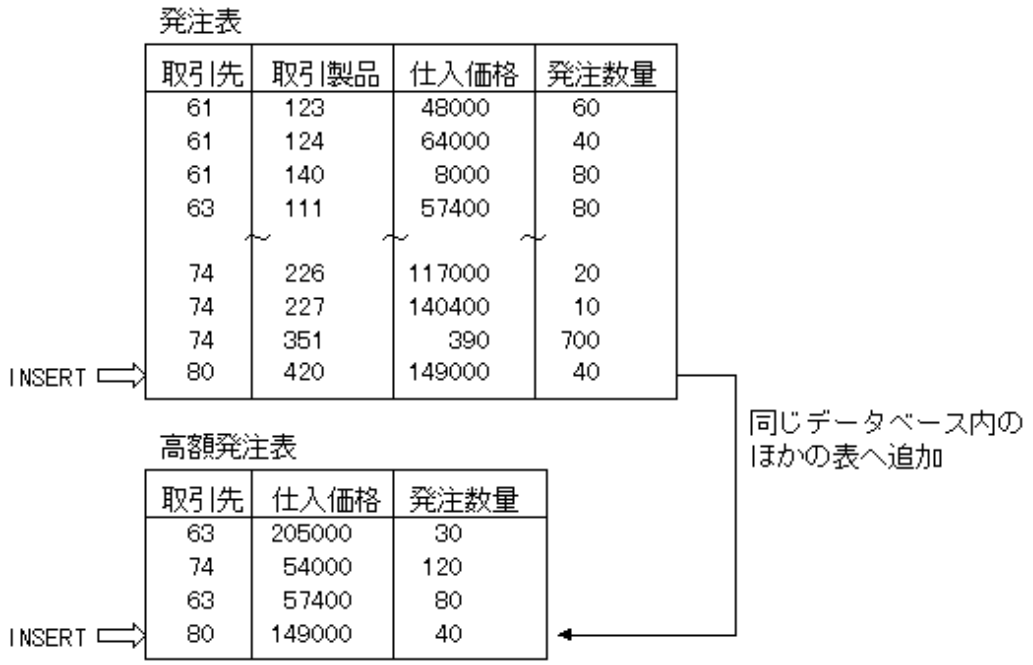
以下に、トリガの定義例を示します。

例1 単純なトリガの場合

発注表に追加された行の発注価格が500万を超えている場合、トリガ“発注トリガ”により、その製品の取引先、仕入価格および発注数量を高額発注表に追加します。

トリガ定義例

```
CREATE TRIGGER  STOCKS.発注トリガ
  AFTER INSERT ON STOCKS.発注表
  REFERENCING NEW AS NEWREC
  FOR EACH ROW
    WHEN(NEWREC.仕入価格 * NEWREC.発注数量 > 5000000)
    INSERT INTO STOCKS.高額発注表
    VALUES(NEWREC.取引先,NEWREC.仕入価格,NEWREC.発注数量)
```



例2 プロシジャルーチンを呼び出すトリガの場合

発注表に行を挿入すると、トリガ“発注トリガ2”により、プロシジャルーチン“発注ルーチン”を呼び出します。“発注ルーチン”は、挿入データに対して整合性チェックを行います。そして、発注した製品の在庫数量について在庫表を更新します。

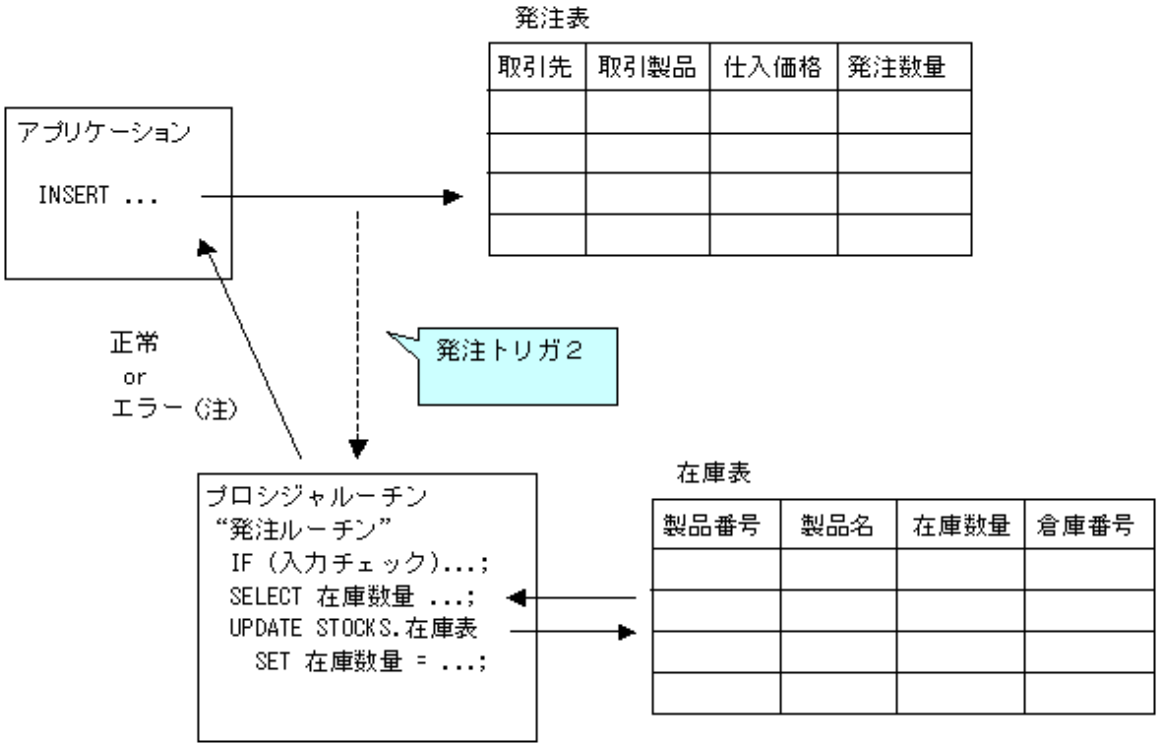
プロシジャルーチン定義例

```
CREATE PROCEDURE STOCKS.発注ルーチン (
    IN  ＮＥＷ取引製品    SMALLINT,
    IN  ＮＥＷ発注数量    SMALLINT
)
BEGIN
    -- SQL変数宣言
    DECLARE SQLSTATE    CHAR(5);
    DECLARE SQLMSG      CHAR(256);
    DECLARE POINT       SMALLINT;
    DECLARE Ｓ在庫数量  INTEGER;
    -- 条件宣言
    DECLARE 入力発注数量不当    CONDITION FOR SQLSTATE'60001';
    DECLARE 入力製品番号不当    CONDITION FOR SQLSTATE'60002';
    DECLARE 在庫不足            CONDITION FOR SQLSTATE'60003';
    DECLARE その他異常          CONDITION FOR SQLSTATE'60999';
    -- ハンドラ宣言
    DECLARE EXIT HANDLER FOR NOT FOUND
    BEGIN
        IF (POINT = 20 OR POINT = 30) THEN
            RESIGNAL 入力製品番号不当
                    '発注: 入力異常: 取引製品の値が不当';
        ELSE
            RESIGNAL その他異常 '発注: 異常: 予想外のデータなし発生';
        END IF;
    END;
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
```

```
        RESIGNAL;  --発生したエラーをそのまま返却する
    END;
    -- 本処理
    -- (1)入力チェック
    SET POINT = 10;
    IF (ＮＥＷ発注数量 < 1) THEN
        SIGNAL 入力発注数量不当 '発注: 入力異常: 発注数量の値が不当';
    END IF;
    -- (2)在庫数量チェック
    SET POINT = 20;
    SELECT 在庫数量 INTO Ｓ在庫数量 FROM STOCKS.在庫表
        WHERE 製品番号 = ＮＥＷ取引製品
        WITH OPTION LOCK_MODE(EXCLUSIVE LOCK);
    IF (ＮＥＷ発注数量 > Ｓ在庫数量) THEN
        SIGNAL 在庫不足 '発注: 異常: 在庫不足です';
    END IF;
    -- (3)発注作業
    SET POINT = 30;
    UPDATE STOCKS.在庫表
        SET 在庫数量 = 在庫数量 - ＮＥＷ発注数量
        WHERE 製品番号 = ＮＥＷ取引製品;
END
```

トリガ定義例

```
CREATE TRIGGER STOCKS.発注トリガ２
    AFTER INSERT ON STOCKS.発注表
    FOR EACH ROW
    CALL STOCKS.発注ルーチン(NEW.取引製品, NEW.発注数量);
```



注)“発注ルーチン”が不当な入力値を検出した際、INSERT文は以下のエラーになります。
「JYP1065E スキーマ “STOCKS”のトリガ“発注トリガ2”の被トリガ SQL文でエラーが発生しました. 詳細メッセージ=
JYP2550E SIGNAL文により例外が送信されました. 例外メッセージ=“発注: 異常: 在庫不足です”」

上記のように、プロシジャルーチンとトリガを定義して、表を更新するSQL文を実行すると、被トリガSQL文で指定したプロシジャルーチンが自動的に実行され、データの一貫性を損なうような更新を抑止することができます。また、プロシジャルーチン中で、お互いに参照関係にある表の結合列の存在をチェックして表間のデータの整合性を保証したり、親となる表の行を削除する契機で、対応する子となる表のデータを自動的に削除したりするようなトリガを定義することもできます。

ただし、複雑な業務ロジックは、トリガで定義しないようにしてください。トリガは起動元となった表の更新の延長で動作します。トリガの中でトランザクションを制御することはできません。複雑な業務ロジックはプロシジャルーチンで作成し、アプリケーションから直接プロシジャルーチンをCALLするようにしてください。

トリガの動作

トリガは以下のように動作します。

- トリガの実行順序は次のようになります。連鎖的に実行されるトリガである場合でも、各トリガは同じルールで行単位に実行されます。

  1. トリガを定義している表を更新するSQL文が実行されます。
  2. SQL文で影響を受ける行をループします。
  3. SQL文がUPDATEまたはDELETEする行を位置づけます。
  4. 定義されているBEFOREトリガを実行します。
  5. 1.のSQL文により、行がUPDATEまたはDELETE、あるいはINSERTされます。
  6. 定義されているAFTERトリガを実行します。

- 同一の表に対して、同一のトリガ事象が複数定義された場合、トリガは順不同に実行されます。同じトリガ動作時点を指定したお互いのトリガは実行順序に影響されないように考慮してください。
- ユーティリティ機能を利用して、データのロード（rdbsloaderコマンド）およびデータベースの再初期化（rdbfmtコマンド）を実行する場合、その実行の対象となる表にトリガが定義されていても、トリガは作用しません。
- トリガを起動する元となったSQL文で更新している表を、被トリガSQL文で更新することはできません。
- 被トリガSQL文で更新する表にトリガが定義されていると、連鎖的にトリガが実行されます。一度実行されたトリガの延長で、再び同じトリガが実行された場合は、エラーになります。
- 被トリガSQL文は、トリガを起動するSQL文を実行したトランザクションと同じトランザクションとして動作します。トリガを起動するSQL文にLOCK_MODEを指定している場合は、被トリガSQL文により実行される各SQLデータ操作文に、暗黙的に同じLOCK_MODEが指定されたものとして動作します。
- 被トリガSQL文で、データベースの更新を行う場合、更新する表に定義されている一意性制約やNOT NULL制約は、SQL文ごとに検査されます。CALL文を指定している場合は、CALL文中のSQL文ごとに制約が検査されます。
- 被トリガSQL文中でエラーが発生した場合は、トリガを起動する元となったSQL文の実行が取り消され、エラーとなります。被トリガSQL文にCALL文を指定した場合は、呼び出されるプロシジャルーチン内の各SQL文の実行でエラーが発生すると、プロシジャルーチンに指定したエラー処理に従ってSQL文の実行を継続することができます。また、CALL文の実行をエラーにして、トリガを起動する元となったSQL文の実行を取り消し、エラーとすることもできます。
- 被トリガSQL文にCALL文を指定する場合、呼び出したプロシジャルーチン内で、トランザクションロールバックの例外が発生した場合は（SQLSTATEの例外コードが40）、トリガを起動する元となったSQL文を含むトランザクションが自動的にロールバックされます。

トリガ定義での注意事項

トリガ定義での注意事項を、以下に示します。

- トリガを起動する元となったSQL文に副問合せを指定している場合、副問合せで指定している表を被トリガSQL文で更新することはできません。
- UPDATE文:探索で表の一意性制約の列を更新すると、SQL文で複数の行を更新している時、一時的にデータの一意性を満たしていない状況が発生することがあります。このような場合に実行されるトリガの被トリガSQL文に指定したCALL文の延長で、トリガを起動する元となった更新中の行を参照すると、一時的に一意性を崩している更新中の

行を参照できないことがあります。このため、CALL文で呼び出すプロシジャルーチンで、トリガの起動元となった表の更新後の行を参照する必要がある場合には、CALL文の引数で必要な列の値を受け渡すようにしてください。

・ トリガ定義時の権限

トリガの定義者は、トリガを定義するスキーマに対するCREATE権、トリガを定義する表に対するTRIGGER権および被トリガSQL文に指定したSQL操作に対応する権限が必要です。

－ INSERT文を指定する場合：被トリガSQL文で指定する表に対するINSERT権が必要です。

－ CALL文を指定する場合：被トリガSQL文で指定するプロシジャルーチンに対する実行権が必要です。

・ 被トリガSQL文実行時の権限チェック

あるSQL文がトリガ契機となり、被トリガSQL文が実行された場合、その被トリガSQL文の権限チェックは行われません。アプリケーションの実行者は、被トリガSQL文で指定した表に対する権限は必要ありません。

第11章　ルーチンを使用したデータ操作の方法

本章では、プロシジャルーチンおよびファンクションルーチンの利用方法を説明します。

11.1 プロシジャルーチンの利用方法

通常のクライアント・サーバ間は、クライアント側にアプリケーション全体を置き、サーバにはデータベースの検索・更新処理だけをさせる形態になっています。この場合、クライアントとサーバ間を検索結果などの大量なデータが行き来して、実際のアプリケーションはクライアント側で制御するようになり、大規模なトランザクション開発に適していません。このような場合は、プロシジャルーチンを利用して、サーバ側でアプリケーションを実行することができます。

本節では、プロシジャルーチンを利用する場合について、以下の項目を説明します。

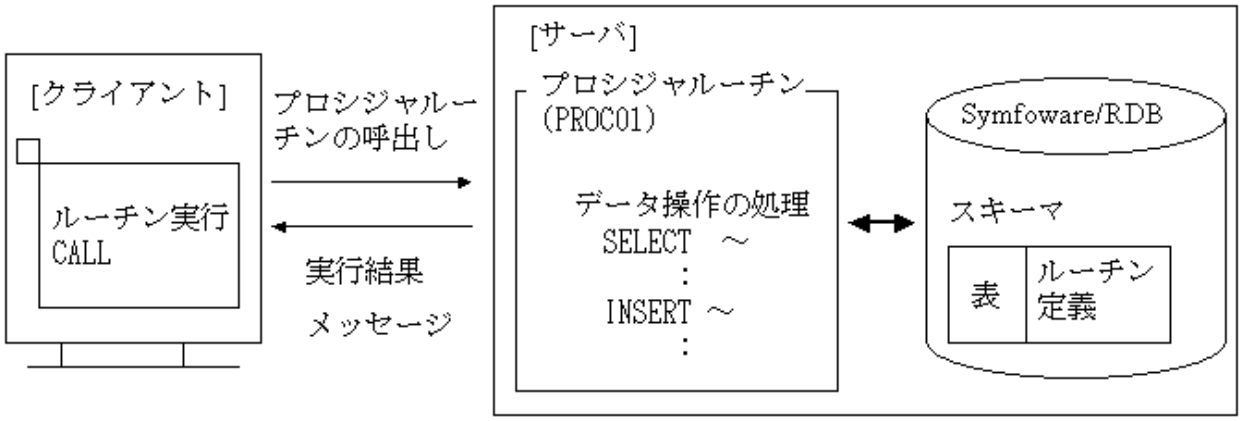
- プロシジャルーチンとは
- プロシジャルーチンの定義

11.1.1 プロシジャルーチンとは

クライアント側にあるアプリケーションをサーバで実行する形態の場合は、クライアントからサーバにSQL文を送信し、サーバ側の実行結果をクライアント側で受信して処理を行っています。このとき、SQL文単位で送信・受信を繰り返すため、クライアント側に通信負荷がかかります。大規模なアプリケーション開発の場合、このようなクライアント側の性能限界を解消し、開発/保守の生産性を向上させるためにはプロシジャルーチンを利用します。プロシジャルーチンとは、サーバに登録する処理手続きのことです。プロシジャルーチンをサーバに登録しておいて、クライアント側からプロシジャルーチンを呼び出し、サーバ側で一連のトランザクション処理を実行します。プロシジャルーチンとして登録する処理手続きは、プロシジャルーチン内で完結する処理が適しています。たとえば、プロシジャルーチン内でアクセスしたデータを基に別の表を作成する処理などです。

以下に、プロシジャルーチンを利用する場合のクライアント・サーバ間の処理概要を示します。

図11.1 プロシジャルーチンを利用したクライアント・サーバ間の処理概要



11.1.2 プロシジャルーチンを利用するアプリケーションの作成の流れ

プロシジャルーチンを利用するアプリケーションは、以下の手順で作成します。

1. プロシジャルーチンの定義
2. プロシジャルーチンの実行

参照

- プロシジャルーチンの実行については、各アプリケーションごとの開発ガイドを参照してください。
- 各SQL文の詳細については、“SQLリファレンス”を参照してください。

11.1.3 プロシジャルーチンの定義

プロシジャルーチンをスキーマに登録するには、以下に示すプロシジャルーチン定義文を使用します。

```
CREATE PROCEDURE  SCM.RNAME ( IN   PARA   INT)
                  (1)   (2)   (3)   (4)   (5)
  BEGIN
    DECLARE VAR   INT;
    SET VAR = PARA + 1;
    INSERT INTO SCM.TBL(ITEM) VALUES(VAR);
              :
  END
```

(1) スキーマ名

(2) ルーチン名

(3) パラメタモード

(4) パラメタ名

(5) データ型

プロシジャルーチンで定義できるSQL文をSQL手続き文と呼びます。プロシジャルーチンはSQL手続き文を組み合わせることで、アプリケーションロジックのルーチンの開発を行います。SQL手続き文には以下のSQL文があります。

- データ操作文
- トランザクション管理文(トリガからの呼出し以外)
  - COMMIT文
  - ROLLBACK文
- SQL制御文
  - 複合文
  - SQL変数宣言
  - IF文
  - CALL文
  - 代入文
  - LOOP文
  - WHILE文
  - REPEAT文
  - LEAVE文
  - GOTO文
  - 分岐先文
  - WHENEVER文
  - 条件宣言
  - ハンドラ宣言
  - SIGNAL文(トリガからの呼出し時)
  - RESIGNAL文(トリガからの呼出し時)

参照

SQL制御文の詳細については、“SQLリファレンス”を参照してください。

プロシジャルーチン定義をファイルに作成し、サーバ側でrdbddlexコマンドを実行してプロシジャルーチンを登録します。
rdbddlexコマンドでプロシジャルーチンを定義する場合、注釈を指定することができます。

S L

定義ファイルからプロシジャルーチンを登録する例と定義ファイルの例を以下に示します。

例

在庫表から、在庫数量が50以下の製品に対する発注を、倉庫番号によって営業所別のテーブルに振り分けます。

[定義ファイルからプロシジャルーチンを登録する例]

```
rdbddlex  -d  RDBDB   -x /home/rdb/DDL/ddl.dat
                                  ↑
                             入力ファイル指定
```

[定義ファイル(ddl.dat)の例]

```
EXEC SQL
  CREATE PROCEDURE 在庫管理.営業所別発注処理(IN 検索キー INT)
    BEGIN
      -- SQL変数宣言
        DECLARE   製品番号V    SMALLINT;
        DECLARE   製品名V      NCHAR(10);
        DECLARE   在庫数量V    INT;
        DECLARE   倉庫番号V    SMALLINT;
        DECLARE   数量V        INT;
        DECLARE   SQLSTATE     CHAR(5);
                  :
                  :
      -- カーソル宣言
      DECLARE CUR1 CURSOR FOR
         SELECT 製品番号, 製品名, 在庫数量, 倉庫番号 FROM 在庫管理.在庫表;

      DECLARE CUR2 CURSOR FOR
         SELECT 取引先, 取引製品, 発注数量 FROM 在庫管理.発注表;
                  :

      -- 検索キーに1が指定された場合,CUR1に対する検索と補充テーブルの
      -- INSERTを行う.
      IF 検索キー = 1 THEN
        OPEN CUR1;
        LABEL1:
        -- データが終了するまでデータを検索する.
        LOOP
          FETCH CUR1 INTO 製品番号V, 製品名V, 在庫数量V, 倉庫番号V;
          IF SQLSTATE <> '00000' THEN
            LEAVE LABEL1;
          END IF;
          -- 在庫数量が50以下の場合, 補充データの作成を行う.
          IF 在庫数量V <= 50 THEN
            -- 補充の数量を算出する.
            SET 数量V = 50 - 在庫数量V;
            -- 倉庫番号によって発注先のテーブルを変更する.
            IF 倉庫番号V = 2 THEN
              INSERT INTO 在庫管理.営業所２(製品番号, 製品名, 数量)
                                    VALUES(製品番号V, 製品名V, 数量V);
            IF SQLSTATE <> '00000' THEN
              LEAVE LABEL1;
```

```
            END IF;
            ELSE
              INSERT INTO 在庫管理.営業所１(製品番号, 製品名, 数量)
                                    VALUES(製品番号Ｖ, 製品名Ｖ, 数量Ｖ);
            IF SQLSTATE <> '00000' THEN
              LEAVE LABEL1;
            END IF;
          END IF;
        END IF;
      END LOOP LABEL1;

    -- 検索キーに2が指定された場合, CUR2に対する処理を行う.
    ELSEIF 検索キー = 2 THEN
      OPEN CUR2;
                :

      END IF;
    END
END-EXEC;
```

上記の例の入力ファイル指定が、“C:¥USERS¥DEFAULT¥DDL.DAT”のような形式になります。

11.2 ファンクションルーチンの利用方法

本節では、ファンクションルーチンを利用する場合について、以下の項目を説明します。

- ファンクションルーチンとは
- ファンクションルーチンの定義

11.2.1 ファンクションルーチンとは

ファンクションルーチンとは、利用者が作成できる関数です。ファンクションルーチンは、プロシジャルーチンと同様にサーバに登録します。ファンクションルーチンの呼出しはSQL文の選択リストや値指定などに指定して実行します。

ファンクションルーチンを利用することで、利用者はSymfoware/RDBが提供する関数以外に自由に関数を作成し、数値関数、データ列値関数、日時値関数と同様にSQL文中で実行することができます。

例

ファンクションルーチンを使用したSQL文の記述例

スキーマ名およびルーチン名は、それぞれ、“管理SCM”、“高額購入伝票データ整形”であるとします。

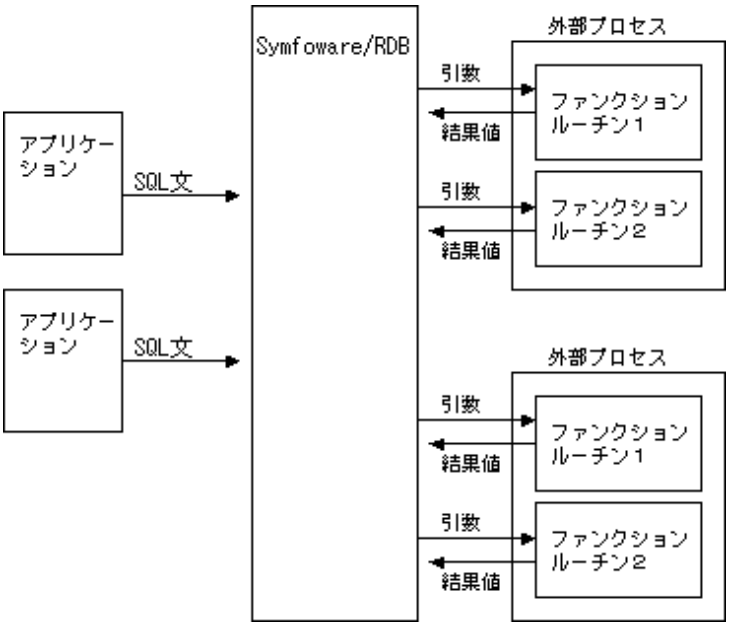
```
SELECT 管理ＳＣＭ. 高額購入伝票データ整形(番号, 製品名, 金額, 発注数)
       FROM 管理ＳＣＭ.伝票テーブル
       WHERE 発注数 > 0 AND 金額 >= :H_YEN ;
```

Symfoware/RDBでは、ファンクションルーチンは外部ルーチンとして提供しています。

外部ルーチンとは、利用者がC言語などで作成したプログラムを動的リンクライブラリとしてサーバに登録し、SQL文から呼び出すものです。外部ルーチンの利点は、文字列型データの整形など、SQLの関数では限界がある複雑な処理を、C言語などを使用することで容易に行うことができる点です。

Symfoware/RDBでは、外部ルーチンを作成する言語として、C言語を使用できます。

ファンクションルーチンの処理を行うプログラムは、Symfoware/RDBプロセス外のファンクションルーチン専用のプロセスで動作し、このプロセスはアプリケーションごとに作成されます。Symfoware/RDBプロセス内のSQL文の処理中に、レコード単位に引数が通知され、ファンクションルーチンの結果値を随時Symfoware/RDBに返却します。



11.2.2 ファンクションルーチンを利用するアプリケーションの作成の流れ

ファンクションルーチンを利用するアプリケーションは、以下の手順で作成します。

1. ファンクションルーチンの定義
2. ライブラリの作成
3. テストドライバの作成・単体テストの実施
4. ファンクションルーチンの実行

参照

- ライブラリの作成、テストドライバの作成・単体テストの実施、およびファンクションルーチンの実行については、各アプリケーションごとの開発ガイドを参照してください。
- 各SQL文の詳細については、“SQLリファレンス”を参照してください。

11.2.3 ファンクションルーチンの定義

ファンクションルーチンをスキーマに登録するには、以下に示すファンクションルーチン定義文を使用します。

S L

Solaris/Linuxの場合

```
CREATE  FUNCTION  SCM1.USERFUNC001  ( IN CHAR(20),  IN INTEGER, … )
                          (1)              (2)

                RETURNS VARCHAR(60)  LANGUAGE  C  NAME  'UserFunc001'
                           (3)                          (4)

                  LIBRARY  '/usr/local/lib/libuserfunc001.so'
                                    (5)
```

(1) ルーチン名

(2) パラメタ宣言

(3) 戻りデータ型

(4) Cプログラムの関数シンボル名

(5) ファンクションルーチンのライブラリ

W

Windows(R)の場合

上記の例のファンクションルーチンのライブラリが、“D:¥FORSYMFO¥FUNCLIB¥USERFUNC001.DLL”のような形式になります。

ファンクションルーチン定義をファイルに作成し、サーバ側でrdbddlexコマンドを実行してファンクションルーチンを登録します。

S L

Solaris/Linuxの場合

ファンクションルーチンを登録する例と定義ファイルの例を以下に示します。

```
rdbddlex  -d  RDBDB    /home/rdb/DDL/userfunc001.dat
                                  ↑
                            入力ファイル指定
```

[定義ファイル(userfunc001.dat)の例]

```
CREATE FUNCTION SCM1.USERFUNC001(IN CHAR(20),
                                 IN INTEGER,
                                 IN INTEGER )
  RETURNS VARCHAR(60)
  LANGUAGE C
  NAME 'UserFunc001'
  LIBRARY '/usr/local/lib/libuserfunc001.so';
```

W

Windows(R)の場合

上記の例の入力ファイル指定が、“D:¥FORSYMFO¥DDL¥USERFUNC001.DAT”のような形式になります。

第12章　XMLデータを操作する方法

本章では、XMLデータを操作する方法を説明します。

12.1 XQueryの概要

本章では、XQueryの概要と、XQueryを構成する上で基本となる構成要素、データモデルについて説明します。

XQueryでは、XMLデータの検索のみが可能であり、更新・削除の機能はありません。そのため、XQueryを実行するための権限は、そのXMLデータを参照する権限と等価です。XMLデータが格納された表に対するSELECT権がこれに当たります。

12.1.1 XQueryの構成要素

XQueryの問合せは、プロローグとそれに続く問合せ本体から構成されます。プロローグは様々な宣言文から構成され、問合せの処理環境を定義します。処理環境をデフォルトの定義から変更しない場合、プロローグは問合せから省略することができます。

問合せ本体は、問合せ結果を定義する式から構成されます。式は、キーワード、記号、オペランド(別の式)の3つから構成されており、XQueryには様々な種類の式が用意されています。式はオペランドを通して入れ子にすることができます。

XQueryでは、式、およびプロローグで使用される英字の大文字と小文字をすべて区別します。

キーワードの表現には小文字を用います。XQueryのキーワードは予約語ではないため、キーワードと同じ文字列を名前としてXQuery式で使用することが可能です。

12.1.2 XQueryのデータモデル

XQueryの問合せを処理できるようにするためには、式の入力となるXMLデータをXDMインスタンスとして表現しなければいけません。XDMインスタンスとは、XMLデータをXQueryで処理できる形に変換した値のことです。XMLデータからXDMインスタンスへの変換処理は、XMLパーザなどで行われ、XQueryを実行する前に終了します。

12.1.2.1 シーケンスおよび項目

値を表すXDMインスタンスは常にシーケンスです。シーケンスとは、0個以上の項目が順序付けされたコレクションです。項目とは、原子値またはノードです。原子値とノードについては“12.1.2.2 原子値”および“12.1.2.3 ノード”を参照してください。

シーケンスの例を以下に示します。

- (10,1,2,3,4)
  この例は、5つの原子値をもつシーケンスです。
- /employees/employee
  この例は、2つのノードをもつシーケンスです。
- 128
  この例は、1つの原子値をもつシーケンスです。
- ()
  この例は、空シーケンスです。

1つの項目しか含まないシーケンスを単一値シーケンスといいます。項目は、その項目を含む単一値シーケンスと同じ意味です。項目を1つも含まないシーケンスを空シーケンスといいます。

シーケンスが入れ子になることはありません。例えば、以下の3つのシーケンスを結合して1つのシーケンスにすると、結果はシーケンス(1, 2, 3)となります。シーケンス(1, (2, 3), ())のような入れ子構造にはなりません。

1

(2,3)

()

12.1.2.2 原子値

原子値とは、XQueryで定義された組込み原子型の値です。Symfowareでは、組込み原子型として文字列型(xs:string)、数値型(xs:double)、日付型(xs:date)、時間型(xs:time)、日時型(xs:dateTime)、ブール値型(xs:boolean)、型なし型(xs:untypedAtomic)の7種類を提供します。これらの型の定義は、XML-Schemaの規約に準拠しています。以下に原子値の例を示します。

- "This is an old pen."
  この例は、xs:string型の原子値です。
- 1024
  この例は、xs:double型の原子値です。
- xs:date("2007-06-13")
  この例は、xs:string型の原子値をキャストして、xs:date型の原子値を生成しています。
- xs:time("13:08:15")
  この例は、xs:string型の原子値をキャストして、xs:time型の原子値を生成しています。
- xs:dateTime("2007-06-13T13:08:15")
  この例は、xs:string型の原子値をキャストして、xs:dateTime型の原子値を生成しています。
- fn:true()
  この例は、ブール値関数fn:true()を利用して、xs:boolean型の原子値を生成しています。
- xs:untypedAtomic("12AB")
  この例は、xs:string型の原子値をキャストして、xs:untypedAtomic型の原子値を生成しています。

12.1.2.3 ノード

ノードとは、XMLデータを構成する要素や属性を表す単位です。XMLデータは、単一のルートノードを頂点とする木構造で表現することができます。XMLデータの例とそのXMLデータを表現する木構造を以下に示します。

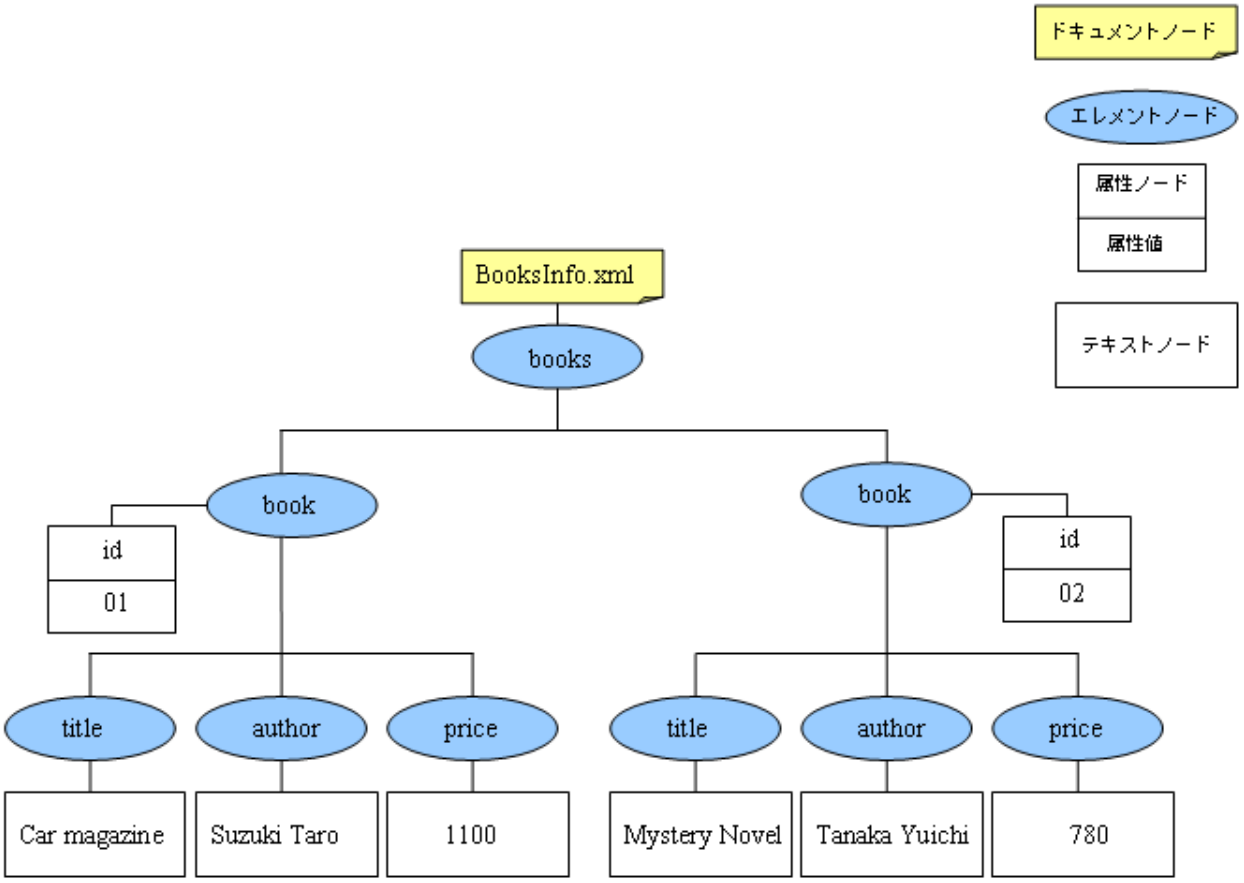
木構造の例

[XMLデータ]

BooksInfo.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
  <book id="01">
     <title>Car magazine</title>
     <author>Suzuki Taro</author>
     <price>1100</price>
  </book>
  <book id="02">
     <title>Mystery Novel</title>
     <author>Tanaka Yuichi</author>
     <price>780</price>
  </book>
</books>
```

[XMLデータを表現する木構造]



ノードの種類

ノードには、ドキュメントノード、エレメントノード、属性ノード、テキストノード、コメントノード、処理命令ノード、ネームスペースノードの7種類があります。各ノードの概要を以下に示します。

表12.1 XMLデータを構成するノード

ノードの種類	概要	親ノード	子ノード
ドキュメントノード	XMLデータのルートノードを表します。XMLデータ内のルート要素（上記例では“books”）のさらに上位に位置する仮想的なノードのことです。	なし	・エレメントノード ・テキストノード ・コメントノード ・処理命令ノード
エレメントノード	XMLデータの要素を表します。要素名をノードの名前として持ちます。	・ドキュメントノード ・エレメントノード	・エレメントノード ・属性ノード ・テキストノード ・コメントノード ・処理命令ノード
属性ノード	XMLデータの属性を表します。属性名をノードの名	・エレメントノード	なし

ノードの種類	概要	親ノード	子ノード
	前として持ち、対応する属性値を保持します。		
テキストノード	XMLデータの要素の内容を表します。	• ドキュメントノード • エレメントノード	なし
コメントノード	XMLデータのコメントを表します。	• ドキュメントノード • エレメントノード	なし
処理命令ノード	XMLデータの処理命令を表します。	• ドキュメントノード • エレメントノード	なし
ネームスペースノード	XMLデータのネームスペースを表します。ネームスペースURIとネームスペース接頭辞またはデフォルトネームスペースとの対応関係を示します。	• エレメントノード	なし

ノード使用例

XQueryで用いられるノードの記述例とその実行結果を以下に示します。

```
/books/book[@id="01"]/price
```

この例は、パス式を利用して、ルートノードの子エレメントノードがbooksであり、その子エレメントノードがbookであり、その子属性ノードidの属性値が“01”に等しい場合の、bookの子エレメントノードpriceの内容“1100”を返します。

12.1.2.4 データ入力源

XQueryでは、データベースに格納されたXMLデータをコンテキストアイテムの初期値に設定することができます。

参照

設定方法の詳細は、“XQueryリファレンス”の“動的コンテキスト”を参照してください。

入力データがコンテキストアイテムの初期値に設定された場合、XQueryはコンテキストアイテム式、または'/'から始まるパス式によってそのデータにアクセスすることができます。

注意

コンテキストアイテムの初期値が設定されていない場合、コンテキストアイテム式、または'/'から始まるパス式を指定したXQueryを実行するとエラーになります。

12.1.3 XQueryの実行例

この節では、XQueryを用いて表現できる問合せの例を示します。ここで示す問合せの内容は、入力源であるXMLデータからのデータ部分抽出です。

部分抽出

部分抽出は、入力源であるXMLデータから特定部分のデータだけを選択して取り出すことであり、XQueryの基本となる問合せです。部分抽出する問合せの例を以下に示します。

例1

```
/employees/employee[@sex="male"]/name
```

この問合せは、入力源であるXML文書「従業員.xml」から、男性従業員の名前を表すデータだけを取り出しています。

例2

```
for $x in /employees/employee
return if ( $x/family/daughter ) then ( ) else $x/name
```

この問合せは、入力源であるXMLデータ「従業員.xml」から、娘のいない従業員の名前を表すデータだけを取り出しています。

12.2 XMLデータを検索する

SQL/XMLを使用してXQueryで条件を指定することで、BLOB型の列に格納されたXMLデータに対して、条件に一致したXMLデータの検索を行うことができます。また、格納されたXMLデータから、部分的にデータを取り出すことも可能です。

ここでは、SQL/XMLを使用する方法を説明します。

12.2.1 SQL/XMLでXMLデータを検索する

SQL/XMLで、BLOB型の列に格納されたXMLデータを検索する場合には、XQueryを使用します。XQueryは、XMLQUERY関数またはXMLEXISTS述語に指定します。XMLQUERY関数またはXMLEXISTS述語を問合せ式に指定することで、表に格納されたXMLデータを検索することができます。

- XMLQUERY関数

  XQueryの条件に一致したXMLデータの検索を行うことができます。格納されたXMLデータから必要な一部分のデータを取り出すこともできます。

- XMLEXISTS述語

  XQueryの問合せ結果が空のシーケンスか否かを評価します。格納されたXMLデータをそのまま取り出す場合には、XMLEXISTS述語の条件指定のみの検索が有効です。

XQueryを使用してXMLデータを検索するときの権限は、XMLデータを格納している列が存在する表への参照権限と同じです。XMLデータが格納された表に対するSELECT権がこれに当たります。

XML構文解析

XML構文解析は、BLOB型の列に格納されたテキスト形式のXMLデータを階層形式に変換する処理です。この処理は、以下のような検索を実施するときに、暗黙的に行われ、データ形式に誤りがあれば、参照処理がエラーになります。

[XML構文解析を実施する検索]

- XMLデータの一部分を取り出す検索
- FLWOR式を指定した検索
- 条件式を指定した検索
- 量化式を指定した検索

参考

rdbunlxコマンドでサポートしているXQuery式は、XML構文解析を実施しません。詳細については、“XQueryリファレンス”の“rdbunlxコマンドのサポート一覧”を参照してください。

また、構文解析時にXMLデータの妥当性検証、内部実体の展開および外部実体の展開は行いません。

12.2.1.1 XMLQUERY関数

XMLQUERY関数は、SQLから、XQuery式を実行するための関数です。XMLQUERY関数は、XQueryの評価結果であるシーケンスの各項目を先頭から順に連結して返却します。このとき、シーケンスの各項目の間を区切る文字はありません。

注意

カンマ記号、空白文字、改行などでシーケンスの各項目が区切られることはありません。ただし、本マニュアルの例では、見易さを考慮して、シーケンスの各項目の間に改行を適宜含めて、XMLQUERY関数の評価結果を表現しています。

XMLQUERY関数でXQuery式を利用すると、格納されたXMLデータから条件に一致する一部分を取り出すことができます。

SQLは大小文字を区別しませんが、XQuery式は大小文字を区別することに注意してください。

以降の説明では、従業員スキーマに定義された従業員表を例として使用します。従業員表は、以下のような構成になっています。

列名	列のデータ型
従業員番号	INTEGER
従業員情報	BLOB(20K)

従業員表には、以下の3件のデータが格納されているとします。

表12.2 従業員スキーマ.従業員表

従業員番号	従業員情報
54319	<従業員情報 従業員番号="54319"> <名前>佐藤　一郎</名前> <所属 code="k0312" >第一開発部</所属> <派遣先>東京都港区</派遣先> <派遣先>東京都千代田区</派遣先> <派遣先>神奈川県横浜市</派遣先> <電話番号>050-1111-2222</電話番号> </従業員情報>
54320	<従業員情報 従業員番号="54320"> <名前>田中　大輔</名前> <所属 code="f0021" >第二開発部</所属> <派遣先>神奈川県横浜市</派遣先> <電話番号>030-2222-1111</電話番号> </従業員情報>
54321	<従業員情報 従業員番号="54321"> <名前>山本　翔太</名前> <所属 code="f0021" >第三開発部</所属> <派遣先>東京都大田区</派遣先> <派遣先>神奈川県横浜市</派遣先> <電話番号>030-2222-2222</電話番号> </従業員情報>

XMLQUERY関数の使用例

従業員表に対して、以下のような問合せを行ったとします。

```
SELECT 従業員番号, XMLQUERY('/従業員情報/派遣先' PASSING 従業員情報) FROM 従業員スキーマ.従業員表
```

問合せの結果、以下の3件のデータが返却されます。

従業員番号	従業員情報
54319	<派遣先>東京都港区</派遣先> <派遣先>東京都千代田区</派遣先> <派遣先>神奈川県横浜市</派遣先>
54320	<派遣先>神奈川県横浜市</派遣先>
54321	<派遣先>東京都大田区</派遣先> <派遣先>神奈川県横浜市</派遣先>

この例は、XMLQUERY関数の一般的な使用方法を示しています。XMLQUERY関数が、検索対象となる表の各行に格納されたXMLデータ1つずつに適用されて、結果の各行が1つのXMLデータからの検索結果を表しています。

最初の行には、3つの<派遣先>要素のシーケンスが含まれているのに対して、2番目の行には1つの<派遣先>要素のシーケンスが含まれています。これは、1番目のXML 文書に3つの<派遣先>要素が含まれていて、XMLQUERY関数はXQuery式を満たすすべての要素のシーケンスを返却したためです。

注意

1件目と3件目に返却された結果にあるXMLデータは、ルート要素が複数存在するため、整形式ではありません。この結果を受け取るアプリケーションが、結果のデータを整形式のXMLデータとして利用したい場合は、以下のいずれかの対処を行ってください。

- アプリケーションにおいて、整形式でない結果のデータを整形式となるように加工してください。
- 結果が必ず整形式となるように、検索対象のXMLデータ、およびXQuery式を修正してください。

XMLQUERY関数がNULLを返却する場合

XQuery式が空のシーケンスを返却する場合、XMLQUERY関数の結果はNULL値になります。

従業員表に対して、以下のような問合せを行ったとします。

```
SELECT 従業員番号, XMLQUERY('/従業員情報/所属[text()="第二開発部"]' PASSING 従業員情報) FROM 従業員スキーマ.従業員表
```

問合せの結果、以下の3件のデータが返却されます。

従業員番号	従業員情報
54319	NULL
54320	<所属 code="f0021" >第二開発部</所属>
54321	NULL

<所属>要素の値が“第二開発部”ではない行に対しては、NULL値が返却されます。2行目については、XQuery式を満たすので<所属>要素が返却されます。

空のシーケンスを含む行の返却を回避するには、WHERE句にXMLEXISTS述語を記述します。たとえば、前述の参照は、以下のようにフィルタを選択リストに指定したXMLQUERY関数ではなくWHERE句に記述することで、空のシーケンスを含む行の返却を回避できます。

```
SELECT 従業員番号, XMLQUERY('/従業員情報/所属' PASSING 従業員情報) FROM 従業員スキーマ.従業員表
WHERE XMLEXISTS('/従業員情報[所属/text()="第二開発部"]' PASSING 従業員情報)
```

問合せの結果、以下の1件のデータが返却されます。

従業員番号	従業員情報
54320	<所属 code="f0021" >第二開発部</所属>

XMLEXISTS述語についての詳細は、“12.2.1.2 XMLEXISTS述語”を参照してください。

12.2.1.2 XMLEXISTS述語

XMLEXISTS述語は、XQuery式が1つ以上の項目のシーケンスを返却するかどうかを評価します。この述語に指定されたXQuery式が空のシーケンスを返す場合、XMLEXISTS述語の結果は偽となります。その他の場合には、真となります。

一般に、XMLQUERY関数は、選択されたXMLデータの一部分を取り出すために選択リストの中で使用されます。XMLQUERY関数のXQuery式に指定された検索条件は、結果として取り出す行を限定するのではなく、XMLデータのどの一部分を取り出すかを決めるためだけに使用されます。

XMLデータの値に応じて結果として取り出す行を限定するには、問合せのWHERE句にXMLEXISTS述語を指定します。

たとえば、次の問合せは、XMLEXISTS述語を使用して、取り出す行を<所属>要素のcode属性の値が“f0021”であるXMLデータを含むものだけに限定する方法を示しています。

SQLは大小文字を区別しませんが、XQuery式は大小文字を区別することに注意してください。

```
SELECT 従業員情報
FROM 従業員スキーマ.従業員表
WHERE XMLEXISTS ('/従業員情報[所属/@code = "f0021" ]' PASSING 従業員情報)
```

注意

結果が正しく返却されるようにするために、XMLEXISTS述語のXQuery式が正しく指定されていることを確認してください。

従業員表から、従業員番号が54321のデータを取り出したい場合を例として説明します。

以下の2つの問合せは、XQuery式が少し異なるために、異なる結果を返します。

```
SELECT *
FROM 従業員スキーマ.従業員表
WHERE XMLEXISTS ('/従業員情報[@従業員番号=54321]' PASSING 従業員情報)
```

```
SELECT *
FROM 従業員スキーマ.従業員表
WHERE XMLEXISTS ('/従業員情報/@従業員番号=54321' PASSING 従業員情報)
```

1つめの問合せでは、XQuery式にパス式を指定しているため、従業員番号が54319、54320の行に対し、XQuery式は空のシーケンスを返します。
したがって、XMLEXISTS述語の結果は偽となり、それらの行が検索結果として取り出されることはなく、従業員番号が54321の行のみが返却されます。

2つめの問合せでは、XQuery式に比較式を指定しているため、従業員番号が54319、54320の行に対し、XQuery式は偽(空ではないシーケンス)を返します。
したがって、XMLEXISTS述語の結果は真となり、検索結果として、XQuery式が真(空ではないシーケンス)を返す、従業員番号が54321の行だけでなく、従業員番号が54319、54320の行も返却されます。このため、予期したとおりの結果を得ることができません。

12.2.1.3 XMLシリアライゼーション

XMLシリアライゼーションは、XML構文解析処理により階層形式に変換したXMLデータを、テキスト形式のXMLデータに変換する処理です。

Symfoware/RDBのXMLシリアライゼーションは、XMLデータ参照時に暗黙的に行われ、XQueryデータモデルの表現から、BLOB型に変換します。

注意

XMLシリアライゼーションの処理内容についての注意点を以下に示します。

- XML宣言は、XMLデータに含まれません。

- DTD宣言は、XMLデータに含まれません。
- XMLデータ内のコメントは、そのまま再現されます。
- 要素の内容の空白は削除します。
- XMLシリアライゼーションを行うときに、対象のシーケンス中に属性要素が含まれている場合には、エラーになります。

12.2.1.4 SQL/XMLで扱えるデータ形式

SQL/XMLで扱えるデータ形式について説明します。

XMLデータの文字コード

SQL/XMLがサポートするXMLデータの文字コードを以下に示します。

- UTF-8コード

XMLデータの形式について

SQL/XMLがサポートするXMLデータの形式について以下に記述します。

SQL/XMLは、XML標準である整形式の条件を満たすXMLデータをサポートします。

整形式の条件について記述します。

- 1つ以上の要素を含んでいる。
- ルートと呼ばれる要素が1つ存在する。
- ルートと呼ばれる要素以外の要素の開始タグが他の要素の内容の中に存在する場合には、終了タグも同じ要素の内容の中に存在する。

ポイント

要素とは、開始タグで始まり、終了タグで終わる単位です。

開始タグと終了タグの間に要素の内容があります。要素の内容が空である場合、空要素タグを記述することができます。

XMLデータの構成要素の記述例を以下に記述します。

```
<Company  Place = "東京">  富士通  </Company>
開始タグ名  属性名  属性値   要素の内容   終了タグ名
```

空要素タグの記述例を以下に記述します。

```
<Company/>
```

QNameを構成するNCNameに記述可能な文字について以下に示します。

構成要素	記述可能な文字
NCName	・ 先頭に記述可能な文字 ― アンダースコア(“_”) ― コロン(“:”) ― Letterクラスの文字 (アルファベット、ひらがな、カタカナ、漢字など) ・ 2文字目以降に記述可能な文字 ― 先頭に記述可能な文字 ― ピリオド(“.”)

構成要素	記述可能な文字
	ー ハイフン(“-”) ー Digitクラスの文字(数字) ー Combining Characterクラスの文字 (アクセント記号など) ー Extenderクラスの文字 (他の文字の後や間に使用される文字)
要素の内容 属性値	キャラクタ(Character)範囲の文字

参照

QNameについては“XQueryリファレンス”を参照してください。

注意

属性値は、二重引用符(“ " ”)、もしくは、引用符(“ ' ”)で囲う必要があり、二重引用符で囲まれている属性値に二重引用符は指定できません。また、引用符で囲まれている属性値に、引用符は指定できません。

以下に正しいXML形式のXMLデータを記述します。

```
<受注伝票>
  <伝票番号>E001001</伝票番号>
  <契約日>2007-01-01</契約日>
  <納入日>2007-01-21</納入日>
  <受注先>
    <社名>ABC銀行</社名>
    <コード>B001</コード>
  </受注先>
  <商品>
    <商品名>FMVM47073</商品名>
    <個数>10個</個数>
  </商品>
</受注伝票>
```

参照

Letterクラスの文字、Digitクラスの文字、Combining Characterの文字、Extenderクラスの文字、および、キャラクタ(character)範囲の文字は、W3Cの勧告として公布されている XML1.0(http://www.w3.org/TR/xml)の内容を参照してください。

12.3 XQueryの検索性能における留意事項

XMLEXISTS述語にXQuery式を指定してXMLデータを検索するとき、XQuery式の指定形式が特定の条件を満たしていれば、XMLデータの検索が速くなる場合があります。これはXQuery式の指定形式に応じて、XMLデータの検索に利用するデータモデルが変化するためです。

以下に示すようなXMLデータの検索を行う際、XQuery式の指定形式が特定の条件に一致していれば、XQueryの検索性能が向上する場合があります。

検索内容	説明
あるノードが存在するかで検索する	検索条件に指定した名前のノードが、XMLデータの中に存在しているかどうかについて検索を行います。

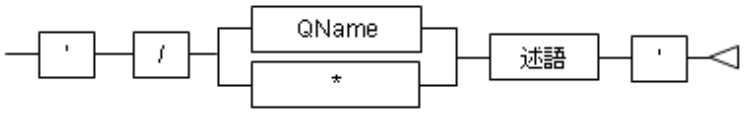
検索内容	説明
あるエレメントノードの属性の値で検索する	検索条件に指定した文字列と、XMLデータ内のあるエレメントノードの属性の値を比較することで検索を行います。
あるエレメントノードの内容で検索する	検索条件に指定した文字列と、XMLデータ内のあるエレメントノードの内容を比較することで検索を行います。

高速検索を可能にするXQuery式

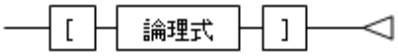
XQueryの検索性能を向上させるためには、以下に示す文法規則に従う必要があります。

記述形式

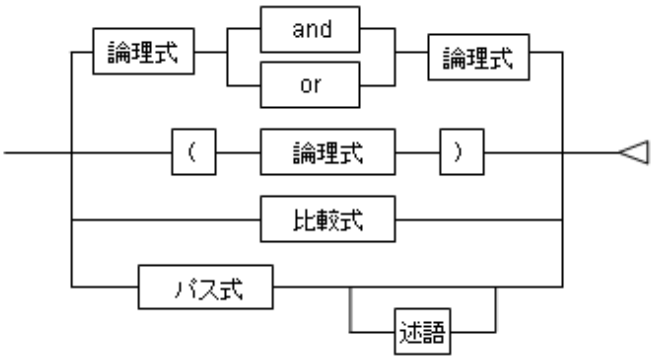
XQuery式



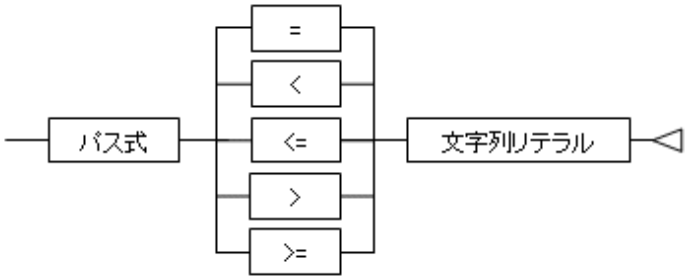
述語



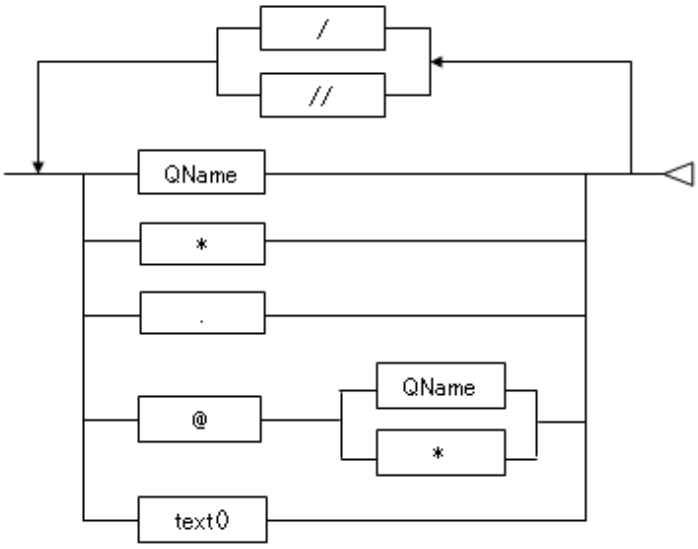
論理式



比較式



パス式



参照

文字列リテラルおよびQNameの指定形式については、“XQueryリファレンス”の“リテラル”または“名前(QName)”を参照してください。

一般規則

XQuery式

XQuery式の最上位には、パス式のように区切り記号“/”または“//”を繰り返し指定することができません。

比較式

- 比較式の右辺には文字列リテラルのみを指定することができます。数値リテラルを指定することはできません。
- 文字列リテラルに既定義エンティティー参照を指定することはできません。

パス式

XQuery式のパス式は、以下の条件をすべて満たしている必要があります。

- “*”または“@*”の直前に“//”が指定されていない。
- “*”または“@*”の直後に“//”が指定されていない。
- “QName”が少なくとも1つ指定されている。
- 述語の直前の式にはエレメントノードの名前(“QName”)が指定されている。
  ただしルートノードの場合だけは、述語の直前でも“*”が指定できる。
- 比較式の左辺のパス式の末尾には、“text()”、“@QName”または“@*”が指定されている。
- 比較演算子に“<”、“<=”、“>”、“>=”が指定されている場合には、比較式の左辺のパス式の末尾に“//text()”、“*/text()”または“@*”が指定されていない。

ここでは、この文法規則に従って高速検索可能なXQuery式を指定するための基本的な考え方について説明します。なお、本節で利用する検索対象のXMLデータは以下のとおりです。

検索対象のXMLデータ

```
<A>
 <B b_id="123">
```

```
  <C c_id="005">test</C>
 </B>
</A>
```

(1) ルートノード名以外の検索条件はすべて述語内に指定する

高速なXQuery検索においては、検索条件の指定箇所が、ルートノード名以外はすべて述語内に指定されていなければなりません。

例えば、ルートノードAの子エレメントノードBの子エレメントノードCが存在しているかどうかでXMLデータを検索したい場合、例1や例2のXQuery式のように“/A”の後に“/B”が指定されていると、高速検索はできません。しかし例3のXQuery式ように“/A”の後に“[”を指定すれば、ルートノード名以外の検索条件が述語内に指定されていることになり、高速検索が可能になります。

例1

```
SELECT XMLデータ列 FROM S1.T1 WHERE XMLEXISTS('/A/B/C' PASSING XMLデータ列)
```

例2

```
SELECT XMLデータ列 FROM S1.T1 WHERE XMLEXISTS('/A/B[C]' PASSING XMLデータ列)
```

例3

```
SELECT XMLデータ列 FROM S1.T1 WHERE XMLEXISTS('/A[B/C]' PASSING XMLデータ列)
```

(2) 検索対象のXMLデータの属性値または内容は、文字列と比較する

高速なXQuery検索においては、XMLデータの属性値または内容を文字列で比較することはできますが、数値で比較することはできません。そのため、数値と文字列で比較結果が同じであれば、文字列で比較する方がXQuery検索を高速化できます。

例えば、例4と例5のXQuery式は、検索対象のXMLデータに対して同じ結果を返すため、数値で比較している例4よりも文字列で比較している例5のXQuery式を指定した方が、高速にXMLデータを検索できます。

これに対し、例6と例7のXQuery式は、検索対象のXMLデータに対して異なる結果を返します。例6の場合は、XMLデータの属性c_idの値“005”は数値5として比較されるため、検索条件は一致します。しかし、例7の場合は、XMLデータの属性c_idの値“005”は文字列として比較されるため、検索条件の文字列“5”とは一致しません。そのため、例6の検索を高速化するためには、検索対象のXMLデータの形式に合わせて、例8のようなXQuery式を指定する必要があります。

例4

属性b_idの値を数値123で検索

```
SELECT XMLデータ列 FROM S1.T1 WHERE XMLEXISTS('/A[B/@b_id=123]' PASSING XMLデータ列)
```

例5

属性b_idの値を文字列“123”で検索

```
SELECT XMLデータ列 FROM S1.T1 WHERE XMLEXISTS('/A[B/@b_id="123"]' PASSING XMLデータ列)
```

例6

属性c_idの値を数値5で検索

```
SELECT XMLデータ列 FROM S1.T1 WHERE XMLEXISTS('/A[B/C/@c_id=5]' PASSING XMLデータ列)
```

例7

属性c_idの値を文字列“5”で検索

```
SELECT XMLデータ列 FROM S1.T1 WHERE XMLEXISTS('/A[B/C/@c_id="5"]' PASSING XMLデータ列)
```

例8

属性c_idの値を文字列“005”で検索

```
SELECT XMLデータ列 FROM S1.T1 WHERE XMLEXISTS('/A[B/C/@c_id="005"]' PASSING XMLデータ列)
```

(3) XQuery式の指定内容を文法規則に合致させる

前述した(1)と(2)の考え方で、基本となるXQuery式の指定内容が決定したら、最後にその指定内容が文法規則に合致しているかどうかを確認します。指定内容が上述の文法規則に合致していれば、XQuery検索は高速になりますが、合致していなければXQuery検索は高速になりません。

例えば、以下のような検索条件でXMLデータを検索する場合、高速検索可能なXQuery式の指定内容は前述した(1)～(3)の考え方で決定します。

検索条件

```
・ルートノードAの子エレメントノードBの属性b_idの値が123である。
・ルートノードBの子エレメントノードCの内容が“test”である。
```

まず、上記の検索条件を満たすXQuery式として例9のようなXQuery式が考えられます。例9のXQuery式に対して、(1)で説明した考え方を適用すると、例10のようなXQuery式に変換できます。さらに例10のXQuery式に対して、(2)で説明した考え方を適用すると、例11のようなXQuery式に変換できます。最後に例11のXQuery式に対して、(3)で説明した考え方を適用し、比較式の左辺にパス式、右辺に文字列リテラルを指定すると、例12のようなXQuery式に変換できます。例12のXQuery式であれば高速なXQuery検索が可能です。

例9

検索条件より作成

```
SELECT XMLデータ列 FROM S1.T1 WHERE XMLEXISTS('/A/B[@b_id=123 and "test"=C/text()]' PASSING XMLデータ列)
```

例10

(1)の考え方を適用

```
SELECT XMLデータ列 FROM S1.T1 WHERE XMLEXISTS('/A[B/@b_id=123 and "test"=B/C/text()]' PASSING XMLデータ列)
```

例11

(2)の考え方を適用

```
SELECT XMLデータ列 FROM S1.T1 WHERE XMLEXISTS('/A[B/@b_id="123" and "test"=B/C/text()]' PASSING XMLデータ列)
```

例12

(3)の考え方を適用

```
SELECT XMLデータ列 FROM S1.T1 WHERE XMLEXISTS('/A[B/@b_id="123" and B/C/text()="test"]' PASSING XMLデータ列)
```

12.4 XMLデータの形式における留意事項

データベースに格納されたXMLデータの形式に関する留意事項は以下のとおりです。

- XMLデータの妥当性検証は実施しません。
- XMLデータの内部実体展開、および、外部実体の展開は行いません。
- XML宣言に記述された文字コードは無視され、UTF-8の文字コードとして扱います。
- XMLデータの属性値の正規化(連続する半角空白、復帰コード、改行コード、タブを半角空白で置き換えること)を行います。

また、XMLEXISTS述語におけるXMLデータの検索では、以下に分類される整形式でないXMLデータを、正しいXMLデータと許容して動作する場合があります。

・ UTF-8形式以外のデータ

XMLデータがUTF-8の形式であるか否かの検証をしていないため、UTF-8形式でなくても許容する場合があります。

・ 制御コードを含むデータ

XMLデータ中に存在する制御コードが、整形式の条件で認められているか否かの検証は行っていないため、整形式の条件で認められていない制御コードを許容する場合があります。

・ 整形式の条件に一致していないデータ

以下のように整形式の条件に一致していないXMLデータであっても、検索対象として動作する場合があります。

― ルートノードの開始タグより前、または終了タグより後に不要な文字が記述されている場合

― 1つの要素内に同じ属性名が重複して記述されている場合(それぞれの属性が検索対象となります)

― 開始タグ名と終了タグ名が異なる場合(開始タグ名が検索対象となり、終了タグ名は無視します)

― 終了タグ内に属性が記述されている場合(その属性は検索対象とならずに、無視します)

― タグ名、または属性名の先頭がピリオド(“.”)、ハイフン(“-”)、数字(0～9)から始まる場合

― コメントに“--”という文字列が指定された場合

― 処理命令(PI)のターゲット名に“XML”や“xml”を指定した場合

12.5 XQueryの指定方法

データベースに以下のような、従業員情報のデータと、製品情報のデータを同じBLOB列に格納してあるとします。このデータベースからSQL/XMLでさまざまな条件で検索を行うことができます。

ここでは、SQL/XMLで検索条件を指定するXQuery式の指定方法について例を用いて説明します。

従業員情報

```
<employees>
  <employee id = "55299">
    <name>
      <family>佐藤</family>
      <first>正志</first>
    </name>
    <position>社員</position>
    <office>丸の内</office>
    <telephone>1111-XXXX</telephone>
    <mail>XXXX01@mail.co.jp</mail>
  </employee>
  <employee id = "55300">
    <name>
      <family>田中</family>
      <first>正文</first>
    </name>
    <position>社員</position>
    <office>丸の内</office>
    <telephone>2222-XXXX</telephone>
    <mail>XXXX02@mail.co.jp</mail>
  </employee>
  <employee id = "55301">
    <name>
      <family>鈴木</family>
      <first>英二</first>
    </name>
    <position>課長</position>
    <office>丸の内</office>
    <telephone>3333-XXXX</telephone>
    <mail>XXXX03@mail.co.jp</mail>
  </employee>
```

```
  <employee id = "55302">
    <name>
      <family>田中</family>
      <first>文雄</first>
    </name>
    <position>社員</position>
    <office>新横浜</office>
    <telephone>4444-XXXX</telephone>
    <mail>XXXX04@mail.co.jp</mail>
  </employee>
  <employee id = "55303">
    <name>
      <family>加藤</family>
      <first>明夫</first>
    </name>
    <position>社員</position>
    <office>新横浜</office>
    <telephone>5555-XXXX</telephone>
    <mail>XXXX05@mail.co.jp</mail>
  </employee>
</employees>
```

製品情報

```
<products>
  <product id="0001">
    <kind>家電製品</kind>
    <item>
      <name>テレビ</name>
      <code>XA-0054</code>
    </item>
    <value>190000</value>
    <stock>10</stock>
    <release>2008-03-03</release>
  </product>
  <product id="0002">
    <kind>家電製品</kind>
    <item>
      <name>パソコン</name>
      <code>XB-3984</code>
    </item>
    <value>220000</value>
    <stock>25</stock>
    <release>2008-02-10</release>
  </product>
  <product id="0003">
    <kind>家電製品</kind>
    <item>
      <name>ドライヤー</name>
      <code>XC-0883</code>
    </item>
    <value>9800</value>
    <stock>0</stock>
    <discount>-5000</discount>
    <release>2007-12-10</release>
  </product>
  <product id="0004">
    <kind>スポーツ用品</kind>
    <item>
      <name>バット</name>
      <code>WB-4432</code>
    </item>
    <value>35000</value>
```

```
    <stock>5</stock>
    <discount>-1000</discount>
    <release>2007-10-20</release>
  </product>
  <product id="0005">
    <kind>スポーツ用品</kind>
    <item>
      <name>テニスラケット</name>
      <code>WC-7763</code>
    </item>
    <value>15800</value>
    <stock>0</stock>
  </product>
</products>
```

12.5.1 ある要素を検索する例

製品情報から、割引(discount)があるデータを検索します。

以下にXQuery式と、返却されるXMLデータを示します。

XQuery式

```
/products/product[discount]
```

返却結果

```
<product id="0003">
  <kind>家電製品</kind>
  <item>
    <name>ドライヤー</name>
    <code>XC-0883</code>
  </item>
  <value>9800</value>
  <stock>15</stock>
  <discount>-5000</discount>
  <release>2007-12-10</release>
</product>
<product id="0004">
  <kind>スポーツ用品</kind>
  <item>
    <name>バット</name>
    <code>WB-4432</code>
  </item>
  <value>35000</value>
  <stock>5</stock>
  <discount>-1000</discount>
  <release>2007-10-20</release>
</product>
```

12.5.2 ある要素の内容を数値の大小比較で検索する例

製品情報から、割引(discount)の値が-3000より小さい値であるデータを検索します。

以下にXQuery式と、返却されるXMLデータを示します。

XQuery式

```
/products/product[discount/text() < -3000 ]
```

返却結果

```
<product id="0003">
  <kind>
```

```
    <name>家電製品</name>
    <code>XC-0883</code>
  </kind>
  <item>ドライヤー</item>
  <value>9800</value>
  <stock>15</stock>
  <discount>-5000</discount>
  <release>2007-12-10</release>
</product>
```

12.5.3 ある要素の内容を検索する例

製品情報から、品名(name)のテキストノードが“テレビ”であるデータを検索します。

以下にXQuery式と、返却されるXMLデータを示します。

XQuery式

```
/products/product[item/name/text()="テレビ"]
```

返却結果

```
<product id="0001">
  <kind>家電製品</kind>
  <item>
    <name>テレビ</name>
    <code>XA-0054</code>
  </item>
  <value>190000</value>
  <stock>10</stock>
  <release>2008-03-03</release>
</product>
```

12.5.4 ある要素の内容をタグを除いて連結した文字列で検索する例

従業員情報から、従業員(employee)の配下に名前(name)があり、その名前(name)の苗字(family) “佐藤”と名前(first)“正志”をつなげた文字列を検索します。

以下にXQuery式と、返却されるXMLデータを示します。

XQuery式

```
/employees/employee[name = "佐藤正志"]
```

返却結果

```
<employee id = "55299">
  <name>
    <family>佐藤</family>
    <first>正志</first>
  </name>
  <position>社員</position>
  <office>丸の内</office>
  <telephone>1111-XXXX</telephone>
  <mail>XXXX01@mail.co.jp</mail>
</employee>
```

12.5.5 すべての要素の内容を検索する例

製品情報から、製品(product)配下のすべてのテキストノードを対象に、値が“テレビ”であるデータを検索します。

以下にXQuery式と、返却されるXMLデータを示します。

XQuery式

```
/products/product[.//text()="テレビ"]
```

返却結果

```
<product id="0001">
  <kind>家電製品</kind>
  <item>
    <name>テレビ</name>
    <code>XA-0054</code>
  </item>
  <value>190000</value>
  <stock>10</stock>
  <release>2008-03-03</release>
</product>
```

12.5.6 ある階層にあるすべての要素の内容を検索する

製品情報から、製品(product)配下の任意のエレメントノードのテキストノードが“スポーツ用品”であるデータを検索します。

以下にXQuery式と、返却されるXMLデータを示します。

XQuery式

```
/products/product[./*/text()="スポーツ用品"]
```

返却結果

```
<product id="0004">
  <kind>スポーツ用品</kind>
  <item>
    <name>バット</name>
    <code>WB-4432</code>
  </item>
  <value>35000</value>
  <stock>5</stock>
  <discount>-1000</discount>
  <release>2007-10-20</release>
</product>
<product id="0005">
  <kind>スポーツ用品</kind>
  <item>
    <name>テニスラケット</name>
    <code>WC-7763</code>
  </item>
  <value>15800</value>
  <stock>0</stock>
</product>
```

12.5.7 ある要素の属性の値を検索する例

製品情報から、製品(product)の属性のidが“0001”であるデータを検索します。

以下にXQuery式と、返却されるXMLデータを示します。

XQuery式

```
/products/product[@id="0001"]
```

返却結果

```
<product id="0001">
  <kind>
```

```
    <name>家電製品</name>
    <code>XA-0054</code>
  </kind>
  <item>テレビ</item>
  <value>190000</value>
  <stock>10</stock>
  <release>2008-03-03</release>
</product>
```

12.5.8 ある要素の内容から現在の日時を検索する例

現在の日付を2008年3月3日とします。

製品情報から、発売日(release)が現在の日付(2008年3月3日)のデータを検索します。

以下にXQuery式と、返却されるXMLデータを示します。

XQuery式

```
/products/product[release/text()=fn:current-date()]
```

返却結果

```
<product id="0001">
  <kind>家電製品</kind>
  <item>
    <name>テレビ</name>
    <code>XA-0054</code>
  </item>
  <value>190000</value>
  <stock>10</stock>
  <release>2008-03-03</release>
</product>
```

12.5.9 ある要素の内容から指定した文字列を検索する例

製品情報から、種類(kind)に“スポーツ”という文字を含んでいるデータを検索します。

以下にXQuery式と、返却されるXMLデータを示します。

XQuery式

```
/products/product[fn:contains(kind/text(),"スポーツ")]
```

返却結果

```
<product id="0004">
  <kind>スポーツ用品</kind>
  <item>
    <name>バット</name>
    <code>WB-4432</code>
  </item>
  <value>35000</value>
  <stock>5</stock>
  <discount>-1000</discount>
  <release>2007-10-20</release>
</product>
<product id="0005">
  <kind>スポーツ用品</kind>
  <item>
    <name>テニスラケット</name>
    <code>WC-7763</code>
  </item>
  <value>15800</value>
```

```
  <stock>0</stock>
</product>
```

12.5.10 2つの条件を論理積を使って検索する例

製品情報から、在庫(stock)がなくなっていて、かつ種類(kind)が“スポーツ用品”であるデータを検索します。

以下にXQuery式と、返却されるXMLデータを示します。

XQuery式

```
/products/product[stock/text()=0 and kind/text()="スポーツ用品" ]
```

返却結果

```
<product id="0005">
  <kind>スポーツ用品</kind>
  <item>
    <name>テニスラケット</name>
    <code>WC-7763</code>
  </item>
  <value>15800</value>
  <stock>0</stock>
</product>
```

12.5.11 ある検索結果を元に検索をする例

製品情報から、割引(discount)がある製品を検索し、その品名(name)とコード(code)と価格(value)を返します。

以下にXQuery式と、返却されるXMLデータを示します。

XQuery式

```
for $prod in /products/product[discount] return ( $prod/item/name/text() , "(" , $prod/item/code/text() ,
")" , "=" , $prod/value/text() )
```

返却結果

```
ドライヤー(XC-0883)=9800
バット(WB-4432)=35000
```

12.5.12 ある条件に応じて検索結果を分岐する

従業員情報から、従業員の役職(position)が“課長”の社員がいたら、従業員氏名(name)、電話番号(telephone)、メールアドレス(mail)、勤務地(office)を返却します。そうでない場合は、従業員氏名(name)、電話番号(telephone)を返却します。

以下にXQuery式と、返却されるXMLデータを示します。

XQuery式

```
for $employee in /employees/employee return
if($employee[position/text()="課長"]) then
($employee[position]/name,
 $employee[position]/telephone,
 $employee[position]/mail,
 $employee[position]/office)
else
($employee[position]/name,
 $employee[position]/telephone)
```

返却結果

```
<name>
  <family>佐藤</family><first>正志</first>
```

```
</name>
<telephone>1111-XXXX</telephone>
<name>
  <family>田中</family><first>正文</first>
</name>
<telephone>2222-XXXX</telephone>
<name>
  <family>鈴木</family><first>英二</first>
</name>
<telephone>3333-XXXX</telephone>
<mail>XXXX03@mail.co.jp</mail>
<office>丸の内</office>
<name>
  <family>田中</family><first>文雄</first>
</name>
<telephone>4444-XXXX</telephone>
<name>
  <family>加藤</family><first>明夫</first>
</name>
<telephone>5555-XXXX</telephone>
```

12.5.13 ある検索結果がある条件を満たしているかを判定する

製品情報から、製品名(name)が“テレビ”で、在庫(stock)が0より多い製品があるかを判定します。その結果が真の場合、種類(kind)が“家電製品”のデータをすべて返却します。

以下にXQuery式と、返却されるXMLデータを示します。

XQuery式

```
if (some $prod in /products/product[item/name = "テレビ"] satisfies  $prod/stock > 0 ) then /products/
product[kind/text() = "家電製品"] else ()
```

返却結果

```
<product id="0001">
  <kind>家電製品</kind>
  <item>
    <name>テレビ</name>
    <code>XA-0054</code>
  </item>
  <value>190000</value>
  <stock>10</stock>
  <release>2008-03-03</release>
</product>
<product id="0002">
  <kind>家電製品</kind>
  <item>
    <name>パソコン</name>
    <code>XB-3984</code>
  </item>
  <value>220000</value>
  <stock>25</stock>
  <release>2008-02-10</release>
</product>
<product id="0003">
  <kind>家電製品</kind>
  <item>
    <name>ドライヤー</name>
    <code>XC-0883</code>
  </item>
  <value>9800</value>
  <stock>0</stock>
  <discount>-5000</discount>
```

```
  <release>2007-12-10</release>
</product>
```

付録A　システム表

Symfoware/RDBは、データベースの運用を効率的に行うために、いくつかの表でデータベースやスキーマを管理しています。この表を、システム表またはRDBディクショナリと呼びます。

システム表は、RDBディクショナリの作成コマンドを実行すると、Symfoware/RDB自身が定義します。また、データベース簡単運用の場合は、Symfoware Serverのインストール時に、自動的に作成されます。

参照

システム表の作成方法については、“セットアップガイド”を参照してください。

ここでは、システム表について、以下の順に説明します。

A.1　システム表の概要

システム表には、表A.1 システム表の構成の表およびビューが含まれています。

表A.1 システム表の構成

表名	定義情報	ビュー表	利用者
RDBII_DATABASE	データベースの定義情報	DATABASE	全データベース
RDBII_SCHEMA	スキーマの定義情報	SCHEMATA	資源の作成、削除可能なスキーマ一覧
RDBII_TABLE	実表、ビュー表、一時表の定義情報	TABLES	アクセス可能な表の一覧
RDBII_TABLE_EX	実表、ビュー表、一時表の定義拡張情報	－	アクセス可能な表の一覧
RDBII_COLUMN	列の定義情報	COLUMNS	アクセス可能な表の行一覧
RDBII_CONSTRAINT	制約の定義情報	TABLE_CONSTRAINTS	資源の作成、削除可能なスキーマ配下の定義情報
RDBII_DESCRIPTION	ビュー表を定義する文字列	－	－
RDBII_TABLE_USAGE	ビュー表、ルーチンで指定された表、およびトリガで指定された表の一覧	－	－
RDBII_COLUMN_USAGE	表を構成する列の一覧	KEY_COLUMN_USAGE	資源の作成、削除可能なスキーマ配下の定義情報
RDBII_COMMENT	スキーマ、実表、ビュー表、一時表、ルーチン、列、インデックス、トリガに指定したコメントの一覧	－	－
RDBII_INDEX	格納構造定義を行わずに定義した表のインデックスに関する情報の一覧	－	－
RDBII_INDEX_EX	格納構造定義を行わずに定義した表のインデックスに関する拡張情報の一覧	－	－
RDBII_PROC	ルーチン定義情報	PROCS	－

表名	定義情報	ビュー表	利用者
RDBII_PROC_EX	ルーチン定義拡張情報	－	－
RDBII_PROC_COL	ルーチンパラメタ情報	PROC_COLS	－
RDBII_PRIVILEGES	権限情報	RSC_PRIVILEGES	利用者の持つ権限情報
RDBII_TRIGGER	トリガ定義情報	TRIGGERS	資源の作成、削除可能なスキーマ配下のトリガ情報
RDBII_TRIG_COLUMNS	トリガ列一覧情報	TRIGGERED_COLUMNS	資源の作成、削除可能なスキーマ配下のトリガ列情報
RDBII_SEQUENCE	順序定義情報	SEQUENCES	アクセス可能な順序の一覧
RDBII_PARAMETER	ユーザパラメタ設定情報	PARAMETERS	－
RDBII_USER	利用者情報	USERS	その利用者の情報
RDBII_ROLE	ロール情報	ROLES	－
RDBII_ROLE_PRIV	ロールに付与された権限の情報	ROLE_PRIVILEGES	－
RDBII_ROLE_GRANT	利用者に対するロールの付与情報	ROLE_GRANTS	－
RDBII_ASSISTTABLE	－	－	－
RDBII_DSI_STATUS	DSI情報	－	－
RDBII_CON_PRM	コネクションの動作環境のパラメタ設定情報(一括出力)	－	－
RDBII_CON_PRM_DIV	コネクションの動作環境のパラメタ設定情報(順序付き出力)	－	－
RDBII_SYS_PRM	システムの動作環境のパラメタ設定情報(一括出力)	－	－
RDBII_SYS_PRM_DIV	システムの動作環境のパラメタ設定情報(順序付き出力)	－	－
RDBII_CON_INF	コネクション情報	－	－

上記の表およびビューのデータベース名とスキーマ名を以下に示します。

データベース名:

RDBII_DICTIONARY

スキーマ名:

RDBII_SYSTEM

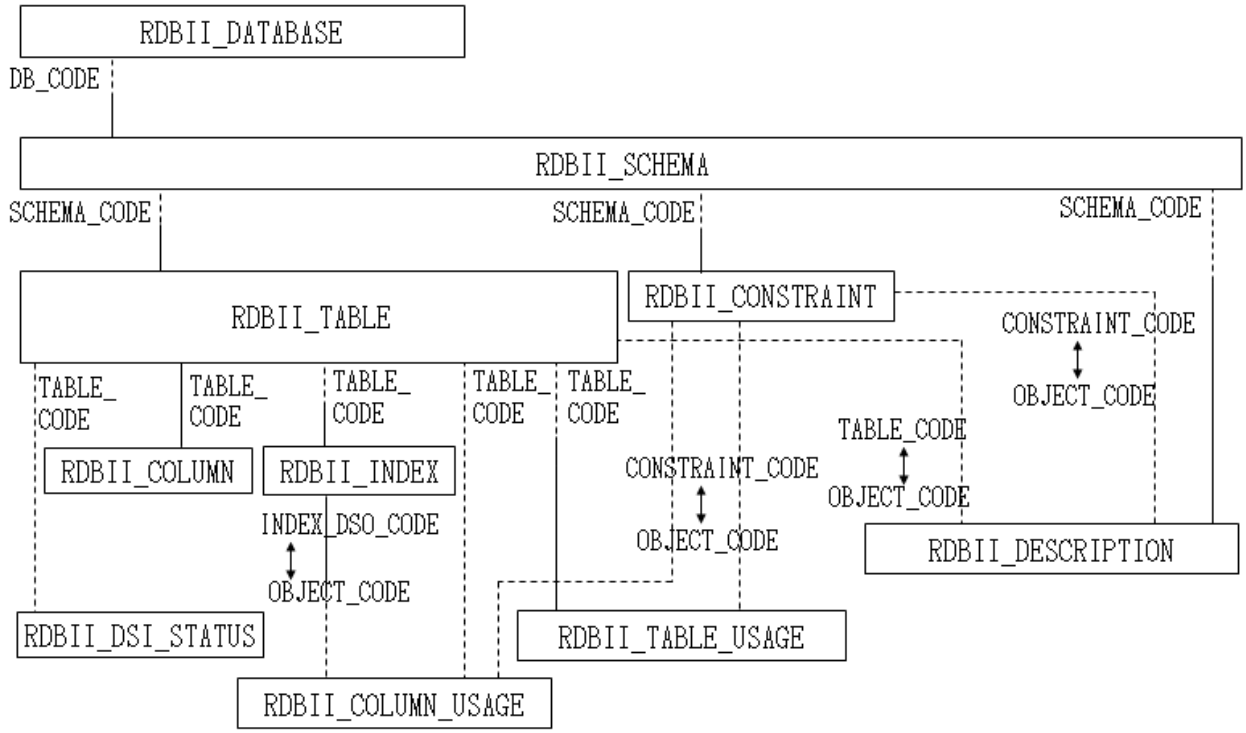
A.2 システム表の参照

システム表は、データベースの定義情報を、アプリケーションから利用したい場合に参照します。また、一般の表またはビュー表と同様に、SELECT文で検索できます。ただし、システム表のデータをSQL文を使用して更新することはできません。

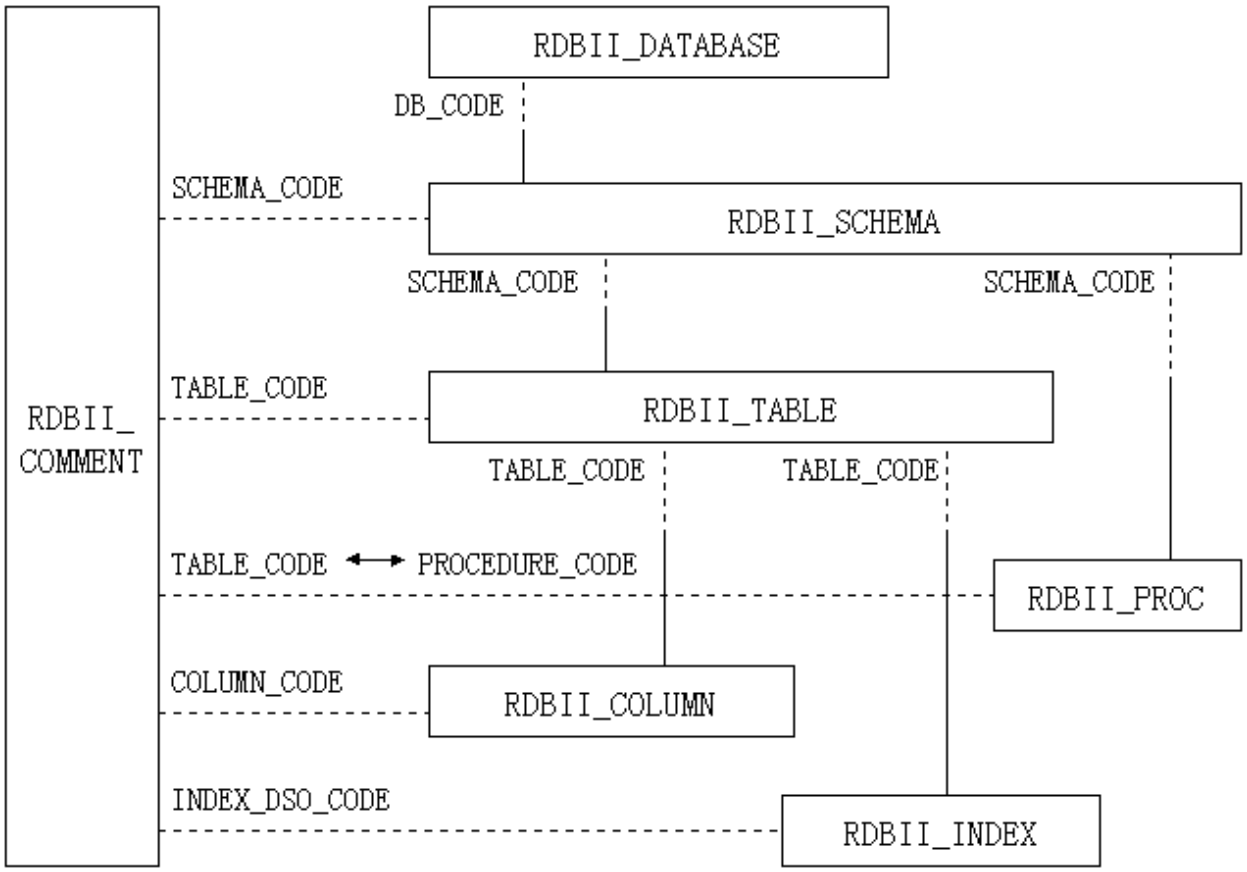
システム表の全体構成を以下に示します。

［基本構成］

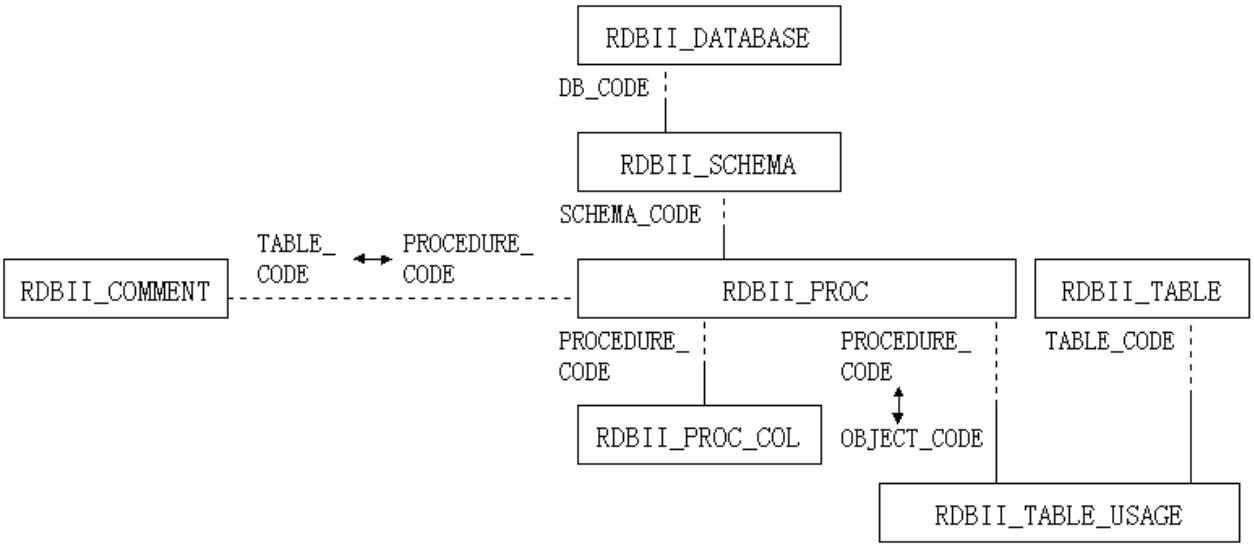
［基本構成］


RDBII_DATABASE
DB_CODE
RDBII_SCHEMA
SCHEMA_CODE
SCHEMA_CODE
SCHEMA_CODE
RDBII_TABLE
RDBII_CONSTRAINT
CONSTRAINT_CODE
OBJECT_CODE
TABLE_ CODE
TABLE_ CODE
TABLE_ CODE
TABLE_ CODE
TABLE_ CODE
TABLE_CODE
OBJECT_CODE
RDBII_COLUMN
RDBII_INDEX
CONSTRAINT_CODE
OBJECT_CODE
INDEX_DSO_CODE
OBJECT_CODE
RDBII_DESCRIPTION
RDBII_DSI_STATUS
RDBII_TABLE_USAGE
RDBII_COLUMN_USAGE

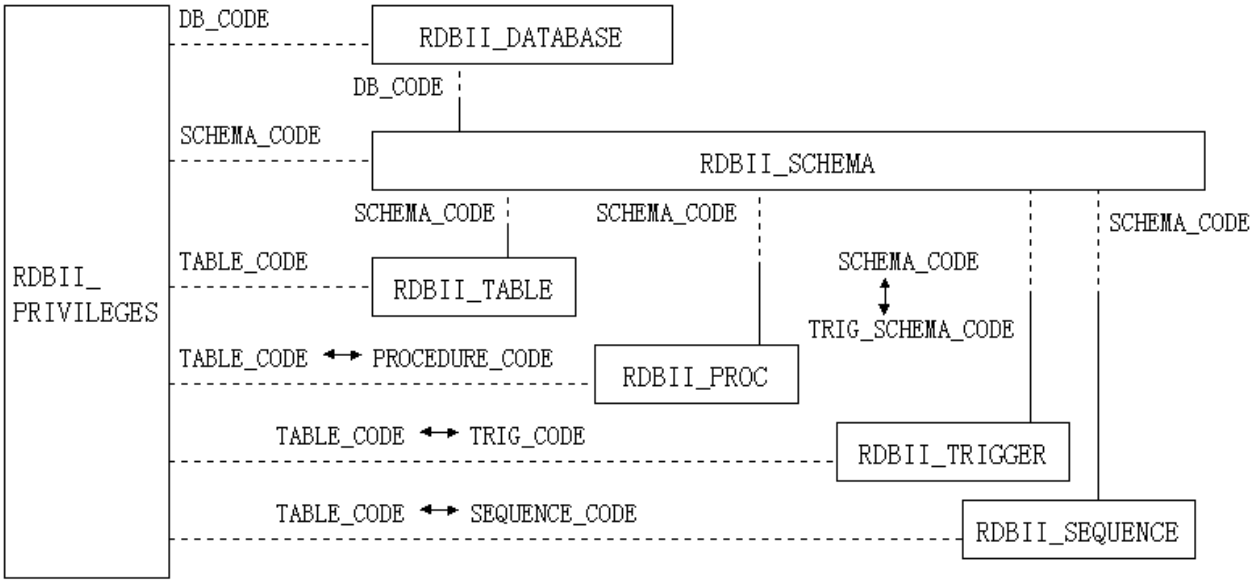
[注釈情報に関連するシステムテーブルの関連図]



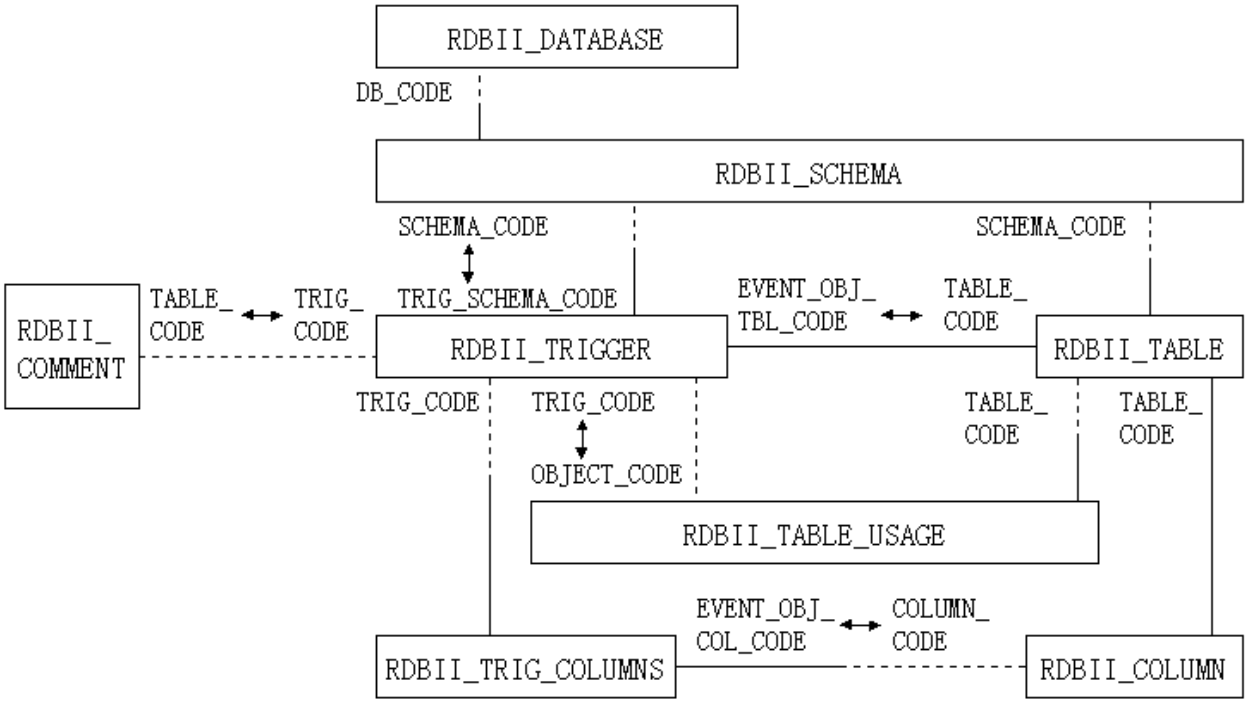
[ルーチン定義に関連するシステムテーブルの関連図]



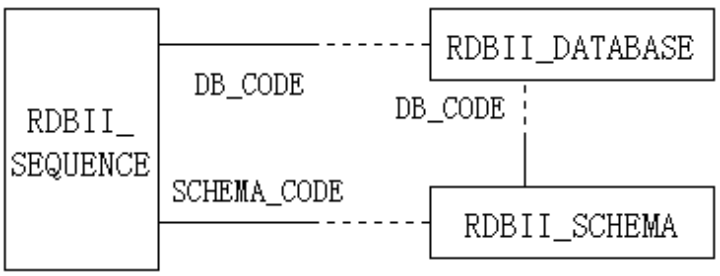
[権限情報に関連するシステムテーブルの関連図]



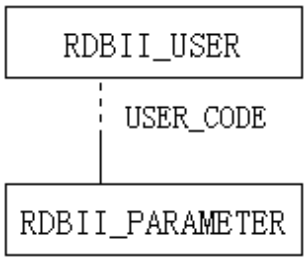
[トリガに関連するシステムテーブルの関連図]



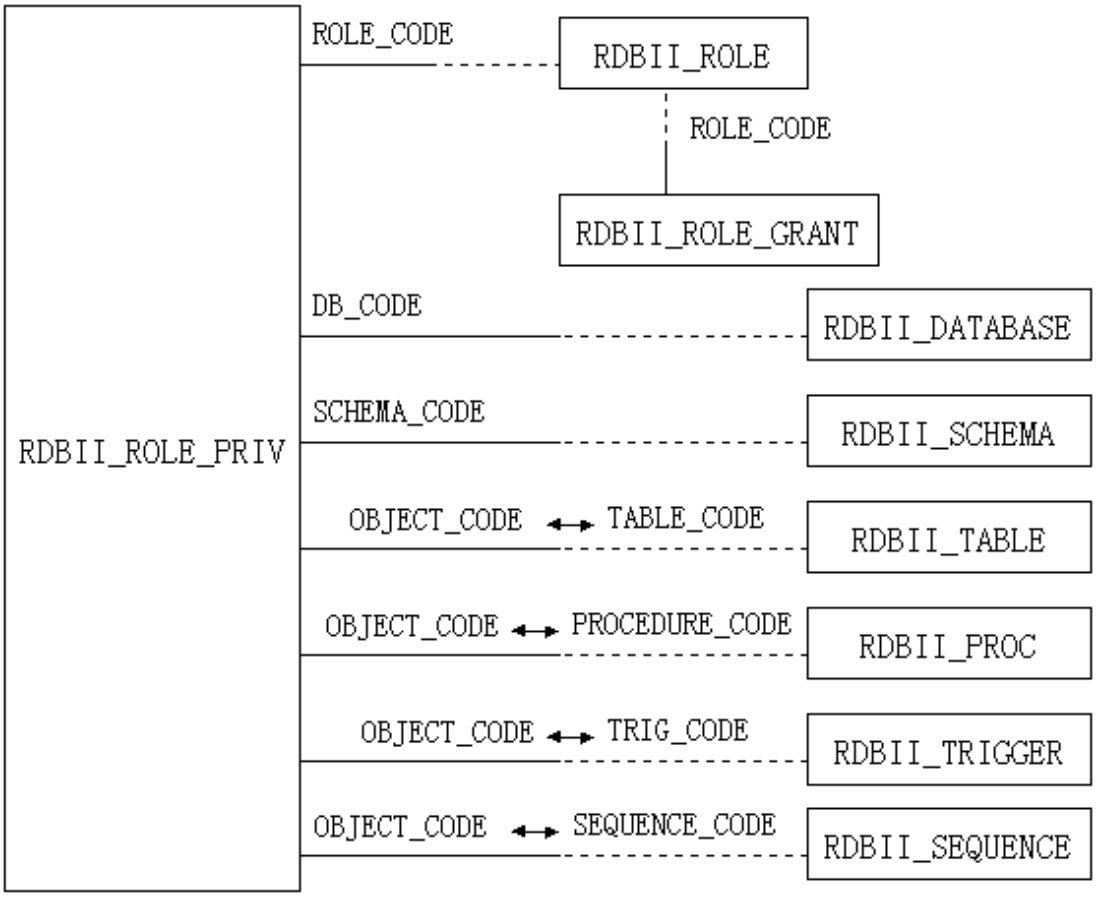
[順序情報に関連するシステムテーブルの関連図]

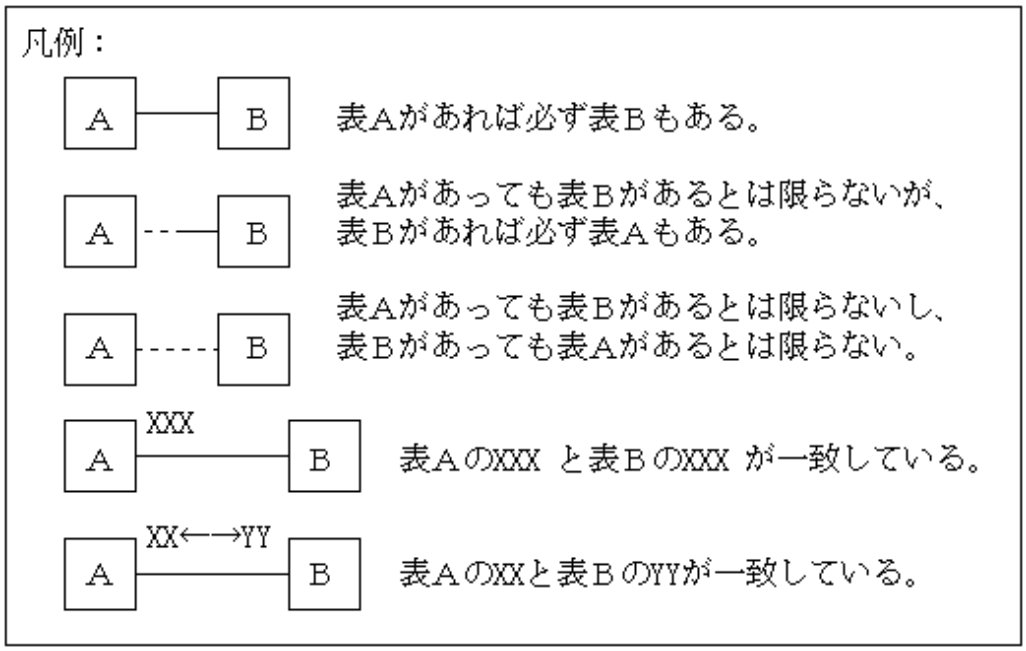


［利用者とパラメタの関連図］



［ロールの関連図］





A.3 システム表の詳細

システム表を構成する各表およびビュー表について説明します。

A.3.1 データベース定義情報

RDBII_DATABASE

RDBII_DATABASEは、データベースの定義情報を出力する表です。RDBII_DATABASEの構成と出力情報を以下に示します。

表A.2 RDBII_DATABASEの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
DB_NAME	CHAR(36) または CHAR(128)	データベース名	1		Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
DB_CODE	INT	データベースコード			
DB_OWNER	CHAR(36) または CHAR(128)	データベース定義者名			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
CREATE_TIME	CHAR(8)	データベース定義日時			
DB_USAGE	CHAR(2)	データベース属性			SY:システム US:ユーザ定義データベース

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。

DATABASE

DATABASEは、データベースの定義情報を出力するビュー表です。ビュー表の構成と出力情報を以下に示します。

表A.3 ビュー表DATABASEの構成と出力情報

列	データ型	情報の意味	補足説明
DB_NAME	CHAR(36) または CHAR(128)	データベース名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
DB_CODE	INT	データベースコード	
DB_OWNER	CHAR(36) または CHAR(128)	データベース定義者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
DB_USAGE	CHAR(2)	データベース属性	SY:システム US:ユーザ定義データベース

A.3.2 スキーマ定義情報

RDBII_SCHEMA

RDBII_SCHEMAは、スキーマの定義情報を出力する表です。RDBII_SCHEMAの構成と出力情報を以下に示します。

表A.4 RDBII_SCHEMAの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
DB_NAME	CHAR(36) または CHAR(128)	データベース名		1	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
SCHEMA_NAME	CHAR(36) または CHAR(128)	スキーマ名	2	2	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
DB_CODE	INT	データベースコード	1		
SCHEMA_CODE	INT	スキーマコード			
SCHEMA_OWNER	CHAR(36) または CHAR(128)	スキーマ定義者名			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
CREATE_TIME	CHAR(8)	スキーマ定義日時			
MAX_OBJECT_TIC	INT	スキーマ内の表制約数			

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。数字は、インデックス内の構成順序を示します。

SCHEMATA

スキーマの定義情報を出力するビュー表です。このビュー表を検索する場合、利用者に権限のあるスキーマの情報のみを出力します。このビュー表の構成と出力情報を以下に示します。

表A.5 ビュー表SCHEMATAの構成と出力情報

列名	データ型	情報の意味	補足説明
SCHEMA_NAME	CHAR(36) または CHAR(128)	スキーマ名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
SCHEMA_OWNER	CHAR(36) または CHAR(128)	スキーマ定義者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
SCHEMA_CODE	INT	スキーマコード	
DB_CODE	INT	データベースコード	

A.3.3 表定義情報

RDBII_TABLE

RDBII_TABLEは、表の定義情報を出力する表です。RDBII_TABLEの構成と出力情報を以下に示します。

表A.6 RDBII_TABLEの構成と出力情報

列名	データ型	情報の意味	P	SI	SI	補足説明
DB_NAME	CHAR(36) または CHAR(128)	データベース名		1		Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
SCHEMA_NAME	CHAR(36) または CHAR(128)	スキーマ名		2		Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TABLE_NAME	CHAR(36) または CHAR(128)	表名	3	3		Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TABLE_OWNER	CHAR(36) または CHAR(128)	テーブル定義者名				Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
CREATE_TIME	CHAR(8)	スキーマ定義日時				
DB_CODE	INT	データベースコード	1		1	
SCHEMA_CODE	INT	スキーマコード	2		2	
TABLE_CODE	INT	表コード、ビュー表コードまたは一時表コード			3	
TABLE_TYPE	CHAR(2)	表の属性				BS:実表 VW:ビュー表

列名	データ型	情報の意味	P	SI	SI	補足説明
						TD:一時表(ON COMMIT DELETE ROWS指定) RT:リモート表 TP:一時表(ON COMMIT PRESERVE ROWS指定)
UPDATABLE	CHAR(1)	表の更新可能性(ビュー表のみ)				Y:更新可能なビュー表 N:更新不可能なビュー表
CHECK_OPTION	CHAR(1)	表の更新時の検査オプション(ビュー表のみ)				Y:WITH CHECK OPTIONあり N:WITH CHECK OPTIONなし

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。数字は、インデックス内の構成順序を示します。

TABLES

表の定義情報を出力するビュー表です。このビュー表を検索する場合、利用者に権限のあるスキーマの情報のみを出力します。このビュー表の構成と出力情報を以下に示します。

表A.7 ビュー表TABLESの構成と出力情報

列名	データ型	情報の意味	補足説明
TABLE_NAME	CHAR(36) または CHAR(128)	表名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TABLE_CODE	INT	表コード	
DB_CODE	INT	データベースコード	
SCHEMA_CODE	INT	スキーマコード	
TABLE_OWNER	CHAR(36) または CHAR(128)	表の定義者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TABLE_TYPE	CHAR(2)	表の属性	BS:実表 VW:ビュー表 TD:一時表(ON COMMIT DELETE ROWS指定) RT:リモート表 TP:一時表(ON COMMIT PRESERVE ROWS指定)

A.3.4 表定義拡張情報

RDBII_TABLE_EX

RDBII_TABLE_EXは、表の定義の拡張情報を出力する表です。利用者が定義した定義情報を出力対象とします。システム表および監査ログデータベースに関する定義情報は出力しません。

RDBII_TABLE_EXの構成と出力情報を以下に示します。

表A.8 RDBII_TABLE_EXの構成と出力情報

列名	データ型	情報の意味	補足説明
DB_NAME	CHAR(36) または CHAR(128)	データベース名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
SCHEMA_NAME	CHAR(36) または CHAR(128)	スキーマ名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TABLE_NAME	CHAR(36) または CHAR(128)	表名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TABLE_OWNER	CHAR(36) または CHAR(128)	テーブル定義者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
CREATE_TIME	CHAR(8)	スキーマ定義日時	
DB_CODE	INT	データベースコード	
SCHEMA_CODE	INT	スキーマコード	
TABLE_CODE	INT	表コード、ビュー表コードまたは一時表コード	
TABLE_TYPE	CHAR(2)	表の属性	BS:実表 VW:ビュー表 TD:一時表(ON COMMIT DELETE ROWS指定) RT:リモート表 TP:一時表(ON COMMIT PRESERVE ROWS指定)
UPDATABLE	CHAR(1)	表の更新可能性 (ビュー表のみ)	Y:更新可能なビュー表 N:更新不可能なビュー表
CHECK_OPTION	CHAR(1)	表の更新時の検査オプション(ビュー表のみ)	Y:WITH CHECK OPTIONあり N:WITH CHECK OPTIONなし
DBS_ALLOCATION	CHAR(1)	格納領域指定に指定したデータベーススペースの種類	N:データベーススペース DSIが定義されていない場合は、空白が表示されます。
DBSPACE_NAME	CHAR(36) または CHAR(128)	格納領域指定に指定したデータベーススペース名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。 DSIが定義されていない場合は、空白が表示されます。
TABLE_LOCK	CHAR(1)	現バージョンでは意味がありません。	値は不定です。

注意

- 本表は、今後列が追加される場合があるため、参照する際は選択リストに列名を列挙するようにしてください。
- 本表を参照する問合せは、独立性水準にREAD UNCOMMITTEDが指定されている場合の動作になります。

A.3.5 列定義情報

RDBII_COLUMN

RDBII_COLUMNは、列の定義情報を出力する表です。RDBII_COLUMNの構成と出力情報を以下に示します。

表A.9 RDBII_COLUMNの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
DB_CODE	INT	データベースコード	1		
SCHEMA_CODE	INT	スキーマコード	2		
TABLE_CODE	INT	表コード	3		
COLUMN_NAME	CHAR(36) または CHAR(128)	列名			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
COLUMN_CODE	INT	列コード	4		
ORDINAL_POSITION	INT	列定義順序番号			
DATA_TYPE	CHAR(2)	列データの属性			CH:CHAR CV:VARCHAR BL:BLOB CN:NCHAR NV:NCHAR VARYING IN:INT SI:SMALLINT NU:NUMERIC DE:DECIMAL FL:FLOAT DP:DOUBLE PRECISION RE:REAL TM:TIMESTAMP DT:DATE TI:TIME IT:INTERVAL
NULLABLE	CHAR(3)	NULL設定可能性			YES:NULL設定可能 NO:NULL設定不可能
CHAR_MAX_LENGTH	INT	文字列系の最大文字数またはBLOB型の最大バイト数			

列名	データ型	情報の意味	P	SI	補足説明
CHAR_OCTET_LENGTH	INT	文字列系の最大バイト数またはBLOB型の最大バイト数			
NUMERIC_RADIX	INT	数値系の基数			2:INT、SMALLINT、FLOAT、DOUBLE PRECISION、REAL 10:NUMERIC、DECIMAL 0:TIMESTAMP、DATE、TIME DATE_TIME_INTERVAL_CODEの値:INTERVAL
NUMERIC_PRECISION	INT	数値系の精度(桁数)			TIMESTAMP、DATE、TIME:0 0以外:INTERVAL
NUMERIC_SCALE	INT	数値系の位取り			0:INT、SMALLINT、FLOAT、DOUBLE PRECISION、REAL、TIMESTAMP、DATE、TIME、INTERVAL n(指定値):NUMERIC、DECIMAL
DEFAULT_VALUE	VARCHAR(3000)	DEFAULT句に指定された文字列			例1:DEFAULT 'AAA'の場合、'AAA'が設定されます。 例2:DEFAULT +10.00の場合、'+10.00'が設定されます。

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。

COLUMNS

列の定義情報を出力するビュー表です。ビュー表の構成と出力情報を以下に示します。

表A.10 COLUMNSの構成と出力情報

列	データ型	情報の意味	補足説明
COLUMN_NAME	CHAR(36)またはCHAR(128)	列名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TABLE_CODE	INT	表コード	
DB_CODE	INT	データベースコード	
COLUMN_CODE	INT	列コード	

列	データ型	情報の意味	補足説明
ORDINAL_POSITION	INT	列定義順序番号	
COLUMN_DEFAULT	VARCHAR(3000)	デフォルト値	例1:DEFAULT 'AAA'の場合、'AAA'が設定されます。 例2:DEFAULT +10.00の場合、'+10.00'が設定されます。
NULLABLE	CHAR(3)	NULL設定可能性	YES:NULL設定可能 NO:NULL設定不可能
DATA_TYPE	CHAR(2)	列データの属性	CH:CHAR CV:VARCHAR BL:BLOB CN:NCHAR NV:NCHAR VARYING IN:INT SI:SMALLINT NU:NUMERIC DE:DECIMAL FL:FLOAT DP:DOUBLE PRECISION RE:REAL TM:TIMESTAMP DT:DATE TI:TIME IT:INTERVAL
CHAR_MAX_LENGTH	INT	文字列系の最大文字数またはBLOB型の最大バイト数	
CHAR_OCTET_LENGTH	INT	文字列系の最大バイト数またはBLOB型の最大バイト数	
NUMERIC_PRECISION	INT	数値系の精度(桁数)	0:TIMESTAMP、DATE、TIME 0以外:先行フィールドの精度
NUMERIC_PREC_RADIX	INT	数値系の基数	2:INT、SMALLINT、FLOAT、REAL、DOUBLE PRECISION 10:NUMERIC、DECIMAL 0:TIMESTAMP、DATE、TIME、INTERVAL DATE_TIME_INTERVAL_CODEの値:INTERVAL
NUMERIC_SCALE	INT	数値系の位取り	0:INT、SMALLINT、FLOAT、DOUBLE RECISION、REAL、TIMESTAMP、 DATE、TIME、INTERVAL n(指定値):NUMERIC、DECIMAL

A.3.6 制約定義情報

RDBII_CONSTRAINT

RDBII_CONSTRAINTは、制約の定義情報を出力する表です。RDBII_CONSTRAINTの表の構成と出力情報を以下に示します。

表A.11 RDBII_CONSTRAINTの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
DB_CODE	INT	データベースコード	1		
SCHEMA_CODE	INT	スキーマコード	2		
CONSTRAINT_NAME	CHAR(36) または CHAR(128)	現バージョンでは意味がありません。			常に空白が設定されます。
CONSTRAINT_CODE	INT	表制約コード	3		
TABLE_CODE	INT	表コード			
DEFERABLE	CHAR(1)	現バージョンでは意味がありません。			常に‘Y’が設定されます。
INITIALY_DEFER	CHAR(1)	現バージョンでは意味がありません。			常に‘I’が設定されます。
CONSTRAINT_TYPE	CHAR(2)	表制約属性			UK:UNIQUE(一意性) PK:PRIMARY(一意性)
MATCH_TYPE	CHAR(1)	現バージョンでは意味がありません。			値は不定です。
MATCH_UNIQUE	CHAR(1)	現バージョンでは意味がありません。			値は不定です。
UPDATE_TRIGER	CHAR(1)	現バージョンでは意味がありません。			値は不定です。
DELETE_TRIGER	CHAR(1)	現バージョンでは意味がありません。			値は不定です。

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。

TABLE_CONSTRAINTS

制約の定義情報を出力するビュー表です。このビュー表の構成と出力情報を以下に示します。

表A.12 ビュー表TABLE_CONSTRAINTSの構成と出力情報

列名	データ型	情報の意味	補足説明
SCHEMA_CODE	INT	スキーマコード	
DB_CODE	INT	データベースコード	

列名	データ型	情報の意味	補足説明
CONSTRAINT_CODE	INT	現バージョンでは意味がありません。	常に空白が設定されます。
CONSTRAINT_NAME	CHAR(36) または CHAR(128)	表制約名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
CONSTRAINT_TYPE	CHAR(2)	表制約属性	UK:UNIQUE(一意性) PK:PRIMARY(一意性)
TABLE_CODE	INT	表コード	

A.3.7 文字列を表現する表情報

RDBII_DESCRIPTION

RDBII_DESCRIPTIONは、定義されたビュー表の問合せ指定を格納する表です。RDBII_DESCRIPTIONの構成と出力情報を以下に示します。

表A.13 RDBII_DESCRIPTIONの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
DB_CODE	INT	ビュー表が属するデータベースのコード	1		
SCHEMA_CODE	INT	ビュー表が属するスキーマのコード	2		
OBJECT_CODE	INT	ビュー表の表コード	3		
DESC_TYPE	CHAR(1)	文字列の内容	4		常に'V'が設定されます。
DESC_VALUE	VARCHAR(30000)	文字列の値			先頭の30,000バイトの情報が格納されます。

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。

A.3.8 一意性制約の定義情報

RDBII_TABLE_USAGE

RDBII_TABLE_USAGEは、定義されたビュー表、ルーチンで指定された表、およびトリガで指定された表の一覧情報を格納する表です。RDBII_TABLE_USAGEの構成と出力情報を以下に示します。

表A.14 RDBII_TABLE_USAGEの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
DB_CODE	INT	ビュー表、ルーチン、またはトリガが属するデータベースのコード	1		
OBJ_SCHEMA_CODE	INT	ビュー表、ルーチン、またはトリガが属するスキーマのコード	2		

列名	データ型	情報の意味	P	SI	補足説明
OBJECT_CODE	INT	ビュー表、ルーチン、またはトリガのコード	3		
LIST_TYPE	CHAR(1)	表の一覧の目的	5		V:ビュー表の問合せ構成一覧 P:ルーチンから参照している表の一覧 T:被トリガ作用から参照している表の一覧
ORDINAL_POSITION	INT	ビュー表、ルーチン、またはトリガに指定された表の記述順序番号	6		
SCHEMA_CODE	INT	ビュー表、ルーチン、またはトリガに指定された表が属するスキーマのコード			
TABLE_CODE	INT	ビュー表、ルーチン、またはトリガに指定された表のコード	4		

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。

A.3.9 列一覧情報

RDBII_COLUMN_USAGE

RDBII_COLUMN_USAGEは、定義されたビュー表、一意性制約を構成する列、およびインデックスのDSO定義で指定された列の一覧の定義情報を出力する表です。このRDBII_COLUMN_USAGEの構成と出力情報を以下に示します。

表A.15 RDBII_COLUMN_USAGEの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
DB_CODE	INT	ビュー表、一意性制約、またはインデックスの属するデータベースコード	1		
OBJ_SCHEMA_CODE	INT	ビュー表、一意性制約、またはインデックスの属するスキーマのコード	2		
OBJECT_CODE	INT	ビュー表、一意性制約、またはインデックスのコード	3		
LIST_TYPE	CHAR(1)	列の一覧の目的	6		V:ビュー表の問合せ構成列一覧 U:一意性制約の列一覧 I:インデックスの構成列一覧

列名	データ型	情報の意味	P	SI	補足説明
ORDINAL_POSITION	INT	一意性制約またはインデックスで指定された列の記述順序番号(注)	7		
SCHEMA_CODE	INT	ビュー表、一意性制約、またはインデックスで指定された列が属するスキーマのコード			
TABLE_CODE	INT	ビュー表、一意性制約、またはインデックスで指定された列が属する表のコード	4		
COLUMN_CODE	INT	ビュー表、一意性制約、またはインデックスで指定された列が属する列のコード	5		

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。

注) LIST_TYPEの値が“V”の場合、ORDINAL_POSITIONの値は、ビュー表の問合せ構成列に割り振った通番です。この値は、ビュー表の問合せ構成列をそのビュー表内で一意に識別するための番号(1以上、ビュー表内で指定された列数以下)です。

KEY_COLUMN_USAGE

KEY_COLUMN_USAGEは、列一覧情報に格納されているレコードのうち、LIST_TYPEが“U”のものを取り出したビュー表です。このビュー表の構成と出力情報を以下に示します。

表A.16 KEY_COLUMN_USAGEの構成と出力情報

列名	データ型	情報の意味	補足説明
SCHEMA_CODE	INT	一意性制約の属するスキーマのコード	
DB_CODE	INT	一意性制約の属するデータベースのコード	
CONSTRAINT_CODE	INT	一意性制約の制約コード	
TABLE_CODE	INT	一意性制約の属する表のコード	
COLUMN_CODE	INT	一意性制約の属する列のコード	
ORDINAL_POSITION	INT	一意性制約で指定された列の記述順序番号	

A.3.10 注釈情報

RDBII_COMMENT

RDBII_COMMENTは、スキーマ、実表、ビュー表、一時表、ルーチン、列、インデックス、トリガに指定したコメントを格納する表です。RDBII_COMMENTの構成と出力情報を以下に示します。

表A.17 RDBII_COMMENTの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
COMMENT_TYPE	CHAR(2)	定義対象の種別	1		SC:スキーマ TV:実表、ビュー表および一時表 PR:ルーチン IX:インデックス CL:列 TR:トリガ
DB_CODE	INT	データベースコード	2		
SCHEMA_CODE	INT	スキーマコード	3		
TABLE_CODE	INT	表コード、ビュー表コード、一時表コード、ルーチンコード、トリガコード	4		
INDEX_DSO_CODE	INT	インデックスのDSOのコード	5		
COLUMN_CODE	INT	列コード	6		
COMMENT_VALUE	VARCHAR(256)	注釈			

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。

定義対象の種別によって、挿入されるデータが異なります。詳細を以下に示します。

オブジェクトの種別	COMMENT_TYPE	DB_CODE	SCHEMA_CODE	TABLE_CODE	INDEX_DSO_CODE	COLUMN_CODE
スキーマ	SC	データベースコード	スキーマコード	0	0	0
実表 ビュー表 一時表	TV	データベースコード	スキーマコード	表コード、 ビュー表コード、 一時表コード	0	0
インデックス	IX	データベースコード	スキーマコード	表コード、 一時表コード	インデックスDSOコード	0
ルーチン	PR	データベースコード	スキーマコード	ルーチンコード	0	0
列	CL	データベースコード	スキーマコード	表コード、 ビュー表コード、 一時表コード	0	列コード
トリガ	TR	データベースコード	スキーマコード	トリガコード	0	0

A.3.11 インデックス定義情報

RDBII_INDEX

RDBII_INDEXは、インデックスに関する情報を格納する表です。RDBII_INDEXの構成と出力情報を以下に示します。

表A.18 RDBII_INDEXの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
DB_CODE	INT	データベースコード	1		
SCHEMA_CODE	INT	スキーマコード	2		
TABLE_CODE	INT	テーブルコード	3		
INDEX_DSO_NAME	CHAR(36) または CHAR(128)	インデックスのDSO名	4		Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
INDEX_DSO_OWNER	CHAR(36) または CHAR(128)	インデックスのDSOの定義者名			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
CREATE_TIME	CHAR(8)	インデックスのDSOの定義日時			
INDEX_DSO_CODE	INT	インデックスのDSOのコード			
CONSTRAINT_CODE	INT	一意性の制約コード			

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。

A.3.12 インデックス定義拡張情報

RDBII_INDEX_EX

RDBII_INDEX_EXは、インデックスに関する拡張情報を格納する表です。利用者が定義した定義情報を出力対象とします。システム表および監査ログデータベースに関する定義情報は出力しません。

RDBII_INDEX_EXの構成と出力情報を以下に示します。

表A.19 RDBII_INDEX_EXの構成と出力情報

列名	データ型	情報の意味	補足説明
DB_CODE	INT	データベースコード	
SCHEMA_CODE	INT	スキーマコード	
TABLE_CODE	INT	テーブルコード	
INDEX_DSO_NAME	CHAR(36) または CHAR(128)	インデックスのDSO名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
INDEX_DSO_OWNER	CHAR(36) または CHAR(128)	インデックスのDSOの定義者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。

列名	データ型	情報の意味	補足説明
CREATE_TIME	CHAR(8)	インデックスのDSOの定義日時	
INDEX_DSO_CODE	INT	インデックスのDSOのコード	
CONSTRAINT_CODE	INT	一意性の制約コード	
DBS_ALLOCATION	CHAR(1)	格納領域指定に指定したデータベーススペースの種類	N:データベーススペース DSIが定義されていない場合は、空白が表示されます。
DBSPACE_NAME	CHAR(36) または CHAR(128)	格納領域指定に指定したデータベーススペース名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。 DSIが定義されていない場合は、空白が表示されます。
INDEX_NAME	CHAR(36) または CHAR(128)	インデックス名 (格納構造定義を行わない場合)	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。 DSIが定義されていない場合は、空白が表示されます。

注意

- 本表は、今後列が追加される場合があるため、参照する際は選択リストに列名を列挙するようにしてください。
- 本表を参照する問合せは、独立性水準にREAD UNCOMMITTEDが指定されている場合の動作になります。

A.3.13 ルーチン定義情報

RDBII_PROC

RDBII_PROCは、定義されたルーチンの情報を格納する表です。RDBII_PROCの構成と出力情報を以下に示します。

表A.20 RDBII_PROCの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
DB_NAME	CHAR(36) または CHAR(128)	データベース名		1	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
SCHEMA_NAME	CHAR(36) または CHAR(128)	スキーマ名		2	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
PROCEDURE_NAME	CHAR(36) または CHAR(128)	ルーチン名	3	3	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場

列名	データ型	情報の意味	P	SI	補足説明
					合はCHAR(128)になります。
PROCEDURE_OWNER	CHAR(36) または CHAR(128)	ルーチン定義者名			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
CREATE_TIME	CHAR(8)	ルーチン定義日時			
DB_CODE	INT	データベースコード	1		
SCHEMA_CODE	INT	スキーマコード	2		
PROCEDURE_CODE	INT	ルーチンコード			
PROCEDURE_TYPE	INT	ルーチン種別情報			1:PROCEDURE 2:FUNCTION
ROUTINE_TYPE	INT	ルーチン本体種別情報			0:SQL手続き文 1:C言語
ROUTINE_NAME	VARCHAR(256)	外部シンボル名			ROUTINE_TYPEが0以外の場合のみ、関数のシンボル名を示します。

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。数字は、インデックスキー内の構成順序を示します。

PROCS

PROCSは、ルーチンの定義情報を出力するビュー表です。このビュー表の構成と出力情報を以下に示します。

表A.21 PROCSの構成と出力情報

列名	データ型	情報の意味	補足説明
DB_NAME	CHAR(36) または CHAR(128)	データベース名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
SCHEMA_NAME	CHAR(36) または CHAR(128)	スキーマ名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
PROCEDURE_NAME	CHAR(36) または CHAR(128)	ルーチン名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
PROCEDURE_OWNER	CHAR(36) または CHAR(128)	ルーチン定義者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
PROCEDURE_TYPE	INT	ルーチン種別情報	1:PROCEDURE 2:FUNCTION
ROUTINE_TYPE	INT	ルーチン本体種別情報	0:SQL手続き文 1:C言語

列名	データ型	情報の意味	補足説明
ROUTINE_NAME	VARCHAR(256)	外部シンボル名	ROUTINE_TYPEが0以外の場合のみ、関数のシンボル名を示します。

A.3.14 ルーチン定義拡張情報

RDBII_PROC_EX

RDBII_PROC_EXは、定義されたルーチンの拡張情報を格納する表です。利用者が定義した定義情報を出力対象とします。システム表および監査ログデータベースに関する定義情報は出力しません。

RDBII_PROC_EXの構成と出力情報を以下に示します。

表A.22 RDBII_PROC_EXの構成と出力情報

列名	データ型	情報の意味	補足説明
DB_NAME	CHAR(36) または CHAR(128)	データベース名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
SCHEMA_NAME	CHAR(36) または CHAR(128)	スキーマ名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
PROCEDURE_NAME	CHAR(36) または CHAR(128)	ルーチン名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
PROCEDURE_OWNER	CHAR(36) または CHAR(128)	ルーチン定義者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
CREATE_TIME	CHAR(8)	ルーチン定義日時	
DB_CODE	INT	データベースコード	
SCHEMA_CODE	INT	スキーマコード	
PROCEDURE_CODE	INT	ルーチンコード	
PROCEDURE_TYPE	INT	ルーチン種別情報	1:PROCEDURE 2:FUNCTION
ROUTINE_TYPE	INT	ルーチン本体種別情報	0:SQL手続き文 1:C言語
ROUTINE_NAME	VARCHAR(256)	外部シンボル名	ROUTINE_TYPEが0以外の場合のみ、関数のシンボル名を示します。
LIBRARY_PATH	VARCHAR(1024)	ライブラリのパス	ROUTINE_TYPEが0以外の場合のみ、ルーチンとして実行される関数が格納されているライブラリのパスを示します。

注意

- 本表は、今後列が追加される場合があるため、参照する際は選択リストに列名を列挙するようにしてください。
- 本表を参照する問合せは、独立性水準にREAD UNCOMMITTEDが指定されている場合の動作になります。

A.3.15 ルーチンパラメタ情報

RDBII_PROC_COL

RDBII_PROC_COLは、定義された資源に割り振られたコードの情報を格納する表です。RDBII_PROC_COLの構成と出力情報を以下に示します。

表A.23 RDBII_PROC_COLの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
DB_CODE	INT	データベースコード	1		
SCHEMA_CODE	INT	スキーマコード	2		
PROCEDURE_CODE	INT	ルーチンコード	3		
COLUMN_NAME	CHAR(36) または CHAR(128)	パラメタ名			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります
COLUMN_CODE	INT	列コード	4		
ORDINAL_POSITION	INT	一意性制約またはインデックスで指定された列の記述順序番号			
COLUMN_TYPE	INT	パラメタタイプ			0:不定 1:入力 2:入出力 3:リザーブ 4:出力 5:復帰値
DATA_TYPE	INT	データタイプ			1:CHAR 2:CHAR VARYING 3:BLOB 4:NCHAR 5:NCHAR VARYING 6:INT 7:SMALLINT 8:NUMERIC 9:DECIMAL 10:FLOAT 11:DOUBLE PRECISION 12:REAL 13:TIMESTAMP 14:DATE

列名	データ型	情報の意味	P	SI	補足説明
					15:TIME 16:INTERVAL
NULLABLE	CHAR(3)	NULL設定可能性			YES:NULL設定可能 NO:NULL設定不可能
NUMERIC_PRECISION	INT	数値系の精度(桁数)			0:TIMESTAMP、DATE、TIME 先行フィールドの精度:INTERVAL
CHAR_MAX_LENGTH	INT	文字列系の最大文字数またはBLOB型の最大キロバイト数			
CHAR_OCTET_LENGTH	INT	文字列系の最大バイト数またはBLOB型の最大キロバイト数			
NUMERIC_SCALE	INT	数値系の位取り			0:INT、SMALLINT、FLOAT、 DOUBLE PRECISION、REAL、TIMESTAMP、DATE n(指定値):NUMERIC、DECIMAL
NUMERIC_RADIX	INT	数値系の基数			2:INT、SMALLINT、FLOAT、 DOUBLE PRECISION、REAL 10:NUMERIC、DECIMAL 0:TIMESTAMP、DATE、TIME DATE_TIME_INTERVAL_CODEの値:INTERVAL
COLUMN_STYLE	INT	現バージョンでは意味がありません。			値は不定です。
VARIANT_TYPE	INT	現バージョンでは意味がありません。			値は不定です。

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。

PROC_COLS

PROC_COLSは、定義された資源に割り振られたコードの情報を出力するビュー表です。このビュー表の構成と出力情報を以下に示します。

表A.24 PROC_COLSの構成と出力情報

列名	データ型	情報の意味	補足説明
DB_NAME	CHAR(36) または CHAR(128)	データベース名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。

列名	データ型	情報の意味	補足説明
SCHEMA_NAME	CHAR(36) または CHAR(128)	スキーマ名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
PROCEDURE_NAME	CHAR(36) または CHAR(128)	ルーチン名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
COLUMN_NAME	CHAR(36) または CHAR(128)	パラメタ名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
COLUMN_CODE	INT	列コード	
ORDINAL_POSITION	INT	一意性制約またはインデックスで指定された列の記述順序番号	
COLUMN_TYPE	INT	パラメタタイプ	0:不定 1:入力 2:入出力 3:リザーブ 4:出力 5:復帰値
DATA_TYPE	INT	データタイプ	1:CHAR 2:CHAR VARYING 3:BLOB 4:NCHAR 5:NCHAR VARYING 6:INT 7:SMALLINT 8:NUMERIC 9:DECIMAL 10:FLOAT 11:DOUBLE PRECISION 12:REAL 13:TIMESTAMP 14:DATE 15:TIME 16:INTERVAL
NULLABLE	CHAR(3)	NULL設定可能性	YES:NULL設定可能 NO:NULL設定不可能
NUMERIC_PRECISION	INT	数値系の精度（桁数）	0:TIMESTAMP、DATE、TIME 先行フィールドの精度: INTERVAL
CHAR_MAX_LENGTH	INT	文字列系の最大文字数またはBLOB型の最大キロバイト数	
CHAR_OCTET_LENGTH	INT	文字列系の最大キロバイト数またはBLOB型の最大キロバイト数	

列名	データ型	情報の意味	補足説明
NUMERIC_SCALE	INT	数値系の位取り	0:INT、SMALLINT、FLOAT、DOUBLE PRECISION、REAL、TIMESTAMP、DATE n(指定値):NUMERIC、DECIMAL
NUMERIC_RADIX	INT	数値系の基数	2:INT、SMALLINT、FLOAT、DOUBLE PRECISION、REAL 10:NUMERIC、DECIMAL 0:TIMESTAMP、DATE、TIME DATE_TIME_INTERVAL_CODEの値 :INTERVAL
COLUMN_STYLE	INT	現バージョンでは意味がありません。	値は不定です。
VARIANT_TYPE	INT	現バージョンでは意味がありません。	値は不定です。

A.3.16 権限情報

RDBII_PRIVILEGES

RDBII_PRIVILEGESは、定義された資源に割り振られたコード情報を格納する表です。

RDBII_PRIVILEGESの構成と出力情報を以下に示します。

表A.25 RDBII_PRIVILEGESの構成と出力情報

列名	データ型	情報の意味	P	SI	SI	補足説明
GRANTOR	CHAR(36) または CHAR(128)	権限付与者名	1			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
GRANTEE	CHAR(36) または CHAR(128)	権限受領者名	2	5	1	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
OBJECT_TYPE	CHAR(2)	権限対象の種別	3	1		DB:データベース SC:スキーマ TV:実表、ビュー表または一時表 PR:ルーチン TR:トリガ SQ:順序
DB_CODE	INT	データベースコード	4	2		
SCHEMA_CODE	INT	スキーマコード	5	3		
TABLE_CODE	INT	表、ビュー表、一時表、ルー	6	4		

列名	データ型	情報の意味	P	SI	SI	補足説明
		チン、トリガまたは順序のコード				
PRIVILEGE_TYPE	CHAR(10)	権限の種別	7			SELECT:実表、ビュー表、一時表および、順序の参照権 INSERT:格納権 UPDATE:更新権 DELETE:削除権 EXECUTE:ルーチンの実行権 CREATE:実表、ビュー表、一時表、ルーチンおよび順序の定義権 TRIGGER:トリガの定義権 DROP :削除権 ALTER:表定義の更新権 INDEX:インデックスの定義権
IS_GRANTABLE	CHAR(3)	付与権				YES:付与可能 NO :付与不可能

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。数字は、インデックスキー内の構成順序を示します。

権限情報テーブルの列の値の組合せを以下に示します。

OBJECT_TYPE	OBJECT_TYPE	DB_CODE	SCHEMA_CODE	TABLE_CODE
データベース	DB	データベースコード	0	0
スキーマ	SC	データベースコード	スキーマコード	0
実表、ビュー表および一時表	TV	データベースコード	スキーマコード	表コード、ビュー表コード、一時表コード
ルーチン	PR	データベースコード	スキーマコード	ルーチンコード
トリガ	TR	データベースコード	スキーマコード	トリガコード
順序	SQ	データベースコード	スキーマコード	順序コード

RSC_PRIVILEGES

RSC_PRIVILEGESは、権限定義情報を出力するビュー表です。

列名	データ型	情報の意味	補足説明
GRANTOR	CHAR(36) または CHAR(128)	権限付与者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
GRANTEE	CHAR(36) または CHAR(128)	権限受領者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。

列名	データ型	情報の意味	補足説明
OBJECT_TYPE	CHAR(2)	権限対象の種別	DB:データベース SC:スキーマ TV:実表、ビュー表または、一時表 PR:ルーチン TR:トリガ SQ:順序
DB_CODE	INT	データベースコード	
SCHEMA_CODE	INT	スキーマコード	
TABLE_CODE	INT	表、ビュー表、一時表、ルーチン、トリガまたは順序のコード	
PRIVILEGE_TYPE	CHAR(10)	権限の種別	SELECT:実表、ビュー表、一時表および、順序の参照権 INSERT:格納権 UPDATE:更新権 DELETE:削除権 EXECUTE:ルーチンの実行権 CREATE:実表、ビュー表、一時表、ルーチンおよび順序の定義権 TRIGGER:トリガの定義権 DROP :削除権 ALTER:表定義の更新権 INDEX:インデックスの定義権
IS_GRANTABLE	CHAR(3)	付与権	YES :付与可能 NO :付与不可能

A.3.17 トリガ定義情報

RDBII_TRIGGER

RDBII_TRIGGERは、定義されたトリガの情報を格納する表です。

RDBII_TRIGGERの構成と出力情報を以下に示します。

表A.26 RDBII_TRIGGERの構成と出力情報

列名	データ型	情報の意味	P	SI	SI	SI	補足説明
TRIG_DB_NAME	CHAR(36) または CHAR(128)	データベース名		1			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TRIG_SCHEMA_NAME	CHAR(36) または CHAR(128)	スキーマ名		2			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TRIG_NAME	CHAR(36) または CHAR(128)	トリガ名	3	3			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。

列名	データ型	情報の意味	P	SI	SI	SI	補足説明
TRIG_OWNER	CHAR(36) または CHAR(128)	トリガ定義者名					Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
CREATE_TIME	CHAR(8)	トリガ定義日時					
TRIG_DB_CODE	INT	データベースコード	1		1	1	
TRIG_SCHEMA_CODE	INT	スキーマコード	2		2		
TRIG_CODE	INT	トリガコード			3		
EVENT_MANIPURATION	CHAR(8)	トリガ契機SQL操作文					'INSERT ' 'UPDATE ' 'DELETE '
EVENT_OBJ_SCH_NAME	CHAR(36) または CHAR(128)	トリガ対象表のスキーマ名					Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
EVENT_OBJ_TBL_NAME	CHAR(36) または CHAR(128)	トリガ対象表の表名					Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
EVENT_OBJ_SCH_CODE	INT	トリガ対象表のスキーマコード				2	
EVENT_OBJ_TBL_CODE	INT	トリガ対象表の表コード				3	
ACTION_ORDER	INT	同一のトリガ契機をもつトリガの実行順番					
ACTION_CONDITION	VARCHAR(15000)	トリガ実行条件					
ACTION_STMT_LIST	VARCHAR(15000)	トリガ契機実行SQL文					
ACTION_ORIENTATION	CHAR(12)	トリガ契機の単位					'ROW ' 'STATEMENT '
COND_TIMING	CHAR(12)	トリガ契機の属性					'BEFORE ' 'AFTER '
COND_REF_OLD_TABLE	CHAR(36) または CHAR(128)	トリガ対象表の更新前相関名／アリアス					Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
COND_REF_NEW_TABLE	CHAR(36) または CHAR(128)	トリガ対象表の更新後相関名／アリアス					Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
COL_LIST_IS_IMPLCT	CHAR(4)	列名リスト指定あり／なし					YES :トリガ契機=UPDATEかつ列名指定リストなし NO :トリガ契機=UPDATEかつ列名指定リストあり

列名	データ型	情報の意味	P	SI	SI	SI	補足説明
							NULL:トリガ契機=INSERTまたはDELETE

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。数字は、インデックスキー内の構成順序を示します。

TRIGGERS

TRIGGERSは、トリガの定義情報を出力するビュー表です。ビュー表の構成と出力情報を以下に示します。

表A.27 ビュー表TRIGGERSの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
TRIG_DB_NAME	CHAR(36) または CHAR(128)	データベース名			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TRIG_SCHEMA_NAME	CHAR(36) または CHAR(128)	スキーマ名			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TRIG_NAME	CHAR(36) または CHAR(128)	トリガ名			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TRIG_OWNER	CHAR(36) または CHAR(128)	トリガ定義者名			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
CREATE_TIME	CHAR(8)	トリガ定義日時			
EVENT_MANIPURATION	CHAR(8)	トリガ契機SQL操作文			'INSERT ' 'UPDATE ' 'DELETE '
EVENT_OBJ_DB_NAME	CHAR(36) または CHAR(128)	トリガ対象表のデータベース名			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
EVENT_OBJ_SCH_NAME	CHAR(36) または CHAR(128)	トリガ対象表のスキーマ名			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
EVENT_OBJ_TBL_NAME	CHAR(36) または CHAR(128)	トリガ対象表の表名			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
ACTION_ORDER	INT	同一のトリガ契機をもつトリガの実行順番			
ACTION_CONDITION	VARCHAR(15000)	トリガ実行条件			

列名	データ型	情報の意味	P	SI	補足説明
ACTION_STMT_LIST	VARCHAR(15000)	トリガ契機実行SQL文			
ACTION_ORIENTATION	CHAR(12)	トリガ契機の単位			'ROW ' 'STATEMENT '
COND_TIMING	CHAR(12)	トリガ契機の属性			'BEFORE ' 'AFTER '
COND_REF_OLD_TABLE	CHAR(36) または CHAR(128)	トリガ対象表の更新前相関名／アリアス			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
COND_REF_NEW_TABLE	CHAR(36) または CHAR(128)	トリガ対象表の更新後相関名／アリアス			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
COL_LIST_IS_IMPLICIT	CHAR(4)	列名リスト指定あり／なし			YES :トリガ契機=UPDATEかつ列名指定リストなし NO :トリガ契機=UPDATEかつ列名指定リストあり NULL:トリガ契機=INSERTまたはDELETE

P : プライマリキーを意味します。

SI : インデックスのキーを意味します。

A.3.18 トリガ列一覧情報

RDBII_TRIG_COLUMNS

RDBII_TRIG_COLUMNSは、トリガ作用で指定されたSQL文でのアクセス対象表についての情報を格納する表です。RDBII_TRIG_COLUMNSの構成と出力情報を以下に示します。

表A.28 RDBII_TRIG_COLUMNSの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
TRIG_DB_CODE	INT	データベースコード	1	1	
TRIG_SCHEMA_CODE	INT	スキーマコード	2		
TRIG_CODE	INT	トリガコード	3		
EVENT_OBJ_SCH_CODE	INT	トリガ対象表のスキーマコード	4	2	
EVENT_OBJ_TBL_CODE	INT	トリガ対象表の表コード	5	3	
EVENT_OBJ_COL_CODE	INT	トリガ対象表の更新対象列コード	6	4	

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。数字は、インデックスキー内の構成順序を示します。

TRIGGERED_COLUMNS

TRIGGERED_COLUMNSは、トリガ列一覧情報の定義情報を出力するビュー表です。ビュー表の構成と出力情報を以下に示します。

表A.29 TRIGGERED_COLUMNSの構成と出力情報

列名	データ型	情報の意味	補足説明
TRIG_DB_NAME	CHAR(36) または CHAR(128)	データベース名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TRIG_SCHEMA_NAME	CHAR(36) または CHAR(128)	スキーマ名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TRIG_NAME	CHAR(36) または CHAR(128)	トリガ名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
EVENT_OBJ_DB_NAME	CHAR(36) または CHAR(128)	トリガ対象表の データベース名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
EVENT_OBJ_SCH_NAME	CHAR(36) または CHAR(128)	トリガ対象表の スキーマ名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
EVENT_OBJ_TBL_NAME	CHAR(36) または CHAR(128)	トリガ対象表の表名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
EVENT_OBJ_COL_NAME	CHAR(36) または CHAR(128)	トリガ対象表の列名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。

A.3.19 順序情報

RDBII_SEQUENCE

RDBII_SEQUENCEは、順序についての情報を格納する表です。

列名	データ型	情報の意味	P	SI	SI	補足説明
DB_NAME	CHAR(36) または CHAR(128)	データベース名		1		Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
SCHEMA_NAME	CHAR(36) または CHAR(128)	スキーマ名		2		Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
SEQUENCE_NAME	CHAR(36) または CHAR(128)	順序名	3	3		Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
SEQUENCE_OWNER	CHAR(36) または CHAR(128)	順序定義者名				Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。

列名	データ型	情報の意味	P	SI	SI	補足説明
CREATE_TIME	CHAR(8)	順序定義日時				
DB_CODE	INT	データベースコード	1		1	
SCHEMA_CODE	INT	スキーマコード	2		2	
SEQUENCE_CODE	INT	順序コード			3	
INCREMENT_VAL	DECIMAL(18,0)	INCREMENT BY句に指定した、順序番号の増分間隔値				省略値は、1になります。
START_WITH_VAL	DECIMAL(18,0)	START WITH句に指定した、順序番号の初期値				省略値は、増分間隔値の値によって以下のようになります。 正の値の場合:1 負の値の場合:－1
MAX_VAL	DECIMAL(18,0)	MAXVALUE句に指定した、順序番号の最大値				省略値は、増分間隔値の値によって以下のようになります。 正の値の場合:(10^{18})-1 負の値の場合:－1
MIN_VAL	DECIMAL(18,0)	MINVALUE句に指定した、順序番号の最小値				省略値は、増分間隔値の値によって以下のようになります。 正の値の場合:1 負の値の場合:-(10^{18})+1
CACHE_VAL	DECIMAL(18,0)	CACHE句に指定した、割当て順序数				省略値は、0になります。
CACHE_MIN_VAL	DECIMAL(18,0)	CACHE句に指定した、保持順序数				省略時は、0になります。
CYCLE_SPEC	CHAR(1)	CYCLE情報				N:CYCLE指定なし Y:CYCLE指定あり
ORDER_SPEC	CHAR(1)	ORDER情報				N:ORDER指定なし Y:ORDER指定あり

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。数字は、インデックスキー内の構成順序を示します。

SEQUENCES

順序定義情報を出力するビュー表です。このビュー表を検索する場合、利用者に権限のある順序の情報のみを出力します。

列名	データ型	情報の意味	補足説明
DB_NAME	CHAR(36) または CHAR(128)	データベース名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異

列名	データ型	情報の意味	補足説明
			なる場合はCHAR(128)になります。
SCHEMA_NAME	CHAR(36) または CHAR(128)	スキーマ名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
SEQUENCE_NAME	CHAR(36) または CHAR(128)	順序名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
SEQUENCE_OWNER	CHAR(36) または CHAR(128)	順序定義者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
DB_CODE	INT	データベースコード	
SCHEMA_CODE	INT	スキーマコード	
SEQUENCE_CODE	INT	順序コード	
INCREMENT_VAL	DECIMAL (18,0)	INCREMENT BY句に指定した、順序番号の増分間隔値	省略値は、1になります。
START_WITH_VAL	DECIMAL (18,0)	START WITH句に指定した、順序番号の初期値	省略値は、増分間隔値の値によって以下のようになります。 正の値の場合:1 負の値の場合:－1
MAX_VAL	DECIMAL (18,0)	MAXVALUE句に指定した、順序番号の最大値	省略値は、増分間隔値の値によって以下のようになります。 正の値の場合:(10^{18})-1 負の値の場合:－1
MIN_VAL	DECIMAL (18,0)	MINVALUE句に指定した、順序番号の最小値	省略値は、増分間隔値の値によって以下のようになります。 正の値の場合:1 負の値の場合:$-(10^{18})+1$
CACHE_VAL	DECIMAL (18,0)	CACHE句に指定した、割当て順序数	省略値は、0になります。
CACHE_MIN_VAL	DECIMAL (18,0)	CACHE句に指定した、保持順序数	省略時は、0になります。
CYCLE_SPEC	CHAR(1)	CYCLE情報	N:指定なし Y:指定あり
ORDER_SPEC	CHAR(1)	ORDER情報	N:指定なし Y:指定あり

A.3.20 利用者情報

RDBII_USER

RDBII_USERは、利用者定義についての情報を格納する実表です。RDBII_USERの構成と出力情報を以下に示します。

表A.30 RDBII_USERの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
USER_NAME	CHAR(36) または CHAR(128)	利用者名	1		Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
USER_CODE	INT	利用者コード		1	
IDENT	CHAR(2)	管理方式			DB:Symfoware/RDBで管理 OS:OSの機能で管理
USER_KIND	CHAR(1)	種別			S:管理者 A:管理者 U:利用者
PASSWORD_STATUS	CHAR(1)	パスワード状態			N:正常状態 L:ロック状態 管理方式が'OS'の場合はNULL値が設定されます。
LAST_LOGIN_DATE	TIMESTAMP	最終ログイン日時			
LOGIN_FAILED_TIME	INT	ログイン失敗回数			

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。数字は、インデックスキー内の構成順序を示します。

USERS

USERSは、利用者情報を出力するビュー表です。ビュー表の構成と出力情報を以下に示します。

表A.31 ビュー表USERSの構成と出力情報

列名	データ型	情報の意味	補足説明
USER_NAME	CHAR(36) または CHAR(128)	利用者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
USER_CODE	INT	利用者コード	
IDENT	CHAR(2)	管理方式	DB:Symfoware/RDBで管理 OS:OSの機能で管理
USER_KIND	CHAR(1)	種別	S:管理者 A:管理者 U:利用者
PASSWORD_STATUS	CHAR(1)	パスワード状態	N:正常状態 L:ロック状態 管理方式が'OS'の場合はNULL値が設定されます。

列名	データ型	情報の意味	補足説明
LAST_LOGIN_DATE	TIMESTAMP	最終ログイン日時	
LOGIN_FAILED_TIME	INT	ログイン失敗回数	

A.3.21 ユーザパラメタ設定情報

RDBII_PARAMETER

RDBII_PARAMETERは、ユーザパラメタ設定情報についての情報を格納する実表です。RDBII_PARAMETERの構成と出力情報を以下に示します。

表A.32 RDBII_PARAMETERの構成と出力情報

列名	データ型	情報の意味	P	SI	SI	補足説明
PARAMETER_NAME	CHAR(36) または CHAR(128)	パラメタ名	1			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
PARAMETER_VALUE	CHAR(256)	パラメタ設定値				
SET_USER_CODE	INT	設定者の利用者コード	2	1		
USER_CODE	INT	利用者コード	3	2	1	システムに対する設定時は-1です。
KIND	CHAR(1)	設定先種別				S:システム U:利用者

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。数字は、インデックスキー内の構成順序を示します。

PARAMETERS

PARAMETERSは、ユーザパラメタ設定情報についての情報を出力するビュー表です。ビュー表の構成と出力情報を以下に示します。

表A.33 ビュー表PARAMETERSの構成と出力情報

列名	データ型	情報の意味	補足説明
PARAMETER_NAME	CHAR(36) または CHAR(128)	パラメタ名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
PARAMETER_VALUE	CHAR(256)	パラメタ設定値	
SET_USER_CODE	INT	設定者の利用者コード	

列名	データ型	情報の意味	補足説明
USER_CODE	INT	利用者コード	システムに対する設定時は-1です。
KIND	CHAR(1)	設定先種別	S:システム U:利用者

A.3.22 ロール情報

RDBII_ROLE

RDBII_ROLEは、ロール定義文で定義されたロールの情報を格納する実表です。RDBII_ROLEの構成と出力情報を以下に示します。

表A.34 RDBII_ROLEの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
ROLE_NAME	CHAR(36) または CHAR(128)	ロール名	1		Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
ROLE_CODE	INT	ロールコード		1	
ROLE_OWNER	CHAR(36) または CHAR(128)	ロール定義者名			Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
CREATE_TIME	CHAR(8)	ロール定義日時			

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。数字は、インデックスキー内の構成順序を示します。

ROLES

ROLESは、ロール情報を出力するビュー表です。ビュー表の構成と出力情報を以下に示します。

表A.35 ビュー表ROLESの構成と出力情報

列名	データ型	情報の意味	補足説明
ROLE_NAME	CHAR(36) または CHAR(128)	ロール名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
ROLE_CODE	INT	ロールコード	
ROLE_OWNER	CHAR(36) または CHAR(128)	ロール定義者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。

A.3.23 ロールに付与された権限の情報

RDBII_ROLE_PRIV

RDBII_ROLE_PRIVは、ロールに付与された権限の情報を格納する表です。RDBII_ROLE_PRIVの構成と出力情報を以下に示します。

表A.36 RDBII_ROLE_PRIVの構成と出力情報

列名	データ型	情報の意味	P	SI	SI	SI	補足説明
ROLE_CODE	INT	ロールコード	1	5	4	3	
GRANTOR	CHAR(36) または CHAR(128)	付与者名	2				Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
OBJECT_TYPE	CHAR(2)	対象資源種別	3	4	3	2	SP:データベーススペース SC:スキーマ TV:実表、ビュー表または、一時表 PR:ルーチン TR:トリガ SQ:順序
DB_CODE	INT	データベースコード	4	1			
SCHEMA_CODE	INT	スキーマコード	5	2	1		
OBJECT_CODE	INT	対象資源コード	6	3	2	1	
PRIVILEGE_TYPE	CHAR(10)	権限の種別	7				SELECT:実表、ビュー表、一時表および順序の参照件 INSERT :格納権 UPDATE :更新権 DELETE :削除権 EXECUTE :ルーチンの実行権 CREATE:実表、ビュー表、一時表、ルーチン、トリガおよび順序の定義権 TRIGGER:トリガの定義権 DROP:削除権 ALTER:表定義の更新権 INDEX:インデックスの定義権 ALLOCATE:データベーススペースの割付け権

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。数字は、インデックスキー内の構成順序を示します。

ロールに付与された権限の情報テーブルの列の値の組合せを以下に示します。

OBJECT_TYPE	OBJECT_TYPE	DB_CODE	SCHEMA_CODE	OBJECT_CODE
データベーススペース	SP	データベースコード	0	データベーススペースコード
スキーマ	SC	データベースコード	スキーマコード	0
実表、ビュー表または、一時表	TV	データベースコード	スキーマコード	表コード、ビュー表コード、または一時表コード
ルーチン	PR	データベースコード	スキーマコード	ルーチンコード
トリガ	TR	データベースコード	スキーマコード	トリガコード
順序	SQ	データベースコード	スキーマコード	順序コード

ROLE_PRIVILEGES

ROLE_PRIVILEGESは、ロール構成権限情報を出力するビュー表です。ビュー表の構成と出力情報を以下に示します。

表A.37 ビュー表ROLE_PRIVILEGESの構成と出力情報

列名	データ型	情報の意味	補足説明
ROLE_CODE	INT	ロールコード	
GRANTOR	CHAR(36) または CHAR(128)	付与者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
OBJECT_TYPE	CHAR(2)	対象資源種別	SP:データベーススペース SC:スキーマ TV:実表、ビュー表または、一時表 PR:ルーチン TR:トリガ SQ:順序
DB_CODE	INT	データベースコード	
SCHEMA_CODE	INT	スキーマコード	
OBJECT_CODE	INT	対象資源コード	
PRIVILEGE_TYPE	CHAR(10)	権限の種別	SELECT :実表、ビュー表、一時表および順序の参照権 INSERT :格納権 UPDATE :更新権 DELETE :削除権 EXECUTE:ルーチンの実行権 CREATE:実表、ビュー表、一時表、ルーチン、トリガおよび順序の定義権 TRIGGER:トリガの定義権 DROP:削除権 ALTER:表定義の更新権 INDEX:インデックスの定義権 ALLOCATE:データベーススペースの割付け権

A.3.24 利用者に対するロールの付与情報

RDBII_ROLE_GRANT

RDBII_ROLE_GRANTは、利用者に対するロールの付与情報を格納する表です。RDBII_ROLE_GRANTの構成と出力情報を以下に示します。

表A.38 RDBII_ROLE_GRANTの構成と出力情報

列名	データ型	情報の意味	P	SI	補足説明
ROLE_CODE	INT	ロールコード	1	2	
GRANTOR	CHAR(36) または CHAR(128)	付与者名	2		Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
GRANTEE	CHAR(36) または CHAR(128)	受領者名	3	1	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
ADMIN_OPTION	CHAR(3)	付与権の保持			YES :付与可能 NO :付与不可能

P : プライマリキーを意味します。数字は、プライマリキー内の構成順序を示します。

SI : インデックスのキーを意味します。数字は、インデックスキー内の構成順序を示します。

ROLE_GRANTS

ROLE_GRANTSは、ロール付与情報を出力するビュー表です。ビュー表の構成と出力情報を以下に示します。

表A.39 ビュー表ROLE_GRANTSの構成と出力情報

列名	データ型	情報の意味	補足説明
ROLE_CODE	INT	ロールコード	
GRANTOR	CHAR(36) または CHAR(128)	付与者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
GRANTEE	CHAR(36) または CHAR(128)	受領者名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
ADMIN_OPTION	CHAR(3)	付与権の保持	YES :付与可能 NO :付与不可能

A.3.25 SQL文の簡易実行情報

RDBII_ASSISTTABLE

RDBII_ASSISTTABLEは、SQL文を簡易的に実行するための表です。この表には1の値を持つデータが1件格納されています。RDBII_ASSISTTABLEに対して、日時値関数、順序などを記述した単一行SELECT文を実行することで、現在の時刻、順序番号などを取得することができます。

RDBII_ASSISTTABLEの構成と出力情報を以下に示します。

表A.40 RDBII_ASSISTTABLEの構成と出力情報

列名	データ型	情報の意味	補足説明
ASSISTCOLUMN	INT		特に意味はありません。

使用例

RDBII_ASSISTTABLEの使用例を説明します。

例1

CURRENT_TIMESTAMPを指定して、現在の時刻を参照します。

```
SELECT CURRENT_TIMESTAMP FROM RDBII_SYSTEM.RDBII_ASSISTTABLE
```

例2

CURRVALを指定して、在庫管理.順序1の現在の順序番号を参照します。

```
SELECT 在庫管理.順序１.CURRVAL FROM RDBII_SYSTEM.RDBII_ASSISTTABLE
```

A.3.26 DSI情報

RDBII_DSI_STATUS

RDBII_DSI_STATUSは、DSIについての情報を出力する表です。RDBII_DSI_STATUSの構成と出力情報を以下に示します。

検索結果をCSV形式で出力する場合には、rdbunlsqlコマンドを使用してください。

S L

ロードシェア運用の場合は、本システム表を検索したノードに定義されているDSIの情報が表示されます。

表A.41 RDBII_DSI_STATUSの構成と出力情報

列名	データ型	情報の意味	補足説明
TYPE	CHAR(1)	対象のDSIの種別	T :表 I :インデックス
CODE	INT	DSIコード	TYPEが‘T’の場合は表のDSIコード TYPEが‘I’の場合はインデックスのDSIコード
NAME	CHAR(36) または CHAR(128)	DSI名	TYPEが‘T’の場合は表のDSI名 TYPEが‘I’の場合はインデックスのDSI名 Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
DB_CODE	INT	データベースコード	
DB_NAME	CHAR(36) または CHAR(128)	データベース名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
SCHEMA_CODE	INT	スキーマコード	
SCHEMA_NAME	CHAR(36) または CHAR(128)	スキーマ名	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
TABLE_CODE	INT	表コード	TYPEが‘I’の場合は関連する表のコードが設定されます。

列名	データ型	情報の意味	補足説明
TABLE_NAME	CHAR(36) または CHAR(128)	表名	TYPEが‘I’の場合は関連する表名が設定されます。 Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
DSO_CODE	INT	DSOコード	TYPEが‘T’の場合は表のDSOコード、 TYPEが‘I’の場合はインデックスDSOのDSOコードが設定されます。
DSO_NAME	CHAR(36) または CHAR(128)	DSO名	TYPEが‘T’の場合は表のDSO名、 TYPEが‘I’の場合はインデックスDSOのDSO名が設定されます。 Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
BASE_DSI_CODE	INT	DSIコード	TYPEが‘T’の場合はNULLが設定されます。 TYPEが‘I’の場合に表のDSIコードが設定されます。
BASE_DSI_NAME	CHAR(36) または CHAR(128)	DSI名	TYPEが‘T’の場合はNULLが設定されます。 TYPEが‘I’の場合に表のDSI名が設定されます。 Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
CREATE_TIME	CHAR(8)	DSI定義日時	
OWNER	CHAR(36) または CHAR(128)	DSI定義ユーザ	Symfoware/RDBの文字コード系とデータベースの文字コード系が異なる場合はCHAR(128)になります。
LAST_REFERENCE_TIME	CHAR(8)	最終参照日時	
LAST_UPDATE_TIME	CHAR(8)	最終更新日時	
REUSE_PAGE_POINT	INT	ページ再使用契機(%)	TYPEが‘T’の場合、かつ、ORDEER(1)が指定されている場合のみ設定されます。 上記以外は0が設定されます。
DEGENERATE	CHAR(1)	インデックス縮退指定	TYPEが‘I’の場合のみ以下のいずれかの値が設定されます。 Y :縮退指定あり N :縮退指定なし TYPEが‘T’の場合は、常にNが設定されます。
REALIGNMENT	CHAR(1)	インデックス自動再配置指定	TYPEが‘I’の場合のみ以下のいずれかの値が設定されます。 Y :再配置指定あり N :再配置指定なし TYPEが‘T’の場合は、常にNが設定されます。

列名	データ型	情報の意味	補足説明
ENTRY_NUM	INT	ページ内エントリ数	
USE_STATUS	CHAR(10)	使用状況	以下のいずれかの値が設定されます。 ACT. . . 使用中 NON. . . 未使用
DSI_STATUS	CHAR(50)	運用情報	下記のいずれかの値が設定されます。 複数の状態を持つ場合は、‘/’で区切られます。 INA : 非活性状態 INH : アクセス禁止 NUP : 更新抑止 NRC : リカバリなし NRW : 参照更新抑止 --- : 運用情報なし
STATUS_SET_CAUSE	CHAR(70)	運用情報設定原因	下記のいずれかの値が設定されます。 複数の原因がある場合は、‘/’で区切られます。 HER : ハードウェア障害 SOF : ソフトウェア障害 DER : データの異常 CRV : ダウンリカバリの異常 MRV : メディアリカバリの異常 TIH : ロールバックの異常 DIR : RDB ディレクトリファイルの異常 ALD : rdbsaloaderコマンドでの追記処理の異常 CMD : RDB コマンドの実行 FMT : フォーマット処理中 IIH : インダウトリカバリ不可閉塞 IND : インダウト状態 FTR : フラッシュトリートメント閉塞状態 --- : 運用情報なし
DSI_TYPE1	CHAR(10)	割り付け対象の種別	以下のいずれかの値を設定します。 DATA : TYPEが‘T’の場合、かつ、SEQUENTIAL格納構造または、OBJECT格納構造の場合の割り付け対象 PRIME : TYPEが‘T’の場合、かつ、RANDOM格納構造の場合の割り付け対象 INDEX : TYPEが‘I’の場合の割り付け対象(インデックス部)
TOTAL_SIZE1	INT	DSI_TYPE1の全体サイズ	単位はキロバイトで設定されます。
USE_SIZE1	INT	DSI_TYPE1の使用サイズ	単位はキロバイトで設定されます。

列名	データ型	情報の意味	補足説明
DSI_TYPE2	CHAR(10)	割り付け対象の種別	以下のいずれかの値を設定します。 OVERFLOW : TYPEが‘T’の場合、かつ、RANDOM格納構造の場合の割り付け対象 BASE : TYPEが‘I’の場合の割り付け対象(データ部) 上記以外はNULLが設定されます。
TOTAL_SIZE2	INT	DSI_TYPE2の全体サイズ	単位はキロバイトで設定されます。 DSI_TYPE2の値がNULLの場合は、NULLが設定されます。
USE_SIZE2	INT	DSI_TYPE2の使用サイズ	単位はキロバイトで設定されます。 DSI_TYPE2の値がNULLの場合は、NULLが設定されます。

注意

本表を参照する問合せは、独立性水準にREAD UNCOMMITTEDが指定されている場合の動作になります。

A.3.27 コネクションの動作環境のパラメタ設定情報(一括出力)

RDBII_CON_PRM

RDBII_CON_PRMは、本システム表にアクセスしている現コネクションで有効になっている動作環境についての情報を出力する表です。

以下に設定可能な実行パラメタについての情報を出力します。

- システム用の動作環境ファイル
- クライアント用の動作環境ファイル
- サーバ用の動作環境ファイル
- 環境変数
- Java連携の場合、default_readonlyオプション、default_isolationオプション、ctuneparamオプションに指定したパラメタ
- ODBC連携の場合、ODBCデータソースのその他パラメタに指定したパラメタ
- .NET Framework連携の場合、ConnectionStringプロパティのtuneparamキーワードに指定した実行パラメタ

Java連携およびODBC連携の場合の実行パラメタ名は、先頭文字“CLI_”を削除したパラメタ名に一致します。

なお、Java連携のdefault_readonlyオプションがDEFAULT_ACCESS_MODE、default_isolationオプションがDEFAULT_ISOLATIONの値として反映されます。default_readonlyオプションにfalseを指定した場合はREAD_WRITE、trueを指定した場合はREAD_ONLYとなります。

ただし、以下の実行パラメタの情報は、出力されません。

- DEFAULT_CONNECTION
- SERVER_ENV_FILEのSQLサーバ名
- SERVER_SPEC
- WAIT_TIME
- SSL_CERT_FILE
- SSL_CLI_CA_CERT_FILE

- SSL_PRIV_KEY_FILE
- DESCRIPTOR_SPEC
- COREFILE_PATH
- DIV_TRACE_FILE
- SET_CALLBACK
- SQL_SNAP
- MSG_PRINT
- JAVA_CONVERTER
- LOWERCASE_NAME

参照

- 動作環境ファイルの詳細については、“付録B 動作環境ファイルのパラメタ一覧”を参照してください。
- 環境変数の詳細については、“アプリケーション開発ガイド(埋込みSQL編)”の“環境変数”を参照してください。
- default_readonlyオプション、default_isolationオプションおよびctuneparamオプションの詳細については、“アプリケーション開発ガイド(JDBCドライバ編)”の“オプションの設定”を参照してください。
- ODBCデータソースのその他パラメタの詳細については、“アプリケーション開発ガイド(ODBCドライバ編)”の“アプリケーションのチューニング”を参照してください。
- tuneparamキーワードの詳細については、“アプリケーション開発ガイド(.NET Data Provider編)”の“tuneparamキーワード”を参照してください。

RDBII_CON_PRMの構成と出力情報を以下に示します。

表A.42 RDBII_CON_PRM の構成と出力情報

列名	データ型	情報の意味	補足説明
PARAMETER_NAME	VARCHAR(256)	実行パラメタ名	
PARAMETER_VALUE	VARCHAR(1024)	パラメタの設定値	現在有効となっているパラメタ設定値が、実行パラメタの指定形式で以下のように設定されます。(注) ・実行パラメタに設定するすべてのパラメタ設定値が設定されます ・複数の設定値がある場合は、コンマ“,”で区切って設定されます

注)データ長が1024バイトを超える場合、先頭から1024バイト分だけが出力されます。

注意

本表を参照する問合せは、独立性水準にREAD UNCOMMITTEDが指定されている場合の動作になります。

例1:コネクションの動作環境のパラメタ設定情報から、実行パラメタ名がDEFAULT_INDEX_SIZEであるパラメタ設定値を取得する方法

システム用の動作環境ファイルに、実行パラメタDEFAULT_INDEX_SIZEを以下の値で設定した場合、実行するSQL文とその結果は、以下のようになります。

システム用の動作環境ファイルの設定例

```
DEFAULT_INDEX_SIZE = (16,8,30720,10240,10240,0)
```

実行するSQL文

```
SELECT PARAMETER_VALUE FROM RDBII_SYSTEM.RDBII_CON_PRM WHERE PARAMETER_NAME = 'DEFAULT_INDEX_SIZE'
```

実行結果

```
PARAMETER_VALUE: 16,8,30720,10240,10240,0
                 (1)
```

(1)初期値の取得結果

例2:コネクションの動作環境のパラメタ設定情報から、実行パラメタ名がACCESS_PLANであるパラメタ設定値を取得する方法

クライアント用の動作環境ファイルに、実行パラメタACCESS_PLANが未設定の場合、実行するSQL文とその結果は、以下のようになります。

実行するSQL文

```
SELECT PARAMETER_VALUE FROM RDBII_SYSTEM.RDBII_CON_PRM WHERE PARAMETER_NAME = 'ACCESS_PLAN'
```

実行結果

```
PARAMETER_VALUE:OFF,,2,ADVICE
                (1)
```

(1)初期値の取得結果

A.3.28 コネクションの動作環境のパラメタ設定情報(順序付き出力)

RDBII_CON_PRM_DIV

RDBII_CON_PRM_DIVは、本システム表にアクセスしている現コネクションで有効になっている動作環境についての情報を出力する表です。

以下に設定可能な実行パラメタについての情報を出力します。

- システム用の動作環境ファイル
- クライアント用の動作環境ファイル
- サーバ用の動作環境ファイル
- 環境変数
- Java連携の場合、default_readonlyオプション、default_isolationオプション、ctuneparamオプションに指定したパラメタ
- ODBC連携の場合、ODBCデータソースのその他パラメタに指定したパラメタ
- .NET Framework連携の場合、ConnectionStringプロパティのtuneparamキーワードに指定した実行パラメタ

Java連携およびODBC連携の場合の実行パラメタ名は、先頭文字“CLI_”を削除したパラメタ名に一致します。

なお、Java連携のdefault_readonlyオプションがDEFAULT_ACCESS_MODE、default_isolationオプションがDEFAULT_ISOLATIONの値として反映されます。default_readonlyオプションにfalseを指定した場合はREAD_WRITE、trueを指定した場合はREAD_ONLYとなります。

ただし、以下の実行パラメタの情報は、出力されません。

- DEFAULT_CONNECTION

- SERVER_ENV_FILEのSQLサーバ名
- SERVER_SPEC
- WAIT_TIME
- SSL_CERT_FILE
- SSL_CLI_CA_CERT_FILE
- SSL_PRIV_KEY_FILE
- DESCRIPTOR_SPEC
- COREFILE_PATH
- DIV_TRACE_FILE
- SET_CALLBACK
- SQL_SNAP
- MSG_PRINT
- JAVA_CONVERTER
- LOWERCASE_NAME

参照

- 動作環境ファイルの詳細については、“付録B 動作環境ファイルのパラメタ一覧”を参照してください。
- 環境変数の詳細については、“アプリケーション開発ガイド(埋込みSQL編)”の“環境変数”を参照してください。
- default_readonlyオプション、default_isolationオプションおよびctuneparamオプションの詳細については、“アプリケーション開発ガイド(JDBCドライバ編)”の“オプションの設定”を参照してください。
- ODBCデータソースのその他パラメタの詳細については、“アプリケーション開発ガイド(ODBCドライバ編)”の“アプリケーションのチューニング”を参照してください。
- tuneparamキーワードの詳細については、“アプリケーション開発ガイド(.NET Data Provider編)”の“tuneparamキーワード”を参照してください。

RDBII_CON_PRM_DIVの構成と出力情報を以下に示します。パラメタ設定値の数が2以上の場合は、異なるレコードとなり、設定値の順番が列ORDINALに設定されます。

表A.43 RDBII_CON_PRM_DIVの構成と出力情報

列名	データ型	情報の意味	補足説明
PARAMETER_NAME	VARCHAR(256)	実行パラメタ名	
ORDINAL	INT	パラメタの順序	実行パラメタの設定値の順序 パラメタの順序は、左から順に1、2、3・・・と割り振られます。
PARAMETER_VALUE	VARCHAR(1024)	パラメタ設定値	現在有効となっているパラメタ設定値が、実行パラメタの指定形式で以下のように設定されます。(注) ・ 実行パラメタに設定する個々のパラメタ設定値が設定されます

注)データ長が1024バイトを超える場合、先頭から1024バイト分だけが出力されます。

注意

本表を参照する問合せは、独立性水準にREAD UNCOMMITTEDが指定されている場合の動作になります。

例:コネクションの動作環境のパラメタ設定情報から、実行パラメタ名がDEFAULT_INDEX_SIZE、かつ、パラメタの順序が2番目であるパラメタ設定値を取得する方法

システム用の動作環境ファイルに、実行パラメタDEFAULT_INDEX_SIZEを以下の値で設定した場合、実行するSQL文とその結果は、以下のようになります。

システム用の動作環境ファイルの設定例

```
DEFAULT_INDEX_SIZE = (16,8,30720,10240,10240,0)
                       (1)
```

(1)2番目のパラメタ設定値

実行するSQL文

```
SELECT PARAMETER_VALUE FROM RDBII_SYSTEM.RDBII_CON_PRM_DIV
 WHERE PARAMETER_NAME = 'DEFAULT_INDEX_SIZE' AND ORDINAL = 2
                                                  (1)
```

(1)2番目のパラメタ設定値の指定

実行結果

```
PARAMETER_VALUE:8
                (1)
```

(1)2番目のパラメタ設定値の取得結果

A.3.29 システムの動作環境のパラメタ設定情報(一括出力)

RDBII_SYS_PRM

RDBII_SYS_PRMは、現在有効となっているシステム用の動作環境ファイルの実行パラメタの情報を出力する表です。

実行パラメタに設定するすべてのパラメタ設定情報を出力します。

以下の動作環境ファイルに設定可能な実行パラメタについての情報を出力します。

- システム用の動作環境ファイル

ただし、以下の実行パラメタの情報は、出力されません。

- SSL_CERT_FILE
- SSL_PRIV_KEY_FILE

動作環境ファイルの詳細については、“付録B 動作環境ファイルのパラメタ一覧”を参照してください。

RDBII_SYS_PRMの構成と出力情報を以下に示します。

表A.44 RDBII_SYS_PRMの構成と出力情報

列名	データ型	情報の意味	補足説明
PARAMETER_NAME	VARCHAR(256)	実行パラメタ名	
KIND	CHAR(3)	運用パラメタの種類	SYS:システム用の動作環境ファイル 常に‘SYS’が設定されます。

列名	データ型	情報の意味	補足説明
PARAMETER_VALUE	VARCHAR(1024)	パラメタ設定値	システム用の動作環境ファイルに設定したパラメタ設定値が、実行パラメタの指定形式で以下のように設定されます。(注1)(注2) ・ 実行パラメタに設定するすべてのパラメタ設定値が設定されます ・ 複数の設定値がある場合は、コンマ“,”で区切って設定されます

注1)データ長が1024バイトを超える場合、先頭から1024バイト分だけが出力されます。

注2)設定値が定義型サイズに満たない場合は、残りの部分に空白が設定されます。

パラメタ設定値を取得する例については、“A.3.27 コネクションの動作環境のパラメタ設定情報(一括出力)”を参照してください。

注意

本表を参照する問合せは、独立性水準にREAD UNCOMMITTEDが指定されている場合の動作になります。

A.3.30 システムの動作環境のパラメタ設定情報(順序付き出力)

RDBII_SYS_PRM_DIV

RDBII_SYS_PRM_DIVは、現在有効となっているシステム用の動作環境ファイルの実行パラメタの情報を出力する表です。

実行パラメタに設定する個々のパラメタの設定情報を出力します。

以下の動作環境ファイルに設定可能な実行パラメタについての情報を出力します。

- システム用の動作環境ファイル

ただし、以下の実行パラメタの情報は、出力されません。

- SSL_CERT_FILE
- SSL_PRIV_KEY_FILE

動作環境ファイルの詳細については、“付録B 動作環境ファイルのパラメタ一覧”を参照してください。

RDBII_SYS_PRM_DIVの構成と出力情報を以下に示します。パラメタ設定値の数が2以上の場合は、異なるレコードとなり、設定値の順番が列ORDINALに設定されます。

表A.45 RDBII_SYS_PRM_DIVの構成と出力情報

列名	データ型	情報の意味	補足説明
PARAMETER_NAME	VARCHAR(256)	実行パラメタ名	
KIND	CHAR(3)	運用パラメタの種類	SYS:システム用の動作環境ファイル 常に‘SYS’が設定されます。

列名	データ型	情報の意味	補足説明
ORDINAL	INT	パラメタの順序	実行パラメタの設定値の順序 パラメタの順序は、左から順に1、2、3・・・と割り振られます。
PARAMETER_VALUE	VARCHAR(1024)	パラメタ設定値	システム用の動作環境ファイルに設定したパラメタ設定値が、実行パラメタの指定形式で以下のように設定されます。(注1)(注2) ・ 実行パラメタに設定する個々のパラメタ設定値が設定されます

注1)データ長が1024バイトを超える場合、先頭から1024バイト分だけが出力されます

注2) 設定値が定義型サイズに満たない場合は、残りの部分に空白が設定されます。

パラメタ設定値を取得する例については、“A.3.28 コネクションの動作環境のパラメタ設定情報(順序付き出力)”を参照してください。

注意

本表を参照する問合せは、独立性水準にREAD UNCOMMITTEDが指定されている場合の動作になります。

A.3.31 コネクション情報

RDBII_CON_INF

RDBII_CON_INFは、RDBシステムに接続中の全コネクションについての情報を出力する表です。

RDBII_CON_INFの構成と出力情報を以下に示します。

表A.46 RDBII_CON_INFの構成と出力情報

列名	データ型	情報の意味	補足説明
NAME	CHAR(128)	モジュール名	以下のいずれかの値が設定されます ・ システム内通信のアプリケーションの場合 アプリケーションのロードモジュール名／ コネクション名 (アプリケーションロードモジュール名が15バイトを超える場合は、 先頭の15バイトだけが出力されます。) ・ トランザクションモニタ配下のアプリケーションの場合 データベース名／コネクション名

列名	データ型	情報の意味	補足説明
			(トランザクションモニタ配下のアプリケーションのデータベース名／ コネクション名は、OPEN-INFO文字列に指定された値が表示されます。) ・ TCP/IP接続のアプリケーションの場合 接続端末のIPアドレス／コネクション名 または接続端末のホスト名／コネクション 名（注1） ・ RDA-SV接続のアプリケーション(SolarisまたはWindows(R)の場合) 接続端末のIPアドレス／コネクション名 ・ RDBコマンドの場合 RDBコマンド名 ・ ODOS連携のODBCアプリケーションの場合 接続端末のIPアドレス／データソース 名、接続端末のホスト名／データソース 名、またはアプリケーションのロードモ ジュール名／データソース名 （注1）(注2)
CONNECTION_ID	CHAR(22)	コネクション識別子	コネクションID RDBシステム内で一意な値が設定されます。(注2)
PROCESS_ID	INT	プロセスID	
ELAPSE	INT	経過時間	RDBシステムに接続してからの経過時間(秒)
IDLE	INT	アイドル時間	使用中のコネクションがアプリケーションの要求を待ってからの経過時間(秒)
CLIENT_TYPE	CHAR(10)	クライアントプロセス種別	以下のいずれかの値が設定されます SQL : システム内通信のアプリケーション SQL-XA : トランザクションモニタ配下のアプリケーション TCP/IP : リモートアクセスのアプリケーション RDA-SV : RDA-SV接続のアプリケーション(SolarisまたはWindows(R)の場合のみ表示) RDBCMD : RDBコマンド

列名	データ型	情報の意味	補足説明
			ODOS : ODOS連携のODBC アプリケーション(注2)
LOGIN_NAME	CHAR(128)	ログイン名	
TRANSACTION_STATUS	CHAR(10)	トランザクション状態	以下のいずれかの値が設定されます ACT : トランザクションは開始されている INACT : トランザクションは開始されていない TINH : トランザクションはテンポラリログファイル閉塞前から開始されている(注2)
ACTION_INF	CHAR(64)	アクション情報	常に空白が設定されます。
CLIENT_INF	CHAR(64)	クライアント情報	Interstage Application Serverと連携している場合、Web サーバ情報が設定されます。 Interstage Application Serverと連携していない場合、空白が設定されます。(注2)
MODULE_INF	CHAR(64)	モジュール情報	Interstage Application Serverと連携している場合、IJServer名が設定されます。 Interstage Application Serverと連携していない場合、空白が設定されます。(注2)
DB_NAME	CHAR(128)	コネクト対象のデータベース名	常に空白が設定されます。

注1) 接続端末のIPアドレスを格納するか、ホスト名を格納するかを、RDB構成パラメタファイルのRDBCLIENTINFで指定します。

注2) 設定値が定義型サイズに満たない場合は、残りの部分に空白が設定されます。

注意

- 本表は、今後列が追加される場合があるため、参照する際は選択リストに列名を列挙するようにしてください。
- 本表を参照する問合せは、独立性水準にREAD UNCOMMITTEDが指定されている場合の動作になります。

例:コネクション情報から、モジュール名がAPL001/CONNECT1であるアプリケーションのプロセスIDとトランザクション状態を取得する方法

アプリケーションロードモジュール名がAPL001、コネクション名がCONNECT1、プロセスIDが3083、トランザクションが開始されている場合、実行するSQL文とその結果は、以下のようになります。

実行するSQL文

```
SELECT PROCESS_ID, TRANSACTION_STATUS FROM RDBII_SYSTEM.RDBII_CON_INF WHERE NAME = 'APL001/CONNECT1'
```

実行結果

```
PROCESS_ID: 3083
TRANSACTION_STATUS: ACT
```

付録B　動作環境ファイルのパラメター一覧

動作環境ファイルに指定できるパラメタの一覧を説明します。

動作環境ファイルには、動作環境ごとに以下の3つのファイルがあります。

- システム用の動作環境ファイル
- クライアント用の動作環境ファイル
- サーバ用の動作環境ファイル

参考

データベース簡単運用の場合には、Symfoware Severのインストール時に、システム用の動作環境として最適な環境が設定されるため、チューニングの必要がありません。よって、システム用の動作環境ファイルは、作成不要です。

動作環境の設定項目の中には、各ファイルに共通して指定できるパラメタがあります。優先順位は、以下のとおりです。

1. サーバ用の動作環境ファイル
2. Java連携の場合、JDBCドライバで指定可能なオプション
   ODBC連携の場合、ODOSのオプションで設定できるパラメタ
   .NET Framework連携の場合、tuneparamキーワード
3. クライアント用の動作環境ファイル
4. システム用の動作環境ファイル

表B.1 動作環境ファイルのパラメター覧

分類	実行パラメタ	概要	優先順位		
			1	2	3
			SV	CL (注1)	SY
通信	BUFFER_SIZE	通信に利用するバッファのサイズ	―	○	―
	CLUSTER_SERVICE_NAME	アプリケーションをフェイルオーバ運用で実行する場合に必要な情報	―	○ (注2)	―
	COMMUNICATION_BUFFER	ローカルアクセスの場合に、サーバ側で使用するバッファのサイズ	―	―	○
	DEFAULT_CONNECTION	サーバとの結合情報のデフォルト	―	○	―
	MAX_CONNECT_SYS	ローカルで接続できるコネクションの数の最大値	―	―	○
	MAX_CONNECT_TCP	1つのSymfoware/RDBに対し、リモートアクセスできるコネクションの数の最大値	―	―	○
	RDB_KEEPALIVE	Symfoware/RDBにリモートアクセスする場合	―	―	○

分類	実行パラメタ	概要	優先順位		
			1	2	3
			SV	CL (注1)	SY
		にKEEPALIVE機能を利用するか否か			
	SERVER_ENV_FILE	使用するサーバ用の動作環境ファイル名	－	○	－
	SERVER_SPEC	リモートのサーバと通信するための情報	－	○	－
	TRAN_SPEC	SQLエラー発生時のトランザクション	－	○	－
	TRAN_TIME_LIMIT	1つのトランザクションの最大使用可能時間	○	○	○
	WAIT_TIME	通信時の待ち時間	－	○	－
通信データの暗号化	SSL_CERT_FILE	サーバ認証を行う場合のサーバ証明書ファイル名	－	－	○
	SSL_CIPHER_SUITES	サーバ認証を行うか否か	－	－	○
	SSL_CLI_CA_CERT_FILE	サーバ認証で使用するCA証明書ファイル名	－	○ (注2)	－
	SSL_PRIV_KEY_FILE	サーバ認証を行う場合のサーバ秘密鍵ファイル名	－	－	○
	SSL_USE	クライアント側からの暗号化した通信を受け入れるか否か	－	－	○
作業領域など	DESCRIPTOR_SPEC	動的SQLのSQL記述子の情報	－	○	－
	MAX_SQL	同時に操作できるSQL文の数	－	○	－
	OPL_BUFFER_SIZE	SQL文の実行手順を格納しておくバッファのサイズ	－	○	－
	RESULT_BUFFER	一括FETCHを行う場合のバッファの数とサイズ	－	○	－
	SORT_MEM_SIZE	作業用ソート領域として使うメモリサイズ	○	○	○
	WORK_ALLOC_SPACESIZE	作業用テーブルおよび作業用ソート領域として使用するファイルサイズ	○	○	－
	WORK_MEM_SIZE	作業用テーブルとして使うメモリのサイズ	○	○	○
	WORK_PATH	作業用テーブルおよび作業用ソート領域のパス	○	○	○

分類	実行パラメタ	概要	優先順位		
			1	2	3
			SV	CL (注1)	SY
データ処理	CAL_ERROR	代入処理でオーバフローが起きた場合の処理	－	○	－
	CHARACTER_TRANSLATE	文字コードの変換をクライアントで行うか否か	－	○	－
	CHAR_SET	アプリケーション中の文字のコード	－	○	－
	JAVA_CONVERTER	JDBCドライバがコード変換時に使用するJavaのエンコーディング方法	－	○	－
	LOWERCASE_NAME	JDBCドライバ利用時において、列名に全角小文字を使用し、かつResultSet.getXXXメソッドを使用する場合に、引数に指定した列名を正しく認識させるか否か	－	○	－
	NCHAR_CODE	アプリケーション中の日本語文字のコード	－	○	－
	SURROGATE_PAIR_NUMBER	UNICODEの補助文字(1～16面の4バイト文字)の文字数	○	○	○
表・インデックス	DEFAULT_DSI_NAME	格納構造定義を行わない表またはインデックスを作成する場合の、Symfoware/RDBが自動的に生成するDSO名またはDSI名の生成方法	－	－	○
	DEFAULT_DSI_TYPE	格納構造定義を行わない表を作成する場合、Symfoware/RDBが自動的に生成する表のDSOの格納構造の選択	－	○ (注2)	○
	DEFAULT_INDEX_SIZE	格納構造定義を行わないインデックスを作成する場合のインデックスのデータ格納域の初期量、拡張量、ページ長など	○ (注2)	○ (注2)	○
	DEFAULT_OBJECT_TABLE_SIZE	格納構造定義を行わない表を作成する場合のOBJECT構造の表のデータ格納域の初期	○ (注2)	○ (注2)	○

分類	実行パラメタ	概要	優先順位		
			1	2	3
			SV	CL (注1)	SY
		量、拡張量、ページ長など			
	DEFAULT_TABLE_SIZE	格納構造定義を行わない表を作成する場合の表のデータ格納域の初期量、拡張量、ページ長など	○ (注2)	○ (注2)	○
	DSI_EXPAND_POINT	DSIの容量拡張を起動するか否か	－	○ (注2)	－
	INCLUSION_DSI	アプリケーションで使用するDSIを限定する	○ (注2)	○ (注2)	－
	INDEX_PREFIX	格納構造定義を行わないインデックスを作成する場合の、Symfoware/RDBが自動的に生成するインデックスのDSO名およびインデックスのDSI名のプレフィックス	－	－	○
	TABLE_PREFIX	格納構造定義を行わない表を作成する場合の、Symfoware/RDBが自動的に生成する表のDSO名および表のDSI名のプレフィックス	－	－	○
	TEMPORARY_INDEX_SIZE	一時表にインデックスを定義する場合のインデックスのデータ格納域の初期量、拡張量など	○ (注2)	○ (注2)	○
	TEMPORARY_TABLE_SIZE	一時表を定義する場合の表のデータ格納域の初期量、拡張量など	○ (注2)	○ (注2)	○
排他	DSO_LOCK	使用するDSOの占有の単位、占有モード	○ (注2)	○ (注2)	－
	ISOLATION_WAIT	占有待ちの方式	－	○	－
	R_LOCK	占有の単位を行とする	－	○	○
トランザクション	DEFAULT_ACCESS_MODE	トランザクションアクセスモードの初期値を指定する	－	○	○
	DEFAULT_ISOLATION	独立性水準の初期値を指定する	－	○	○

分類	実行パラメタ	概要	優先順位		
			1	2	3
			SV	CL (注1)	SY
	TRANSACTION_MODE_CHECK	8.x系以前のSymfoware Server クライアント機能からの接続をエラーとするか否か	―	―	○
デバッグ	COREFILE_PATH	アプリケーションで異常が発生した場合のダンプ出力先	―	○	―
	DIV_TRACE_FILE	複数プロセス配下でアプリケーションが動作している場合、個別のトレース情報を出力するか否か	―	○	―
	EXTERNAL_PROCESS_CORE	Symfoware/RDBの外部で動作するプロセスで異常が発生した場合のダンプ出力先	―	―	○
	MAX_EXTPROC_CORE_NUM	Symfoware/RDBの外部で動作するプロセスで異常が発生した場合のダンプ出力最大数	―	―	○
	ROUTINE_SNAP	ROUTINE_SNAP機能を利用するか否か	―	○	―
	SET_CALLBACK	コールバック関数の動的登録機能を利用するか否か	―	○	―
	SQL_SNAP	SQL_SNAP機能を利用するか否か	―	○	―
アクセスプラン・性能情報	ACCESS_PLAN	アプリケーション単位でアクセスプランを取得するか否か SQL文に対するアドバイスを出力するか否か	―	○	―
	CHOOSE_TID_UNION	WHERE句にブール演算子“OR”、または行値構成子を指定した場合、TIDユニオンマージのアクセスプランのみを作成するか否か	○	○	○
	GROUP_COL_COND_MOVE	導出表を絞り込む探索条件を指定した場合、その探索条件を導出表のWHERE句に移動するか否か	○	○	○
	IGNORE_INDEX	インデックスを使用しないアクセスプランを選択するか否か	○	○	―

分類	実行パラメタ	概要	優先順位		
			1	2	3
			SV	CL (注1)	SY
	INACTIVE_INDEX_SCAN	非活性状態のインデックスDSIを含むインデックスを使用したアクセスプランを選択するか否か	○	○	○
	JOIN_ORDER	結合表と他の表のジョイン順	○	○	○
	JOIN_RULE	ジョインする方法	○	○	○
	MAX_SCAN_RANGE	インデックス、クラスタキー、または分割キーの検索範囲の最大数	○	○	○
	SAME_COST_JOIN_ORDER	最適化情報を設定していない場合、ジョイン順をV5以前と同じにするか否か	○	○	○
	SCAN_KEY_ARITHMETIC_RANGE	四則演算の検索範囲について、インデックスの範囲検索またはクラスタキーの検索を行うか否か	○	○	○
	SCAN_KEY_CAST	探索条件のCASTオペランドに指定した列でインデックスの範囲検索、または、クラスタキー検索を行うか否か	○	○	○
	SORT_HASHAREA_SIZE	ソート処理がレコードをハッシングして格納するための領域サイズ	○	○	○
	SQL_ADVISOR	SQL文に対するアドバイスを出力するか否か	－	－	○
	SQL_TRACE	アプリケーション単位でSQL性能情報を取得するか否か	－	○	－
	SS_RATE	述語ごとの検索範囲の選択率の値	○	○	○
	TID_SORT	インデックス検索と表データ取得のアクセスモデルでTIDソートを利用するか否か	○	○	○
	TID_UNION	TIDユニオンマージのアクセスモデルを有効にするか否か	○	○	○
	USQL_LOCK	UPDATE文:探索またはDELETE文:探索の更新標的レコードを位	○	○	○

分類	実行パラメタ	概要	優先順位		
			1	2	3
			SV	CL (注1)	SY
		置づける部分の占有モード			
プロセス制御	EXTERNAL_PROCESS_USER	Symfoware/RDBの外部で動作するプロセスの実効ユーザID指定	－	－	○
メッセージ	CONSOLE_MSG	メッセージログファイルへメッセージを出力するか否か	－	－	○
	MSG_LANG	表示するメッセージの言語種を設定する	－	○	－
	MSG_PRINT	SQL文実行時にエラーメッセージを表示するか否か	－	○	－
リカバリ	RCV_MODE	アプリケーションのリカバリ水準を指定する	－	○ (注2)	－
予約語とSQL機能	SQL_LEVEL	アプリケーションの予約語とSQL機能のレベルを設定する	－	○	－
並列クエリ	MAX_PARALLEL	データベースを並列に検索する場合の多重度	○ (注2)	○ (注2)	○
	PARALLEL_SCAN	アプリケーション単位またはコネクション単位に、データベースを並列に検索するか否か	○ (注2)	○ (注2)	－
その他	ALTER_CHECK	表の設計変更により表の列を変更した場合、アプリケーションへの影響をチェックするか否か	－	○ (注2)	－
	ARC_FULL	アーカイブログ満杯時にエラー復帰するか否か	－	○ (注2)	○
	NAME_SIZE_CHECK	資源名のサイズをチェックアウトするか否か	－	－	○
	READ_ERROR_INHIBIT	データベースの読込みでI/Oエラーが発生した場合に、読込み対象データをアクセス禁止にするか否か	－	－	○
	ROLLBACK_MEM_ERROR	ロールバックがメモリ不足によって失敗した場合に、システムを強制停止させるか、ロール	－	－	○

分類	実行パラメタ	概要	優先順位		
			1	2	3
			SV	CL (注1)	SY
		バック対象データをアクセス禁止にするか			
	S L SIGNAL_INF	シグナルをアプリケーションで使用するか否か	－	○	－
	MAX_FUNCTION	関数の入れ子の数、または演算式の数が定量制限値を超えた場合、チェックアウトするか否か	－	－	○
	MAX_JOINED_TABLE	結合表の入れ子の数が定量制限値を超えた場合、チェックアウトするか否か	－	－	○
	MAX_UNION	UNIONの数が定量制限値を超えた場合、チェックアウトするか否か	－	－	○

SV:サーバ用の動作環境ファイルへの指定が可能であるか否かを表します。

CL:クライアント用の動作環境ファイルへの指定が可能であるか否かを表します。

SY:システム用の動作環境ファイルへの指定が可能であるか否かを表します。

○:指定可

－:指定不可

注1) Java連携の場合はctuneparamオプション、ODBC連携の場合はODOSのオプションで設定できるパラメタ、.NET Framework連携の場合はtuneparamキーワードで、クライアント用の動作環境ファイルに相当する指定をします。Java連携の場合は、クライアント用の動作環境ファイルを利用することもできます。

注2)データベース簡単運用の場合は、指定できません。

B.1 システム用の動作環境ファイル

システム用の動作環境ファイルには、データベースシステム全体に共通する実行環境を、実行パラメタとして記述します。

参考

データベース簡単運用の場合には、Symfoware Severのインストール時に、システム用の動作環境として最適な環境が設定されるため、チューニングの必要がありません。よって、システム用の動作環境ファイルは、作成不要です。

参照

- システム用の動作環境ファイルの詳細については、“セットアップガイド”を参照してください。

・ アクセスプランおよび性能情報に関する実行パラメタの詳細については、“SQLTOOLユーザーズガイド”を参照してください。

システム用の動作環境ファイルに記述できる実行パラメタは、以下に示す種類があります。

表B.2 システム用の動作環境ファイルの実行パラメタの種類

分類	実行パラメタ	概要	記述数	記述の省略
通信	COMMUNICATION_BUFFER	ローカルアクセスの場合に、サーバ側で使用するバッファのサイズ	単一	省略可能
	MAX_CONNECT_SYS	ローカルで接続できるコネクションの数の最大値	単一	省略可能
	MAX_CONNECT_TCP	1つのSymfoware/RDBに対し、リモートアクセスできるコネクションの数の最大値	単一	省略可能
	RDB_KEEPALIVE	Symfoware/RDBにリモートアクセスする場合にKEEPALIVE機能を利用するか否か	単一	省略可能
	TRAN_TIME_LIMIT	1つのトランザクションの最大使用可能時間	単一	省略可能
通信データの暗号化	SSL_CERT_FILE	サーバ認証を行う場合のサーバ証明書ファイル名	単一	サーバ認証を行う場合は省略不可。
	SSL_CIPHER_SUITES	サーバ認証を行うか否か	単一	省略可能
	SSL_PRIV_KEY_FILE	サーバ認証を行う場合のサーバ秘密鍵ファイル名	単一	サーバ認証を行う場合は省略不可。
	SSL_USE	クライアント側からの暗号化した通信を受け入れるか否か	単一	省略可能
作業領域	SORT_MEM_SIZE	作業用ソート領域として使うメモリサイズ	単一	省略可能
	WORK_MEM_SIZE	作業用テーブルとして使うメモリのサイズ	単一	省略可能
	WORK_PATH	作業用テーブルおよび作業用ソート領域のパス	単一	省略可能
データ処理	SURROGATE_PAIR_NUMBER	UNICODEの補助文字(1～16面の4バイト文字)の文字数	単一	省略可能
表・インデックス	DEFAULT_DSI_NAME	格納構造定義を行わない表またはインデックスを作成する場合の、Symfoware/RDBが自動的に生成するDSO名またはDSI名の生成方法	単一	省略可能

分類	実行パラメタ	概要	記述数	記述の省略
	DEFAULT_DSI_TYPE	格納構造定義を行わない表を作成する場合、Symfoware/RDBが自動的に生成する表のDSOの格納構造の選択	単一	省略可能
	DEFAULT_INDEX_SIZE	格納構造定義を行わないインデックスを作成する場合のインデックスのデータ格納域の初期量、拡張量、ページ長など	単一	省略可能
	DEFAULT_OBJECT_TABLE_SIZE	格納構造定義を行わない表を作成する場合のOBJECT構造の表のデータ格納域の初期量、拡張量、ページ長など	単一	省略可能
	DEFAULT_TABLE_SIZE	格納構造定義を行わない表を作成する場合の表のデータ格納域の初期量、拡張量、ページ長など	単一	省略可能
	INDEX_PREFIX	格納構造定義を行わないインデックスを作成する場合の、Symfoware/RDBが自動的に生成するインデックスのDSO名およびインデックスの DSI名のプレフィックス	単一	省略可能
	TABLE_PREFIX	格納構造定義を行わない表を作成する場合の、Symfoware/RDBが自動的に生成する表のDSO名および表のDSI名のプレフィックス	単一	省略可能
	TEMPORARY_INDEX_SIZE	一時表にインデックスを定義する場合のインデックスのデータ格納域の初期量、拡張量など	単一	省略可能
	TEMPORARY_TABLE_SIZE	一時表を定義する場合の表のデータ格納域の初期量、拡張量など	単一	省略可能
排他	R_LOCK	占有の単位を行とする	単一	省略可能
トランザクション	DEFAULT_ACCESS_MODE	トランザクションアクセスモードの初期値を指定する	単一	省略可能
	DEFAULT_ISOLATION	独立性水準の初期値を指定する	単一	省略可能
	TRANSACTION_MODE_CHECK	8.x系以前のSymfoware Server クライアント機能からの接続をエラーとするか否か	単一	省略可能
デバッグ	EXTERNAL_PROCESS_CORE	Symfoware/RDBの外部で動作するプロセスで異常が発生した場合のダンプ出力先	単一	省略可能
	MAX_EXTPROC_CORE_NUM	Symfoware/RDBの外部で動作するプロセスで異常が発生した場合のダンプ出力最大数	単一	省略可能

分類	実行パラメタ	概要	記述数	記述の省略
アクセスプラン・性能情報	CHOOSE_TID_UNION	WHERE句にブール演算子”OR”、または、行値構成子を指定した場合、TIDユニオンマージのアクセスプランのみを作成するか否か	単一	省略可能
	GROUP_COL_COND_MOVE	導出表を絞り込む探索条件を指定した場合、その探索条件を導出表のWHERE句に移動するか否か	単一	省略可能
	INACTIVE_INDEX_SCAN	非活性状態のインデックスDSIを含むインデックスを使用したアクセスプランを選択するか否か	単一	省略可能
	JOIN_ORDER	結合表と他の表のジョイン順	単一	省略可能
	JOIN_RULE	ジョインする方法	単一	省略可能
	MAX_SCAN_RANGE	インデックス、クラスタキー、または分割キーの検索範囲の最大数	単一	省略可能
	SAME_COST_JOIN_ORDER	最適化情報を設定していない場合、ジョイン順をV5以前と同じにするか否か	単一	省略可能
	SCAN_KEY_ARITHMETIC_RANGE	四則演算の検索範囲について、インデックスの範囲検索またはクラスタキーの検索を行うか否か	単一	省略可能
	SCAN_KEY_CAST	探索条件のCASTオペランドに指定した列でインデックスの範囲検索またはクラスタキーの検索を行うか否か	単一	省略可能
	SORT_HASHAREA_SIZE	ソート処理がレコードをハッシングして格納するための領域サイズ	単一	省略可能
	SQL_ADVISOR	SQL文に対するアドバイスを出力するか否か	単一	省略可能
	SS_RATE	述語ごとの検索範囲の選択率の値	単一	省略可能
	TID_SORT	インデックス検索と表データ取得のアクセスモデルでTIDソートを利用するか否か	単一	省略可能
	TID_UNION	TIDユニオンマージのアクセスモデルを有効にするか否か	単一	省略可能
	USQL_LOCK	UPDATE文:探索またはDELETE文:探索の更新標的レコードを位置づける部分の占有モード	単一	省略可能

分類	実行パラメタ	概要	記述数	記述の省略
プロセス制御	EXTERNAL_PROCESS_USER	Symfoware/RDBの外部で動作するプロセスの実効ユーザID指定	単一	省略可能
メッセージ	CONSOLE_MSG	S L メッセージログファイルへメッセージを出力するか否か W イベントログへメッセージを出力するか否か	単一	省略可能
並列クエリ	MAX_PARALLEL	データベースを並列に検索する場合の多重度	単一	省略可能
その他	ARC_FULL	アーカイブログ満杯時にエラー復帰するか否か	単一	省略可能
	NAME_SIZE_CHECK	資源名のサイズをチェックアウトするか否か	単一	省略可能
	READ_ERROR_INHIBIT	データベースの読込みでI/Oエラーが発生した場合に、読込み対象データをアクセス禁止にするか否か	単一	省略可能
	ROLLBACK_MEM_ERROR	ロールバックがメモリ不足によって失敗した場合に、システムを強制停止させるか、ロールバック対象データをアクセス禁止にするか	単一	省略可能
	MAX_FUNCTION	関数の入れ子の数、または演算式の数が定量制限値を超えた場合、チェックアウトするか否か	単一	省略可能
	MAX_JOINED_TABLE	結合表の入れ子の数が定量制限値を超えた場合、チェックアウトするか否か	単一	省略可能
	MAX_UNION	UNIONの数が定量制限値を超えた場合、チェックアウトするか否か	単一	省略可能

B.2 クライアント用の動作環境ファイル

クライアント用の動作環境ファイルには、アプリケーションの実行に必要な環境情報として実行パラメタを記述します。

参照

- クライアント用の動作環境ファイルの詳細については、以下を参照してください。
  - ―“アプリケーション開発ガイド(埋込みSQL編)”の“クライアント用の動作環境ファイルによる定義”
  - ―“アプリケーション開発ガイド(JDBCドライバ編)”の“クライアント用の動作環境ファイルの指定について”
- アクセスプランおよび性能情報に関する実行パラメタの詳細については、“SQLTOOLユーザーズガイド”を参照してください。

実行パラメタは、以下に示す種類があります。

表B.3 クライアント用の動作環境ファイルの実行パラメタの種類

分類	実行パラメタ	概要	記述数	記述の省略	埋込みSQL	JDBC
通信	BUFFER_SIZE	通信に利用するバッファ(クライアント側)のサイズ	単一	省略可能	○	○
	CLUSTER_SERVICE_NAME	アプリケーションをフェイルオーバ運用で実行する場合に必要な情報	単一	省略可能	○ (注)	－
	DEFAULT_CONNECTION	サーバとの結合情報のデフォルト	単一	CONNECT文にDEFAULTを指定する場合は省略不可。	○	－
	SERVER_ENV_FILE	使用するサーバ用の動作環境ファイル名	複数可	省略可能	○	○
	SERVER_SPEC	リモートのサーバと通信するための情報	複数可	リモートのデータベースにアクセスする場合省略不可。ローカルの場合は指定できない。	○	－
	TRAN_SPEC	SQLエラー発生時のトランザクション	単一	省略可能	○	○
	TRAN_TIME_LIMIT	1つのトランザクションの最大使用可能時間	単一	省略可能	○	○
	WAIT_TIME	通信時の待ち時間	単一	省略可能	○	○
通信データの暗号化	SSL_CLI_CA_CERT_FILE	サーバ認証を行う場合のCA証明書ファイル名	複数可	サーバ認証を行う場合は省略不可。	○ (注)	○ (注)
作業領域など	DESCRIPTOR_SPEC	動的SQLのSQL記述子の情報	単一	省略可能	○	－
	MAX_SQL	同時に操作できるSQL文の数	単一	省略可能	○	○
	OPL_BUFFER_SIZE	SQL文の実行手順を格納しておくバッファのサイズ	単一	省略可能	○	○
	RESULT_BUFFER	一括FETCHを行う場合のバッファの数とサイズ	単一	省略可能	○	○
	SORT_MEM_SIZE	作業用ソート領域として使うメモリサイズ	単一	省略可能	○	○
	WORK_ALLOC_SPACESIZE	作業用テーブルおよび作業用ソート領域	単一	省略可能	○	○

分類	実行パラメタ	概要	記述数	記述の省略	埋込みSQL	JDBC
		として使用するファイルサイズ				
	WORK_MEM_SIZE	作業用テーブルとして使うメモリのサイズ	単一	省略可能	○	○
	WORK_PATH	作業用テーブルおよび作業用ソート領域のパス	単一	省略可能	○	○
データ処理	CAL_ERROR	代入処理でオーバフローが起きた場合の処理	単一	省略可能	○	○
	CHARACTER_TRANSLATE	文字コードの変換をクライアントで行うか否か	単一	省略可能	○	○
	CHAR_SET	アプリケーション中の文字のコード	単一	省略可能	○	－
	JAVA_CONVERTER	JDBCドライバがコード変換時に使用するJavaのエンコーディング方法	単一	省略可能	－	○
	LOWERCASE_NAME	JDBCドライバ利用時において、列名に全角小文字を使用し、かつResultSet.getXXXメソッドを使用する場合に、引数に指定した列名を正しく認識させるか否か	単一	省略可能	－	○
	NCHAR_CODE	アプリケーション中の日本語文字のコード	単一	省略可能	○	－
	SURROGATE_PAIR_NUMBER	UNICODEの補助文字(1～16面の4バイト文字)の文字数	単一	省略可能	○	○
表・インデックス	DEFAULT_DSI_TYPE	格納構造定義を行わない表を作成する場合、Symfoware/RDBが自動的に生成する表のDSOの格納構造の選択	単一	省略可能	○(注)	－
	DEFAULT_INDEX_SIZE	格納構造定義を行わないインデックスを作成する場合のインデックスのデータ格納域の初期量、拡張量、ページ長など	単一	省略可能	○(注)	○(注)
	DEFAULT_OBJECT_TABLE_SIZE	格納構造定義を行わない表を作成する場合のOBJECT構造の表のデータ格納	単一	省略可能	○(注)	○(注)

分類	実行パラメタ	概要	記述数	記述の省略	埋込みSQL	JDBC
		域の初期量、拡張量、ページ長など				
	DEFAULT_TABLE_SIZE	格納構造定義を行わない表を作成する場合の表のデータ格納域の初期量、拡張量、ページ長など	単一	省略可能	○(注)	○(注)
	DSI_EXPAND_POINT	DSIの容量拡張を起動するか否か	単一	省略可能	○(注)	○(注)
	INCLUSION_DSI	アプリケーションで使用するDSIを限定する	単一	省略可能	○(注)	○(注)
	TEMPORARY_INDEX_SIZE	一時表にインデックスを定義する場合のインデックスのデータ格納域の初期量、拡張量など	単一	省略可能	○(注)	○(注)
	TEMPORARY_TABLE_SIZE	一時表を定義する場合の表のデータ格納域の初期量、拡張量など	単一	省略可能	○(注)	○(注)
排他	DSO_LOCK	使用するDSOの占有の単位、占有モード	単一	省略可能	○(注)	○(注)
	ISOLATION_WAIT	占有待ちの方式	単一	省略可能	○	○
	R_LOCK	占有の単位を行とする	単一	省略可能	○	○
トランザクション	DEFAULT_ACCESS_MODE	トランザクションアクセスモードの初期値を指定する	単一	省略可能	○	○
	DEFAULT_ISOLATION	独立性水準の初期値を指定する	単一	省略可能	○	○
デバッグ	COREFILE_PATH	アプリケーションで異常が発生した場合のダンプ出力先	単一	省略可能	○	－
	DIV_TRACE_FILE	複数プロセス配下でアプリケーションが動作している場合、個別のトレース情報を出力するか否か	単一	省略可能	○	－
	ROUTINE_SNAP	ROUTINE_SNAP機能を利用するか否か	単一	省略可能	○	○
	SET_CALLBACK	コールバック関数の動的登録機能を利用するか否か	単一	省略可能	○	－
	SQL_SNAP	SQL_SNAP機能を利用するか否か	単一	省略可能	○	－

分類	実行パラメタ	概要	記述数	記述の省略	埋込みSQL	JDBC
アクセスプランおよび性能情報	ACCESS_PLAN	アプリケーション単位でアクセスプランを取得するか否かおよびSQL文に対するアドバイスを出力するか否か	単一	省略可能	○	○
	CHOOSE_TID_UNION	WHERE句にブール演算子”OR”、または、行値構成子を指定した場合、TIDユニオンマージのアクセスプランのみを作成するか否か	単一	省略可能	○	○
	GROUP_COL_COND_MOVE	導出表を絞り込む探索条件を指定した場合、その探索条件を導出表のWHERE句に移動するか否か	単一	省略可能	○	○
	IGNORE_INDEX	インデックスを使用しないアクセスプランを選択するか否か	単一	省略可能	○	○
	INACTIVE_INDEX_SCAN	非活性状態のインデックスDSIを含むインデックスを使用したアクセスプランを選択するか否か	単一	省略可能	○	○
	JOIN_ORDER	結合表と他の表のジョイン順	単一	省略可能	○	○
	JOIN_RULE	ジョインする方法	単一	省略可能	○	○
	MAX_SCAN_RANGE	インデックス、クラスタキー、または分割キーの検索範囲の最大数	単一	省略可能	○	○
	SAME_COST_JOIN_ORDER	最適化情報を設定していない場合、ジョイン順をV5以前と同じにするか否か	単一	省略可能	○	○
	SCAN_KEY_ARITHMETIC_RANGE	四則演算の検索範囲について、インデックスの範囲検索またはクラスタキーの検索を行うか否か	単一	省略可能	○	○
	SCAN_KEY_CAST	探索条件のCASTオペランドに指定した列でインデックスの範囲検索、または、クラスタキー検索を行うか否か	単一	省略可能	○	○

分類	実行パラメタ	概要	記述数	記述の省略	埋込みSQL	JDBC
	SORT_HASHAREA_SIZE	ソート処理がレコードをハッシングして格納するための領域サイズ	単一	省略可能	○	○
	SQL_TRACE	アプリケーション単位でSQL性能情報を取得するか否か	単一	省略可能	○	○
	SS_RATE	述語ごとの検索範囲の選択率の値	単一	省略可能	○	○
	TID_SORT	インデックス検索と表データ取得のアクセスモデルでTIDソートを利用するか否か	単一	省略可能	○	○
	TID_UNION	TIDユニオンマージのアクセスモデルを有効にするか否か	単一	省略可能	○	○
	USQL_LOCK	UPDATE文:探索またはDELETE文:探索の更新標的レコードを位置づける部分の占有モード	単一	省略可能	○	○
メッセージ	MSG_LANG	表示するメッセージの言語種を設定する	単一	省略可能	○	－
	MSG_PRINT	SQL文実行時にエラーメッセージを表示するか否か	単一	省略可能	○	－
リカバリ	RCV_MODE	アプリケーションのリカバリ水準を指定する	単一	省略可能	○ (注)	○ (注)
予約語とSQL機能	SQL_LEVEL	アプリケーションの予約語とSQL機能のレベルを設定する	単一	省略可能	○	○
並列クエリ	MAX_PARALLEL	データベースを並列に検索する場合の多重度	単一	省略可能	○ (注)	○ (注)
	PARALLEL_SCAN	アプリケーション単位またはコネクション単位に、データベースを並列に検索するか否か	単一	省略可能	○ (注)	○ (注)
その他	ALTER_CHECK	表の設計変更により表の列を変更した場合、アプリケーションへの影響をチェックするか否か	単一	省略可能	○ (注)	－
	ARC_FULL	アーカイブログ満杯時にエラー復帰するか否か	単一	省略可能	○ (注)	○ (注)

分類	実行パラメタ	概要	記述数	記述の省略	埋込みSQL	JDBC
	S L SIGNAL_INF	シグナルをアプリケーションで利用するか否か	単一	省略可能	○	－

埋込みSQL : 埋込みSQLを利用したアプリケーションにおいてパラメタが有効になるか否かを表します。

JDBC : JDBCドライバを利用したアプリケーションにおいてパラメタが有効になるか否かを表します。

○:指定可

－:指定不可

注)データベース簡単運用の場合は、指定できません。

B.3 サーバ用の動作環境ファイル

サーバ用の動作環境ファイルはコネクション接続時に読み込まれます。

サーバ用の動作環境ファイルのパラメタをチューニングする場合は、サーバ用の動作環境ファイルを編集後、コネクションを再接続する必要があります。

ここでは、サーバ用の動作環境ファイルを定義して、アプリケーション中のコネクション単位に動作環境を設定する方法について説明します。

参照

- パラメタの記述形式および意味については、“アプリケーション開発ガイド(埋込みSQL編)”の“クライアント用の動作環境ファイルによる定義”を参照してください。
- アクセスプランおよび性能情報に関する実行パラメタの詳細については、“SQLTOOLユーザーズガイド”を参照してください。

コネクション単位に動作環境を変更するには、まず、汎用エディタを利用して、実行パラメタを設定します。ファイル形式は、テキスト形式です。次に、クライアント用の動作環境ファイルのSERVER_ENV_FILEパラメタで、サーバ用の動作環境ファイル名を指定します。

サーバ用の動作環境ファイルにおいて実行パラメタを省略した場合は、“表B.1 動作環境ファイルのパラメタ一覧”の優先順位に従い設定されます。

サーバ用の動作環境ファイルに指定できるパラメタを以下に示します。

表B.4 サーバ用の動作環境ファイルの実行パラメタの種類

分類	実行パラメタ	概要	記述数	記述の省略
通信	TRAN_TIME_LIMIT	1つのトランザクションの最大使用可能時間	単一	省略可能
作業領域	SORT_MEM_SIZE	作業用ソート領域として使うメモリサイズ	単一	省略可能
	WORK_ALLOC_SPACESIZE	作業用テーブルおよび作業用ソート領域として使用するファイルサイズ	単一	省略可能
	WORK_MEM_SIZE	作業用テーブルとして使うメモリのサイズ	単一	省略可能
	WORK_PATH	作業用テーブルおよび作業用ソート領域のパス	単一	省略可能

分類	実行パラメタ	概要	記述数	記述の省略
データ処理	SURROGATE_PAIR_NUMBER	UNICODEの補助文字(1～16面の4バイト文字)の文字数	単一	省略可能
表・インデックス	DEFAULT_INDEX_SIZE(注)	格納構造定義を行わないインデックスを作成する場合のインデックスのデータ格納域の初期量、拡張量、ページ長など	単一	省略可能
	DEFAULT_OBJECT_TABLE_SIZE(注)	格納構造定義を行わない表を作成する場合のOBJECT構造の表のデータ格納域の初期量、拡張量、ページ長など	単一	省略可能
	DEFAULT_TABLE_SIZE(注)	格納構造定義を行わない表を作成する場合の表のデータ格納域の初期量、拡張量、ページ長など	単一	省略可能
	INCLUSION_DSI(注)	アプリケーションで使用するDSIを限定する	単一	省略可能
	TEMPORARY_INDEX_SIZE(注)	一時表にインデックスを定義する場合のインデックスのデータ格納域の初期量、拡張量など	単一	省略可能
	TEMPORARY_TABLE_SIZE(注)	一時表を定義する場合の表のデータ格納域の初期量、拡張量など	単一	省略可能
排他	DSO_LOCK(注)	使用するDSOの占有の単位、占有モード	単一	省略可能
アクセスプラン・性能情報	CHOOSE_TID_UNION	WHERE句にブール演算子”OR”、または、行値構成子を指定した場合、TIDユニオンマージのアクセスプランのみを作成するか否か	単一	省略可能
	GROUP_COL_COND_MOVE	導出表を絞り込む探索条件を指定した場合、その探索条件を導出表のWHERE句に移動するか否か	単一	省略可能
	IGNORE_INDEX	インデックスを使用しないアクセスプランを選択するか否か	単一	省略可能
	INACTIVE_INDEX_SCAN	非活性状態のインデックスDSIを含むインデックスを使用したアクセスプランを選択するか否か	単一	省略可能
	JOIN_ORDER	結合表と他の表のジョイン順	単一	省略可能
	JOIN_RULE	ジョインする方法	単一	省略可能
	MAX_SCAN_RANGE	インデックス、クラスタキー、または分割キーの検索範囲の最大数	単一	省略可能
	SAME_COST_JOIN_ORDER	最適化情報を設定していない場合、ジョイン順をV5以前と同じにするか否か	単一	省略可能
	SCAN_KEY_ARITHMETIC_RANGE	四則演算の検索範囲について、インデックスの範囲検索またはクラスタキーの検索を行うか否か	単一	省略可能

分類	実行パラメタ	概要	記述数	記述の省略
	SCAN_KEY_CAST	探索条件のCASTオペランドに指定した列でインデックスの範囲検索、または、クラスタキー検索を行うか否か	単一	省略可能
	SORT_HASHAREA_SIZE	ソート処理がレコードをハッシングして格納するための領域サイズ	単一	省略可能
	SS_RATE	述語ごとの検索範囲の選択率の値	単一	省略可能
	TID_SORT	インデックス検索と表データ取得のアクセスモデルでTIDソートを利用するか否か	単一	省略可能
	TID_UNION	TIDユニオンマージのアクセスモデルを有効にするか否か	単一	省略可能
	USQL_LOCK	UPDATE文:探索またはDELETE文:探索の更新標的レコードを位置づける部分の占有モード	単一	省略可能
並列クエリ	MAX_PARALLEL(注)	データベースを並列に検索する場合の多重度	単一	省略可能
	PARALLEL_SCAN(注)	アプリケーション単位またはコネクション単位に、データベースを並列に検索するか否か	単一	省略可能

注)データベース簡単運用の場合は、指定できません。

付録C　資源の見積り

アプリケーション開発に関連する資源の見積りについて説明します。

C.1　ソート作業域の見積り

Symfoware/RDBは、データ操作中に中間結果の保存が必要になると、作業用テーブルにデータを保存します。また、データのソート処理が必要な場合は、作業用ソート領域を使用します。作業用テーブルや作業用ソート領域には、一定量までは仮想記憶を使用しますが、それを超えると、ハードディスク上の領域を使用します。この領域をソート作業域と呼びます。ソート作業域には、利用者が獲得した作業用のディレクトリを指定します。

ソート作業域には、共用ソート作業域と固有ソート作業域の2つがあります。共用ソート作業域は、Symfoware/RDB内のアプリケーションで共用するソート作業域であり、システム用の動作環境ファイルで指定します。固有ソート作業域は、特定のアプリケーションが使用するソート作業域であり、クライアント用の動作環境ファイルで指定します。固有ソート作業域を指定しない場合は、共用ソート作業域が使用されます。固有ソート作業域を指定した場合は、固有ソート作業域が使用され、共用ソート作業域は使用されません。

アプリケーション実行時にソート作業域が不足した場合は、アプリケーションのエラーとなります。

C.1.1　共用ソート作業域

Symfoware/RDB内のアプリケーションで共用するソート作業域を、システム用の動作環境ファイルで指定します。

参考

データベース簡単運用の場合は、Symfoware Severのインストール時に、システムの動作環境として最適なディレクトリパス名が設定されるため、システム用の動作環境ファイルでの設定は不要です。

ファイル容量

共用ソート作業域として必要なファイル容量を正確に見積もることは困難ですが、同時に実行する各アプリケーションで使用するソート作業域容量の総計を、参考にしてください。

```
ΣPi　　Pi＝1アプリケーション中のSQL文で使用する検索結果データ量の
　　　　大きさの2倍以上
```

検索結果データ量は、検索結果に含まれる列の定義長で見積りを行ってください。ただし、検索結果にBLOB型列や可変長文字列が含まれる場合、以下の値で見積りを行ってください。

32キロバイト未満で定義されたBLOB型列、および可変長文字列の場合：定義長

32キロバイト以上のBLOB型列かつ、32ビット製品をご使用の場合：4バイト

32キロバイト以上のBLOB型列かつ、64ビット製品をご使用の場合：8バイト

指定方法

動作環境ファイルのパラメタWORK_PATHにソート作業域のディレクトリパス名を指定します。動作環境ファイルのパラメタの詳細は、“付録B 動作環境ファイルのパラメタ一覧”を参照してください。

```
WORK_PATH ＝（パス名 [, パス名]・・・）
```

C.1.2　固有ソート作業域

個々のアプリケーションが使用する固有ソート作業域を、クライアント用の動作環境ファイルで指定します。

固定ソート作業域は、大量データ処理で大量にソート作業域を使用する場合に、一時作業領域をクライアント用の動作環境ファイルで指定するものです。

ファイル容量

固有ソート作業域として必要なファイル容量は、対象となるアプリケーション中のSQL文で使用する検索結果データ量の大きさの2倍以上になります。

```
ファイル容量＝1アプリケーション中のSQL文で使用する検索結果データ量
              の大きさの2倍以上
```

検索結果データ量は、検索結果に含まれる列の定義長で見積りを行ってください。ただし、検索結果にBLOB型列や可変長文字列が含まれる場合、以下の値で見積りを行ってください。

32キロバイト未満で定義されたBLOB型列、および可変長文字列の場合:定義長

32キロバイト以上のBLOB型列かつ、32ビット製品をご使用の場合:4バイト

32キロバイト以上のBLOB型列かつ、64ビット製品をご使用の場合::8バイト

指定方法

動作環境ファイルのパラメタWORK_PATHにソート作業域のディレクトリパス名を指定します。動作環境ファイルのパラメタの詳細は、“付録B 動作環境ファイルのパラメタ一覧”を参照してください。

```
WORK_PATH＝（パス名 [, パス名]・・・）
```

C.2 メモリの見積り式

アプリケーションの実行時に、Symfoware/RDBのライブラリが使用するメモリの見積り式について説明します。

C.2.1 アプリケーション実行時のメモリ見積り

アプリケーションの実行時に、Symfoware/RDBのライブラリが使用するメモリ量の概算は、以下の式で見積もってください。

```
アプリケーション実行時の使用メモリ量 ＝ Σセション単位の使用メモリ量

  セション単位の使用メモリ量 ＝ 30KB
                              ＋ 通信に利用するバッファサイズ
                              ＋ 一括FETCHを行う場合のバッファサイズ
                              ＋ SQL文を実行する際の情報の保持域サイズ
                              ＋ ΣSQL記述子域サイズ

  通信に利用するバッファサイズ
    ＝ 動作環境パラメタBUFFER_SIZEに指定したバッファサイズ
      （1列に対する列のデータ長がこのサイズを超えた場合、自動的に拡張されます）

  一括FETCHを行う場合のバッファサイズ
    ＝ 動作環境パラメタRESULT_BUFFERに指定したバッファサイズ ×
                        動作環境パラメタRESULT_BUFFERに指定したバッファの個数

  SQL文を実行する際の情報の保持域サイズ
    ＝ 動作環境パラメタMAX_SQLに指定した数×0.5KB
    ＋ 準備中の動的SQL文の中で使用している動的パラメタおよび選択リストの数×0.1KB

  SQL記述子域サイズ(注1)
    ＝ ALLOCATE DESCRIPTOR文で指定したWITH MAX値
      （省略時は動作環境パラメタDESCRIPTOR_SPECに指定した値） ×0.1KB
    ＋ Σ各項目記述子域に設定されたデータの合計サイズ
```

備考.動作環境パラメタは、クライアント用の動作環境ファイルの実行パラメタのことです。

注1) Java連携で、バッチ更新機能を使用する場合には、SQL記述子域サイズを以下の式で見積もってください。

```
SQL記述子域サイズ
    ＝ ALLOCATE DESCRIPTOR文で指定したWITH MAX値
       （省略時は動作環境パラメタDESCRIPTOR_SPECに指定した値） ×0.1KB
    ＋ Σ各項目記述子域に設定されたデータの合計サイズ(注2)×addBatch()の実行回数×3B
```

注2) バッチ更新で指定したSQL文中で、連結演算子(||)を使用する場合は、32002バイトとなります。

参照

セションの詳細については、“アプリケーション開発ガイド(埋込みSQL編)”の“マルチスレッド環境におけるデータ操作”を参照してください。

C.3 ファイルディスクリプタ数の見積り

アプリケーション実行時に使用するファイルディスクリプタ数について、ファイルディスクリプタ数のデフォルト値を超える場合は、以下の見積り式を使用して、OSのファイルディスクリプタ数の設定を行ってください。

Connection Managerを使用してデータベースに接続している場合

```
初回のデータベース接続時 ..... +5
2回目以降のデータベース接続時 ..... +3
データベースへの接続の切断時 ..... -3
```

Connection Managerを使用せずにデータベースに接続している場合

```
初回のデータベース接続時 ..... +3
2回目以降のデータベース接続時 ..... +1
データベースへの接続の切断時 ..... -1
```

データベースに接続すると、使用するファイルディスクリプタ数が増えます。

データベースへの接続を切断すると、ファイルディスクリプタ数が減ります。

ただし、コネクションプーリング機能によって、コネクションが切断されずにプールされている場合は、ファイルディスクリプタ数は減りません。

アプリケーションプロセス起動後、初めてデータベースに接続した時、同じアプリケーションプロセス内での、2回目以降の接続と比較して、ファイルディスクリプタ数を、以後の操作よりも2つ多く使用します。この2つ分のファイルディスクリプタは、アプリケーションプロセス終了まで継続して使用します。

付録D　アプリケーションおよびRDBコマンド間の排他

データベースの同一資源に対して、アプリケーションおよびRDBコマンドが同時に動作した場合に、それぞれの処理を矛盾なく実行するために、Symfoware/RDBが排他制御を行います。

表D.1 Symfoware/RDBの各機能の占有の資源と占有の強さ(ディクショナリ資源の占有)および表D.3 Symfoware/RDBの各機能の占有の資源と占有の強さ(データベース資源の占有)に、各機能が排他する資源と排他の強さを示します。

また、表D.2 占有の強さの関係に排他の強さの関係を示します。

各機能はオプションや入力データで指定された資源を排他して処理を行います。資源を排他する期間は次のとおりです。

- アプリケーション

  ディクショナリ資源:

  　アプリケーションの最初のSQL文発行時点から、アプリケーションの終了(コネクション切断)まで

  データベース資源:

  　トランザクションの開始から終了まで

- RDBコマンド

  ディクショナリ資源:

  　コマンド投入時点から応答メッセージの出力まで

  データベース資源:

  　トランザクションの開始から終了まで

資源の占有を行うことによって、各機能を同時に実行した場合、資源の占有エラーとなったり、資源の占有が解除されるまで待ちに入ったりする場合があります。

同一資源に対して各機能を同時に実行した場合の振る舞いについて、表D.4 Symfoware/RDBの同一資源に対する各機能の同時実行時の振る舞いに示します。ただし、以下の処理については、資源の占有に関係なく同一資源に対して他の機能と同時に実行することはできません。

- データベースのリカバリ(rdbrcvコマンド、rdbadjrcvコマンド、rdbexspcコマンド、rdbxarcvコマンド、rdbmrrcvコマンド)
- 監査ログデータベースのリカバリ(rdbrcvadtコマンド)
- ディクショナリ操作(rdbcrdicコマンド、rdbaldicコマンド、rdbgcdicコマンド、Symfoware/RDB停止中のrdbrcvdicコマンド、rdbscldirコマンド)

rdbdmpdicコマンド、rdbprdicコマンドおよびSymfoware/RDB運用中のrdbrcvdicコマンドは、他の機能と同時実行可能ですが、定義操作との同時実行においては占有エラーとなる場合があります。

rdbbackupコマンドは、同一RDBシステムに対して同時に実行することはできません。

データベース簡単運用の場合のデータベースのリカバリは、同一RDBシステムに対して同時に実行することはできません。また、他の機能と同時に実行することも出来ません。

表D.1 Symfoware/RDBの各機能の占有の資源と占有の強さ(ディクショナリ資源の占有)

機能	コマンド		占有資源													
			論理						格納					物理	利用者認可識別子	ROLE
			DB	SCH	SEQ	TBL	RTN	TRG	DSO	DSO(IX)	DSI	DSI(IX)	SCOP	DBS		
データベース定義	rdbddlex	CREATE DATABASE	EX	－	－	－	－	－	－	－	－	－	－	－	－	－
データベース消去	rdbddlex	DROP DATABASE	EX	－	－	－	－	－	－	－	－	－	－	－	－	－

機能	コマンド		占有資源													
			論理						格納					物理	利用者認可識別子	ROLE
			DB	SCH	SEQ	TBL	RTN	TRG	DSO	DSO (IX)	DSI	DSI (IX)	SCOP	DBS		
スキーマ定義	rdbddlex	CREATE SCHEMA	SH	EX	—	—	—	—	—	—	—	—	—	—	—	—
スキーマ消去	rdbddlex	DROP SCHEMA (CASCADEなし)	SH	EX	—	—	—	—	—	—	—	—	—	—	—	—
		DROP SCHEMA (CASCADEあり)	SH	EX	—	EX	EX	EX	EX	EX	EX	EX	EX	SH	EX	—
順序定義	rdbddlex	CREATE SEQUENCE	SH	SH	EX	—	—	—	—	—	—	—	—	—	—	—
順序削除	rdbddlex	DROP SEQUENCE	SH	SH	EX	—	—	—	—	—	—	—	—	—	—	—
表定義	rdbddlex	CREATE TABLE	SH	SH	—	EX	—	—	—	—	—	—	—	—	—	—
表消去	rdbddlex	DROP TABLE (CASCADEなし)	SH	SH	—	EX	—	—	—	—	—	—	—	—	—	—
		DROP TABLE (CASCADEあり)	SH	SH	—	EX	EX	EX	EX	EX	EX	EX	EX	SH	EX	—
ビュー定義	rdbddlex	CREATE VIEW	SH	SH	—	EX	—	—	—	—	—	—	—	—	—	—
ビュー消去	rdbddlex	DROP VIEW	SH	SH	—	EX	—	—	—	—	—	—	—	—	—	—
ルーチン定義	rdbddlex	CREATE PROCEDURE	SH	SH	—	—	EX	—	—	—	—	—	—	—	—	—
		CREATE FUNCTION	SH	SH	—	—	EX	—	—	—	—	—	—	—	—	—
ルーチン削除	rdbddlex	DROP PROCEDURE	SH	SH	—	—	EX	—	—	—	—	—	—	—	—	—
		DROP FUNCTION	SH	SH	—	—	EX	—	—	—	—	—	—	—	—	—
トリガ定義	rdbddlex	CREATE TRIGGER	SH	SH	—	EX/SH (注1)	—	EX	—	—	—	—	—	—	—	—
トリガ削除	rdbddlex	DROP TRIGGER	SH	SH	—	EX/SH (注1)	—	EX	—	—	—	—	—	—	—	—

機能	コマンド		占有資源													
			論理						格納					物理	利用者認可識別子	ROLE
			DB	SCH	SEQ	TBL	RTN	TRG	DSO	DSO (IX)	DSI	DSI (IX)	SCOP	DBS		
表定義変更	rdbddlex	ALTER TABLE	SH	SH	—	EX (注2)	—	—	EX (注2)	EX (注2)	EX (注2)	EX (注2)	—	—	—	—
表交換	rdbddlex	SWAP TABLE	SH	SH	—	EX	—	—	EX	EX	EX	EX	—	—	—	—
格納構造定義	rdbddlex	CREATE DSO	SH	SH	—	EX	—	—	EX	—	—	—	—	—	—	—
		CREATE DSO (IX)	SH	SH	—	EX	—	—	EX	EX	EX	—	—	—	—	—
		CREATE DSI	SH	SH	—	EX (注3)	—	—	EX (注3)	—	EX	—	EX (注3)	SH	EX (注3)	—
		CREATE DSI (IX)	SH	SH	—	EX (注3)	—	—	EX (注3)	EX (注3)	SH	EX	—	SH	—	—
		CREATE SCOPE	SH	—	—	—	—	—	—	—	EX	—	EX	—	—	—
格納構造消去	rdbddlex	DROP DSO (CASCADE なし)	SH	SH	—	EX	—	—	EX	—	—	—	—	—	—	—
		DROP DSO (CASCADE あり)	SH	SH	—	EX	—	—	EX	EX	EX	EX	EX	SH	EX	—
		DROP DSO (IX) (CASCADE なし)	SH	SH	—	EX	—	—	EX	EX	—	—	—	—	—	—
		DROP DSO (IX) (CASCADE あり)	SH	SH	—	EX	—	—	EX	EX	—	EX	—	SH	—	—
		DROP DSI (CASCADE なし)(注4)	SH	SH	—	EX	—	—	EX	—	EX	—	EX	SH	EX	—
		DROP DSI (CASCADE あり)(注4)	SH	SH	—	EX	—	—	EX	—	EX	EX	EX	SH	EX	—
		DROP DSI (IX)(注4)	SH	SH	—	EX	—	—	EX	EX	SH	EX	EX	SH	EX	—
		DROP SCOPE	SH	—	—	—	—	—	—	—	—	—	EX	—	EX	—

機能	コマンド		占有資源													
			論理						格納					物理	利用者認可識別子	ROLE
			DB	SCH	SEQ	TBL	RTN	TRG	DSO	DSO (IX)	DSI	DSI (IX)	SCOP	DBS		
DSI変更	rdbddlex	ALTER DSI	SH	SH	—	EX (注5)	—	—	EX (注5)	—	EX (注5)	—	—	SH	—	—
スコープ適用	rdbddlex	APPLY SCOPE	SH	—	—	—	—	—	—	—	—	—	EX	—	EX	—
スコープ解除	rdbddlex	RELEASE SCOPE	SH	—	—	—	—	—	—	—	—	—	EX	—	EX	—
スペース定義	rdbddlex	CREATE DBS	SH	—	—	—	—	—	—	—	—	—	—	EX	—	—
スペース消去	rdbddlex	DROP DBS	SH	—	—	—	—	—	—	—	—	—	—	EX	—	—
権限情報定義	rdbddlex	GRANT (SCH)	SH	MOD	—	—	—	—	—	—	—	—	—	—	SH	MOD
		GRANT (TBL)	SH	SH	—	MOD	—	—	—	—	—	—	—	—	SH	MOD
		GRANT (DBS)	SH	—	—	—	—	—	—	—	—	—	—	MOD	SH	MOD
		GRANT (ROUTINE)	SH	SH	—	—	MOD	—	—	—	—	—	—	—	SH	MOD
		GRANT (TRG)	SH	SH	—	—	—	MOD	—	—	—	—	—	—	SH	MOD
		GRANT (SEQUENCE)	SH	SH	MOD	—	—	—	—	—	—	—	—	—	SH	MOD
		GRANT(ROLE)	—	—	—	—	—	—	—	—	—	—	—	—	SH	MOD
権限情報削除	rdbddlex	REVOKE (SCH)	SH	MOD	—	—	—	—	—	—	—	—	—	—	SH	MOD
		REVOKE (TBL)	SH	SH	—	MOD	—	—	—	—	—	—	—	—	SH	MOD
		REVOKE (DBS)	SH	—	—	—	—	—	—	—	—	—	—	MOD	SH	MOD

機能	コマンド		占有資源													
			論理						格納					物理	利用者認可識別子	ROLE
			DB	SCH	SEQ	TBL	RTN	TRG	DSO	DSO (IX)	DSI	DSI (IX)	SCOP	DBS		
		REVOKE (ROUTINE)	SH	SH	—	—	MOD	—	—	—	—	—	—	—	SH	MOD
		REVOKE (TRG)	SH	SH	—	—	—	MOD	—	—	—	—	—	—	SH	MOD
		REVOKE (SEQUENCE)	SH	SH	MOD	—	—	—	—	—	—	—	—	—	SH	MOD
		REVOKE(ROLE)	—	—	—	—	—	—	—	—	—	—	—	—	SH	MOD
利用者定義	rdbddlex	CREATE USER	—	—	—	—	—	—	—	—	—	—	—	—	EX	—
利用者変更	rdbddlex	ALTER USER	—	—	—	—	—	—	—	—	—	—	—	—	MOD	—
利用者削除	rdbddlex	DROP USER	—	—	—	—	—	—	—	—	—	—	—	—	EX	—
ロール定義	rdbddlex	CREATE ROLE	—	—	—	—	—	—	—	—	—	—	—	—	—	EX
ロール削除	rdbddlex	DROP ROLE	—	—	—	—	—	—	—	—	—	—	—	—	—	EX
パラメタ設定	rdbddlex	SET SYSTEM PARAMETER	—	—	—	—	—	—	—	—	—	—	—	—	SH	MOD
最適化情報の設定	rdbddlex	SET STATISTICS (TBL)	SH	SH	—	SH	—	—	MOD	MOD	MOD	MOD	—	SH	—	—
		SET STATISTICS (DSO IX)	SH	SH	—	SH	—	—	—	MOD	—	SH	—	SH	—	—
		SET STATISTICS (DSI)	SH	SH	—	SH	—	—	—	—	MOD	—	—	SH	—	—
		SET STATISTICS (DSI IX)	SH	SH	—	SH	—	—	—	—	—	MOD	—	SH	—	—
最適化情報の表示	rdbddlex	PRINT STATISTICS (TBL)	SH	SH	—	SH	—	—	SH	SH	SH	SH	—	SH	—	—

機能	コマンド		占有資源													
			論理						格納					物理	利用者認可識別子	ROLE
			DB	SCH	SEQ	TBL	RTN	TRG	DSO	DSO (IX)	DSI	DSI (IX)	SCOP	DBS		
		PRINT STATISTICS (DSO IX)	SH	SH	—	SH	—	—	—	SH	—	SH	—	SH	—	—
		PRINT STATISTICS (DSI)	SH	SH	—	SH	—	—	—	—	SH	—	—	SH	—	—
		PRINT STATISTICS (DSI IX)	SH	SH	—	SH	—	—	—	—	—	SH	—	SH	—	—
一時表定義	rdbddlex	CREATE GLOBAL TEMPORARY TABLE	SH	SH	—	EX	—	—	—	—	—	—	—	SH	—	—
一時表インデックス定義	rdbddlex	CREATE INDEX	SH	SH	—	EX	—	—	—	—	—	—	—	SH	—	—
DSI初期化	rdbfmt	DSI	SH	SH	—	SH	—	—	—	—	SH	SH	—	SH	—	—
		DSI (IX)	SH	SH	—	SH	—	—	—	—	SH	SH	—	SH	—	—
データロード	rdbsloader	DSI	SH	SH	—	SH	—	—	SH	SH	SH	SH	—	SH	—	—
		DSI (IX)	SH	SH	—	SH	—	—	SH	SH	SH	SH	—	SH	—	—
データアンロード	rdbunl	DSI	SH	SH	—	SH	—	—	—	—	SH	—	—	SH	—	—
	rdbunlx	DSI	SH	SH	—	SH	—	—	—	—	SH	—	—	SH	—	—
	rdbunlsql		SH	SH	—	SH	—	—	SH	SH	SH	SH	SH	—	SH	—
データベース再編成	rdbgcdsi	DSI	SH	—	—	—	—	—	—	—	SH	SH	—	—	—	—
		DSI (IX)	SH	—	—	—	—	—	—	—	SH	SH	—	—	—	—
データベースの更新	rdbupt		SH	SH	—	SH	—	—	SH	SH	SH	SH	SH	—	SH	—
	rdbsaloader		SH	SH	—	SH	—	—	SH	SH	SH	SH	—	SH	—	—
	rdbsuloader		SH	SH	—	SH	—	—	SH	SH	SH	SH	—	SH	—	—
自動DSI領域拡張	rdbalmdsi	DSI	SH	—	—	—	—	—	—	—	SH	—	—	—	—	—

機能	コマンド		占有資源													
			論理						格納					物理	利用者認可識別子	ROLE
			DB	SCH	SEQ	TBL	RTN	TRG	DSO	DSO (IX)	DSI	DSI (IX)	SCOP	DBS		
		DSI (IX)	SH	―	―	―	―	―	―	―	―	SH	―	―	―	―
DSIの除外	rdbexdsi	DSI	―	―	―	―	―	―	―	―	SH	―	―	―	―	―
最適化情報更新	rdbups	TBL	SH	SH	―	SH	―	―	MOD	MOD	MOD	MOD	―	SH	―	―
		DSO	SH	SH	―	SH	―	―	MOD	―	SH	―	―	SH	―	―
		DSO (IX)	SH	SH	―	SH	―	―	―	MOD	―	SH	―	SH	―	―
		DSI	SH	SH	―	SH	―	―	―	―	MOD	―	―	SH	―	―
		DSI (IX)	SH	SH	―	SH	―	―	―	―	―	MOD	―	SH	―	―
管理情報印刷	rdbprt	定義情報	SH	SH	―	SH	―	―	―	―	―	―	―	―	―	―
スペース情報印刷	rdbspcinf		SH	―	―	―	―	―	―	―	SH	SH	―	SH	―	―
DSI使用状況出力	rdbudsi	DSI	SH	―	―	―	―	―	―	―	SH	―	―	―	―	―
		DSI (IX)	SH	―	―	―	―	―	―	―	―	SH	―	―	―	―
退避データ作成	rdbdmp	DSI指定	―	―	―	―	―	―	―	―	SH	―	―	―	―	―
データベースリカバリ	rdbrcv	DSI指定	―	―	―	―	―	―	―	―	MOD	―	―	―	―	―
		DBS指定	―	―	―	―	―	―	―	―	MOD	―	―	SH	―	―
	rdbadjrcv		―	―	―	―	―	―	―	―	―	―	―	―	―	―
一括バックアップ	rdbbackup		―	―	―	―	―	―	―	―	SH(注7)	―	―	―	―	―
アーカイブ管理	rdblog		―	―	―	―	―	―	―	―	―	―	―	―	―	―

機能	コマンド		占有資源													
			論理						格納					物理	利用者認可識別子	ROLE
			DB	SCH	SEQ	TBL	RTN	TRG	DSO	DSO (IX)	DSI	DSI (IX)	SCOP	DBS		
運用情報取得	rdbinf	DIC指定	SH	—	—	—	—	—	—	—	SH	SH	—	—	—	—
		DSI指定	—	—	—	—	—	—	—	—	SH	SH	—	—	—	—
		DBS指定	SH	—	—	—	—	—	—	—	SH	SH	—	SH	—	—
実行情報表示	rdbps		—	—	—	—	—	—	—	—	—	—	—	—	—	—
資源占有情報表示	rdblkinf		—	—	—	—	—	—	—	—	—	—	—	—	—	—
退避媒体情報表示	rdbdvinf		—	—	—	—	—	—	—	—	—	—	—	—	—	—
利用規定設定	rdbrtr	DIC指定	—	—	—	—	—	—	—	—	SH	SH	—	—	—	—
		DSI指定	—	—	—	—	—	—	—	—	SH	SH	—	—	—	—
		DBS指定	—	—	—	—	—	—	—	—	SH	SH	—	SH	—	—
利用規定解除	rdbrls	DIC指定	—	—	—	—	—	—	—	—	SH	SH	—	—	—	—
		DSI指定	—	—	—	—	—	—	—	—	SH	SH	—	—	—	—
		DBS指定	—	—	—	—	—	—	—	—	SH	SH	—	SH	—	—
アクセス禁止設定	rdbinh	DSI指定	—	—	—	—	—	—	—	—	SH	SH	—	—	—	—
		DBS指定	—	—	—	—	—	—	—	—	SH	SH	—	SH	—	—
アクセス禁止解除	rdbpmt	DSI指定	—	—	—	—	—	—	—	—	SH	SH	—	—	—	—
		DBS指定	—	—	—	—	—	—	—	—	SH	SH	—	SH	—	—
プリプロセッサ	sqlpc/sqlpcob		—	—	—	—	—	—	—	—	—	—	—	—	—	—
データベーススペースの切離しおよび接続	rdbexspc	mdetach	—	—	—	—	—	—	—	—	—	—	—	MOD	—	—
		mattach	—	—	—	—	—	—	—	—	—	—	—	MOD	—	—
		mp	—	—	—	—	—	—	—	—	—	—	—	SH	—	—

機能	コマンド		占有資源													
			論理						格納					物理	利用者認可識別子	ROLE
			DB	SCH	SEQ	TBL	RTN	TRG	DSO	DSO (IX)	DSI	DSI (IX)	SCOP	DBS		
RDBディレクトリファイルの切離しおよび接続	rdbexdir		―	―	―	―	―	―	―	―	―	―	―	―	―	―
監査ログデータの退避	rdbunladt	-mb -mc	SH	SH	―	SH	―	―	―	―	SH	―	―	SH	―	―
監査ログエレメントの初期化	rdbunladt	-mb -mf	SH	SH	―	SH	―	―	―	―	SH	―	―	SH	―	―
監査ログデータベース定義	rdbaudit	-c	EX	EX	―	EX	―	―	EX	EX	EX	EX	―	EX	―	―
監査ログデータベース削除	rdbaudit	-d	EX	EX	―	EX	―	―	EX	EX	EX	EX	―	EX	―	―
監査ログエレメントの追加	rdbaudit	-a	SH	SH	―	EX	―	―	EX	EX	EX	EX	―	EX	―	―
監査ログエレメントの切替	rdbaudit	-e	―	―	―	―	―	―	―	―	MOD	MOD	―	MOD	―	―
監査ログエレメントの表示	rdbaudit	-v	SH	―	―	―	―	―	―	―	SH	SH	―	SH	―	―
監査ログエレメントのリカバリ	rdbrcvadt		―	―	―	―	―	―	―	―	MOD	―	―	SH	―	―
アプリケーション(動的SQL文あり)	―		SH	SH	SH	SH	SH	SH	SH	SH	SH	SH	SH	―	SH	SH
アプリケーション(動的SQL文なし)	―		SH	SH	SH	SH	SH	SH	SH	SH	SH	SH	SH	―	SH	SH
RDBの起動	rdbstart		―	―	―	―	―	―	―	―	―	―	―	―	―	―
RDBの停止	rdbstop		―	―	―	―	―	―	―	―	―	―	―	―	―	―
RDBの動作確認	rdbchksanity		―	―	―	―	―	―	―	―	―	―	―	―	―	―
共用バッファプール 開設	rdbcrbf		―	―	―	―	―	―	―	―	―	―	―	―	―	―
共用バッファプール 閉鎖	rdbdrbf		―	―	―	―	―	―	―	―	―	―	―	―	―	―
共用バッファプール 対応登録	rdbconbf		SH	―	―	―	―	―	―	―	SH	SH	―	―	―	―

機能		コマンド	占有資源													
			論理						格納					物理	利用者認可識別子	ROLE
			DB	SCH	SEQ	TBL	RTN	TRG	DSO	DSO (IX)	DSI	DSI (IX)	SCOP	DBS		
	対応解除	rdbdisbf	SH	—	—	—	—	—	—	—	SH	SH	—	—	—	—
	表示	rdbprtbf	—	—	—	—	—	—	—	—	—	—	—	—	—	—
DSIの常駐/常駐解除		rdbresident	SH	—	—	—	—	—	—	—	SH	SH	—	—	—	—
インデックス縮退		rdbalidx	SH	—	—	—	—	—	—	—	—	EX	—	—	—	—
定義情報のメモリ展開		rdbpldic	SH	SH	—	SH	—	—	—	—	—	—	—	—	—	—
RDBディクショナリの作成		rdbcrdic	—	—	—	—	—	—	—	—	—	—	—	—	—	—
RDBディクショナリの容量拡張		rdbaldic	—	—	—	—	—	—	—	—	—	—	—	—	—	—
RDBディクショナリの再配置		rdbgcdic	—	—	—	—	—	—	—	—	—	—	—	—	—	—
RDBディクショナリの領域の状況表示		rdbprdic	—	—	—	—	—	—	—	—	—	—	—	—	—	—
RDBディクショナリのリカバリ		rdbrcvdic	—	—	—	—	—	—	—	—	—	—	—	—	—	—
RDBディクショナリの退避データ作成		rdbdmpdic	—	—	—	—	—	—	—	—	—	—	—	—	—	—
性能監視		rdbsar	—	—	—	—	—	—	—	—	—	—	—	—	—	—
システムの性能情報の表示		rdbpmreport	—	—	—	—	—	—	—	—	—	—	—	—	—	—
SQL文、アクセスプランの表示		rdbpmsqllist	—	—	—	—	—	—	—	—	—	—	—	—	—	—
パフォーマンスモニタの環境設定		rdbpmsetup	—	—	—	—	—	—	—	—	—	—	—	—	—	—
リカバリポイントの設定		rdbsetrp	—	—	—	—	—	—	—	—	—	—	—	—	—	—
コネクションの回収リカ		rdbterm	—	—	—	—	—	—	—	—	—	—	—	—	SH	—

機能	コマンド	占有資源													
		論理						格納					物理	利用者認可識別子	ROLE
		DB	SCH	SEQ	TBL	RTN	TRG	DSO	DSO (IX)	DSI	DSI (IX)	SCOP	DBS		
バリ・状態の表示	rdbcninf	－	－	－	－	－	－	－	－	－	－	－	－	－	－
SQL文の実行	rdbexecsql	S H	S H	－	S H	S H	S H	S H	S H	S H	S H	S H	－	S H	－
XA配下のリカバリ	rdbxarcv	－	－	－	－	－	－	－	－	－	－	－	－	－	－
XA配下のXID一覧表示	rdbprxid	－	－	－	－	－	－	－	－	－	－	－	－	－	－
RDBメッセージの表示(注6)	rdbprtmsg	－	－	－	－	－	－	－	－	－	－	－	－	－	－
リソースの登録、変更、表示、削除	rdbhsrsc	－	－	－	－	－	－	－	－	－	－	－	－	－	－
	rdbpcrsc	－	－	－	－	－	－	－	－	－	－	－	－	－	－
ユーザロググループの偏在先の情報表示	rdbloggroup	－	－	－	－	－	－	－	－	－	－	－	－	－	－
ホットスタンバイの宣言・プレオープン情報出力	rdbhsuty	S H	－	－	－	－	－	－	－	S H	－	S H	－	－	－
Advanced Backup Controllerでの退避データの取得	rdbmrdmp	－	－	－	－	－	－	－	－	M O D	M O D	－	M O D	－	－
Advanced Backup Controllerでバックアップしたリカバリ制御ファイルの表示	rdbmrinf	－	－	－	－	－	－	－	－	－	－	－	－	－	－
Advanced Backup Controllerでの退避データによるリカバリ	rdbmrrcv	－	－	－	－	－	－	－	－	M O D	M O D	－	M O D	－	－
ユーザロググループ用RDBディレクトリファイルの運用	rdbscldir	－	－	－	－	－	－	－	－	－	－	－	－	－	－

機能	コマンド		占有資源													
			論理						格納					物理	利用者認可識別子	ROLE
			DB	SCH	SEQ	TBL	RTN	TRG	DSO	DSO(IX)	DSI	DSI(IX)	SCOP	DBS		
Textアダプタ定義の登録、削除、登録内容の表示	rdbtadef		—	—	—	—	—	—	—	—	—	—	—	—	—	—
Accela BizSearchインデックス作成用定義ファイルの作成	rdbtais	-r all	SH	SH	—	SH	—	—	—	—	SH	—	—	SH	—	—
		-r diff	—	—	—	—	—	—	—	—	—	—	—	—	—	—

DB : データベース

SCH : スキーマ

SEQ:順序

TBL : 実表

RTN : ルーチン

TRG : トリガ

DSO : 表のDSO

DSO(IX): インデックスのDSO

DSI : 表のDSI

DSI(IX): インデックスのDSI

SCOP : スコープ

DBS : データベーススペース

DIC : RDBディクショナリ

SH : 共有モード

MOD : 弱共有モード

EX : 非共有モード

- : 排他対象外

注1) ON句で指定した表をEXで排他します。被トリガSQL文で指定した表をSHで排他します。

注2) rdbddlexコマンドの準備モード(lオプション)で列定義を追加した場合は、MODになります。

注3) rdbddlexコマンドの準備モード(lオプション)で、分割運用を適用する表のDSIを定義した場合、またはインデックスのDSIを定義した場合は、MODになります。

注4) デフォルトバッファや共有バッファと関連付いているDSIは関連付けを解除しないと削除できません。

DSIとバッファの関連付けの解除は以下の方法で実施してください。

- 利用しているアプリケーションのコネクションを切断する。(デフォルトバッファのみ有効)
- rdbdisbufコマンドを実行する。(共有バッファのみ有効)
- Symfoware/RDBを停止・再起動する。(デフォルトバッファ・共有バッファともに有効)

注5) 動的定義変更を行うためにrdbexdsiコマンドですべての人に対してDSIが運用から除外されている場合はSHとなります。

注6) SolarisおよびLinuxで使用可能です。

注7) 必要に応じて排他にすることがあります。

表D.2 占有の強さの関係

		後続		
		EX	MOD	SH
先行	EX	×	×	×
	MOD	×	×	○
	SH	×	○	○

○:2つのプロセス間で、同一資源の多重処理が許されます。

×:2つのプロセス間で、同一資源の多重処理が許されません。

表D.3 Symfoware/RDBの各機能の占有の資源と占有の強さ(データベース資源の占有)

機能	コマンド		占有資源(注)	
			表	インデックス
アプリケーション(データベースのデータ操作)	参照処理		SH	SH
	更新処理		EX	EX
DSI初期化	rdbfmt	DSI	EX	EX
		DSI(IX)	SH	EX
データロード	rdbsloader	DSI	EX	EX
		DSI(IX)	SH	EX
データベース更新	rdbsuloader		EX	EX
	rdbsaloader		EX	EX
	rdbupt		EX	EX
データアンロード	rdbunl	DSI	SH	-
	rdbunlx	DSI	SH	-
	rdbunlsql		SH	SH
データベース再編成	rdbgcdsi	DSI	EX	EX
		DSI(IX)	SH	EX
最適化情報更新	rdbups	TBL	SH	-
		DSO	SH	-
		DSO(IX)	-	SH
		DSI	SH	-
		DSI(IX)	-	SH
監査ログデータの退避	rdbunladt	DSI	SH	-
監査ログエレメントの初期化	rdbunladt	DSI	EX	EX

注)占有資源は以下のようになります。

アプリケーションで実行した場合：動作環境ファイルの実行パラメタ(DSO_LOCKまたはR_LOCK)の設定により、DSI、ページ、レコードのいずれかになります。

ユーティリティコマンドで実行した場合：DSIになります。

表D.4 Symfoware/RDBの同一資源に対する各機能の同時実行時の振る舞い

		後行機能			
		定義操作のコマンド	データ操作のアプリケーション	データ操作のユーティリティコマンド	運用操作のコマンド
先行機能	定義操作のコマンド	× 占有待ち	× 占有エラー	× 占有エラー	× 占有エラー
	データ操作のアプリケーション	× 占有エラー	△ 占有待ち	△ 占有エラー (注4)	○ (注1)
	データ操作のユーティリティコマンド	× 占有エラー	△ 占有待ち (注3)(注4)	△ 占有エラー	△ 占有エラー (注2)
	運用操作のコマンド	× 占有エラー	○	△ 占有エラー (注2)	○ (注5)(注6)

○：同時実行可能

△：参照/参照は同時実行可能。参照/更新は同時実行不可

×：同時実行不可

注1) アプリケーションが更新処理のトランザクション中の場合、rdbdmpコマンド、rdbsetrpコマンドおよびrdbmrdmpコマンドは、トランザクション終了まで待ちになります。アプリケーションが参照処理または更新処理のトランザクション中の場合、rdbrtrコマンドおよびrdbrlsコマンドは、占有エラーとなります。

注2) rdbdmpコマンドおよびrdbmrdmpコマンドは更新系のユーティリティコマンドと同時に実行できません。

注3) 先行が更新系のユーティリティコマンドの場合、後行のアプリケーションは、利用規定されている旨のエラーとなります。更新系のユーティリティコマンドとは、rdbfmtコマンド、rdbsloaderコマンド、rdbsaloaderコマンド、rdbsuloaderコマンド、rdbgcdsiコマンド、rdbunladtコマンド(mbオプションまたはmfオプション)を示します。

注4) rdbudsiコマンドおよびrdbupsコマンド(cオプション)はデータ操作のアプリケーションと同時に実行可能です。

注5) 以下の場合、同時実行は不可となります。

		後行コマンド					
		rdbinh -i	rdbpmt -i	rdbrtr -i	rdbrls -i	rdbinf -i	rdbexdsi -i
先行コマンド	rdbinh -i	×	×	×	×	×	×
	rdbpmt -i	×	×	×	×	×	×
	rdbrtr -i	×	×	×	×	×	×
	rdbrls -i	×	×	×	×	×	×
	rdbinf -i	×	×	×	×	○	×
	rdbexdsi -i	×	×	×	×	×	×

		後行コマンド					
		rdbinh -p	rdbpmt -p	rdbrtr -p	rdbrls -p	rdbinf -p	rdbexdsi -p
先行コマンド	rdbinh -p	×	×	×	×	×	×
	rdbpmt -p	×	×	×	×	×	×
	rdbrtr -p	×	×	×	×	×	×
	rdbrls -p	×	×	×	×	×	×
	rdbinf -p	×	×	×	×	×	×
	rdbexdsi -p	×	×	×	×	×	×

		後行コマンド
		rdbinf -i
先行コマンド	rdbgcdsi -E(容量拡張)	×

		後行コマンド
		rdbinf -p
先行コマンド	rdbgcdsi -c(拡張域返却)	×

		後行コマンド	
		rdbinf -i	rdbinf -p
先行コマンド	rdbfmt -c(拡張域返却)	×	×

		後行コマンド		
		rdbrtr -z	rdbrls -z	rdbinf -z
先行コマンド	rdbgcdic	×	×	×

		後行コマンド
		rdbudsi
先行コマンド	データ更新UTYコマンド	×

		後行コマンド	
		rdbgcdsi -E	rdbfmt -c
先行コマンド	rdbresident	×	×

注6) 以下のコマンドで、pオプションで指定したDBスペースに含まれるDSIをiオプションで指定した場合、以下のとおり、同時実行は、不可となります。

		後行コマンド					
		rdbinh -i	rdbpmt -i	rdbrtr -i	rdbrls -i	rdbinf -i	rdbexdsi -i
先行コマンド	rdbinh -p	×	×	×	×	×	×
	rdbpmt -p	×	×	×	×	×	×
	rdbrtr -p	×	×	×	×	×	×
	rdbrls -p	×	×	×	×	×	×
	rdbinf -p	×	×	×	×	○	×
	rdbexdsi -p	×	×	×	×	×	×

		後行コマンド					
		rdbinh -p	rdbpmt -p	rdbrtr -p	rdbrls -p	rdbinf -p	rdbexdsi -p
先行コマンド	rdbinh -i	×	×	×	×	×	×
	rdbpmt -i	×	×	×	×	×	×
	rdbrtr -i	×	×	×	×	×	×
	rdbrls -i	×	×	×	×	×	×
	rdbinf -i	×	×	×	×	○	×
	rdbexdsi -i	×	×	×	×	×	×

表D.4 Symfoware/RDBの同一資源に対する各機能の同時実行時の振る舞いで示した各機能の詳細を以下に示します。

定義操作

rdbddlexコマンド、rdbexecsqlコマンドまたはアプリケーションによる、Symfoware/RDBの各資源の定義/定義更新/定義削除の処理です。

rdbalidxコマンドはこの分類に属します。また、監査ログデータベースの作成/削除を行うrdbauditコマンドもこの分類に属します。

データ操作のアプリケーション

SQL文によりデータベースのデータをアクセスするアプリケーションの処理です。

rdbuptコマンド、rdbunlsqlコマンドおよびrdbexecsqlコマンドはこの分類に属します。

データ操作のユーティリティコマンド

データベースの初期化、創成、データの追加/更新、データの抽出または再編成を行うユーティリティコマンドの処理です。

以下のコマンドがあります。

- rdbfmtコマンド
- rdbsloaderコマンド
- rdbsaloaderコマンド
- rdbsuloaderコマンド
- rdbgcdsiコマンド
- rdbunlコマンド
- rdbunladt コマンド

rdbupsコマンドおよびrdbudsiコマンド(データベースの参照)はこの分類に属します。

運用操作のコマンド

以下の運用操作コマンドの処理です。

運用操作の分類	コマンド
データベースの運用操作	rdbinhコマンド rdbpmtコマンド rdbrtrコマンド rdbrlsコマンド rdbtermコマンド rdbhsutyコマンド
データベースの保守および管理	rdbalmdsiコマンド rdbexdsiコマンド
データベースのバックアップ・リカバリ関連	rdbdmpコマンド rdbmrdmpコマンド rdbdvinfコマンド rdbsetrpコマンド
ログ運用操作	rdblogコマンド(アーカイブログ関連)
データベースのチューニング	rdbcrbfコマンド rdbdrbfコマンド rdbconbfコマンド rdbdisbfコマンド rdbprtbfコマンド rdbpldicコマンド rdbresidentコマンド
性能情報出力	rdbsarコマンド rdbpsコマンド rdblkinfコマンド
運用情報出力	rdbinfコマンド rdbprtコマンド rdbspcinfコマンド rdbprdicコマンド

運用操作の分類	コマンド
	rdbmrinfコマンド rdbcninfコマンド rdbprtmsgコマンド rdbchksanityコマンド rdbprxidコマンド
ロードシェア機能を利用した運用	rdbloggroupコマンド

付録E　SQLSTATE値

アプリケーション中で、SQL文の実行結果は状態変数のSQLSTATEに通知されます。したがって、アプリケーションは、SQL文を実行するたびにSQLSTATEをチェックして、Symfoware/RDBの処理結果を確認しながら処理を進めることができます。

SQLSTATE

状態変数のSQLSTATEは、5桁の文字列型の変数です。SQL文を実行した結果、SQLSTATEに設定される値を“状態コード”と呼びます。状態コードの先頭の2桁を“例外コード”と呼び、うしろの3桁を“副例外コード”と呼びます。

SQL文の実行結果に対するコードの値を以下に示します。

参照

クラスタシステム利用時のクラスタ切替え時に返却されるSQLSTATEについては、“クラスタ導入運用ガイド”の“SQLSTATE値”を参照してください。

表E.1 SQLSTATE値

例外コード	例外条件	副例外コード	副例外コードの意味
00	正常終了	000	なし
01	警告	005	なし
02	データなし	000	なし
07	動的SQLエラー	000	なし
		001	USING句が被準備文内の動的パラメタと対応しない
		002	USING句が被準備文内の選択リスト(相手指定)と一致しない
		003	被準備文が＜カーソル指定＞である
		004	動的パラメタ指定に対するUSING句が指定されていない
		005	被準備文が＜カーソル指定＞でない
		006	埋め込みホスト変数属性違反(制限されたデータ型属性を指定)
		007	(相手指定に対する)結果USINGが指定されていない
		008	SQL記述子域の個数不当
		009	SQL記述子域のインデックス不当
08	コネクション違反	000	コネクションの利用方法の誤り
		002	使用中のコネクション
		003	コネクションが存在しない
		004	SQLサーバがSQLコネクションの接続を拒絶した
09	被トリガ動作例外	000	なし
0A	未サポート機能	000	なし
0P	ロール名不当	000	なし

例外コード	例外条件	副例外コード	副例外コードの意味
10	XQuery式エラー	000	なし
21	基数違反	000	単一行SELECT文の実行で2行以上が検索された、または、比較述語に直接含まれる副問合せの結果が2行以上検索された
22	データ例外	001	文字データの右側を切り捨てた(トランケート)
		002	NULL値を設定する標識変数の指定がない
		003	扱える数値の範囲を超えた
		005	代入規則違反
		008	扱える日時の範囲を超えた
		00M	正しくないXML形式のデータ
		011	部分文字列の指定に誤り
		012	除数0による除算
		015	扱える時間隔の範囲を超えた
		018	型変換を行う文字列の形式に誤り
		019	エスケープ文字の不当
		024	正しくない文字型ホスト変数(入力ホスト変数の定義サイズ内にNULL文字が存在していない)
		027	TRIM文字の指定に誤り
		701	正しくない可変長文字列(長さ部の値が0未満、または、定義サイズを超えている)
		702	DSI範囲外アクセス
		703	正しくない数値(固定小数点の符号部の値が不当)
		704	正しくない文字(各国語文字列の値が不当)
23	NOT NULL制約違反、一意性制約違反	000	なし
24	カーソル状態不当	000	なし
25	トランザクション状態不当	000	なし
26	SQL文識別子不当	000	なし
28	パスワードが不当	000	なし
2B	依存する権限記述子がまだ存在する(注1)	000	なし
2E	コネクション名が不当	000	なし
2F	ルーチン呼び出しエラー	000	なし
		003	禁止されているSQL文の実行
33	記述子名不当	000	なし
34	カーソル名不当	000	なし

例外コード	例外条件	副例外コード	副例外コードの意味
37	SQL文構文エラー	000	なし
3C	あいまいなカーソル名	000	なし
3D	カタログ名不当	000	なし
3F	スキーマ名不当	000	なし
40	トランザクションロールバック	001	トランザクション直列化に失敗または実時間オーバ
		003	文終了不明（処理時間オーバまたは強制終了）
		701	テンポラリログ領域不足
		702	COMMIT実行時エラー
		703	アクセス中のノードダウン
		704	アーカイブログファイル不足
		705	Connection Managerの強制終了
		801	定義系処理異常
42	アクセス規則違反	000	権限なし
44	ビューに対する更新でWITH CHECK OPTIONに違反	000	なし
60	利用者送信例外	XXX (注2)	利用者が送信した例外
61	SQL文実行キャンセル	000	なし
70	定量制限	000	なし
71	環境不整合または運用矛盾	000	OSの制限値を超えた
		001	Symfoware/RDBが未起動
		002	サーバシステムが未起動
		003	ダウン中ノードへのアクセス要求
		004	外部プロセスダウン
		010	環境変数または、動作環境ファイルの形式が不当
		020	Symfoware/RDBの制限値を超えた
		030	Symfoware/RDB環境矛盾
		031	コード変換ライブラリロード失敗
		210	DSIが未定義またはDSIが未選択
		220	DSIが未フォーマット
		300	カーソル矛盾
		400	利用規定（注3）
		410	アクセス禁止
		500	データベース未定義またはDSO未定義
		510	検索パス未創成
		600	セションロック不可状態

例外コード	例外条件	副例外コード	副例外コードの意味
		700	トランザクションの実行多重度オーバ
		710	リカバリ要状態
		720	データベースまたは、RDB ディクショナリの矛盾
		730	定義系処理異常
		740	分割転送での矛盾
		750	一時表の制限値オーバ
		800	DSI自動容量拡張での異常
		900	待機系のRDBシステムでのSQL文の実行
		A00	順序の利用順違反
72	外部記憶領域などの不足	000	DSIの格納領域不足
		010	データベーススペース不足
		100	ディレクトリ領域不足
		200	作業用テーブル領域不足
73	メモリの領域不足	100	ローカルメモリのプール数が設定上限値を超えた
		110	ローカルメモリの不足
		200	共用メモリのプール数が設定上限値を超えた
		210	共用メモリの不足
		300	共用バッファの不足
74	外部記憶領域への入出力異常	000	データベーススペースの入出力異常
		010	データベーススペースのアクセス権なし
		100	ディレクトリ領域への入出力異常
		110	ディレクトリ領域のアクセス権なし
		200	作業用ファイルの入出力異常
		210	作業用ファイルのアクセス権なし
		400	テンポラリログ領域の入出力異常
		410	テンポラリログ領域のアクセス権なし
		500	スナップファイルへの入出力異常
		600	その他のファイルの入出力異常
		610	アクセス権なし
		720	RDBディクショナリの入出力異常
75	コネクション環境異常	000	コネクション実行環境の異常
76	システム環境異常	000	システム異常
77	他製品の動作異常	000	PowerAIM/TJNLで異常検出
		010	TJNLが未起動
		100	Accela BizSearchで異常検出

注1) 権限記述子は、以下の項目からなります。

・ 権限受領者が所有する資源(表、ビューなど)

・ 権限受領者が他の認可識別子に付与した権限

注2) SIGNAL文の副例外コードで設定する0～9およびA～Zまでの任意の文字を3桁で表現します。

注3) 処理対象のDSI に対するrdbfmtコマンドを実行中の場合にも設定されることがあります。

参照

ODBCに対応したパソコンツールとの連携で対応するコードが表に存在しない場合には、“MSDNライブラリ Visual Studio” で参照してください。

SQLSTATE値とトランザクションの関係

SQL文を実行した結果により、トランザクションは開始された状態が継続される場合と、終了してアプリケーションに復帰する場合があります。また、SQL文でエラーが発生した場合のトランザクションの扱いを、動作環境ファイルのパラメタTRAN_SPECで指定することができます。動作環境ファイルについては、“付録B 動作環境ファイルのパラメタ一覧”を参照してください。

これらの関係を以下に示します。

表E.2 SQLSTATE値とトランザクションの関係

TRAN_SPEC（注）	SQLSTATEの例外コード	復帰時のトランザクションの状態	アプリケーションでの対処
NONE	00、02	トランザクションは開始された状態が継続しています。	そのまま次の処理に進みます。
	40	トランザクションはロールバックされ終了しています。	無効となったトランザクションを再実行(リトライ)するかどうかを決定します。
	上記以外	トランザクションは開始された状態が継続しています。該当のSQL 文によるデータベースの更新は、実行直前の状態に戻されています。ただし、それまでに獲得されたデータベース資源のロックとカーソルの位置はそのまま保持されます。	エラーの原因によりトランザクションを継続させるか終了させるかを決定します。
TRANSACTION_ROLLBACK	00、02	トランザクションは開始された状態が継続しています。	そのまま次の処理に進みます。
	40	トランザクションはロールバックされ終了しています。	無効となったトランザクションを再実行(リトライ)するかどうかを決定します。
	上記以外	トランザクションはロールバックされ終了しています。	エラーの原因を取り除いて、トランザクションの処理を最初からやり直す必要があります。このためSQL 文の実行でエラーを起こさないように、アプリケーションで入力データのチェックなどを行います。

注) 動作環境ファイルのパラメタ“TRAN_SPEC”に設定された値を示します。

被トリガSQL文の実行でエラーの場合の注意事項

アプリケーションの実行時に、被トリガSQL文の実行がエラーとなった場合、トリガ契機となったSQL文のSQLSTATEの値は、以下に示す値になります。

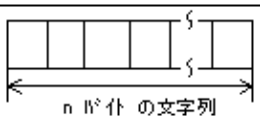
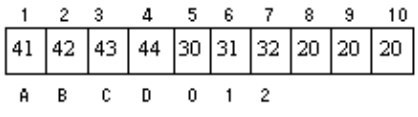
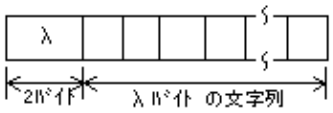
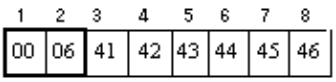
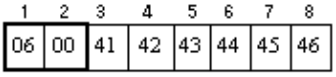
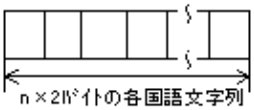
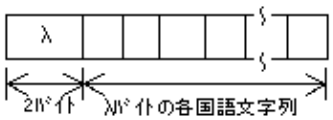
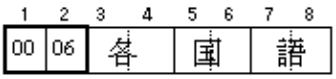
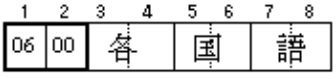
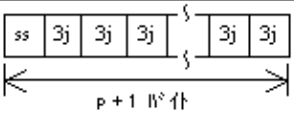
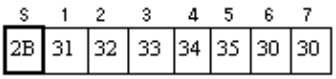
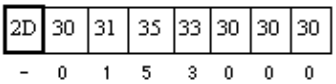
表E.3 トリガ契機となったSQL文のSQLSTATE

被トリガSQL文のSQLSTATE	トリガ契機となったSQL文のSQLSTATE	備考
4xxxx	4xxxx	SQLSTATEとエラーメッセージが、そのままトリガ契機となったSQL文の実行結果に引き継がれます。
7xxxx	7xxxx	上記と同じです。
その他	09000	トリガ契機となったSQL文のSQLSTATEは、09000となり、そのエラーメッセージは、被トリガのSQL文のエラーメッセージが埋め込まれます。

付録F　ファンクションルーチンのパラメタのデータ型と対応するデータ形式

ファンクションルーチンのパラメタに指定されたデータベースのデータ型に対応するデータ形式について、以下に示します。

図F.1 ファンクションルーチンのパラメタのデータ型と対応するデータ形式

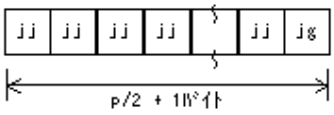
パラメタのデータ型	データ形式	指定例とデータ形式
CHARACTER(n)		CHAR(10) データ例： "ABCD012　" 
CHARACTER VARYING(n)	 λ：有効なデータ長をバイト数で表現します	CHAR VARYING(10) データ例： "ABCDEF" Solarisの場合  LinuxまたはWindows(R)の場合 
NATIONAL CHARACTER(n)		NCHAR(5) データ例： “各国語文字”
NATIONAL CHARACTER VARYING(n)	 λ：有効なデータ長をバイト数で表現します	NCHAR VARYING(10) データ例： “各国語” Solarisの場合  LinuxまたはWindows(R)の場合 
NUMERIC(p, q)	 ss：符号(正：16進数の2B 負：16進数の2D) 3 ：16進数の3 j ：データに対応する数値 (16進数の0~9)	NUMERIC(7, 3) データ例： 1234.5  データ例： -153 

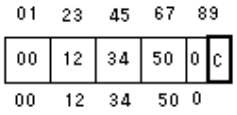
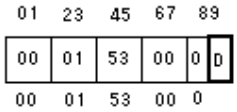
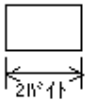
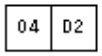
備考. 文字型および各国語文字列型データの文字コード系はデータベースの文字コード系となります。また、各国語文字列型のデータは以下のコードとなります。

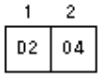
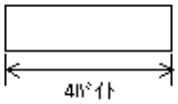
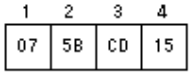
EUCコードの場合:COBOL_EUC

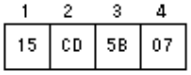
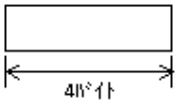
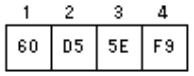
UNICODEの場合: UCS-2形式

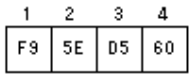
なお、UCS-2形式では、UNICODEの補助文字(1～16面の4バイト文字)はUCS-2の2文字として扱われます。

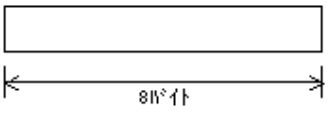
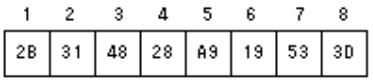
<table>
<tr><th>ﾊﾟﾗﾒﾀのﾃﾞｰﾀ型</th><th>データ形式</th><th colspan="2">指定例とデータ形式</th></tr>
<tr><td>DECIMAL(p,q)</td><td rowspan="2"> 

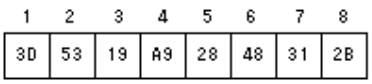
j:データに対応する数値
(16進数の0～9)
g:符号(正:16進数のA,C,E,F
負:16進数のB,D)</td><td>DECIMAL(8,3)
データ例:
1234.5</td><td> 
</td></tr>
<tr><td>データ例:
-153</td><td> 
</td></tr>
<tr><td>SMALLINT</td><td> 
</td><td>SMALLINT
データ例:
1234</td><td>Solarisの場合
 

LinuxまたはWindows(R)の場合
 
</td></tr>
<tr><td>INTEGER</td><td> 
</td><td>INTEGER
データ例:
123456789</td><td>Solarisの場合
 

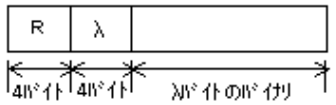
LinuxまたはWindows(R)の場合
 
</td></tr>
<tr><td>REAL</td><td> 
</td><td>REAL
データ例:
1.230E+20</td><td>Solarisの場合
 

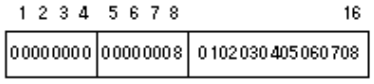
IEEE 形式
LinuxまたはWindows(R)の場合
 

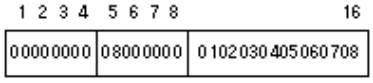
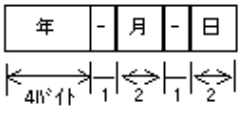
IEEE 形式</td></tr>
<tr><td>DOUBLE
PRECISION</td><td> 
</td><td>DOUBLE
PRECISION
データ例:
1.2345E-100</td><td>Solarisの場合
 

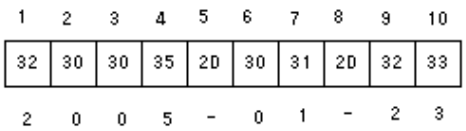
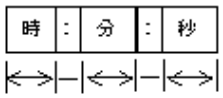
IEEE 形式
LinuxまたはWindows(R)の場合
 

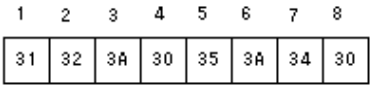
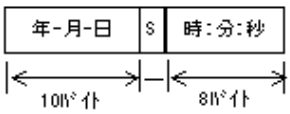
IEEE 形式</td></tr>
</table>

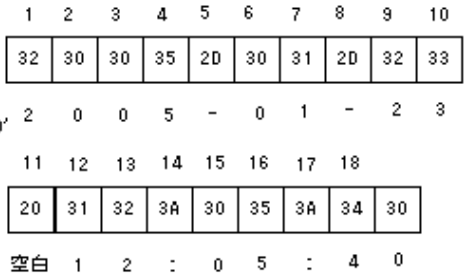
<table>
<tr><th>ﾊﾟﾗﾒﾀのﾃﾞｰﾀ型</th><th>データ形式</th><th>指定例とデータ形式</th></tr>
<tr><td>BINARY LARGE
OBJECT(n)</td><td> 

λ：有効なデータ長をバイト数で
表現します
R：16進数で00000000を指定します</td><td>BLOB(4K)
データ例：
01020304
05060708
(16進)</td><td>Solarisの場合
 

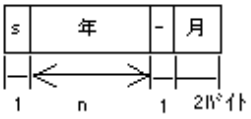
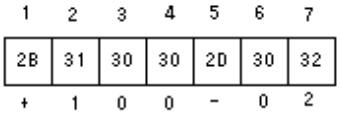
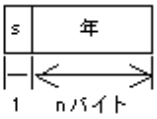
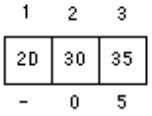
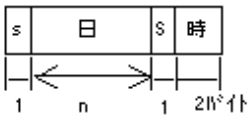
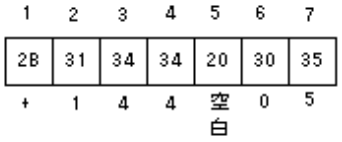
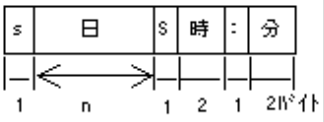
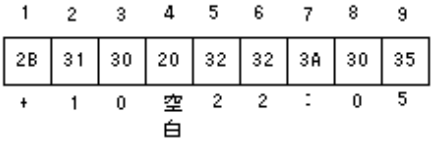
LinuxまたはWindows(R)の場合
 
</td></tr>
<tr><td>DATE</td><td> 

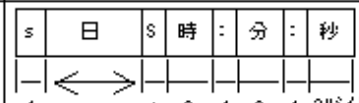
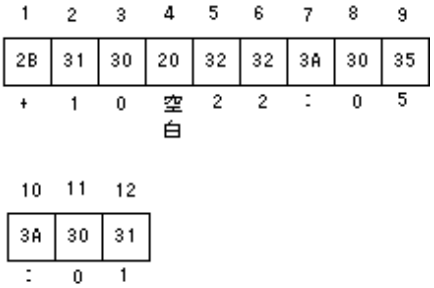
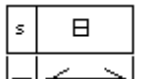
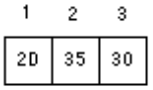
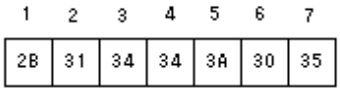
年：数字4桁で表現します
月：数字2桁で表現します
日：数字2桁で表現します
-：文字'-'を指定します</td><td>DATE
データ例：
DATE '2005-01-23'</td><td> 
</td></tr>
<tr><td>TIME</td><td> 

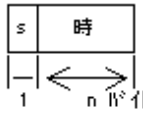
時,分,秒：数字2桁で表現します
':'　　：文字':'を指定します</td><td>TIME
データ例：
TIME '12:05:40'</td><td> 
</td></tr>
<tr><td>TIMESTAMP</td><td> 

年-月-日：DATE型と同じです
時:分:秒：TIME型と同じです
S　　：半角空白1バイトを
指定します</td><td>TIMESTAMP
データ例：
TIMESTAMP
'2005-01-23 12:05:40'</td><td> 
</td></tr>
</table>

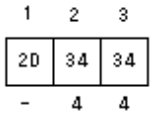
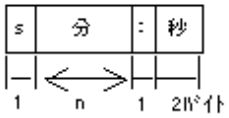
備考.日時型については、文字型のデータとして表現します。

ﾊﾟﾗﾒﾀのﾃﾞｰﾀ型	データ形式	指定例とデータ形式	
INTERVAL YEAR(n) TO MONTH	 s ：符号(正：文字'+' 負：文字'-') 年：数字n桁で表現します 月：数字2桁で表現します - ：文字'-'を指定します	INTERVAL YEAR(3) TO MONTH データ例： INTERVAL +'100-02' YEAR(3) TO MONTH	
INTERVAL YEAR(n)	 s ：符号(正：文字'+' 負：文字'-') 年：数字n桁で表現します	INTERVAL YEAR(2) データ例： INTERVAL -'05' YEAR(2)	
INTERVAL DAY(n) TO HOUR	 s ：符号(正：文字'+' 負：文字'-') 日：数字n桁で表現します 時：数字2桁で表現します S ：半角空白1バイトを 指定します	INTERVAL DAY(3) TO HOUR データ例： INTERVAL +'144 05' DAY(3) TO HOUR	
INTERVAL DAY(n) TO MINUTE	 s ：符号(正：文字'+' 負：文字'-') 日 ：数字n桁で表現します 時,分：数字2桁で表現します S ：半角空白1バイトを 指定します ':' ：文字':'を指定します	INTERVAL DAY(2) TO MINUTE データ例： INTERVAL +'10 22:05' DAY(2) TO MINUTE	

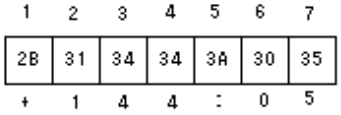
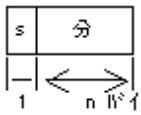
ﾊﾟﾗﾒﾀのﾃﾞｰﾀ型	データ形式	指定例とデータ形式	
INTERVAL DAY(n) TO SECOND	 s　：符号(正：文字'+' 負：文字'-') 日　：数字n桁で表現します 時,分,秒：数字2桁で表現します S　：半角空白1バイトを 指定します ':'　：文字':'を指定します	INTERVAL DAY(2) TO SECOND データ例： INTERVAL +'10 22:05:01' DAY(2) TO SECOND	
INTERVAL DAY(n)	 s ：符号(正：文字'+' 負：文字'-') 日：数字n桁で表現します	INTERVAL DAY(n) データ例： INTERVAL -'50' DAY(2)	
INTERVAL HOUR(n) TO MINUTE	 s　：符号(正：文字'+' 負：文字'-') 時　：数字n桁で表現します 分　：数字2桁で表現します ':'：文字':'を指定します	INTERVAL HOUR(3) TO MINUTE データ例： INTERVAL +'144:05' HOUR(3) TO MINUTE	
INTERVAL HOUR(n) TO SECOND	 s　：符号(正：文字'+' 負：文字'-') 時　：数字n桁で表現します 分,秒：数字2桁で表現します ':'　：文字':'を指定します	INTERVAL HOUR(2) TO SECOND データ例： INTERVAL +'10:22:05' HOUR(2) TO SECOND	

<table>
<tr><th>ﾊﾟﾗﾒﾀのﾃﾞｰﾀ型</th><th>データ形式</th><th colspan="2">指定例とデータ形式</th></tr>
<tr><td>INTERVAL
HOUR(n)</td><td> 

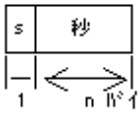
s ：符号(正：文字'+'
負：文字'-')
日：数字n桁で表現します</td><td>INTERVAL
HOUR(2)

データ例：
INTERVAL
-'44' HOUR(2)</td><td> 
</td></tr>
<tr><td>INTERVAL
MINUTE(n)
TO SECOND</td><td> 

s ：符号(正：文字'+'
負：文字'-')
分 ：数字n桁で表現します
秒 ：数字2桁で表現します
':'：文字':'を指定します</td><td>INTERVAL
MINUTE(3)
TO SECOND

データ例：
INTERVAL
-'144:05'
MINUTE(3)
TO SECOND</td><td> 
</td></tr>
<tr><td>INTERVAL
MINUTE(n)</td><td> 

s ：符号(正：文字'+'
負：文字'-')
分：数字n桁で表現します</td><td>INTERVAL
MINUTE(3)

データ例：
INTERVAL +'500'
MINUTE(3)</td><td>
</td></tr>
<tr><td>INTERVAL
SECOND(n)</td><td> 

s ：符号(正：文字'+'
負：文字'-')
秒：数字n桁で表現します</td><td>INTERVAL
SECOND(5)

データ例：
INTERVAL
+'1234'
SECOND(5)</td><td>
</td></tr>
</table>

備考.時間隔型については、文字型のデータとして表現します。

付録G　定量制限

Symfoware Serverの定量制限について説明します。

G.1 Symfoware/RDBの定量制限

Symfoware/RDBの定量制限を以下に示します。

表G.1 定量制限

項目			定量制限
データベース数			無制限
名標の長さ	英字	データベース名	36文字以内 (注1)
		スキーマ名	36文字以内 (注1) (注2)
		表名	36文字以内 (注1) (注2)
		列名	36文字以内 (注1)
		インデックス名	36文字以内 (注1) (注2)
		DSO名	36文字以内 (注1)
		DSI名	36文字以内 (注1)
		データベーススペース名	36文字以内 (注1)
		カーソル名	36文字以内 (注1)
		相関名	36文字以内 (注1)
		ルーチン名	36文字以内 (注1)
		スコープ名	36文字以内 (注1)
		トリガ名	36文字以内 (注1)
		SQL文識別子	36文字以内 (注1)
		SQL変数名	36文字以内 (注1)
		文ラベル	36文字以内 (注1)
		パラメタ名	36文字以内 (注1)
		条件名	36文字以内 (注1)
		認可識別子	18文字以内
		記述子名	36文字以内
		コネクション名	36文字以内
		SQLサーバ名	36文字以内
		順序名	36文字以内 (注1)
		ロール名	36文字以内 (注1)
		ロググループ名	18文字以内
		共用バッファ識別子	18文字以内
	日本語	データベース名	18文字以内
		スキーマ名	18文字以内 (注2)
		表名	18文字以内 (注2)
		列名	18文字以内
		インデックス名	18文字以内 (注2)

項目			定量制限
		DSO名	18文字以内
		DSI名	18文字以内
		データベーススペース名	18文字以内
		カーソル名	18文字以内
		相関名	18文字以内
		ルーチン名	18文字以内
		スコープ名	18文字以内
		トリガ名	18文字以内
		SQL文識別子	18文字以内
		SQL変数名	18文字以内
		文ラベル	18文字以内
		パラメタ名	18文字以内
		条件名	18文字以内
		認可識別子	9文字以内
		記述子名	18文字以内
		コネクション名	18文字以内
		SQLサーバ名	18文字以内
		順序名	18文字以内
		ロール名	18文字以内
		ロググループ名	18文字以内または12文字以内 (注3)
		共用バッファ識別子	18文字以内
スキーマ要素	実表数／スキーマ		無制限
	列数／表		32,000個
	表の行長	SEQUENTIAL構造(BLOB型を含む場合)	2,000メガバイト
		SEQUENTIAL構造(BLOB型を含まない場合)またはRANDOM構造	32,000バイト
		OBJECT構造	2,047メガバイト＋32,000バイト
	一意性制約を構成する列数		64列
	一意性制約を構成するデータ長		1,000バイト
	既定値オプションのデータ長		3,000バイト (注4)
	容量／表		ペタバイトオーダ
	注釈定義		256バイト
	トリガ定義文の探索条件の文字列長		15,000バイト
	トリガ定義文の被トリガSQL文の文字列長		15,000バイト
	ビュー定義文の問合せ指定の文字列長		32,000バイト (注5)
	プロシジャルーチン定義の文字列長		無制限
格納構造要素	実表	DSO数／スキーマ	無制限
		DSI数	無制限

項目				定量制限
		DSI数／DSO		無制限
		分割条件を指定する固定長文字列型の1つあたりのデータ長	CHARACTER	254文字
			NATIONAL CHARACTER	127文字
		キー構成列数		64列
		クラスタのキーの長さ		1,000バイト
		ページ長(BLOBが31キロバイト以内)		1、2、4、8、16、32キロバイト
		ページ長(BLOBが32キロバイト以上)		32キロバイト
		容量／DSI		2テラバイト未満
		DSIの静的容量拡張／拡張量		2,097,151キロバイト
		DSIの動的容量拡張／1回の拡張量		2,097,151キロバイト
		DSIの動的容量拡張／拡張契機		2テラバイト未満
		DSI領域のアラームポイント		2テラバイト未満
	インデックス	DSO数		無制限
		DSI数		無制限
		DSI数／DSO		無制限
		キー構成列数		64列
		キーの長さ		1,000バイト
		ページ長		1、2、4、8、16、32キロバイト
		容量／DSI		2テラバイト未満
		DSIの静的容量拡張／拡張量		2,097,151キロバイト
		DSIの動的容量拡張／1回の拡張量		2,097,151キロバイト
		DSIの動的容量拡張／拡張契機		2テラバイト未満
		DSI領域のアラームポイント		2テラバイト未満
扱えるデータ種と属性(C)	文字		固定長	32,000バイト
			可変長	32,000バイト
	日本語文字		固定長	32,000バイト
			可変長	32,000バイト
	数字	外部10進表現		18桁
		内部2進表現	2バイト	-32,768 から 32,767
			4バイト	-2,147,483,648 から 2,147,483,647
			8バイト	-999,999,999,999,999,999 から 999,999,999,999,999,999
		内部10進表現		18桁
		浮動小数点表現	4バイト	3.4E +/- 38 (7 桁)
			8バイト	1.7E +/- 308 (15 桁)
	BLOB			2ギガバイト

項目				定量制限
扱えるデータ種と属性(COBOL)	文字		固定長	32,000文字
			可変長	32,000文字
	日本語文字(UTF-32形式以外)		固定長	16,000文字
			可変長	16,000文字
	日本語文字(UTF-32形式)		固定長	8,000文字
			可変長	8,000文字
	数字	外部10進表現		18桁
		内部2進表現	2バイト	4桁
			4バイト	9桁
		内部10進表現		18桁
		浮動小数点表現	4バイト	仮数桁数6以下
			8バイト	仮数桁数7以上
	BLOB			2ギガバイト
プロシジャルーチン定義	SQLパラメタ宣言リストに指定できるSQLパラメタ宣言の数			32,767個
	局所宣言リストに指定できる局所宣言の数			32,767個
	SQL変数宣言に指定できるSQL変数名の数			32,767個
	ハンドラ宣言に指定できる条件値の数			32,767個
	SQL文リストに指定できるSQL手続き文の数			32,767個
GRANT文/REVOKE文	権限受領者に指定できる認可識別子またはロール名の数			32,767個
	ロール付与またはロール剥奪に指定できるロール名の数			32,767個
データ操作文	アプリケーションの1プロセスあたりの最大コネクション数		ローカルアクセス	512
			リモートアクセス	無制限
	選択リストに指定可能な値式数			32,000個
	選択リストに指定可能な列の合計長			2ギガバイト－1(注6)
	表式のFROM句に指定可能な表参照の数			128個
	GROUP BY句に指定可能な値式数			32,000個
	ORDER BY句に指定可能な値式数			32,000個
	1つの問合せ式に指定できる問合せ項の数			64個 (注7)
	1つの参照表に指定できる結合表の入れ子の数			128個 (注8)
	1つの値式に指定できる関数または演算式の数			128個 (注9)
	1つの行値構成子に指定できる値式の数			127個
	1つのファンクションルーチン指定の引数リストに指定可能な値式の合計長			2ギガバイト-1 (注10)
	1つのセションで同時に処理可能なカーソル数			無制限
	1つのセションで同時に接続できるコネクション			16
	1つのSQL文の文字列長		静的SQL	無制限
			動的SQL	32キロバイト(注11)

項目		定量制限
	1つのSQL文に指定できる動的パラメタ指定の数	32,767個
	IN述語の限定値リストに指定できる値指定の数	32,767個
	USING句に指定できる引数の数	32,767個
	結合表に指定できる結合演算項の数 (注12)	128個
	表宣言に指定できる表名の数	32,767個
	COALESCEに指定できる値式の数	32,767個
	CASE式に指定できる単純WHEN句または探索WHEN句の数	32,767個
	作業用テーブルおよび作業用ソート領域に格納可能な列の合計長 (注13)	64キロバイト (注14)
	作業用テーブルおよび作業用ソート領域に格納可能なレコード数	2147483647件
ASSIST指定	ASSIST要素に指定できる制約ASSISTの数	128個
	ジョインASSISTに指定できるASSIST表要素の数	128個
RDBシステムで管理可能なデータベース規模		128ペタバイト
ロググループ数 (注15)		100個 (注16)
データベーススペースの割付けサイズ		1キロバイト以上2テラバイト未満(注17)
RDBディクショナリの割付けサイズ		2テラバイト未満(注17)
DSIの割付けサイズ		2テラバイト未満
RDBディレクトリファイルの割付けサイズ		4テラバイト未満(注17)
テンポラリログファイル	BIログ域サイズ	16ギガバイト未満
	AIログ域サイズ	16ギガバイト未満
	インダウトログファイルのサイズ	16ギガバイト未満
アーカイブログファイルのサイズ		制限なし
アンロードファイル(rdbunl、rdbunlsql、rdbunlx、rdbunladt)のサイズ		制限なし
入力データファイル(rdbsloader、rdbsaloader、rdbsuloader)のサイズ		制限なし
入力データファイル(rdbupt)のサイズ		2ギガバイト未満
DSI退避ファイル(rdbdmp)のサイズ		制限なし
RDBディクショナリ退避ファイル(rdbdmpdic)のサイズ		制限なし
アーカイブログ退避ファイル(rdblog -B)のサイズ		制限なし
アプリケーションの1プロセスあたりの最大コネクション数	ローカルアクセス	512
	リモートアクセス	制限なし
ソート作業域のサイズ (rdbgcdsi、rdbsaloader、rdbsloader、rdbsuloader、rdbrcv、rdbups)		制限なし
1つのデータベーススペース内に作成可能なサブエクステント数 (注18)		262,138個
パフォーマンスデータ格納ファイルのサイズ		制限なし
コアファイルのサイズ		制限なし

注1) Symfoware/RDBと組み合わせて使用する製品で18文字を超える識別子を扱えない場合は、システム用の動作環境ファイルでNAME_SIZE_CHECK=YESを指定することにより、19文字以上の資源名を定義できないようにチェックすることができます。19文字以上の資源名の定義に対しては、エラーが返却されます。

注2) 格納構造定義を行わない場合は8文字以内です。ただし、システム用の動作環境ファイルでDEFAULT_DSI_NAME=CODEを指定した場合、または、デフォルトデータベーススペースに表またはインデックスを作成する場合は、定量制限どおりとなります。

注3) Symfoware/RDBの文字コード系がEUCコードまたはシフトJISコードの場合は18文字以内となります。Symfoware/RDBの文字コード系がUNICODEの場合は12文字以内となります。

注4) 既定値オプションを文字列定義で指定する場合、データ中の引用符は2バイトとして数えます。

注5) 30,000バイトを超える問合せ指定を定義した場合、システム表のテーブルRDBII_SYSTEM.RDBII_DESCRIPTIONの、列DESC_VALUEには、30,000バイトまでしか格納されません。

注6) 以下の見積り式で2ギガバイト－1まで指定可能です。

```
列の合計長 ＝選択リストに指定した列の合計長
           ＋（選択リストに指定したBLOB列数 × 8）
           ＋ NULL値を許可する列数（表定義でNOT NULLを指定していない列数）
           ＋（可変長の列数 × 2）
```

なお、選択リストに定数を指定した場合、返却データのデータ型は指定する定数に依存します。

参照

定数を指定した場合のデータ型については、“SQLリファレンス”を参照してください。

注7) 1つの問合せ式として、問合せ項UNION(またはUNION ALL)を複数指定した場合のUNION(またはUNION ALL)の数の定量制限値です。

以下にこの定量制限値でのUNION(またはUNION ALL)の数え方の例を示します。

例

以下の例の場合、UNIONの数は2個です。

```
SELECT 倉庫番号 FROM 在庫管理.在庫表
    UNION SELECT 取引先 FROM 在庫管理.発注表
    UNION SELECT 会社番号 FROM 在庫管理.会社表
```

上記例の数え方については、以下のとおりです。

```
SELECT 倉庫番号 FROM 在庫管理.在庫表
  UNION SELECT 取引先 FROM 在庫管理.発注表        1個目
  UNION SELECT 会社番号 FROM 在庫管理.会社表      2個目
```

注8) FROM句に指定できる1つの表参照として、結合表を入れ子で指定した場合の結合表の数の定量制限値です。

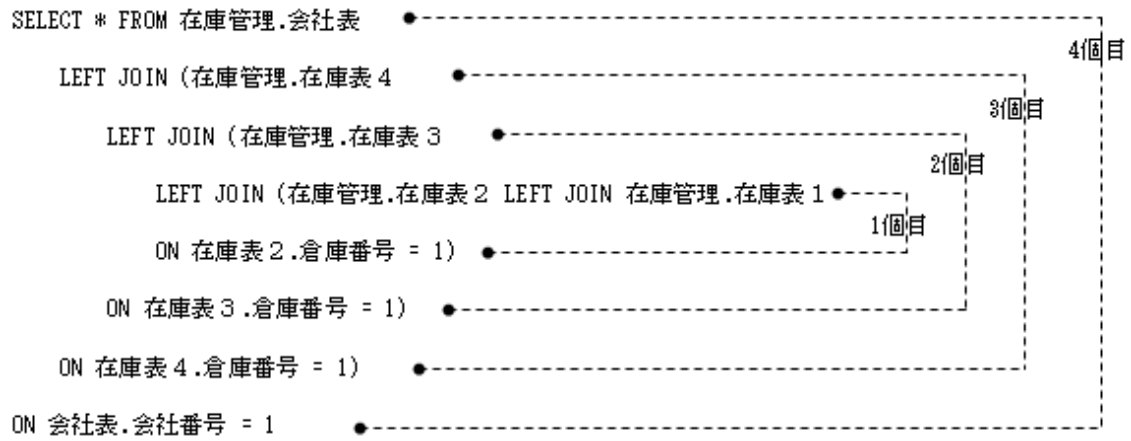
以下にこの定量制限値での結合表の数え方の例を示します。

例

以下の例の場合、結合表の数は4個です。

```
SELECT * FROM 在庫管理.会社表
    LEFT JOIN（在庫管理.在庫表４
        LEFT JOIN（在庫管理.在庫表３
            LEFT JOIN（在庫管理.在庫表２ LEFT JOIN 在庫管理.在庫表１
            ON 在庫表２.倉庫番号 = 1）
        ON 在庫表３.倉庫番号 = 1）
    ON 在庫表４.倉庫番号 = 1）
ON 会社表.会社番号 = 1
```

上記例の数え方については、以下のとおりです。



注9) 1つの値式に指定した関数または演算式の合計数の定量制限値です。

関数の場合、1つの値式として、関数を入れ子で指定した場合の入れ子の数です。

演算式の場合、1つの値式として、演算式を複数指定した場合の演算式の数です。

なお対象となる値式は、以下のとおりです。

- 列指定
- 数値関数
- データ列値関数
- 日時値関数
- ファンクションルーチン指定
- CASE式
- CAST指定
- 数値式(単項演算子または二項演算子)
- データ列値値式(連結演算子)
- 日時値式(日時演算)
- 時間隔値式(時間隔演算)

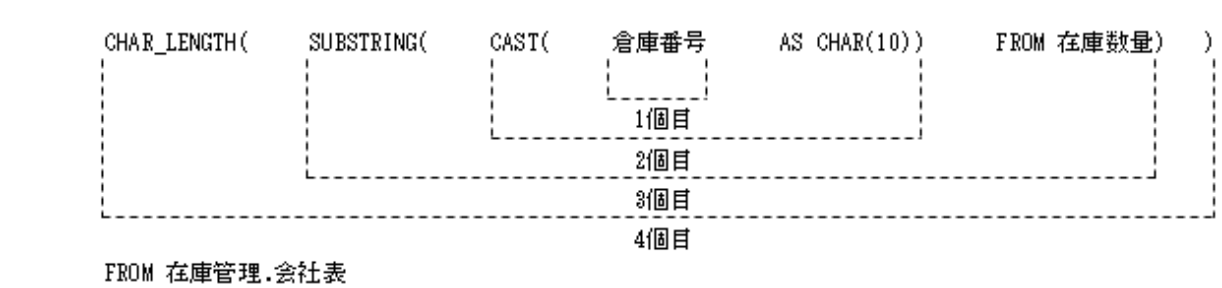
以下にこの定量制限値での関数および演算式の数え方の例を示します。

例1:関数の場合

以下の例の場合、関数の数は4個です。

```
SELECT
    CHAR_LENGTH(SUBSTRING(CAST(倉庫番号 AS CHAR(10)) FROM 在庫数量))
    FROM 在庫管理.会社表
```

上記例の関数の数え方については、以下のとおりです。

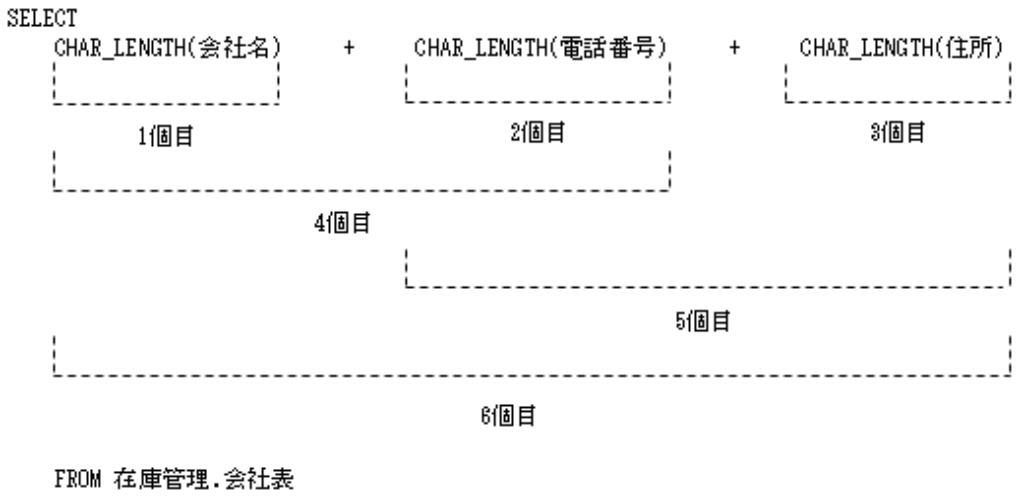


例2:演算式の場合

以下の例の場合、演算式の数は6個です。

```
SELECT
    CHAR_LENGTH(会社名)+CHAR_LENGTH(電話番号)+CHAR_LENGTH(住所)
    FROM 在庫管理.会社表
```

上記例の演算式の数え方については、以下のとおりです。



注10) 以下の見積り式で2ギガバイト-1まで指定可能です。

```
引数リストの合計長 ＝引数リストに指定した値式の合計長
                  ＋ 戻りデータ型の長さ
                  ＋ 2MB

引数リストに指定した値式の合計長 :
SQL文に複数のファンクションルーチンを指定している場合、各ファンクションルーチンについて引数リストに指定した値式の合計長を求め、その合計長のうち、最長の値を指定してください。
引数リストに指定した値式がBLOB型を含む、かつ、値式の合計長が128K以上の場合、BLOB型の値式については実データ長で算出してください。BLOB型以外の値式については定義長で算出してください。

戻りデータ型の長さ:
SQL文に複数のファンクションルーチンを指定している場合、各ファンクションルーチンの戻りデータ型の定義長のうち、最長の値を指定してください。
```

注11) 埋込みSQL連携の場合の定量制限値です。Java連携、ODBC連携および.NET Framework連携の場合は、無制限です。

注12) FROM句に指定できる1つの結合表として、結合演算項の数の定量制限値です。

以下にこの定量制限値での結合演算項の数え方の例を示します。

例

以下の例の場合、結合演算項の数は4個です。

```
SELECT * FROM 在庫管理.会社表
    LEFT JOIN 在庫管理.在庫表４ ON 会社表.会社番号 = 1
    LEFT JOIN 在庫管理.在庫表３ ON 在庫表４.倉庫番号 = 1
    LEFT JOIN 在庫管理.在庫表２ ON 在庫表３.倉庫番号 = 1
    LEFT JOIN 在庫管理.在庫表１ ON 在庫表２.倉庫番号 = 1
```

上記例の数え方については、以下のとおりです。

```
SELECT * FROM 在庫管理.会社表
    LEFT JOIN 在庫管理.在庫表４ ON 会社表.会社番号 = 1    ●---------- 1個目
    LEFT JOIN 在庫管理.在庫表３ ON 在庫表４.倉庫番号 = 1  ●---------- 2個目
    LEFT JOIN 在庫管理.在庫表２ ON 在庫表３.倉庫番号 = 1  ●---------- 3個目
    LEFT JOIN 在庫管理.在庫表１ ON 在庫表２.倉庫番号 = 1  ●---------- 4個目
```

注13) 選択リストに記述した項目に列が含まれる場合、列のデータ長は定義長です。ただし、32キロバイトを超えるBLOB列については、格納構造がSEQENTIAL構造の場合は24バイト、OBJECT構造の場合は18バイトです。

以下のいずれかに該当する場合、標識変数の1バイトを含めてください。

- NOT NULL制約を指定していない列
- LEFTを指定した結合表(OUTER JOIN)の右側の表の列
- RIGHTを指定した結合表(OUTER JOIN)の左側の表の列

データ長は、以下で確認できます。

- アクセスプラン:エレメント情報の詳細情報 insert record length
- SQL性能情報 :実行エレメント情報の詳細情報 record length(byte)

注14) カーソル宣言のSCROLLの指定の有無により、しきい値は異なります。カーソル宣言にSCROLLを指定した場合のしきい値は65516バイト、カーソル宣言にSCROLLを指定しない場合のしきい値は65519バイトです。

注15) ロググループ数とは、システムロググループ数(1個)とユーザロググループ数の合計値です。ユーザロググループには、監査ログ用ロググループも含まれます。

注16) ロードシェア運用の場合は、ロードシェアシステム全体で最大1600個のロググループが定義できます。

注17) ローデバイスに配置する場合、Solaris 10の場合は1テラバイト未満、Solaris 11の場合は2テラバイト未満です。

注18) rdbprtコマンドでデータベーススペースの定義情報を出力した時に、“DSI information”に表示されるDSI情報の数が、データベーススペース内のサブエクステント数です。

コンパイル時の定量制限

1つのコンパイル単位における定量制限を以下に示します。

表G.2 コンパイル時の定量制限

項目	定量制限	備考
ホストラベル識別子	32個	
日本語名標	－	ホスト変数名など、C言語では使用不可

G.2 Textアダプタの定量制限(Solarisの場合)

Textアダプタの定量制限を以下に示します。

表G.3 Textアダプタの定量制限

項目		定量制限
Textアダプタ定義	Textアダプタ定義名長	32バイト
	Textアダプタ定義数	無制限
	Columnタグ数	64個
	フィールド名長	15バイト

項目		定量制限
TEXT検索式	TEXT検索式	8192バイト

G.3 XMLアダプタの定量制限

XMLアダプタの定量制限を以下に示します。

表G.4 XMLアダプタの定量制限

項目			定量制限
名標の長さ	英字	XMLグループ名	36文字以内
	日本語	XMLグループ名	18文字以内
格納構造要素	XML構造のインデックス	DSO数	無制限
		DSI数	無制限
		DSOの中に存在するDSI数	無制限
		キー構成列数	1個(BLOB型)
		ページ長	1、2、4、8、16、32キロバイト
		実表のDSOに対するXML構造のインデックスのDSO数	1個
XMLグループ定義	XMLグループ定義数		無制限
	パス定義数		1024個
	パスの段数		50段
	パスの文字列長		500バイト
	型指定	STRING	1000バイト
		NUMBER	18桁
	選択条件	列名リスト	64個

付録H　アクセスプランの詳細

Symfoware Serverのアクセスモデルおよびエレメントの種類について説明します。

H.1　アクセスモデルの種類

Symfoware Serverのアクセスモデルについて説明します。

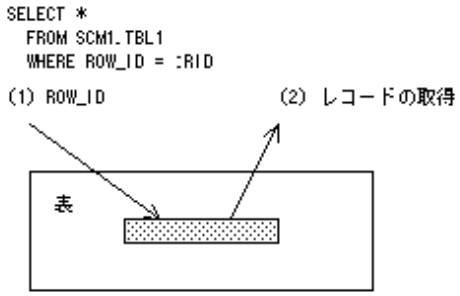
H.1.1　制約系アクセスモデル

制約系アクセスモデルには、以下のものがあります。

アクセスモデル名
ROW_ID検索
クラスタキー検索
インデックス検索
インデックス検索と表データ取得
TIDユニオンマージ
表の全件検索
並列スキャン

ROW_ID検索

WHERE探索条件に指定したROW_IDに対する条件を元に、検索対象のレコードを取り出すアクセスモデルです。WHERE探索条件にROW_IDを指定すると、Symfoware/RDBはこのアクセスモデルを選択します。



(1) WHERE探索条件に指定したROW_IDにより、TABLE KEY SCANを行う。

(2) レコードを取得してアプリケーションに返却する。

[アクセスプランの出力例]

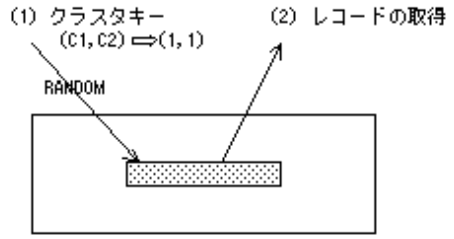
```
================================================================================
Main query
=sno===sectname=====input1================input2=============output/update======
   1 : SCAN        [TBL1_BASE_DSO       ][                 ][APPL              ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        TABLE KEY SCAN
        dso name         TBL1_BASE_DSO          [PAGE/SH]
        condition evaluation  No
        scan record number    1
```

```
  [  2] OUTPUT ELEMENT
        record length         200
```

クラスタキー検索

WHERE探索条件に指定した条件が、クラスタキーを一意に識別する場合に、クラスタキーにより検索対象レコードを取り出すアクセスモデルです。このアクセスモデルは表の格納構造がRANDOMの表に対してのみ可能です。このアクセスモデルが可能な場合は、Symfoware/RDBはクラスタキー検索のアクセスモデルを選択します。

```
SELECT *
  FROM SCM1.TBL1
  WHERE C1=1 AND C2=1
```



(1) WHERE探索条件より、クラスタキー値を生成し、TABLE KEY SCANを行う。

(2) レコードを取得してアプリケーションに返却する。

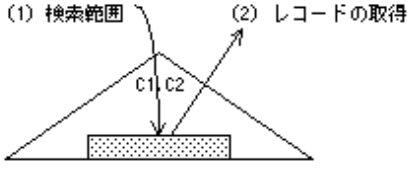
[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1================input2============output/update======
   1 : SCAN        [TBL1_BASE_DSO       ][                 ][APPL              ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        dsi access       ONE DSI
        scan type        TABLE KEY SCAN
        dso name         TBL1_BASE_DSO          [PAGE/SH]
        condition evaluation  No
        scan record number    1
  [  2] OUTPUT ELEMENT
        record length         200
```

インデックス検索

WHERE探索条件に指定した条件から、インデックスの検索範囲を作成して、インデックスを検索するアクセスモデルです。SQL文中に指定した列がすべてインデックスで取得できる場合に、表にアクセスを行わずに、インデックスのみからデータを取り出します。行単位の排他を指定している場合は、必ず表にアクセスするため、このアクセスモデルは選択しません。

```
SELECT C1,C2
  FROM SCM1.TBL1
  WHERE C1=1
```



(1) WHERE探索条件よりインデックス検索範囲を作成し、INDEX KEY SCANを行う。検索範囲を生成できない場合は、INDEX ALL SCANを行うこともある。

(2) レコードを取得してアプリケーションに返却する。

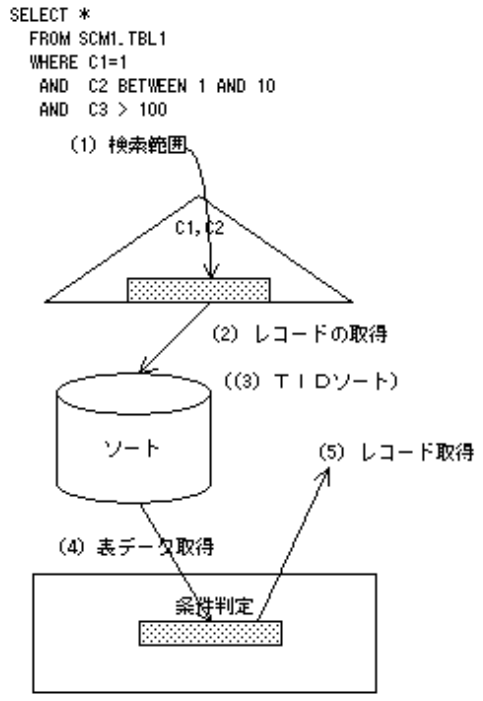
[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1==============input2============output/update======
   1 : SCAN        [TBL1_IX1_DSO      ][                 ][APPL              ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        INDEX KEY SCAN(1)
        dso name         TBL1_IX1_DSO        [PAGE/SH]
        condition evaluation  No
        scan record number    10
  [  2] OUTPUT ELEMENT
        record length         8
```

インデックス検索と表データ取得

WHERE探索条件に指定した条件から、インデックスの検索範囲を作成して、インデックスを検索し、インデックスから取得できないデータを表から取得するアクセスモデルです。

また、行単位の排他を指定している場合にインデックスを使用して検索を行うと、必ず表からデータを取得します。



(1) WHERE探索条件よりインデックス探索範囲を作成し、INDEX KEY SCANを行う。

(2) TIDを取得する。

(3) 表に対するI/O効率向上のためにTIDソートを行う場合は、TIDでソートを行う。

(4) ソートからTIDを取得(TABLE ALL SCAN)し、表をTABLE KEY SCANする。

(5) レコードを取得してアプリケーションに返却する。

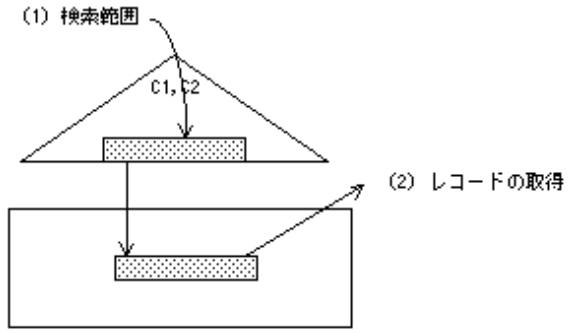
[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : SCAN        [TBL1_IX1_DSO      ][                  ][SORT0001          ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        INDEX KEY SCAN(2)
        dso name         TBL1_IX1_DSO        [PAGE/SH]
        condition evaluation  No
        scan record number    10
  [  2] INSERT ELEMENT
        table name       SORT0001
        insert record length  12
--------------------------------------------------------------------------------
   2 : SCAN        [SORT0001          ][TBL1_BASE_DSO     ][APPL              ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SORT0001
        scan type        TABLE ALL SCAN
        condition evaluation  No
  [  2] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        TABLE KEY SCAN
        dso name         TBL1_BASE_DSO       [PAGE/SH]
        condition evaluation  Yes
        scan record number    1
  [  3] OUTPUT ELEMENT
        record length         200
```

参考

TIDソートは、TIDを元に表データを取得するときに、表に対するI/O効率を高める目的で行います。インデックスのデータ部のレコードは、インデックスキー値+TIDで昇順に並んでいます。このため、インデックスを範囲検索して取得したTIDで順次表からデータを取得すると、表のページを再読込みする可能性があります。TIDでソートすることにより表ページの再読込みを抑止しています。なお、インデックスを構成するすべての列に対してWHERE探索条件で比較述語“=”で条件を与えると、TIDの昇順にデータを取り出すことができるため、TIDソートは行いません。この場合は、INDEX TABLE SCANのアクセス方式が使用されます。また、TIDソートはチューニングパラメタ“TID_SORT”に“NO”を指定することにより抑止することができます。

```
SELECT *
  FROM SCM1.TBL1
  WHERE C1=1
   AND  C2=1
```

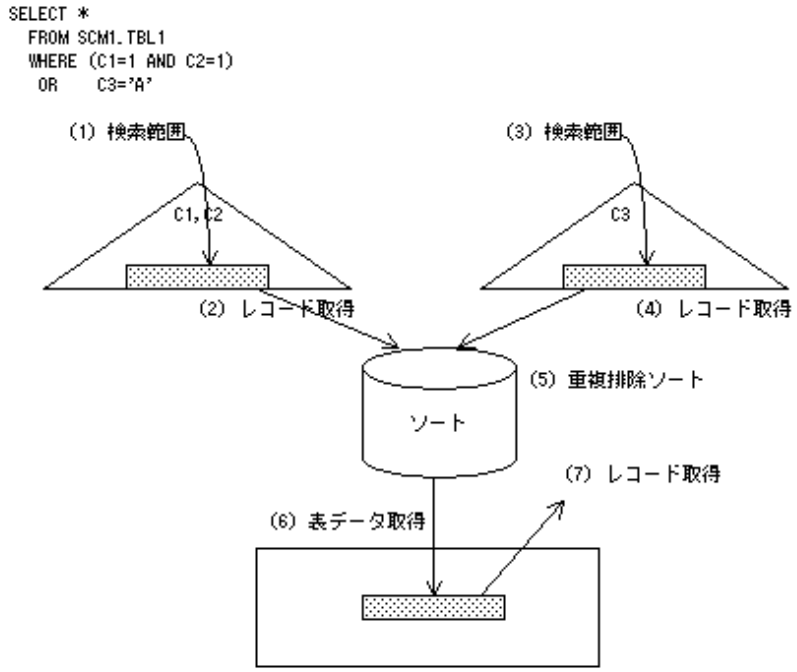


(1) WHERE探索条件よりインデックス検索範囲を作成し、INDEX TABLE SCANを行う。

(2) レコードを取得してアプリケーションに返却する。

TIDユニオンマージ

WHERE探索条件にブール演算子“OR”を指定した場合に、ORで結合された条件のそれぞれについてインデックスの検索範囲を作成してインデックスを検索します。それぞれの結果を1つのソートテーブルに格納します。ソートテーブルからの取出し時に、TIDの重複を排除してから、表データを取得するアクセスモデルです。



(1) WHERE探索条件よりインデックス検索範囲を作成し、INDEX KEY SCANを行う。

(2) TIDを取得し、ソートテーブルにINSERTする。

(3) WHERE探索条件よりインデックス検索範囲を作成し、INDEX KEY SCANを行う。

(4) TIDを取得し、ソートテーブルにINSERTする。

(5) 重複レコードを省くため、TIDソートを行う。

(6) ソートからTIDを取得(TABLE ALL SCAN)し、表をTABLE KEY SCANする。

(7) レコードを取得してアプリケーションに返却する。

[アクセスプランの出力例]

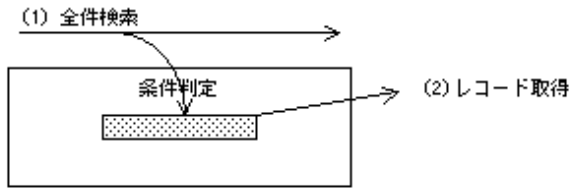
```
================================================================================
Main query
=sno===sectname=====input1=============input2=============output/update======
   1 : SCAN        [TBL1_IX1_DSO      ][                 ][SORT0001          ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        INDEX KEY SCAN(2)
        dso name         TBL1_IX1_DSO        [PAGE/SH]
        condition evaluation  No
        scan record number    10
  [  2] INSERT ELEMENT
        table name       SORT0001
        insert record length  12
```

```
--------------------------------------------------------------------------------
  2 : SCAN        [TBL1_IX2_DSO     ][                  ][SORT0001          ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        INDEX KEY SCAN(1)
        dso namev        TBL1_IX2_DSO        [PAGE/SH]
        condition evaluation  No
        scan record number    30
  [  2] INSERT ELEMENT
        table name       SORT0001
        insert record length  12
--------------------------------------------------------------------------------
  3 : SCAN        [SORT0001         ][TBL1_BASE_DSO     ][APPL              ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SORT0001
        scan type        TABLE ALL SCAN
        condition evaluation  No
  [  2] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        TABLE KEY SCAN
        dso name         TBL1_BASE_DSO       [PAGE/SH]
        condition evaluation  No
        scan record number    1
  [  3] OUTPUT ELEMENT
        record length         200
```

表の全件検索

表の全レコードを読み込みながら、表データを取得するアクセスモデルです。表の全データを読み込むため、検索効率が悪く、Symfoware/RDBは、他に有効なアクセスモデルを選択できない場合に、このアクセスモデルを選択します。

```
SELECT *
  FROM SCM1.TBL1
  WHERE C1 NOT BETWEEN 1 AND 10
```



(1) TABLE ALL SCANを行う。

(2) レコードを取得してアプリケーションに返却する。

[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1=============input2=============output/update======
  1 : SCAN        [TBL1_BASE_DSO    ][                  ][APPL              ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        TABLE ALL SCAN
        dso name         TBL1_BASE_DSO       [DSI/SH]
        condition evaluation  Yes
        scan record number    800
```

```
  [  2] OUTPUT ELEMENT
        record length         200
```

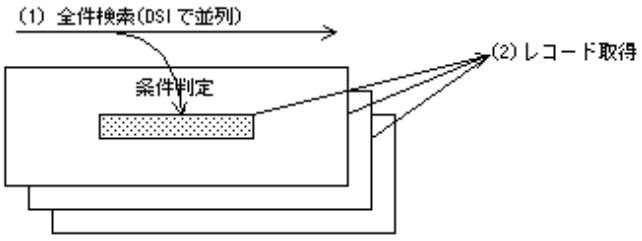
並列スキャン

表のレコードをDSIごとに並列に読み込んで表データを取得するアクセスモデルです。並列スキャンが動作する環境下で、SQL文に並列指定を指定するか、動作環境ファイルに並列クエリで動作するように指定すると、Symfoware/RDBはこのアクセスモデルを選択します。

なお、以下のいずれかの条件を満たす場合、並列スキャンのアクセスモデルは選択しません。

- SQL文に並列指定を指定した場合
  - 表をDSI分割していない、または1つのDSIに対するアクセスである場合
  - ROW_ID検索のアクセスモデルが利用できる場合
  - クラスタキー検索のアクセスモデルが利用できる場合
  - ASSIST指定のUSE_INDEXを指定した場合
- 動作環境ファイルにデータベースを並列に検索するように指定した場合
  - 表をDSI分割していない、または1つのDSIに対するアクセスである場合
  - ROW_ID検索のアクセスモデルが利用できる場合
  - クラスタキー検索のアクセスモデルが利用できる場合
  - インデックス検索のアクセスモデルが利用できる場合
  - インデックス検索と表データ取得のアクセスモデルが利用できる場合
  - TIDユニオンマージのアクセスモデルが利用できる場合

```
SELECT C1,C2,C3
  FROM SCM1.TBL1
  WHERE C1 < 10 AND C2 <> 'A'
  PARALLEL
```



(1) DSIごとに並列にTABLE ALL SCANを行う。

(2) レコードを取得してアプリケーションに返却する。

[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : SCAN        [TBL1_BASE_DSO     ][                  ][APPL              ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        dsi access       ALL DSI
        scan type        TABLE PARALLEL SCAN
        dso name         TBL1_BASE_DSO       [DSI/SH]
        condition evaluation  Yes
        scan record number    500
```

```
[  2] OUTPUT ELEMENT
      record length         12
```

H.1.2 ジョイン系アクセスモデル

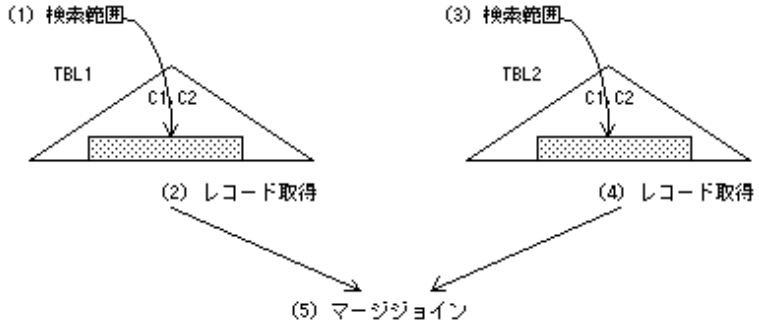
ジョイン系アクセスモデルには以下のものがあります。

アクセスモデル名	種類
マージジョイン	1) マージジョイン(インデックスを利用してジョインする) 2) マージジョイン(インデックスを利用してジョインをして表データも取得する) 3) マージジョイン(ソートテーブルを利用してジョインする)
フェッチジョイン	4) フェッチジョイン(インデックスを利用してジョインする) 5) フェッチジョイン(クラスタキーを利用してジョインする) 6) フェッチジョイン(実表を直接読み込んでジョインする)
ネスティドループジョイン	7) ネスティドループジョイン(実表を直接読み込んでジョインする) 8) ネスティドループジョイン(ワークテーブルを利用してジョインする)

1) マージジョイン(インデックスを利用してジョインする)

ジョインキーを昇順に読み込む方法として、インデックスを利用するマージジョインのアクセスモデルです。

```
SELECT TBL1.C1,TBL1.C2,TBL2.C2
  FROM SCM1.TBL1,SCM1.TBL2
  WHERE TBL1.C1=TBL2.C1
   AND  TBL1.C1 BETWEEN 1 AND 10
```



(1)(3)WHERE探索条件によりインデックス検索範囲を作成し、INDEX KEY SCANを行う。

(インデックスに検索範囲が生成できない場合は、INDEX ALL SCANを行う)

(2)(4)インデックスキー(ジョインキー)の昇順にレコードを取得する。

(5)ソートマージ法でジョインを行い、結合したデータをアプリケーションに返却する。

[アクセスプランの出力例]

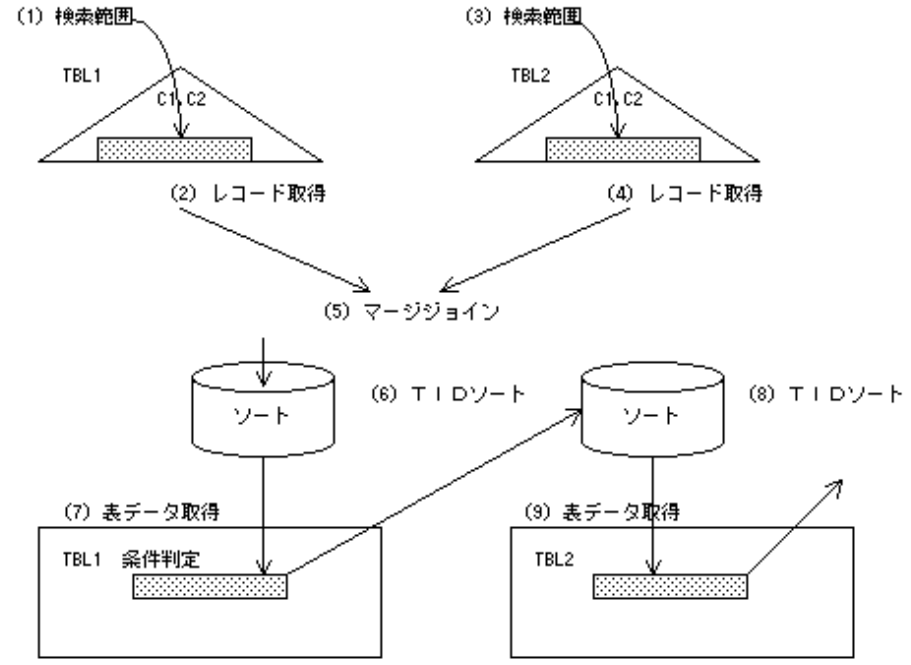
```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : M-JOIN      [TBL1_IX1_DSO       ][                  ][                  ]
                   [TBL2_IX1_DSO       ][                  ][APPL              ]
--------------------------------------------------------------------------------
  [  1] MERGE JOIN ELEMENT
       1) table name       SCM1.TBL1
          scan type        INDEX KEY SCAN(1)
          dso name         TBL1_IX1_DSO         [PAGE/SH]
          join key column       1
```

```
          condition evaluation  No
          scan record number    100
       2) table name       SCM1.TBL2
          scan type        INDEX KEY SCAN(1)
          dso name         TBL2_IX1_DSO         [PAGE/SH]
          join key column       1
          condition evaluation  No
          scan record number    200
[  2] OUTPUT ELEMENT
      record length         22
```

2) マージジョイン(インデックスを利用してジョインをして表データも取得する)

ジョインキーを昇順に読み込む方法として、インデックスを利用します。インデックスだけでは必要な列のすべてを獲得できない表に対して、ジョイン後に各表からデータを取り出すアクセスモデルです。

```
SELECT *
  FROM SCM1.TBL1,SCM1.TBL2
  WHERE TBL1.C1=TBL2.C1
   AND  TBL1.C2=TBL2.C2
   AND  TBL1.C1 BETWEEN 1 AND 10
   AND  TBL1.C3 > 0
```



(1)(3)WHERE探索条件よりインデックス検索範囲を作成し、INDEX KEY SCANを行う。

（インデックスに検索範囲が生成できない場合は、INDEX ALL SCANを行う）

(2)(4)インデックスキー（ジョインキー）の昇順にレコードを取得する。

(5)ソートマージ法でジョインを行う。

(6)TBL1のTIDでソートを行う。

(7)ソートからレコードを取り出し、表をTABLE KEY SCANする。

(8)TBL2のTIDでソートを行う。

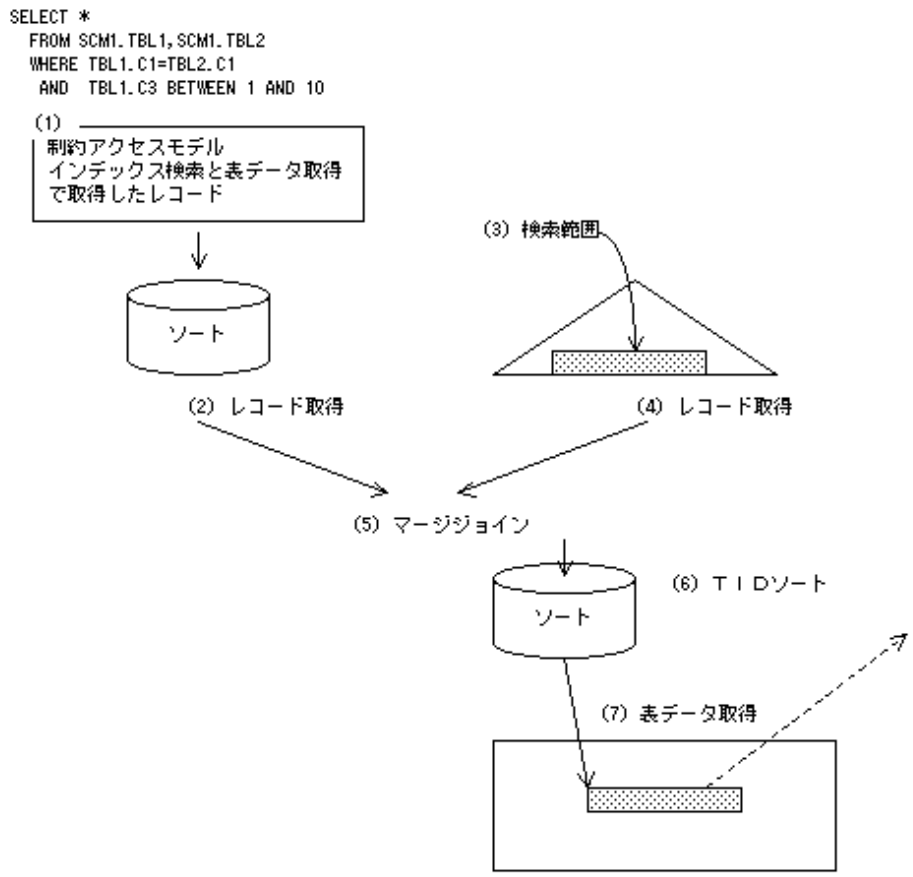
(9) ソートからレコードを取り出し、表をTABLE KEY SCANして、結果データをアプリケーションに返却する。

[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : M-JOIN      [TBL1_IX1_DSO      ][                  ][                  ]
                   [TBL2_IX1_DSO      ][                  ][SORT0001          ]
--------------------------------------------------------------------------------
  [  1] MERGE JOIN ELEMENT
        1) table name       SCM1.TBL1
           scan type        INDEX KEY SCAN(1)
           dso name         TBL1_IX1_DSO        [PAGE/SH]
           join key column       2
           condition evaluation  No
           scan record number    100
        2) table name       SCM1.TBL2
           scan type        INDEX KEY SCAN(1)
           dso name         TBL2_IX1_DSO        [PAGE/SH]
           join key column       2
           condition evaluation  No
           scan record number    200
  [  2] INSERT ELEMENT
        table name       SORT0001
        insert record length  24
--------------------------------------------------------------------------------
   2 : SCAN        [SORT0001          ][TBL1_BASE_DSO     ][SORT0002          ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SORT0001
        scan type        TABLE ALL SCAN
        condition evaluation  No
  [  2] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        TABLE KEY SCAN
        dso name         TBL1_BASE_DSO       [PAGE/SH]
        condition evaluation  Yes
        scan record number    1
  [  3] INSERT ELEMENT
        table name       SORT0002
        insert record length  88
--------------------------------------------------------------------------------
   3 : SCAN        [SORT0002          ][TBL2_BASE_DSO     ][APPL              ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SORT0002
        scan type        TABLE ALL SCAN
        condition evaluation  No
  [  2] SCAN ELEMENT
        table name       SCM1.TBL2
        scan type        TABLE KEY SCAN
        dso name         TBL2_BASE_DSO       [PAGE/SH]
        condition evaluation  No
        scan record number    1
  [  3] OUTPUT ELEMENT
        record length         182
```

3) マージジョイン(ソートテーブルを利用してジョインする)

ジョインするいくつかの表を制約アクセスモデルを利用して結果を求めておき、その結果をジョインキーでソートしてからマージジョインを行うアクセスモデルです。



(1) 制約アクセスモデルを利用して得られたレコードをソートテーブルにINSERTする。

(2) 検索範囲を生成できないのでINDEX ALL SCANを行う。

(3)(4)ソートテーブルとインデックスからジョインキーの昇順にレコードを取り出す。

(5) ソートマージ法でジョインを行う。

(6) TBL2のTIDでソートを行う。

(7) ソートからレコードを取り出し、表をTABLE KEY SCANして、結果データをアプリケーションに返却する。

[アクセスプランの出力例]

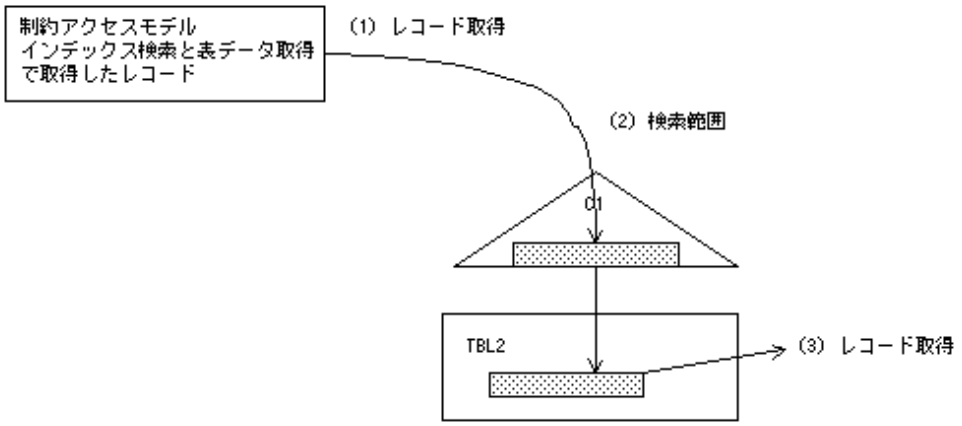
```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : SCAN        [TBL1_IX1_DSO      ][                  ][SORT0002          ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        INDEX KEY SCAN(1)
        dso name         TBL1_IX1_DSO        [PAGE/SH]
        condition evaluation  No
        scan record number    100
  [  2] INSERT ELEMENT
        table name       SORT0002
        insert record length  12
--------------------------------------------------------------------------------
   2 : SCAN        [SORT0002          ][TBL1_BASE_DSO     ][SORT0001          ]
--------------------------------------------------------------------------------
```

```
  [  1] SCAN ELEMENT
        table name       SORT0002
        scan type        TABLE ALL SCAN
        condition evaluation  No
  [  2] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        TABLE KEY SCAN
        dso name         TBL1_BASE_DSO       [PAGE/SH]
        condition evaluation  No
        scan record number    1
  [  3] INSERT ELEMENT
        table name       SORT0001
        insert record length  88
--------------------------------------------------------------------------------
   3 : M-JOIN      [SORT0001          ][                 ][                 ]
                   [TBL2_IX1_DSO      ][                 ][SORT0003         ]
--------------------------------------------------------------------------------
  [  1] MERGE JOIN ELEMENT
        1) table name       SORT0001
           scan type        TABLE ALL SCAN
           join key column       1
           condition evaluation  No
        2) table name       SCM1.TBL2
           scan type        INDEX ALL SCAN
           dso name         TBL2_IX1_DSO       [DSI/SH]
           join key column       1
           condition evaluation  No
           scan record number    1000
  [  2] INSERT ELEMENT
        table name       SORT0003
        insert record length  100
--------------------------------------------------------------------------------
   4 : SCAN        [SORT0003          ][TBL2_BASE_DSO     ][APPL             ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SORT0003
        scan type        TABLE ALL SCAN
        condition evaluation  No
  [  2] SCAN ELEMENT
        table name       SCM1.TBL2
        scan type        TABLE KEY SCAN
        dso name         TBL2_BASE_DSO       [PAGE/SH]
        condition evaluation  No
        scan record number    1
  [  3] OUTPUT ELEMENT
        record length         182
```

4) フェッチジョイン(インデックスを利用してジョインする)

制約アクセスモデルにより読み込んだデータを元に、結合する表のインデックスをキー値検索してジョインを行うアクセスモデルです。

```
SELECT *
  FROM SCM1.TBL1,SCM1.TBL2
  WHERE TBL1.C1=TBL2.C1
   AND  TBL1.C3 BETWEEN 1 AND 3
```



(1) 制約アクセスモデルを利用して得られたレコードを取り出す。

(2) インデックスの検索範囲を作成し、INDEX TABLE SCANを行う。

(3) 結果データをアプリケーションに返却する。

[アクセスプランの出力例]

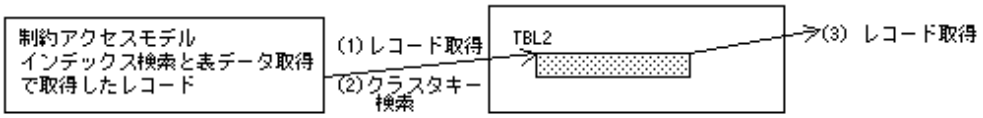
```
================================================================================
Main query
=sno===sectname=====input1=============input2=============output/update======
   1 : SCAN        [TBL1_IX1_DSO     ][                 ][SORT0001          ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        INDEX KEY SCAN(1)
        dso name         TBL1_IX1_DSO        [PAGE/SH]
        condition evaluation  No
        scan record number    5
  [  2] INSERT ELEMENT
        table name       SORT0001
        insert record length  12
--------------------------------------------------------------------------------
   2 : SCAN        [SORT0001         ][                 ][                  ]
       F-JOIN      [TBL1_BASE_DSO    ][                 ][                  ]
                   [TBL2_IX1_DSO     ][TBL2_BASE_DSO    ][APPL              ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SORT0001
        scan type        TABLE ALL SCAN
        condition evaluation  No
  [  2] FETCH JOIN ELEMENT
        1) table name       SCM1.TBL1
           scan type        TABLE KEY SCAN
           dso name         TBL1_BASE_DSO       [PAGE/SH]
           condition evaluation  No
           scan record number    1
        2) table name       SCM1.TBL2
           scan type        INDEX TABLE SCAN(1)
           dso name         TBL2_IX1_DSO        [PAGE/SH]
                            TBL2_BASE_DSO       [PAGE/SH]
```

```
          condition evaluation  No
          scan record number    3
  [  3] OUTPUT ELEMENT
        record length         182
```

5) フェッチジョイン(クラスタキーを利用してジョインする)

制約アクセスモデルにより読み込んだデータを元に、結合する表をクラスタキーで検索してジョインを行うアクセスモデルです。

```
SELECT *
  FROM SCM1.TBL1,SCM1.TBL2
  WHERE TBL1.C1=TBL2.C1
   AND  TBL1.C2 =1
```



(1) 制約アクセスモデルを利用して得られたレコードを取り出す。

(2) クラスタキー値を作成し、TBL2に対してTABLE KEY SCANを行う。

(3) 結果データをアプリケーションに返却する。

[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : F-JOIN      [TBL1_IX1_DSO      ][TBL1_BASE_DSO     ][                  ]
                   [TBL2_BASE_DSO     ][                  ][APPL              ]
--------------------------------------------------------------------------------
  [  1] FETCH JOIN ELEMENT
        1) table name       SCM1.TBL1
           scan type        INDEX TABLE SCAN(1)
           dso name         TBL1_IX1_DSO        [PAGE/SH]
                            TBL1_BASE_DSO       [PAGE/SH]
           condition evaluation  No
           scan record number    1
        2) table name       SCM1.TBL2
           scan type        TABLE KEY SCAN
           dso name         TBL2_BASE_DSO       [PAGE/SH]
           condition evaluation  No
           scan record number    1
  [  2] OUTPUT ELEMENT
        record length         182
```

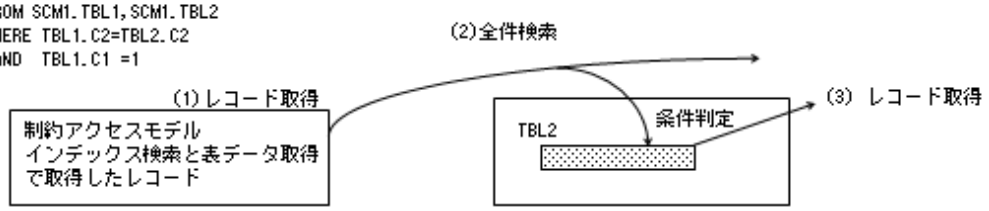
6) フェッチジョイン(実表を直接読み込んでジョインする)

制約アクセスモデルにより読み込んだデータごとに、結合する表を直接読み込みながらジョインを行うアクセスモデルです。

注意

本アクセスモデルは、ASSIST指定のLEADING_TABLEと、FETCHを指定したJOIN_RULEを同時に指定し、結合される側の表のインデックスやクラスタキーが使えない場合に有効になります。

```
SELECT /* ASSIST LEADING_TABLE(TBL1,TBL2) JOIN_RULE(FETCH) */ *
  FROM SCM1.TBL1,SCM1.TBL2
  WHERE TBL1.C2=TBL2.C2
   AND  TBL1.C1 =1
```



(1) 制約アクセスモデルを利用して得られたレコードを取り出す。

(2) TBL2についてTABLE ALL SCANを行う。

(3) 結果データをアプリケーションに返却する。

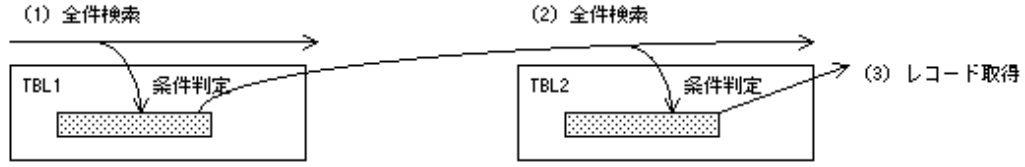
[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1=============input2=============output/update======
   1 : SCAN        [TBL1_IX1_DSO      ][                  ][SORT0001          ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        INDEX KEY SCAN(1)
        dso name         TBL1_IX1_DSO        [PAGE/SH]
        condition evaluation  No
        scan record number    5
  [  2] INSERT ELEMENT
        table name       SORT0001
        insert record length  12
--------------------------------------------------------------------------------
   2 : SCAN        [SORT0001          ][                  ][                  ]
       F-JOIN      [TBL1_BASE_DSO     ][                  ][                  ]
                   [TBL2_BASE_DSO     ][                  ][APPL              ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SORT0001
        scan type        TABLE ALL SCAN
        condition evaluation  No
  [  2] FETCH JOIN ELEMENT
        1) table name       SCM1.TBL1
           scan type        TABLE ALL SCAN
           dso name         TBL1_BASE_DSO        [DSI/SH]
           condition evaluation  Yes
           scan record number    1
        2) table name       SCM1.TBL2
           scan type        TABLE ALL SCAN
           dso name         TBL2_BASE_DSO        [DSI/SH]
           condition evaluation  Yes
           scan record number    1
  [  2] OUTPUT ELEMENT
        record length         182
```

7) ネスティドループジョイン(実表を直接読み込んでジョインする)

データベースの表を直接読み込みながら、レコードの組合わせを作成するアクセスモデルです。

```
SELECT *
  FROM SCM1.TBL1,SCM1.TBL2
  WHERE TBL1.C1>TBL2.C1
   AND  TBL1.C2 = 'A'
```



(1) TBL1についてTABLE ALL SCANを行う。

(2) (1)のレコードごとに、TBL2についてTABLE ALL SCANを行う。

(3) 結果データをアプリケーションに返却する。

[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : N-JOIN      [TBL1_BASE_DSO     ][                  ][                  ]
                   [TBL2_BASE_DSO     ][                  ][APPL              ]
--------------------------------------------------------------------------------
  [  1] NESTED LOOP JOIN ELEMENT
        1) table name       SCM1.TBL1
           scan type        TABLE ALL SCAN
           dso name         TBL1_BASE_DSO        [DSI/SH]
           condition evaluation  Yes
           scan record number    1
        2) table name       SCM1.TBL2
           scan type        TABLE ALL SCAN
           dso name         TBL2_BASE_DSO        [DSI/SH]
           condition evaluation  Yes
           scan record number    1
  [  2] OUTPUT ELEMENT
        record length         182
```

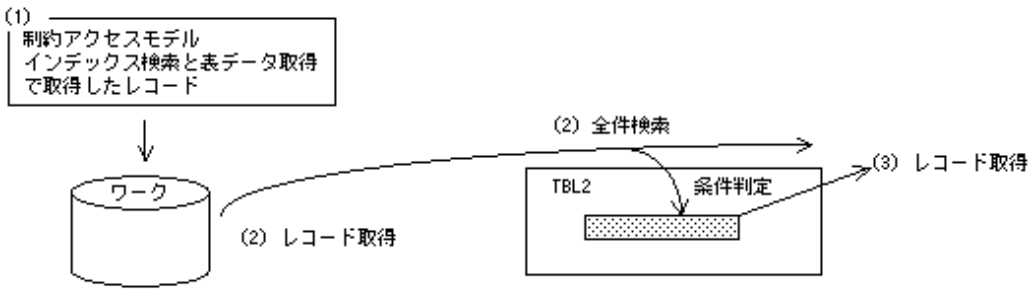
8) ネスティドループジョイン(ワークテーブルを利用してジョインする)

制約アクセスモデルまたはジョイン系アクセスモデルの結果データをワークテーブルに格納し、そのワークテーブルを読み込みながらレコードの組合わせを作成するアクセスモデルです。

参考

以下の例では、ジョインする表が1)WORK0001、2)SCM1.TBL2と並んでいますが、1)SCM1.TBL2、2)WORK0001のように、ジョインする表の順序が入れ替わることもあります。

```
SELECT TBL2.C2
  FROM SCM1.TBL1,SCM1.TBL2
  WHERE TBL1.C1>TBL2.C1
   AND  TBL1.C2 = 1
```



(1) 制約アクセスモデルを利用して得られたレコードをワークテーブルにINSERTする。

(2) ワークテーブルからレコードを取り出しながらレコードごとにTBL2をTABLE ALL SCANする。

(3) 結果データをアプリケーションに返却する。

[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : SCAN        [TBL1_IX1_DSO      ][TBL1_BASE_DSO     ][WORK0001          ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        INDEX TABLE SCAN(1)
        dso name         TBL1_IX1_DSO         [PAGE/SH]
                         TBL1_BASE_DSO        [PAGE/SH]
        condition evaluation  No
        scan record number    1
[ 2] INSERT ELEMENT
        table name       WORK0001
        insert record length  4
--------------------------------------------------------------------------------
   2 : N-JOIN      [WORK0001          ][                  ][                  ]
                   [TBL2_BASE_DSO     ][                  ][APPL              ]
--------------------------------------------------------------------------------
  [  1] NESTED LOOP JOIN ELEMENT
        1) table name       WORK0001
           scan type        TABLE ALL SCAN
           condition evaluation  No
        2) table name       SCM1.TBL2
           scan type        TABLE ALL SCAN
           dso name         TBL2_BASE_DSO        [DSI/SH]
           condition evaluation  Yes
           scan record number    50
  [  2] OUTPUT ELEMENT
        record length         4
```

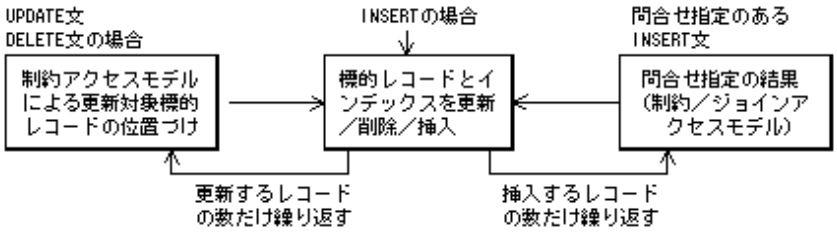
参考

ワークテーブルだけを利用してネスティドループジョインを行うアクセスモデルもあります。

この場合は、上記例におけるTBL2を制約アクセスモデルを利用してワークテーブルに格納して、ワークテーブルを読み込みながらネスティドループジョインすると考えてください。

H.1.3 更新系アクセスモデル

更新系のアクセスモデルは、更新する標的レコードを位置づけるまでの部分と、標的レコードを更新する部分から構成されます。



標的レコードを位置づける部分

標的レコードを位置づける部分は、制約系アクセスモデルのいずれかを選択します。ただし、インデックス検索のアクセスモデル(表データを取得しないアクセスモデル)は、更新する標的レコードを位置づけるまでの制約アクセスモデルとはなりません。なぜなら、更新処理は必ず表のデータを更新するからです。また、並列スキャンのアクセスモデルも、UPDATE文:探索またはDELETE文:探索のアクセスプランとして選択されません。

[UPDATE文:探索またはDELETE文:探索で更新標的レコードを位置づけるまでの部分]

制約アクセスモデル名
ROW_ID検索
クラスタキー検索
インデックス検索
TIDユニオンマージ
表の全件検索

UPDATE文:位置づけまたはDELETE文:位置づけでは、カーソルにより位置づけられた現在行を更新または削除します。標的レコードの位置づけにはTABLE KEY SCANのアクセス方式を使用します。

標的レコードを更新する部分

標的レコードの更新は、データベースの表のレコードの更新(更新、削除または挿入)と、インデックスのレコードの更新(更新、削除または挿入)を行います。インデックスは次の場合に更新されます。

- INSERT文を実行した場合

  その表に定義しているすべてのインデックスにレコードが挿入されます。

- DELETE文を実行した場合

  その表に定義しているすべてのインデックスから表に対応するレコードが削除されます。

- UPDATE文を実行した場合

  ― UPDATE文のSET句にインデックスを構成する列が指定されているインデックスが更新されます。

  ― 表のレコードの位置が物理的に移動した場合に、TIDを更新するため、その表に定義しているすべてのインデックスが更新されます。

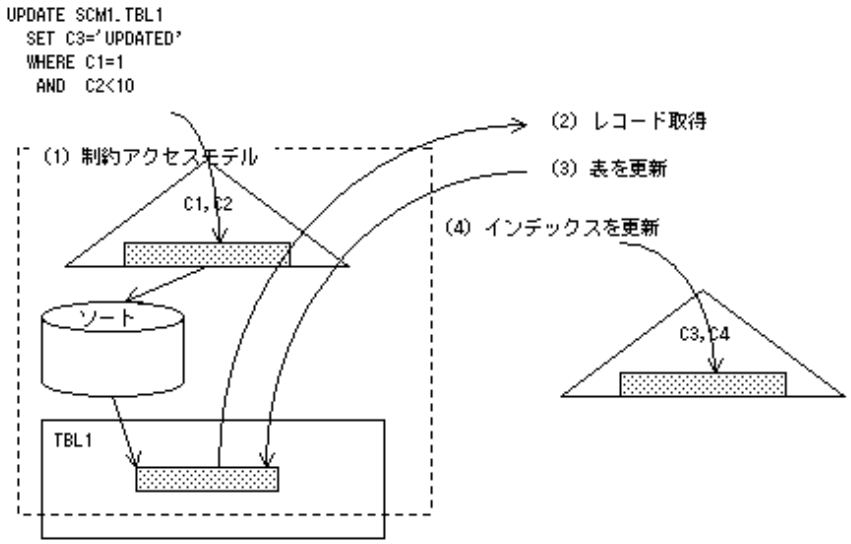
参考

表のレコードの位置は、次の操作を行った場合に移動することがあります。

- 表の分割条件に指定した列を更新する場合
- 表のクラスタキーを更新する場合
- VARCHAR/NCHAR VARYING/BLOBの列を前より長いデータに更新する場合

1) UPDATE文:探索

WHERE探索条件に指定した条件から、制約アクセスモデルのいずれかを選択して、データベースを更新するアクセスモデルです。



(1) 制約アクセスモデルにより更新標的レコードを位置づける。

（上記の例では、インデックス検索と表データ取得のアクセスモデルを採用している）

(2) 表レコードを取得する。

(3) 表レコードのC3部分を更新する。

(4) 更新前のデータからインデックスの当該レコードを削除して、新しいレコードを挿入する。

[アクセスプランの出力例]

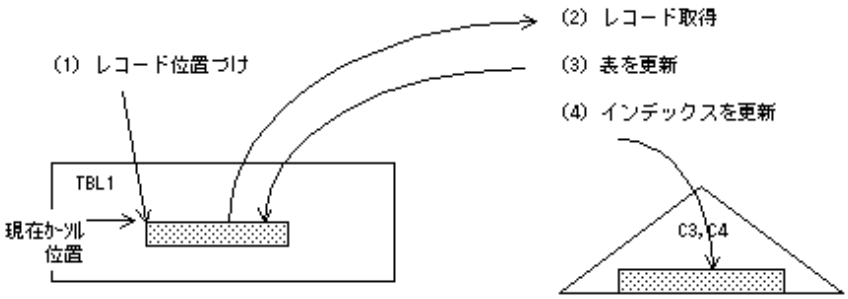
```
================================================================================
Main query
=sno===sectname=====input1=============input2=============output/update======
   1 : SCAN        [TBL1_IX1_DSO      ][                  ][SORT0001          ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        INDEX KEY SCAN(2)
        dso name         TBL1_IX1_DSO        [PAGE/SH]
        condition evaluation  No
        scan record number    100
  [  2] INSERT ELEMENT
        table name       SORT0001
        insert record length  12
--------------------------------------------------------------------------------
   2 : SCAN        [SORT0001          ][TBL1_BASE_DSO     ][                  ]
       UPDATE      [                  ][                  ][TBL1_BASE_DSO     ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
```

```
       table name        SORT0001
       scan type         TABLE ALL SCAN
       condition evaluation    No
  [  2] SCAN ELEMENT
       table name       SCM1.TBL1
       scan type        TABLE KEY SCAN
       dso name         TBL1_BASE_DSO       [PAGE/SH]
       condition evaluation  No
       scan record number    1
  [  3] UPDATE ELEMENT
       table name       SCM1.TBL1
       dso name         TBL1_BASE_DSO       [PAGE/EX]
                        TBL1_IX2_DSO        [PAGE/EX]
```

2) UPDATE文:位置づけ

カーソルにより位置づけられているレコードを更新するアクセスモデルです。

```
UPDATE SCM1.TBL1
  SET C3='UPDATED'
  WHERE CURRENT OF CURSOR01
```



(1) 現在カーソル位置のレコードをTABLE KEY SCANで位置づける。

(2) 表レコードを取得する。

(3) 表レコードのC3部分を更新する。

(4) 更新前のデータからインデックスの当該レコードを削除して、新しいレコードを挿入する。

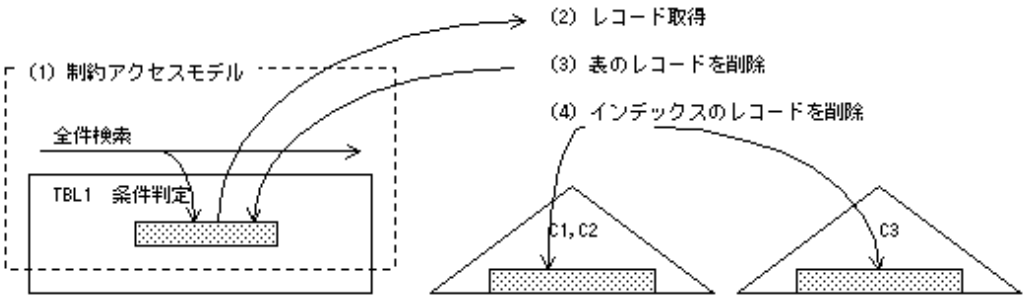
[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : SCAN        [TBL1_BASE_DSO     ][                  ][                  ]
       UPDATE      [                  ][                  ][TBL1_BASE_DSO     ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
       table name       SCM1.TBL1
       scan type        TABLE KEY SCAN
       dso name         TBL1_BASE_DSO       [PAGE/EX]
       condition evaluation  No
       scan record number    1
  [  2] UPDATE ELEMENT
       table name       SCM1.TBL1
       dso name         TBL1_BASE_DSO       [PAGE/EX]
                        TBL1_IX2_DSO        [PAGE/EX]
```

3) DELETE文:探索

WHERE探索条件に指定した条件から、制約アクセスモデルのいずれかを選択して、データベースのレコードを削除するアクセスモデルです。

```
DELETE FROM SCM1.TBL1
  WHERE C4 < DATE '2000-02-01'
```



(1) 制約アクセスモデルにより更新標的レコードを位置づける。

（上記の例では、表の全件検索を採用している）

(2) 表レコードを取得する。

(3) 表レコードを削除する。

(4) 更新前のデータからインデックスの当該レコードを削除する。

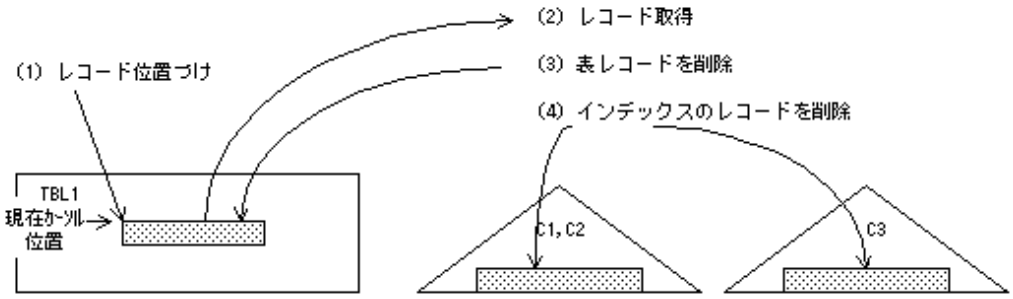
[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : SCAN        [TBL1_BASE_DSO     ][                  ][                  ]
       UPDATE      [                  ][                  ][TBL1_BASE_DSO     ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        TABLE ALL SCAN
        dso name         TBL1_BASE_DSO       [DSI/SH]
        condition evaluation  Yes
        scan record number    100
  [  2] DELETE ELEMENT
        table name       SCM1.TBL1
        dso name         TBL1_BASE_DSO       [DSI/EX]
                         TBL1_IX1_DSO        [DSI/EX]
                         TBL1_IX2_DSO        [DSI/EX]
```

4) DELETE文:位置づけ

カーソルにより位置づけられているレコードを削除するアクセスモデルです。

```
DELETE FROM SCM1.TBL1
  WHERE CURRENT OF CURSOR 01
```



(1) 現在カーソル位置のレコードをTABLE KEY SCANで位置づける。

(2) 表レコードを取得する。

(3) 表レコードを削除する。

(4) 更新前のデータからインデックスの当該レコードを削除する。

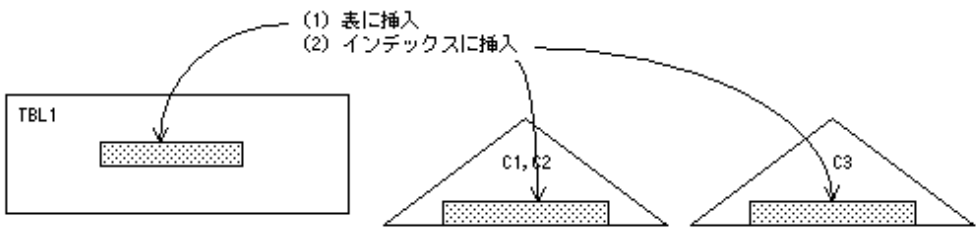
[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : SCAN        [TBL1_BASE_DSO     ][                  ][                  ]
       UPDATE      [                  ][                  ][TBL1_BASE_DSO     ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL1
        scan type        TABLE KEY SCAN
        dso name         TBL1_BASE_DSO       [PAGE/EX]
        condition evaluation  No
        scan record number    1
  [  2] DELETE ELEMENT
        table name       SCM1.TBL1
        dso name         TBL1_BASE_DSO       [PAGE/EX]
                         TBL1_IX1_DSO        [PAGE/EX]
                         TBL1_IX2_DSO        [PAGE/EX]
```

5) 挿入値リストを指定したINSERT文

データベースの表に挿入値リストで指定したレコードを挿入するアクセスモデルです。

```
INSERT INTO SCM1.TBL1(C1,C2,C3,C4,C5)
  VALUES (1,1,'INSERT_REC',DATE'2000-01-10',1)
```



(1) 表のレコードを作成し、表に挿入する。

(2) インデックスのレコードを作成し、インデックスに挿入する。

[アクセスプランの出力例]

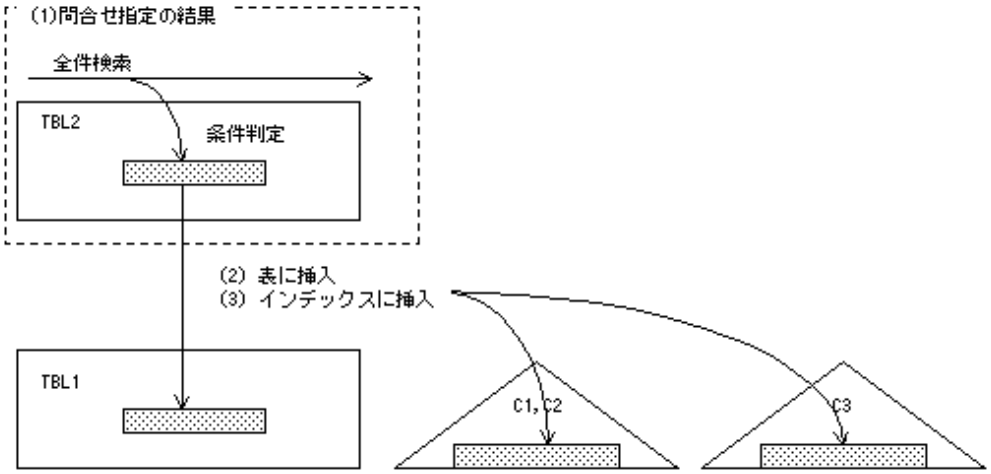
```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : UPDATE      [                  ][                  ][TBL1_BASE_DSO     ]
--------------------------------------------------------------------------------
  [  1] INSERT ELEMENT
        table name       SCM1.TBL1
        dso name         TBL1_BASE_DSO       [PAGE/EX]
                         TBL1_IX1_DSO        [PAGE/EX]
                         TBL1_IX2_DSO        [PAGE/EX]
```

6) 問合せ指定を指定したINSERT文

データベースの表に問合せ指定の結果を挿入するアクセスモデルです。

問合せ指定の結果は、制約系アクセスモデルやジョイン系アクセスモデルおよびその組合わせにより求められた結果です。

```
INSERT INTO SCM1.TBL1(C1,C2,C3,C4,C5)
  SELECT C1,C2,C3,C4,C5 FORM SCM1.TBL2 WHERE C4 < DATE '2000-02-01'
```



(1) 問合せ指定の結果を求める。

(2) 表のレコードを作成し、表に挿入する。

(3) インデックスのレコードを作成し、インデックスに挿入する。

[アクセスプランの出力例]

```
================================================================================
Main query
=sno===sectname=====input1==============input2==============output/update======
   1 : SCAN        [TBL2_BASE_DSO     ][                  ][                  ]
       UPDATE      [                  ][                  ][TBL1_BASE_DSO     ]
--------------------------------------------------------------------------------
  [  1] SCAN ELEMENT
        table name       SCM1.TBL2
        scan type        TABLE ALL SCAN
        dso name         TBL2_BASE_DSO       [DSI/SH]
        condition evaluation  Yes
        scan record number    100
  [  2] INSERT ELEMENT
        table name       SCM1.TBL1
        dso name         TBL1_BASE_DSO       [DSI/EX]
```

```
                    TBL1_IX1_DSO        [DSI/EX]
                    TBL1_IX2_DSO        [DSI/EX]
```

H.2 エレメントの種類

エレメントは、データベースアクセスにおいて、データを読み込む、データを加工する、データを出力するなどの処理単位です。

エレメントには、以下のものがあります。

エレメント名	エレメントの処理内容の簡単な説明	参照箇所
SCAN	表、インデックス、ソートテーブルおよびワークテーブルからデータを取り出す。	H.2.1
MERGE JOIN	マージジョインを行う。	H.2.2
FETCH JOIN	フェッチジョインを行う。	H.2.3
NESTED LOOP JOIN	ネスティドループジョインを行う。	H.2.4
INSERT	表、インデックス、ソートテーブルおよびワークテーブルにレコードを挿入する。	H.2.5
DELETE	表、インデックスからレコードを削除する。	H.2.6
UPDATE	表、インデックスのデータを更新する。	H.2.7
OUTPUT	問合せの結果をアプリケーションに返却する。	H.2.8
GROUPING	複数のレコードをグループ化する。	H.2.9
TRIGGER EXECUTION	トリガを実行する。	H.2.10

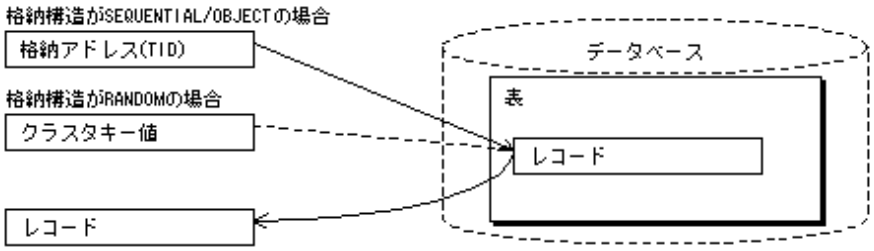
H.2.1 SCANエレメント

SCANエレメントは、データベースやソートテーブル、ワークテーブルからデータを取り出す処理単位です。データの取出し方法には、複数のアクセス方式があります。

アクセス方式	アクセス対象
1) TABLE KEY SCAN	データベースの表
2) TABLE ALL SCAN	データベースの表 ソートテーブル ワークテーブル
3) TABLE PARALLEL SCAN	データベースの表
4) INDEX KEY SCAN	インデックス
5) INDEX ALL SCAN	インデックス
6) INDEX TABLE SCAN	データベースの表とインデックス

1) TABLE KEY SCAN

データベースの表の目的のレコードを、表のクラスタキー値あるいはレコードの格納アドレスで検索するアクセス方式です。レコードの格納アドレスはTID(Tuple ID)と呼びます。



レコードの格納アドレス:

レコードのTIDでレコードを取り出します。TIDは、表の格納構造がSEQUENTIALまたはOBJECTの場合に、インデックスのデータ部に表レコードとの関係付けのために格納されています。

クラスタキー値:

表の格納構造がRANDOMの場合、クラスタキー値を作成して、目的のレコードが格納されているバケットを位置づけて、バケット中のレコード群から目的のレコードを取り出します。クラスタキー値は、RANDOM構造の表の場合に、インデックスのデータ部に表レコードとの関係付けのために格納されています。

SQL文のWHERE探索条件に表のクラスタキー値を一意に指定するような条件を指定すると、指定した条件からクラスタキー値を作成して、TABLE KEY SCAN(クラスタキー値検索)を行うことができます。

例1

クラスタキー値を利用したTABLE KEY SCANができる場合(TBL1の格納構造はRANDOMで、クラスタキーを(C1,C2)とします。)

```
SELECT *
 FROM TBL1
 WHERE C1 = 1.....(注)
   AND C2 = 1.....(注)
   注)クラスタキー値を作成してTABLE KEY SCAN
```

例2

クラスタキー値を利用したTABLE KEY SCANができない場合(TIの格納構造はRANDOMで、クラスタキーを(C1,C2)とします。)

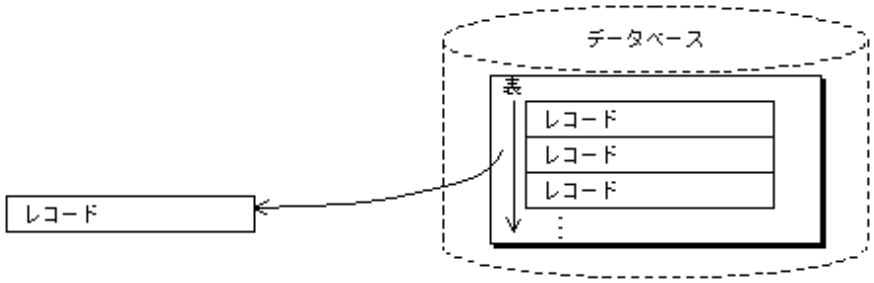
```
SELECT *
 FROM TBL1
 WHERE C1 = 1.....(注)
   AND C2 > 1.....(注)
   注)条件を“=”またはIN述語で与えないと、クラスタキー値を一意に指定できません。
```

2) TABLE ALL SCAN

表やソートテーブルまたはワークテーブルの全レコードを取り出すアクセス方式です。

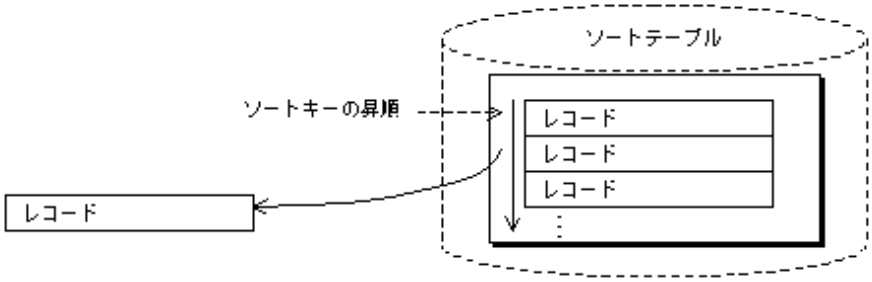
検索対象がデータベースの表である場合

データベースに物理的に格納されている順番に表の全レコードを読み込みながら、指定されたWHERE探索条件を判定して条件が成立するレコードを取り出します。



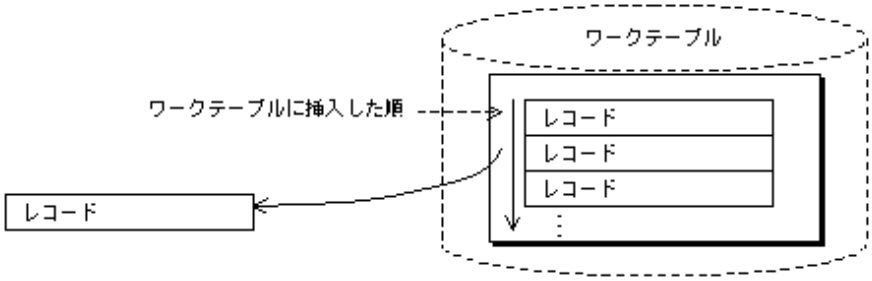
検索対象がソートテーブルである場合

ソートテーブルからソートキーの昇順となるようにレコードを取り出します。



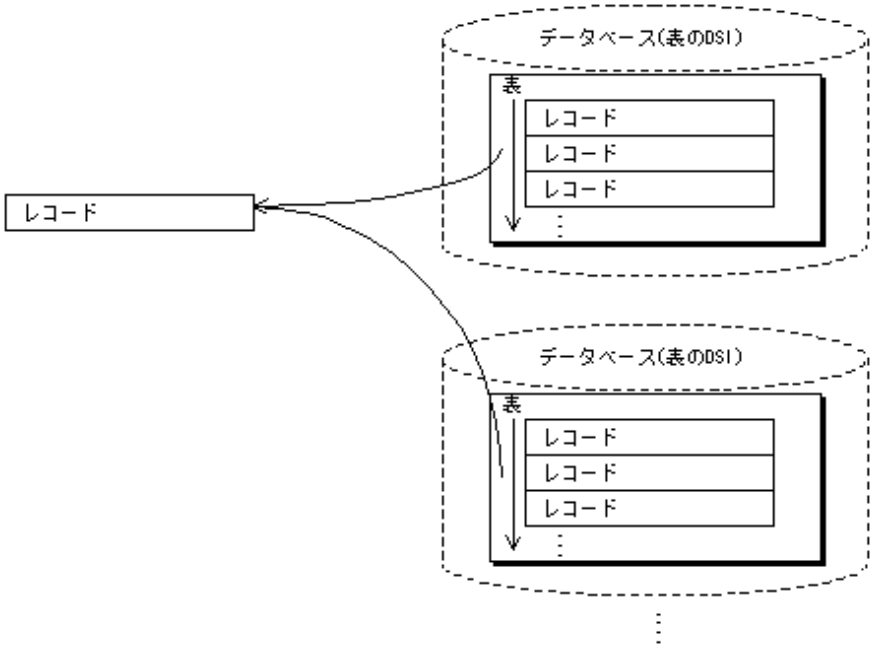
検索対象がワークテーブルである場合

ワークテーブルに挿入した順にレコードを取り出します。



3) TABLE PARALLEL SCAN

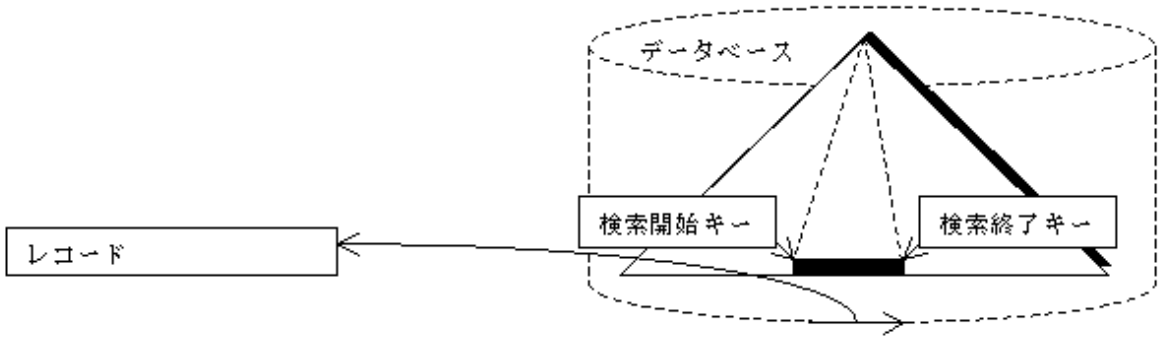
複数のDSIに分割されたデータベースを並列に同時に検索してデータを取り出すアクセス方式です。



4) INDEX KEY SCAN

インデックスに対して、検索範囲(検索開始キーまたは検索終了キー)を指定して、その範囲にあるインデックスデータ部のレコードを検索するアクセス方式です。

インデックスの検索範囲は、SQL文に指定したWHERE探索条件から生成します。



インデックスの検索範囲の検索開始キー値、検索終了キー値の生成は、次のように行います。

なお、下表では列“C1”を先頭に持つインデックスを想定しています。:Hはホスト変数を表現しています。

条件の種別	インデックスの検索範囲	備考
比較述語 =	例 C1 = :H :H <= (C1) <= :H	
比較述語 >(=)	例 C1 >= :H :H <(=) (C1) <= max	
比較述語 <(=)	例 C1 <= :H min <= (C1) <(=) :H	
BETWEEN述語	例 C1 BETWEEN :H1 AND :H2 :H1 <= (C1) <= :H2	

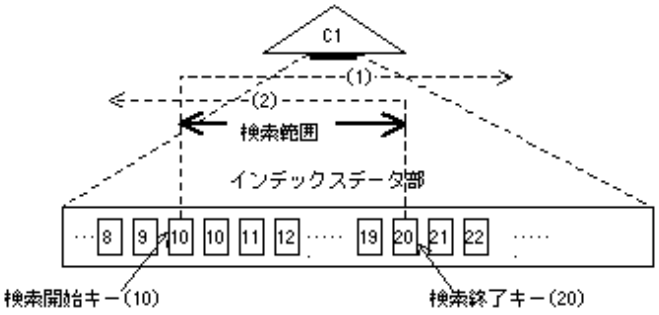
条件の種別	インデックスの検索範囲	備考
IN述語	例 C1 IN (:H1, :H2, ・・) :H1 <= (C1) <= :H1 :H2 <= (C1) <= :H2 ・・	複数範囲となる。
LIKE述語	例 C1 LIKE :H :Hlow <= (C1) <= :Hhigh	“%”、“_”またはESCAPE文字までの文字列より，検索開始キー、検索終了キーを生成する。
NULL述語	例 C1 IS NULL NULLデータ <= (C1) <= NULLデータ	NULL値データを検索する検索開始キー値、終了キー値を生成する。

同一の列に対して複数の条件をブール演算子“AND”でつないだ場合、検索範囲の検索開始キーまたは検索終了キーはデータベースアクセス前にマージされ、作成された範囲のデータのみを検索します。

例1

WHERE C1 >= 10(1) AND C1 <= 20(2)の場合

インデックスの検索範囲は下図のようになる。

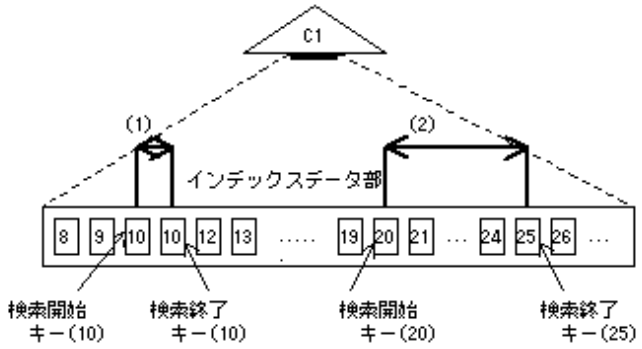


同一の列に対して複数の条件をブール演算子“OR”でつないだ場合、複数の検索範囲が生成され、各検索範囲のデータを検索します。

例2

WHERE C1 = 10(1) OR C1 BETWEEN 20 AND 25(2)の場合

インデックスの検索範囲は下図のようになる。

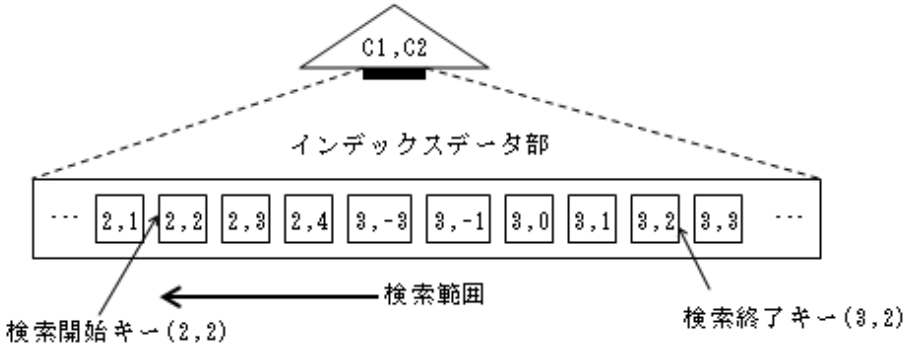


複数列から構成されるインデックスの検索範囲

複数列から構成されるインデックスは、インデックスの構成列(複数の列)でインデックスのキー値を構成し、インデックスのデータ部にはこのキー値の昇順にデータが格納されています。マルチカラムインデックスの検索範囲は、この複数列で検索開始キーおよび検索終了キーを作成して、その範囲のデータを検索します。

例1

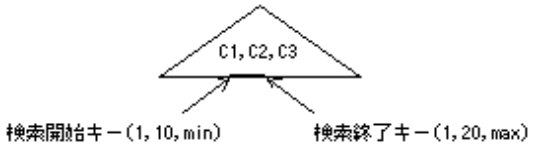
WHERE C1 BETWEEN 2 AND 3 AND C2 = 2の場合



複数列から構成されるインデックスのn番目の構成列が検索範囲として有効でない場合は、その列の取りうる最小値および最大値を補って検索範囲を作成します。

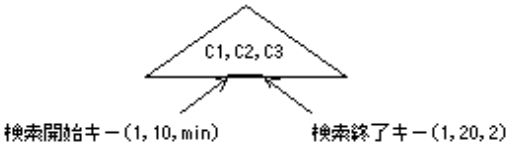
例2

WHERE C1 = 1 AND C2 BETWEEN 10 AND 20の場合



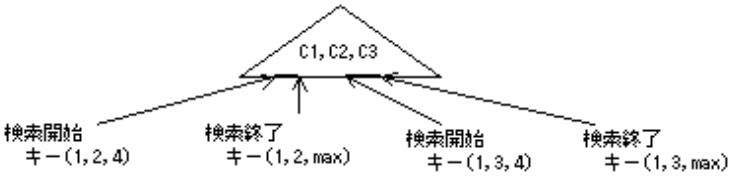
例3

WHERE C1 = 1 AND C2 > 10 AND C2 <=20 AND C3 = 2の場合



例4

WHERE C1 = 1 AND (C2 = 2 OR C2 = 3) AND C3 > 4の場合



例5

WHERE C1 > 1 AND C2 = 1 AND C3 = 1の場合

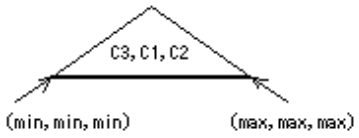
参考

インデックスキー順について

- 第1構成列に条件が指定されないような、複数列から構成されるインデックスは、検索範囲を作成できません。

```
WHERE   C1 =1 AND C2 = 1
```

インデックスの全レコードを検索する必要がある。
このため、このインデックスはINDEX KEY SCANとして使用されない。



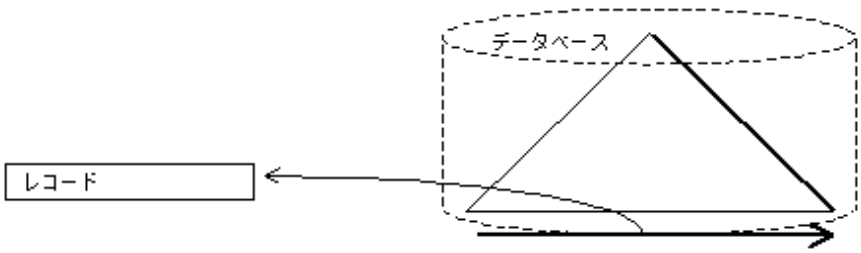
- 比較述語“=”で条件を指定する列を優先的に先頭に配置したほうが、検索するデータ量が少なくなり、効果的です。

```
WHERE   C1 =1 AND C2 BETWEEN 10 AND 20
```

5) INDEX ALL SCAN

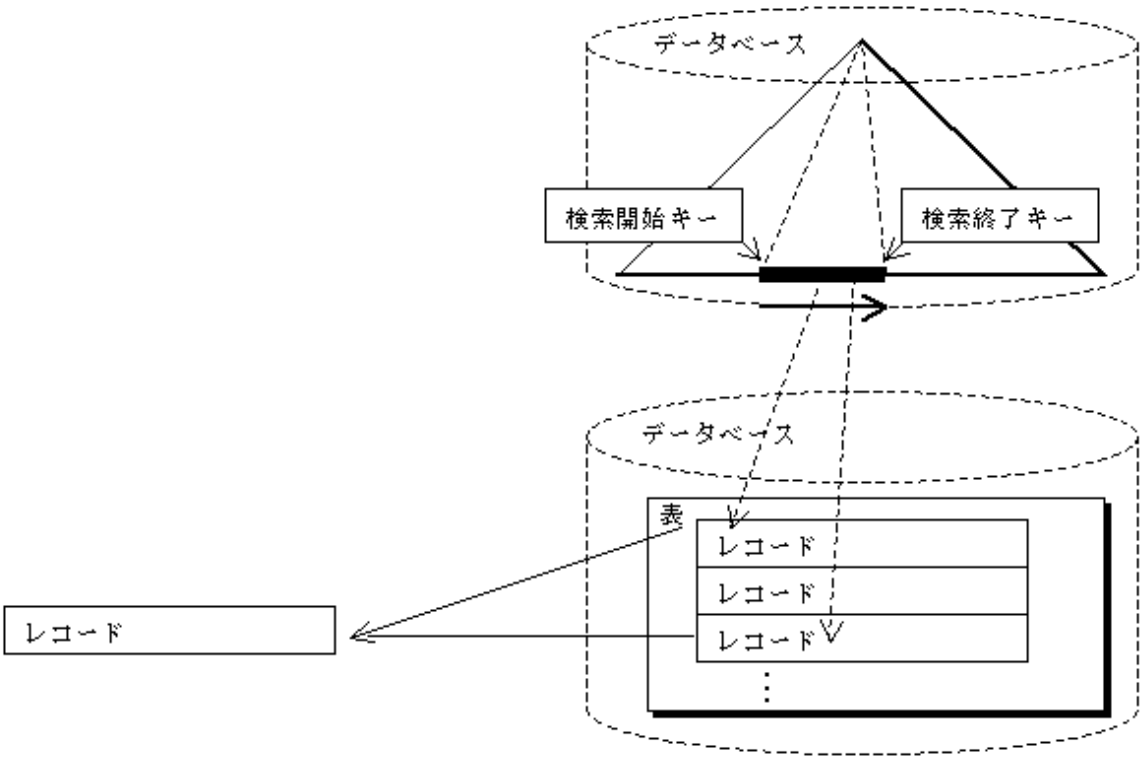
インデックスのデータ部の全データを検索するアクセス方式です。

インデックスのデータ部の全レコードを読み込み、インデックスのキー値の昇順でデータを取り出します。



6) INDEX TABLE SCAN

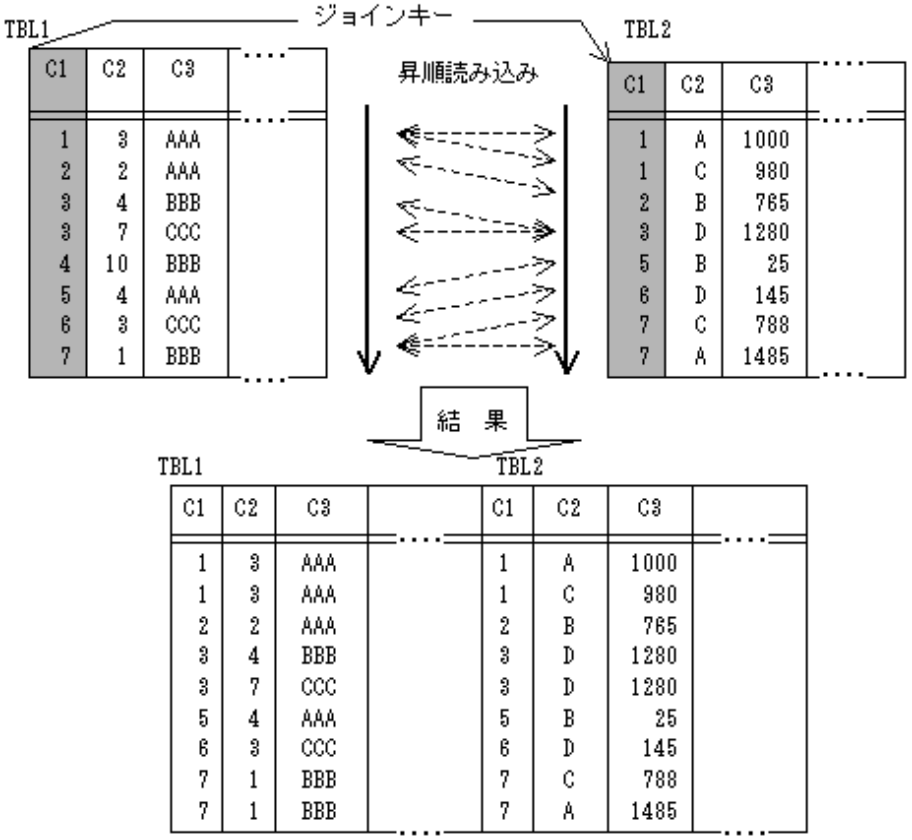
インデックスを範囲検索しながら、同時に対応する表のレコードも取り出すアクセス方式です。



H.2.2 MERGE JOINエレメント

MERGE JOINエレメントは、ソートマージ法で表を結合する処理単位です。ソートマージ法とは、複数の表をジョインキー(ジョインで使用する列)のデータの昇順に読み込み、ジョインキーの等しい行を結合する手法です。ソートマージ法の動作を以下に示します。

```
SELECT  *
 FROM   TBL1,TBL2
 WHERE  TBL1.C1 = TBL2.C1
```



C1	C2	C3
1	3	AAA
2	2	AAA
3	4	BBB
3	7	CCC
4	10	BBB
5	4	AAA
6	3	CCC
7	1	BBB

C1	C2	C3
1	A	1000
1	C	980
2	B	765
3	D	1280
5	B	25
6	D	145
7	C	788
7	A	1485

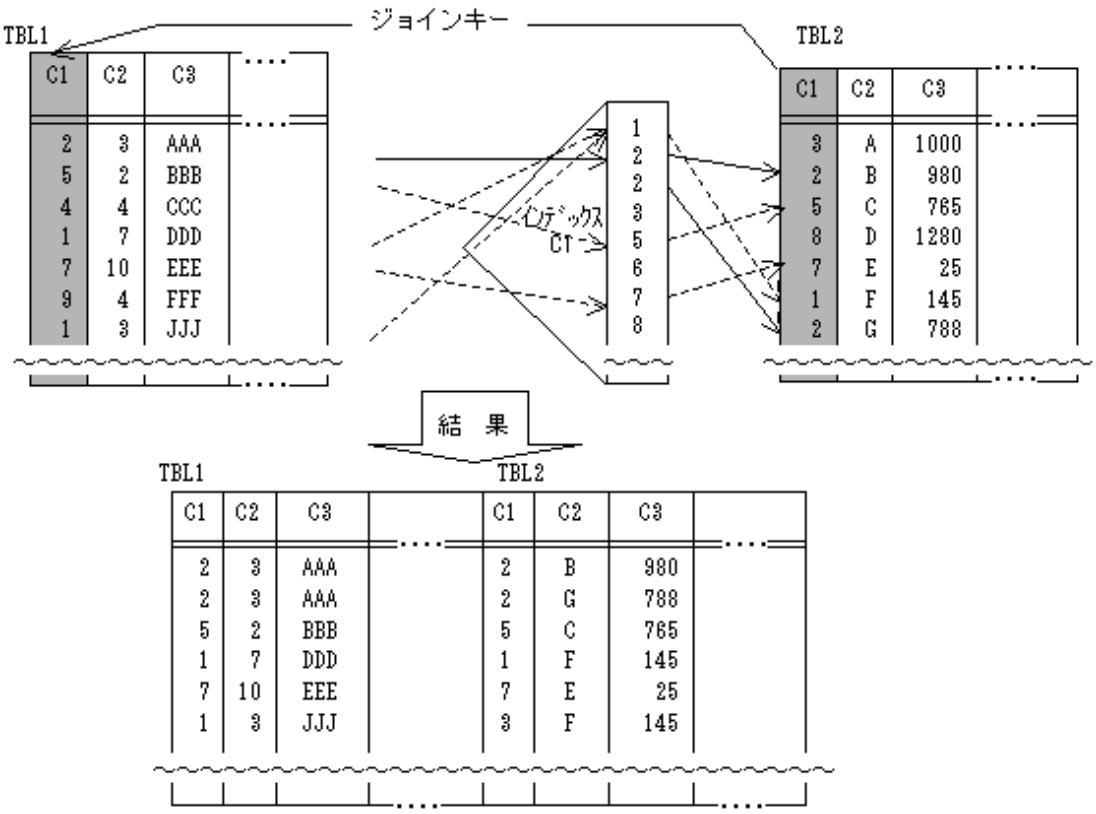
TBL1 C1	TBL1 C2	TBL1 C3	TBL2 C1	TBL2 C2	TBL2 C3
1	3	AAA	1	A	1000
1	3	AAA	1	C	980
2	2	AAA	2	B	765
3	4	BBB	3	D	1280
3	7	CCC	3	D	1280
5	4	AAA	5	B	25
6	3	CCC	6	D	145
7	1	BBB	7	C	788
7	1	BBB	7	A	1485

MERGE JOINエレメントは、ジョインする表のデータをジョインキーの昇順に読み込むため、そのアクセス対象は、インデックス/ソートテーブル/データが昇順に格納されたワークテーブルというバリエーションがあります。

H.2.3 FETCH JOINエレメント

FETCH JOINエレメントは、一方の表のレコードごとに、ジョインする表にアクセスして、ジョインキーの等しいレコードの組合わせを得る処理単位です。一方の表のレコードごとに、ジョインキーで他方の表へのアクセスを繰り返すため、大量データの検索を行う場合に十分な共用バッファがないと、結合される側の表のI/O効率が低下します。一般的には、大量データのジョインには向きません。反面、少量データのジョインには向いています。フェッチジョインの動作を以下に示します。

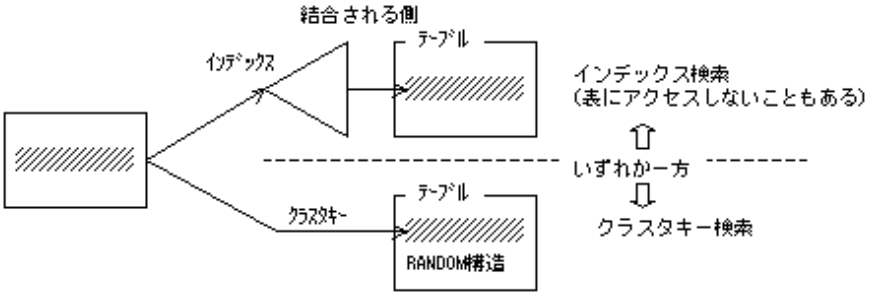
```
SELECT  *
 FROM   TBL1,TBL2
 WHERE  TBL1.C1 = TBL2.C1
```



FETCH JOINエレメントのジョインする表のアクセスモデルには、以下のバリエーションがあります。

- インデックスを利用したアクセスモデル
- クラスタキーを利用したアクセスモデル
- データベースの表を直接読み込む

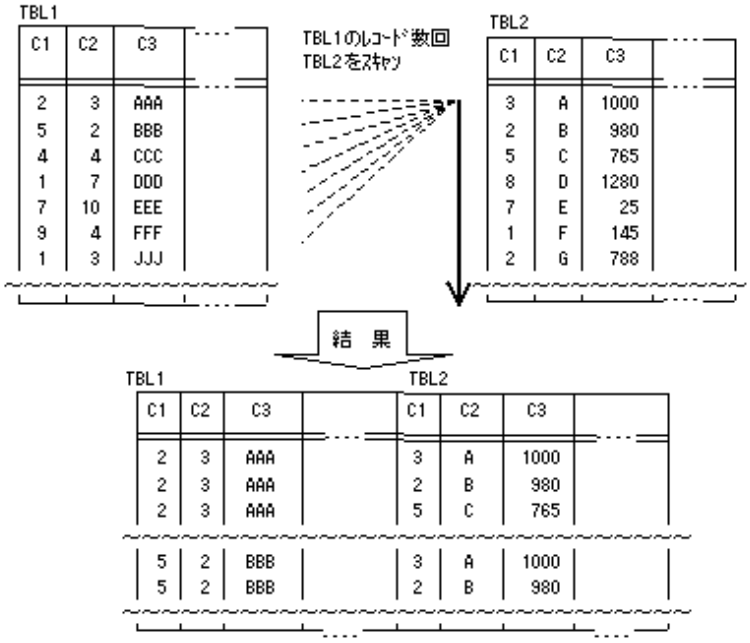
結合される側の表がDSI分割されている場合、探索条件によってアクセスするDSIを決めてからそのDSIのみにダイレクトにアクセスします。



H.2.4 NESTED LOOP JOINエレメント

NESTED LOOP JOINエレメントは、ジョインする表のすべてのレコードを組み合わせて結合を行う処理単位です。このエレメントは、WHERE探索条件や結合表の結合条件に、各表の列を比較述語“=”で結合していない場合に採用します。表のレコードの全組合わせを作成するため、ジョイン結果を求めるのに時間がかかります。NESTED LOOP JOINの動作を以下に示します。

```
SELECT  *
 FROM   TBL1,TBL2
```



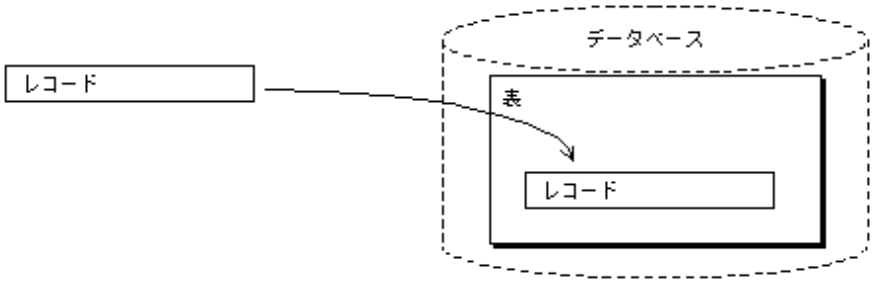
NESTED LOOP JOINエレメントは、データベースの表を直接読み込むか、データベースの表をワークテーブルに格納して、そのワークテーブルを読み込んでジョインするかのバリエーションがあります。

H.2.5 INSERTエレメント

表やインデックス、ソートテーブルまたはワークテーブルにレコードを挿入する処理単位です。

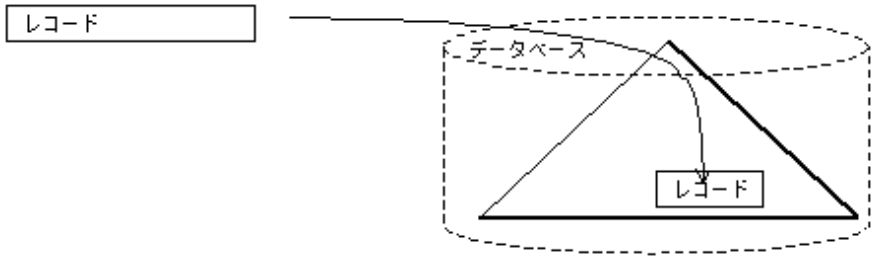
挿入対象がデータベースの表である場合

- SEQUENTIAL構造またはOBJECT構造の表である場合は、基本的にレコードを挿入された順番で格納します。格納オプションで削除領域を再利用する指定がある場合は、空き領域にレコードを格納します。
- RANDOM構造の表である場合は、挿入するレコードのクラスタキー値によりハッシュ関数で求めたバケットにレコードを格納します。



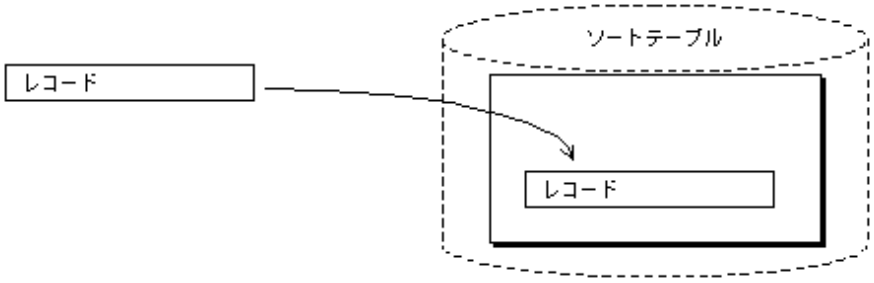
挿入対象がインデックスである場合

インデックスのキー値の順にデータ部のレコードが並ぶ位置にレコードを挿入します。データの挿入に伴い、インデックスデータ部のページスプリットが発生することがあります。



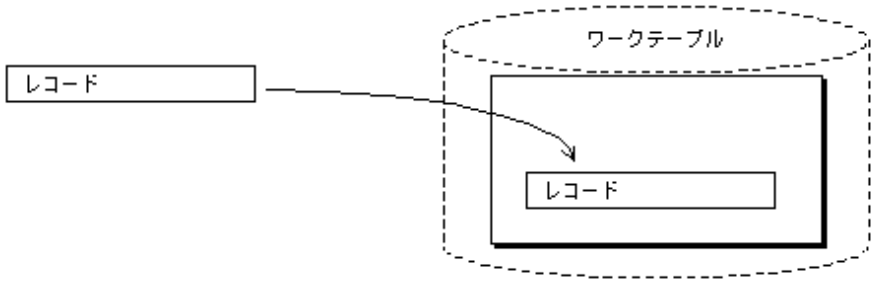
挿入対象がソートテーブルである場合

ソートテーブルへのレコードの挿入は、ソートメモリ上にデータを挿入して、ソートメモリが足りなくなると、部分的にソートされた形で二次記憶に書き出します。



挿入対象がワークテーブルである場合

ワークテーブルへのレコードの挿入は、挿入された順番にレコードを格納します。ワークメモリが足りなくなると、二次記憶に書き出します。

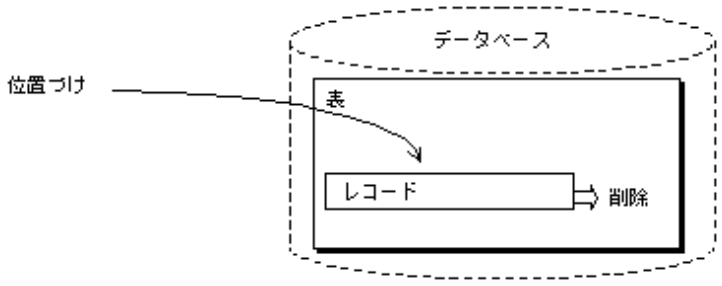


H.2.6 DELETEエレメント

表やインデックスからレコードを削除する処理単位です。

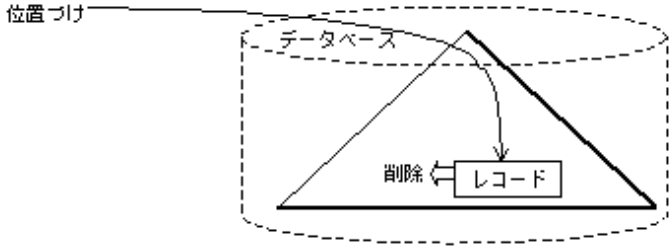
削除対象がデータベースの表である場合

削除対象レコードの位置づけは、TABLE KEY SCANまたはTABLE ALL SCANにより位置づけられます。



削除対象がインデックスである場合

インデックスのレコード削除は、インデックスのデータ部からレコードを削除します。削除対象レコードの位置づけは、削除した表レコードから取り出して作成したインデックスキーと表データとの対応情報(SEQUENTIAL構造またはOBJECT構造の場合はTID、RANDOM構造の場合はクラスタキー値)で行います。

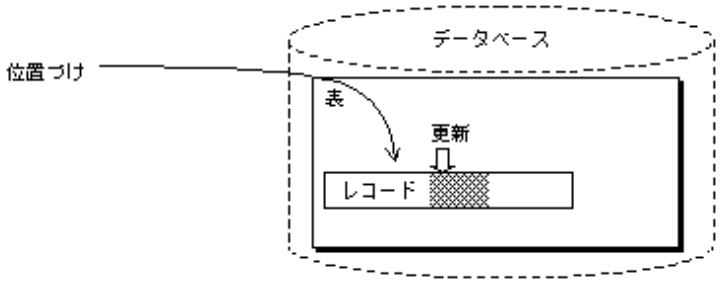


H.2.7 UPDATEエレメント

表やインデックスのレコードの内容を更新する処理単位です。

更新対象がデータベースの表である場合

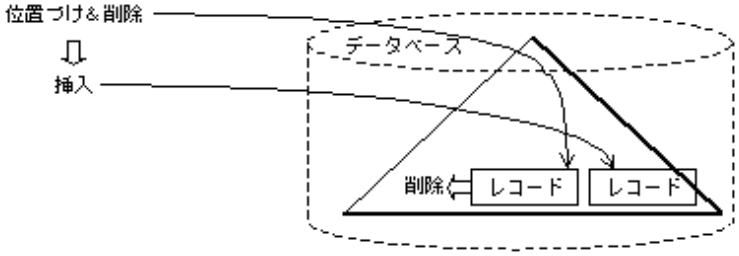
更新対象レコードの位置づけは、TABLE KEY SCANまたはTABLE ALL SCANにより位置づけられます。



更新対象がインデックスである場合

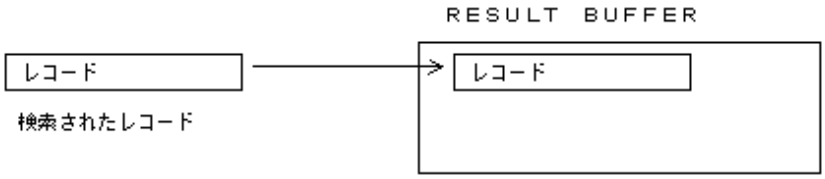
インデックスのレコードの更新は、まず更新対象のインデックスのレコードを削除します。削除するレコードの位置づけは、更新する前の表レコードから取り出して作成したインデックスキーと表データとの対応情報(SEQUENTIAL構造またはOBJECT構造の場合はTID、RANDOM構造の場合はクラスタキー値)で行います。レコードを削除した後、更新された表レコードのデータを元にインデックスキーと表データとの対応情報からインデックスレコードを作成して挿入します。

このように、インデックスの更新は、インデックスのキーを更新していなくても、表のクラスタキーが更新されたり、対応する表レコードの物理的な位置が移動する場合にも動作します。



H.2.8 OUTPUTエレメント

OUTPUTエレメントは検索した結果をアプリケーションに返却する処理単位です。

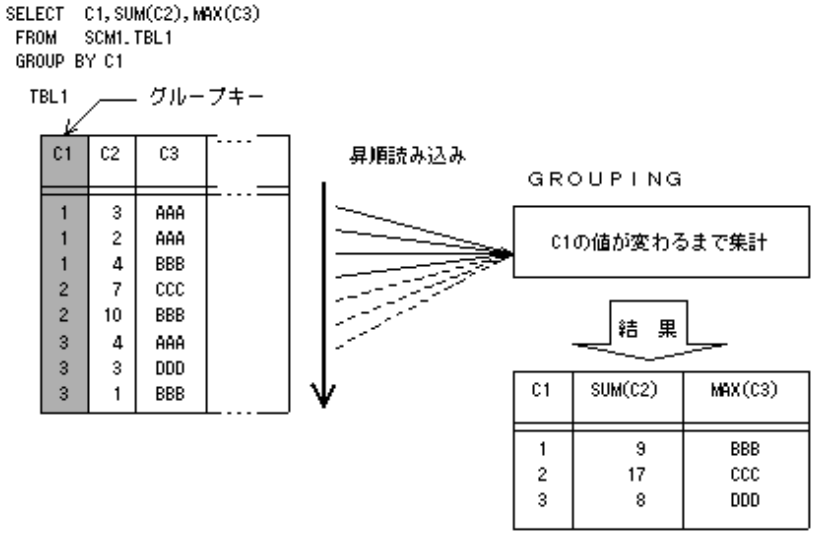


H.2.9 GROUPINGエレメント

GROUPINGエレメントは、問合せ指定にGROUP BY句を指定した場合や、問合せ指定に集合関数を指定した場合に、複数のレコードをグループ化する処理単位です。

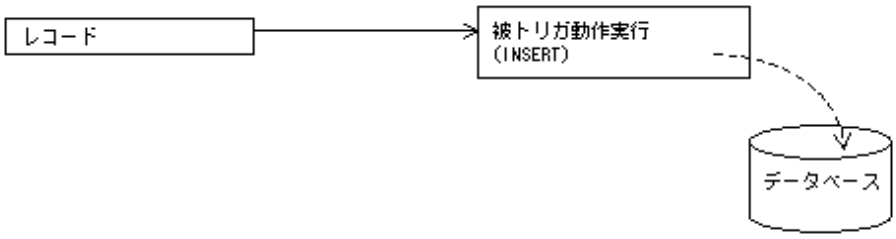
GROUP BYを指定した場合は、GROUP BYで指定した列の昇順にレコードを読み込んで、GROUP BYで指定した列データの値が同じものを1つのグループにまとめます。列の昇順にデータを読み込むには、一般的にはソートテーブルを利用します。

GROUP BYを指定しない場合は、すべてのレコードを1つのグループにまとめます。



H.2.10 TRIGGER EXECUTIONエレメント

TRIGGER EXECUTIONエレメントは、トリガを実行する処理単位です。



付録I　トランザクションと排他制御(データベース簡単運用の場合)

Symfoware/RDBでは、トランザクション単位にデータベースの更新が行われます。複数のトランザクションが同時に同じデータベースを参照または更新すると、データの矛盾が発生してしまいます。このようなことを防ぐために、Symfoware/RDBは、排他制御を行います。ここでは、データベース簡単運用でのトランザクション処理と排他制御について説明します。

I.1　トランザクション制御の概要

SQL文を使用してデータ操作を行った場合に、Symfoware/RDBが操作の内容をどのようにデータベースへ反映するかを説明します。

トランザクションの開始

トランザクションは、Symfoware/RDBにより自動的に開始されます。アプリケーション中のSQL文を実行する場合、トランザクションが開始された状態でなければ、SQL文の処理に先立ってトランザクションが開始されます。トランザクションが開始されていない状態とは、アプリケーションが実行開始したあと、またはトランザクション制御用のSQL文によってトランザクションが終了したあと、SQL文が一度も実行されていない状態です。

SQL文には、トランザクションを開始しないSQL文もあります。

- トランザクションを開始するSQL文
  - ー カーソル宣言以外のデータ操作文
  - ー 動的カーソル宣言以外の動的SQL文
  - ー CALL文
- トランザクションを開始しないSQL文
  - ー トランザクション制御文
  - ー コネクション管理文
  - ー セション管理文
  - ー 埋込み例外宣言
  - ー カーソル宣言
  - ー 動的カーソル宣言
  - ー 資源操作文
  - ー SET USER PASSWORD文
  - ー SET ROLE文

トランザクションの終了

トランザクションは、以下のSQL文を実行することによって、終了させることができます。また、アプリケーションが正常終了すると、自動的に終了します。

表I.1 トランザクションを制御するSQL文

SQL文	機能
COMMIT文	現在のトランザクションを終了します。 トランザクション内でのデータ操作(注)をすべてデータベースに反映します。 オープン中のすべてのカーソルをクローズします。ただし、トランザクションを超越するカーソルはクローズしません。 トランザクション内で獲得したロックをすべて解放します。

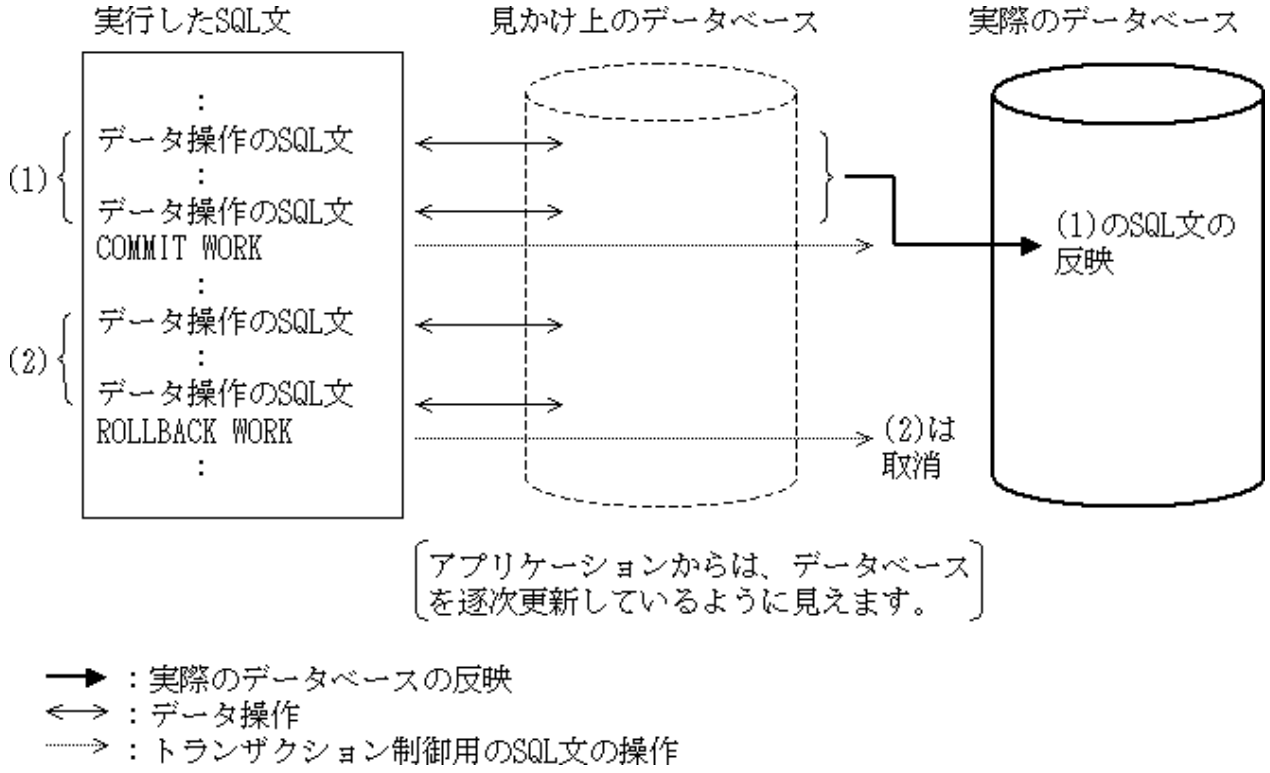
SQL文	機能
ROLLBACK文	現在のトランザクションを終了します。 トランザクション内でのデータ操作(注)はすべて取り消します。 オープン中のすべてのカーソルをクローズします。 トランザクション内で獲得したロックをすべて解放します。

注) データ操作には、挿入、更新および削除があります。

トランザクションを制御する例

トランザクションを制御する例を以下に示します。データ操作のSQL文を実行すると、トランザクションごとに、そのトランザクション内で処理されたデータがデータベースに反映されます。

図I.1 データベースの更新処理



I.2 トランザクションの終了方法

トランザクションの終了方法には、以下の3つの方法があります。

- COMMIT文によるトランザクションの終了
- ROLLBACK文によるトランザクションの終了
- アプリケーションの終了によるトランザクションの終了

COMMIT文によるトランザクションの終了

COMMIT文は、トランザクションの区切りを指定するSQL文です。COMMIT文により行われる処理は、以下のとおりです。

- 処理中のトランザクションを終了し、このトランザクションで行われたデータ操作をデータベースに反映します。
- オープン中のすべてのカーソルをクローズします。ただし、トランザクションを超越するカーソルはクローズしません。
- このトランザクションで獲得した、データベース資源のロックをすべて解放します。

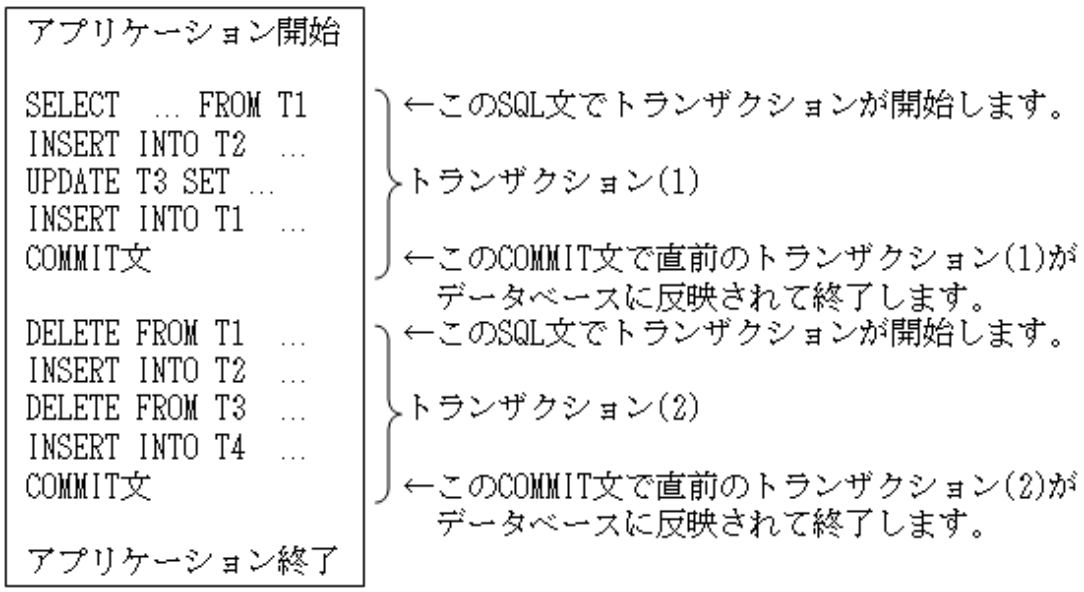
前に実行したCOMMIT文またはROLLBACK文以後、当該COMMIT文までに行ったデータベースの処理をデータベースに反映します。次に、現在のトランザクションを終了します。一度もCOMMIT文またはROLLBACK文を実行していない場合は、アプリケーションの実行開始から当該COMMIT文までのデータベースの処理をデータベースに反映します。

COMMIT文の形式を以下に示します。

```
COMMIT WORK
```

COMMIT文によるデータベース更新の例を以下に示します。

図I.2 COMMIT文によるデータベース更新の例



ROLLBACK文によるトランザクションの終了

ROLLBACK文は、トランザクション内で行ったデータベースの処理を取り消すためのSQL文です。ROLLBACK文により行われる処理を以下に示します。

- 処理中のトランザクションを終了し、このトランザクションで行われたすべての変更を取り消します。データベースは更新されません。
- オープン中のすべてのカーソルをクローズします。
- このトランザクションで獲得した、データベース資源のロックをすべて解放します。

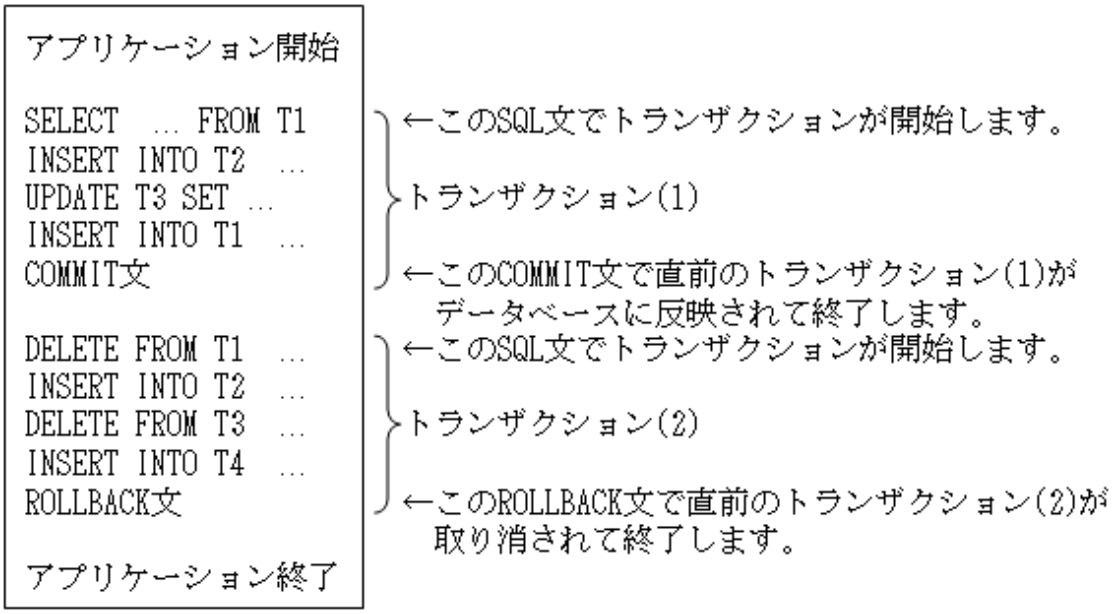
前に実行したCOMMIT文またはROLLBACK文実行以後、当該ROLLBACK文までに行ったデータベースの処理を取り消します。次に、現在のトランザクションを終了します。一度もCOMMIT文またはROLLBACK文を実行していない場合は、アプリケーションの実行開始から当該ROLLBACK文までのデータベースの処理を取り消します。

ROLLBACK文の形式を以下に示します。

```
ROLLBACK WORK
```

ROLLBACK文によるトランザクション取消しの例を以下に示します。

図I.3 ROLLBACK文によるトランザクション取消しの例

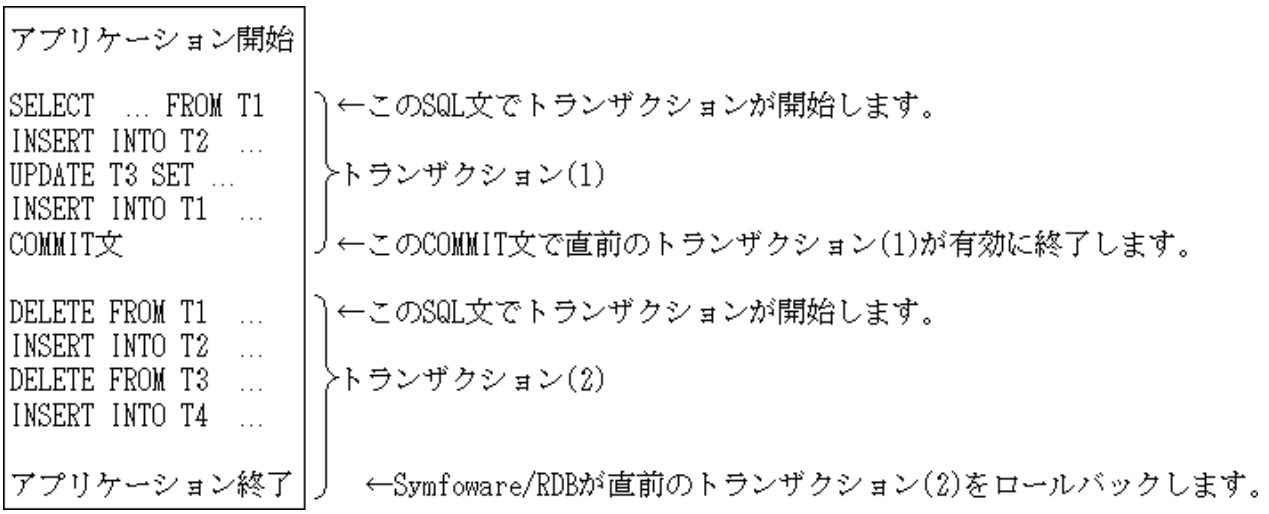


アプリケーションの終了によるトランザクションの終了

アプリケーションがトランザクション開始状態のままCOMMIT文もROLLBACK文も実行せずに終了する場合には、Symfoware/RDBがトランザクションをロールバックします。

トランザクションが開始状態のままアプリケーションが終了する例を以下に示します。

図I.4 トランザクション開始状態でのアプリケーションの終了の例



I.3 トランザクションモード

複数のトランザクションが同じ資源にアクセスする場合、なにも制御を行わないと、データの矛盾が発生してしまいます。このようなことが起こらないように、トランザクションモードを設定し、排他制御を行います。

トランザクションモードには、以下の2つの項目があります。

- アクセスモード
- 独立性水準

I.3.1　アクセスモード

トランザクションのアクセスモードでは、実行できるSQL文の種類を設定します。アクセスモードには、以下の2種類があります。

READ ONLY:

読込み専用モードです。トランザクション内で参照系のSQL文のみ、実行することができます。

READ WRITE:

更新可能モードです。トランザクション内で参照系および更新系のSQL文を実行することができます。

アクセスモードの初期値は、READ_WRITEです。

I.3.2　独立性水準

独立性水準では、あるトランザクションがアクセスしているデータに対し、他のトランザクションによる参照または更新を制御します。また、他のトランザクションが更新中の資源に対し、自トランザクションが参照または更新できるかを制御します。

独立性水準には以下の4種類があります。

- SERIALIZABLE
- REPEATABLE READ
- READ COMMITTED
- READ UNCOMMITTED

独立性水準の初期値は、READ COMMITTEDになります。

独立性水準は、動作環境ファイルのDEFAULT_ISOLATIONまたはSET TRANSACTION文で変更可能です。

ただし、動作環境ファイルのR_LOCKの指定により、独立性水準は以下のようになります。

R_LOCK	DEFAULT_ISOLATION または SET TRANSACTION 文	独立性水準
NO	REPEATABLE_READ	SERIALIZABLE
	上記以外	SET TRANSACTION文の指定に従う
YES	SERIALIZABLE	REPEATABLE READ
	上記以外	SET TRANSACTION文の指定に従う

参考

SET TRANSACTION文の詳細は、“SQLリファレンス”を参照してください。

あるトランザクションの実行中に現れる可能性のある現象を以下に示します。

- 現象1)

  トランザクションT1が行を更新する。T1がトランザクションを終了する前に、トランザクションT2がその行を読み込む。T1がROLLBACK文を実行した場合、T2は存在しなかった行を読むことになる。

- 現象2)

  トランザクションT1が行を読み込む。トランザクションT2が、その行を更新または削除し、COMMIT文を実行する。T1がその行を再び読もうとした場合、更新した値を受け取るかまたはその行が削除されたことを知る。

・ 現象3)

トランザクションT1が、ある探索条件を満たす行の集合を読み込む。トランザクションT2が、その探索条件を満たす行を追加する。T1が、同じ探索条件で再度行の集合を読み込んだ場合、最初とは異なった行の集合を得ることになる。

独立性水準	現象1)	現象2)	現象3)
SERIALIZABLE	不可能	不可能	不可能
REPEATABLE READ	不可能	不可能	可能
READ COMMITTED	不可能	可能	可能
READ UNCOMMITTED	可能	可能	可能

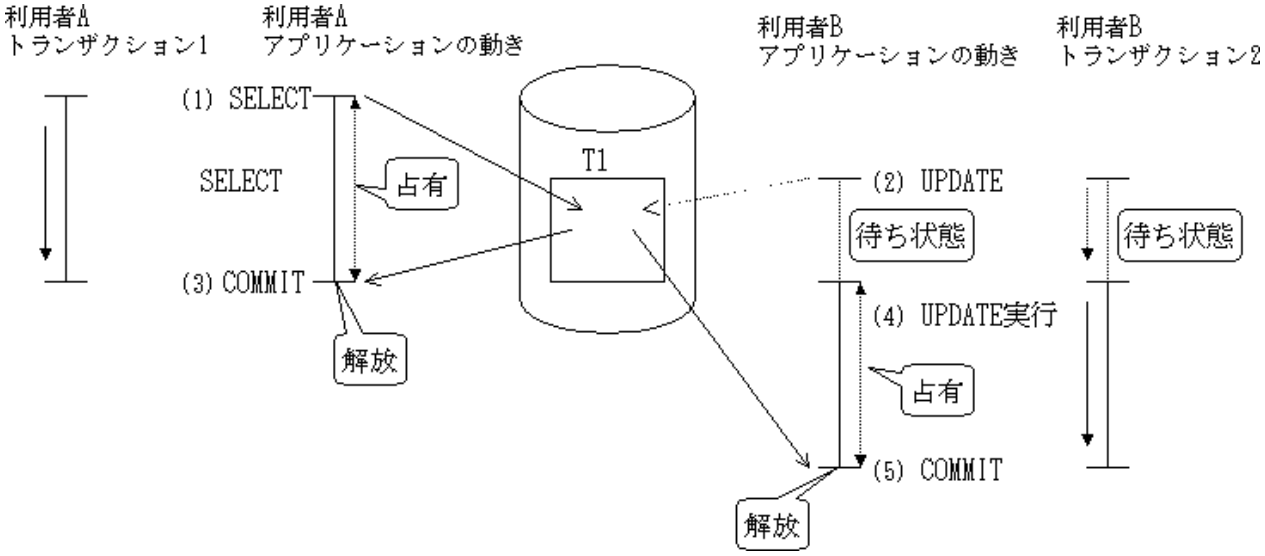
SERIALIZABLE

あるトランザクションが参照または更新した資源は、そのトランザクションが終了するまで、他のトランザクションから更新されないことが保証されます。

動作環境ファイルのパラメタR_LOCKで行単位の排他が指定されていない場合に有効です。

SERIALIZABLEの場合の排他処理を以下に示します。

図I.5 SERIALIZABLEの場合の排他



(1) 利用者AがCOMMIT(ROLLBACK)終了するまで資源を占有します。

(2) 利用者Bは、利用者Aが資源を占有しているため、待ち状態となります。

(3) 利用者AがCOMMIT(ROLLBACK)するので、利用者Aの資源は解放されます。

(4) 利用者Aが資源を解放したので、利用者Bの更新が実行されます。このとき、資源を占有します。

(5) 利用者BがCOMMIT(ROLLBACK)するので、利用者Bの資源は解放されます。

REPEATABLE READ

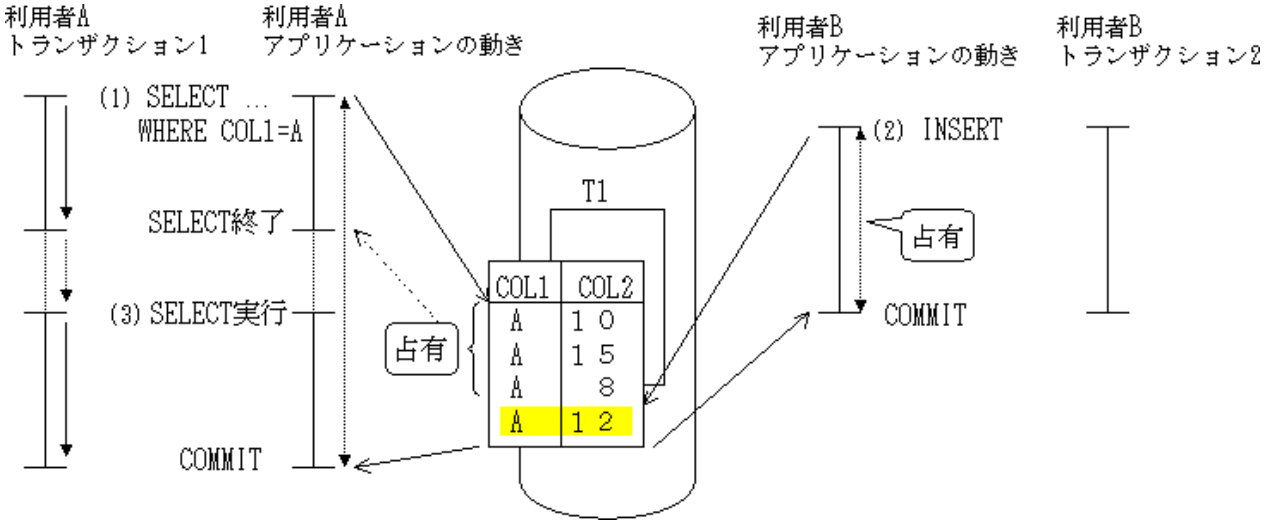
あるトランザクションが参照または更新した資源は、そのトランザクションが終了するまで、他のトランザクションからの更新がないことが保証されます。

動作環境ファイルのR_LOCKパラメタで行単位の排他が指定されている場合に有効です。

排他の単位が行単位でかつREPEATABLE READが指定された場合は、該当のトランザクションがアクセスしていない行は占有しないので、同一トランザクション内で、同一のSQL文をくり返し実行しても、異なる集合が検索される場合があります。

REPEATABLE READの場合の排他処理を以下に示します。

図I.6 REPEATABLE READの場合の排他



(1) 利用者Aは、COL1がAの行を参照します。トランザクションが終了するまでこの行の集合を占有します。

(2) 利用者Bは、利用者Aが参照した行に対しては更新することができませんが、INSERT文で行を追加することはできます。

(3) 利用者Aが、再びCOL1がAの行を参照します。このとき前回参照した行に加えて、利用者Bが追加した行も検索されます。

READ COMMITTED

動作環境ファイルのパラメタR_LOCKで行単位の排他が指定しているか否かで行の見え方が異なる場合があります。次に行単位の排他を指定した場合と、指定しない場合について説明します。

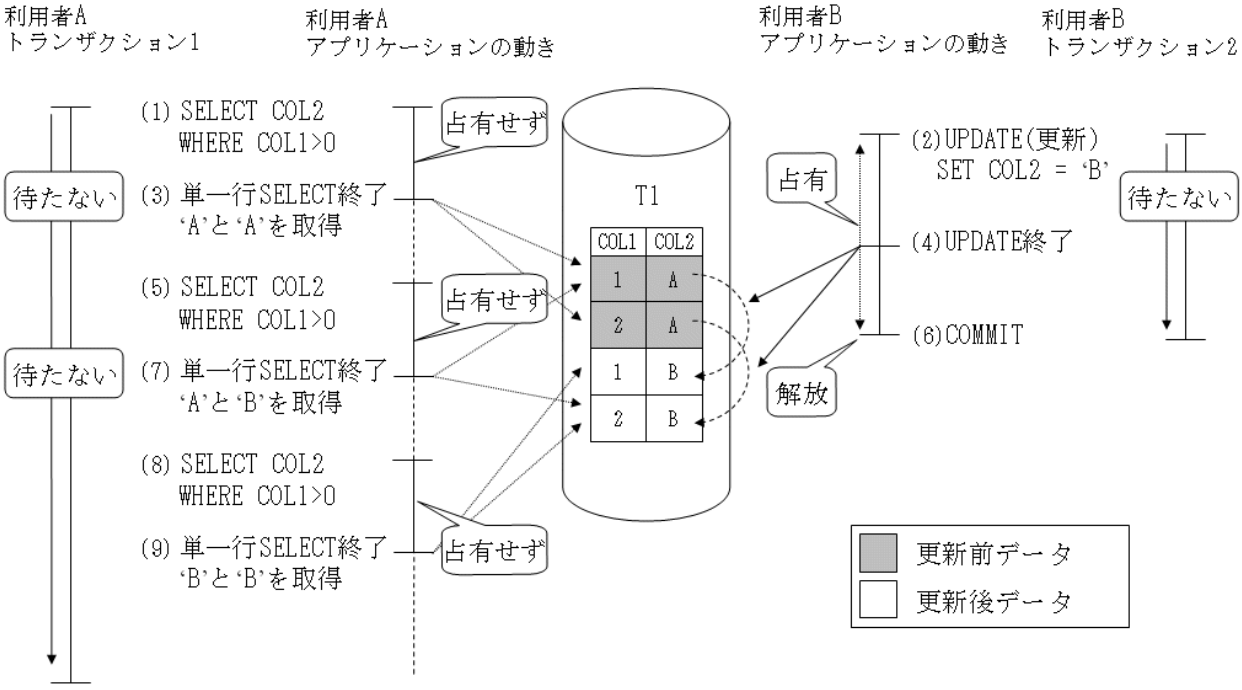
READ COMMITTED を指定して、かつ行単位を指定している場合

以下のSQL文を実行した場合、表の1行に対応する更新前のデータと更新中のデータを2重に保持しているため、検索する行が更新途中であっても、排他待ちすることなく、更新が発生する前のコミット済みのデータを検索することができます。

- 単一行SELECT文
- 更新可能性句に、FOR READ ONLYを指定したカーソル、または更新可能性句を省略したカーソルのOPEN文

READ COMMITTED を指定して、かつ、行単位を指定している場合の排他処理を以下に示します。

図I.7 READ COMMITTED を指定して、かつ、行単位を指定している場合の排他



(1) 利用者AがCOL1>0の行を参照します。このとき、行は占有しません。

(2) 利用者Bは利用者Aが資源を参照中に更新を行うことができます。このとき、資源を占有します。データベースには更新前の行に加え、更新後の行が格納されます。

(3) 利用者Aは利用者Bが行を更新占有中であっても参照を完了させることができます。検索される行には、利用者Bが更新途中の結果は反映されません。

(4) 利用者Bの更新が終了しますが、行の占有は続きます。

(5) 利用者AがCOL1>0の行を参照します。このとき、利用者Bが更新占有中の行であっても参照することができます。

(6) 利用者Bがトランザクションを終了し、行を解放します。

(7) 利用者Aの参照が終了します。参照中に利用者BがCOMMITしたため、検索される行には利用者Bの更新結果が一部反映されます。

(8) 利用者AがCOL1>0の行を参照します。このとき、行は占有しません。

(9) 利用者Aの参照が終了します。このとき、検索される行には利用者Bの更新結果がすべて反映されます。

注意

以下のSQL文は、各行の読み込み時の最新のコミット済みデータを参照します。

- 単一行SELECT文
- 更新可能性句に、FOR READ ONLYを指定したカーソル、または更新可能性句を省略したカーソルのOPEN文

このため、上記のSQL文実行中に更新業務がコミットした場合、次のような現象が起こることがあります。

- SQL文の検索結果に、ある行については更新業務が更新する前の行、別の行については更新業務が更新した後の行が含まれる場合があります。
- SQL文の検索結果に、ある行の更新前後の行がともに含まれる場合や、いずれも含まれない場合があります。

また、更新トランザクションを実行する場合、ある行について検索してから更新や削除するまでの区間に、他のトランザクションが更新や削除を行ってコミットする可能性があります。このような場合、データの整合性を保証するために、更新トランザクションについて以下の対処を行ってください。

- カーソルの更新可能性句にFOR UPDATEを指定する。

ー 占有モード指定にSHARE LOCK、またはEXCLUSIVE LOCKを指定する。

ー 上記対処を行わないと以下の現象になります。

ー 他のトランザクションが更新した行を上書きにより更新してしまう場合がある。

ー 他のトランザクションが削除した行についてUPDATE文:位置づけやDELETE文:位置づけを行うと、対象行が存在しないのでエラーになる。

ー 他のトランザクションが削除した行についてUPDATE文:探索やDELETE文:探索を行うと、対象行が存在しないのでデータなしになる。

READ COMMITTED を指定して、かつ行単位を指定していない場合

あるトランザクションが更新した資源は、そのトランザクションが終了するまで、他のトランザクションから更新されないことが保証されます。しかし、以下のいずれかのSQL文で参照した資源に対しては、他のトランザクションから更新される可能性があります。

ー 単一行SELECT文

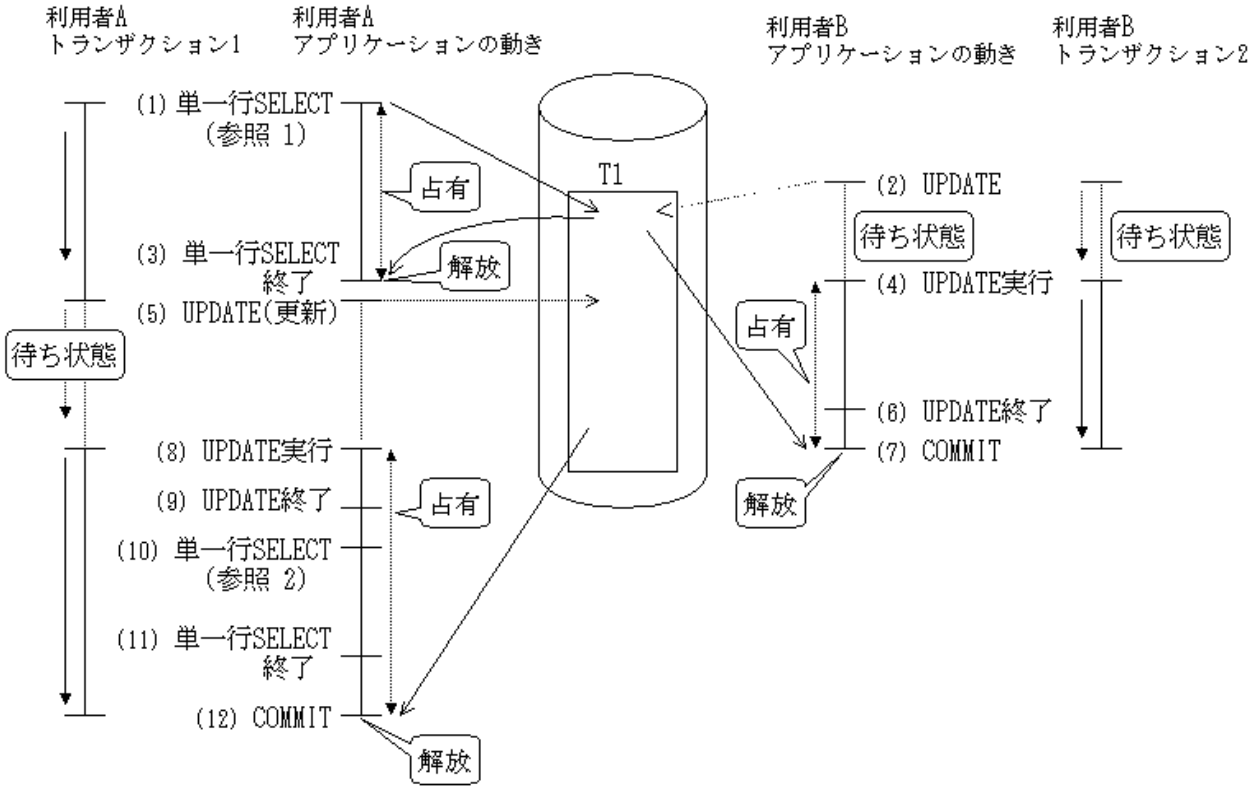
ー 更新可能性句に、FOR READ ONLYを指定したカーソルのOPEN文

ー トランザクションアクセスモードが読込み専用モードのときのOPEN文

したがって、同一トランザクション内でも、テーブル参照ごとに内容が異なる場合があります。

READ COMMITTED を指定して、かつ、行単位を指定していない場合の排他処理を以下に示します。

図I.8 READ COMMITTED を指定して、かつ、行単位を指定していない場合の排他



(1) 利用者Aが参照しているため(参照1)、この参照が終了するまで資源を占有します。

(2) 利用者Aが資源を解放するまで、利用者Bは更新を行えず、待ち状態となります。

(3) 利用者Aが参照終了(参照1)し、資源を解放します。

(4) 利用者Aが資源を解放したので、利用者Bの更新が実行されます。このとき、資源を占有します。

(5) 利用者Bが資源を解放するまで利用者Aは更新(更新)を行えず、待ち状態となります。

(6) 利用者Bの更新が終了します。

(7) 利用者Bがトランザクションを終了し、資源を解放します。

(8) 利用者Bが資源を解放したので、利用者Aの更新(更新)が実行されます。このとき、資源を占有します。

(9) 利用者Aが更新終了(更新)しますが、資源の占有は続きます。

(10) 利用者Aが参照(参照2)を行います。このとき検索される行は、利用者Bが更新した行を含みます。

(11) 利用者Aが参照(参照2)を終了しますが、資源の占有は続きます。

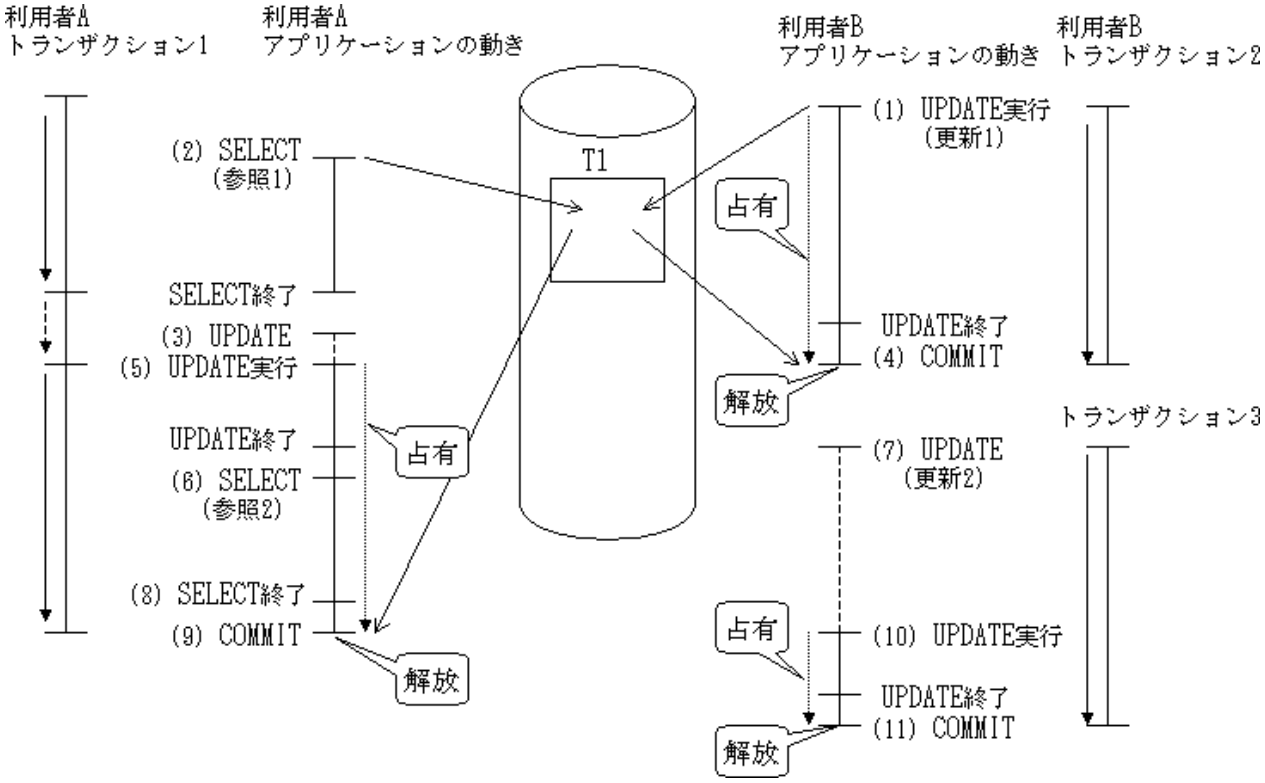
(12) 利用者Aがトランザクションを終了します。これによって資源が解放されます。

READ UNCOMMITTED

あるトランザクションが参照した資源は、他のトランザクションの参照および更新をまったく制限しません。また他のトランザクションが更新した資源も、トランザクションの終了を待たずに参照することができます。トランザクションが更新した資源は、トランザクション終了まで他のトランザクションの参照および更新を待たせます。

READ UNCOMMITTEDの場合の排他処理を以下に示します。

図I.9 READ UNCOMMITTEDの場合の排他



(1) 利用者Bは、更新処理(更新1)のため、COMMIT(ROLLBACK)終了するまで資源を占有します。

(2) 利用者Aは、資源を占有しないため、利用者Bが資源を占有していても、待ち状態にならずに参照処理(参照1)を行います。

(3) 利用者Aは利用者Bが資源を解放するまで更新を行えず、待ち状態となります。

(4) 利用者BがCOMMIT(ROLLBACK)します。利用者Bの資源は解放されます。

(5) 利用者Bが資源を解放したので、利用者Aの更新が実行されます。このとき、実際に更新した資源を占有します。

(6) 利用者Aが参照(参照2)を行います。このとき検索される行は、利用者Bが更新した行を含みます。

(7) 利用者Bは利用者Aが資源を解放するまで更新を行えず、待ち状態となります。

(8) 利用者Aが参照(参照2)を終了しますが、資源の占有は続きます。

(9) 利用者AがCOMMIT(ROLLBACK)します。

(10) 利用者Aが資源を解放したので、利用者Bの更新(更新2)が実行されます。このとき、資源を占有します。

(11) 利用者BがCOMMIT(ROLLBACK)します。

I.4　トランザクションとカーソル操作

データ操作において、操作の対象とする行を指示する指示子のことを“カーソル”と呼びます。

カーソルについての詳細は、“第7章 カーソルを使用するデータ操作の方法”を参照してください。

カーソルは、一般に、トランザクション終了時にクローズされます。しかし、次の場合、カーソルはクローズされません。

- トランザクションの独立性水準指定がREAD COMMITTEDで、カーソル指定の更新可能性句がFOR READ ONLYの場合
- カーソル宣言に、カーソルモード指定がある場合

以下に独立性水準指定にREAD COMMITTEDを指定し、更新可能性句にFOR READ ONLYを指定して実現する、トランザクションを超越するカーソル操作のSQL文の例を示します。

```
EXEC SQL DECLARE CSR1 CURSOR FOR                              ……(1)
                 SELECT * FROM 取引先会社
                 WHERE  電話番号 LIKE  :hcomp_word
                 FOR  READ  ONLY;

EXEC SQL SET TRANSACTION  READ  ONLY,                         ……(2)
                          ISOLATION LEVEL  READ  COMMITTED;
EXEC SQL OPEN  CSR1;                                          ……(3)
EXEC SQL FETCH CSR1 INTO :hnumber,:hproduct,:hstock;          ……(4)
            :
EXEC SQL COMMIT WORK;                                         ……(5)


EXEC SQL FETCH CSR1 INTO :hnumber,:hproduct,:hstock;          ……(6)
            :
      （データの終了）

EXEC SQL CLOSE CSR1;                                          ……(7)
```

(1) カーソル“CSR1”を宣言します。更新可能性句に“FOR READ ONLY”を設定します。

(2) トランザクションアクセスモードに“READ ONLY”を設定します。独立性水準に“READ COMMITTED”を設定します。

(3) カーソル“CSR1”をオープンします。

(4) 探索条件が真となる行にカーソルを位置づけ、その行を読み込みます。

(5) COMMITによりトランザクションを終了します。

(6) (5)でカーソルがクローズされないため、カーソルを位置づけることができます。

(7) データ終了後、カーソル“CSR1”をクローズします。

また、以下にカーソルモード指定によって実現する、トランザクションを超越するカーソル操作のSQL文の例を示します。

```
EXEC SQL DECLARE CSR1 CURSOR FOR                              ……(1)
         SELECT * FROM  取引先会社
         WHERE 電話番号 LIKE :hcomp_word
         WITH OPTION LOCK_MODE(FREE LOCK), CURSOR_MODE(HOLD);

EXEC SQL OPEN CSR1;                                           ……(2)

EXEC SQL FETCH CSR1 INTO :hnumber,:hproduct,:hstock;          ……(3)
                    :
EXEC SQL COMMIT WORK;                                         ……(4)
```

```
EXEC SQL FETCH CSR1 INTO :hnumber,:hproduct,:hstock;              ……(5)
                        :
                （データの終了）
EXEC SQL CLOSE CSR1;                                              ……(6)
```

(1) カーソル“CSR1”を宣言します。カーソルモード指定に“HOLD”を設定します。

(2) カーソル“CSR1”をオープンします。

(3) 探索条件が真となる行にカーソルを位置づけ、その行を読み込みます。

(4) COMMITによりトランザクションを終了します。

(5) (4)でカーソルがクローズされないため、カーソルを位置づけることができます。

(6) データ終了後、カーソル“CSR1”をクローズします。

トランザクションを超越するカーソルを使用する場合の注意事項

トランザクションを超越するカーソルを使用する場合、カーソルのオープン中に、そのカーソルが参照する表の定義や格納構造定義が変更されると、次のトランザクションでFETCH文の実行が、SQLSTATE値が24000(カーソル不当)のエラーとなります。このエラーが発生した場合は、カーソルをクローズして、再度オープンするようにしてください。

I.5　SQL文の処理結果異常とトランザクション

データ操作のSQL文の処理中に異常が発生した場合には、アプリケーションにエラーで復帰します。エラーで復帰したときのトランザクションの状態は、エラーの原因によって異なります。エラーの原因を示すコードは、状態変数“SQLSTATE”に設定されます。

トランザクションがロールバックされるエラーの場合のSQLSTATE:

40xxx

トランザクションが継続されるエラーの場合のSQLSTATE:

40xxx以外

トランザクションが継続されるエラー

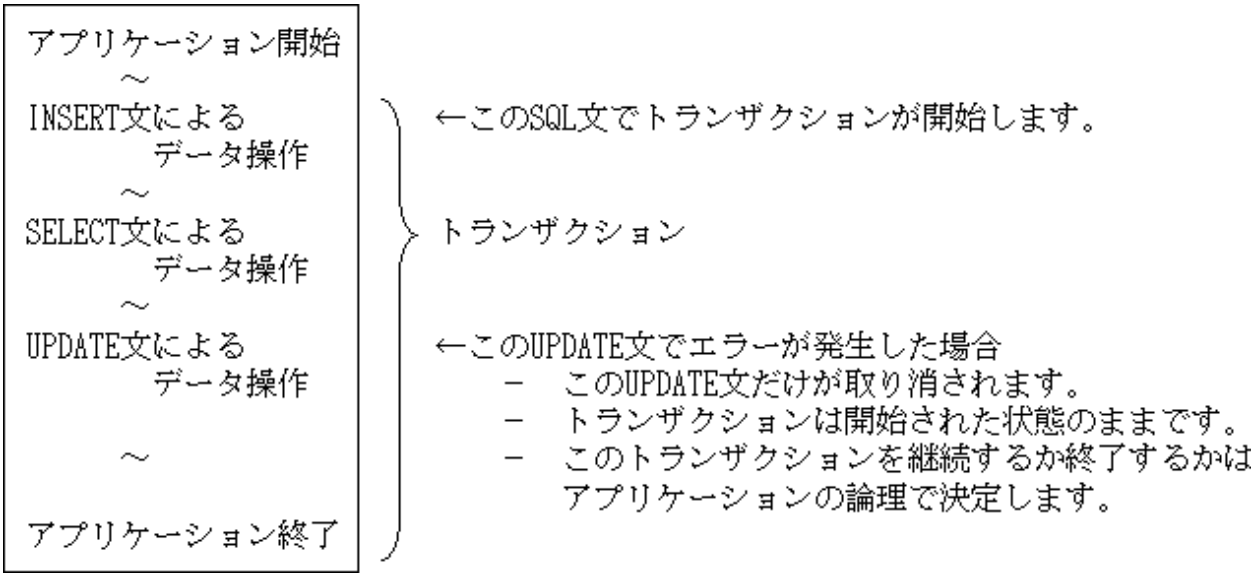
エラーが発生したSQL文による更新は、そのSQL文を実行する直前の状態に戻されてアプリケーションへ復帰します。ただし、このSQL文の実行までに獲得されたデータベース資源の占有とカーソルの位置はそのまま保持されます。このとき、トランザクションは継続されたままになっています。トランザクションをそのまま継続させるか終了させるかは、エラーの原因ごとに利用者がアプリケーションの論理に従って決定します。たとえば、INSERT文で表にデータを繰り返し挿入するような処理では、あるデータが表の一意性制約に違反するエラーとなっても、そのデータを無視して引き続きデータを挿入すればよい場合には、トランザクションは継続させて処理を続行します。また、互いに関連のある複数の表を更新するような処理では、ある表の更新時にエラーが発生した場合、表の間での整合を保つために、ROLLBACK文を実行してトランザクションをロールバックします。

なお、動作環境ファイルのパラメタ“TRAN_SPEC”の指定により、トランザクションの対処方法を変更することができます。

動作環境ファイルについては、“2.1.1 動作環境ファイル”を参照してください。

データ操作のSQL文の実行でエラーが発生した場合の例を以下に示します。

図I.10 データ操作のSQL文の実行でエラーが発生した場合の例



トランザクションがロールバックされるエラー

トランザクションはロールバックされ終了してアプリケーションへ復帰するため、このSQL文の実行までに獲得されたデータベース資源のロックは解放され、オープン中のカーソルは閉じられます。

デッドロックが発生した場合のアプリケーションの対処方法については、“I.10 デッドロックの対処方法”で説明します。

I.6 アプリケーションの異常終了とトランザクション

アプリケーションは実行中に異常終了することがあります。このとき、トランザクションが開始状態であれば、トランザクションはロールバックされて終了します。アプリケーションが異常終了するケースには以下のような場合があります。

- アプリケーションの実行環境の定義の誤り
- アプリケーションの誤りによる異常終了

I.7 排他制御

アプリケーションからデータベースへアクセスする場合には、トランザクション単位にSymfoware/RDBで排他制御を行います。

データベースの同一資源に対して、アプリケーションおよびRDBコマンドが同時に動作した場合のSymfoware/RDBの排他制御の詳細については、“付録D アプリケーションおよびRDBコマンド間の排他”を参照してください。

I.7.1 排他の属性と選択方法

排他の属性には、以下のものがあります。

- 占有の資源
- 占有の単位
- 占有の範囲
- 占有モード
- 占有待ちの有無

以下に、排他の属性の指定方法と、排他の属性の各項目の概要を説明します。

排他の属性の指定方法

排他の属性を指定するには、以下の方法があります。

- 動作環境ファイルの実行パラメタによる指定

  動作環境ファイルの実行パラメタによって設定できる属性は、以下のとおりです。

  - ISOLATION_WAIT:

    占有待ちの有無について指定します。

  - R_LOCK:

    占有単位について指定します。

参照

各パラメタの記述形式については、“アプリケーション開発ガイド(埋込みSQL編)”の“クライアント用の動作環境ファイルによる定義”を参照してください。

- SET TRANSACTION文による指定

  アプリケーション中にSET TRANSACTION文の独立性水準指定によって、占有の強さについて設定することができます。

参照

SET TRANSACTION文の詳細は、“SQLリファレンス”を参照してください。

- Symfoware/RDBによる選択

  動作環境ファイルの実行パラメタや、SET TRANSACTION文が指定されなかった場合、Symfoware/RDBがSQL文を解釈し、その結果をもとに選択します。

- SQL文の占有モード指定による指定

  SQL文に占有モード指定をすることにより、読み込んだデータベース資源(データおよびインデックス)に対して、SQL文単位に明示的に占有モードを設定することができます。この機能を、イルシデーションロック(elucidation lock)といいます。

参照

占有モード指定の詳細は、“SQLリファレンス”を参照してください。

占有の資源

占有する対象の資源は、そのアプリケーションで使用する表およびインデックスです。

占有の単位

占有制御によって資源を占有する単位には、行単位、表単位およびインデックス単位があります。

行単位の占有の場合には、アプリケーションがアクセスするデータを含む行が占有の対象になります。このとき、データの参照または更新のための走査対象行も占有の範囲に含まれます。表単位の占有の場合には、アプリケーションがアクセスするデータを含む表が占有の対象になります。占有単位を指定するには動作環境ファイルの実行パラメタR_LOCKを使用してください。

R_LOCKの指定	占有単位
YES	行単位で占有
NO	表またはインデックス単位で占有

占有の範囲

表やインデックスの占有範囲はその資源の検索範囲となります。表の検索範囲は、SQL文の探索条件をインデックスで評価して絞り込んだ部分です。このため、インデックスを使用しない場合は、探索条件の評価結果に関わらず、表の全範囲が占有対象となります。インデックスの検索範囲については、“H.2.1 SCANエレメント”の“4) INDEX KEY SCAN”を参照してください。

占有モード

データベースにアクセスする場合の占有モードには、共有モードと非共有モードの2種類があります。

共有モードの場合、ロックされたデータは、ほかのトランザクションからのデータ操作において、参照だけが許され更新操作はできません。一般に、データのロックを獲得中のトランザクションのコミットが行われるまで、ほかのトランザクションのデータの更新操作は、待ち状態となります。非共有モードの場合、ロックされたデータは、ほかのトランザクションでデータ操作することはできません。一般に、データのロックを獲得中のトランザクションのコミットが行われるまで、ほかのトランザクションの実行は、待ち状態となります。

占有モードを指定するには、以下の方法があります。

- Symfoware/RDBによる選択
- SET TRANSACTION文での指定
- SQL文における占有モード指定での指定

Symfoware/RDBによる選択

データに対する占有

Symfoware/RDBは、アクセスするデータに対して、以下に示す条件で、SQL文ごとにデータに対する占有モードを選択します。

表I.2 Symfoware/RDBによる占有モードの選択

占有モード	選択条件
共有モード	データを参照する処理の部分の場合
非共有モード	データを更新する処理の部分の場合

データ検索を行う以下のSQL文については、アクセスするデータに対する占有モードは共有モードになります。また、これらのSQL文については、インデックスに対する更新が行われないので、インデックスに対するモードは共有モードになります。

- OPEN文(ただし、カーソル宣言の更新可能性句にFOR UPDATEを指定したカーソルのOPEN文は、非共有モードで占有します。)
- 単一行SELECT文
- FETCH文

データの更新、削除および挿入を行う以下のSQL文については、処理(更新、削除および挿入)対象のデータに対する占有モードは、非共有モードになります。ただし、データ更新、削除および挿入を行うSQL文であっても、UPDATE文やDELETE文の更新対象のデータを位置づけるための検索やINSERT文に指定した問合せ指定の検索に関しては、共有モードになります。

- UPDATE文:探索
- UPDATE文:位置づけ
- DELETE文:探索
- DELETE文:位置づけ
- INSERT文

インデックスに対する占有

Symfoware/RDBは、SQL文の処理で使用するインデックスの更新の有無により、以下のように選択します。

表I.3 Symfoware/RDBによるインデックスに対する占有モードの選択

インデックスに対する占有モード	インデックス更新の有無
共有モード	インデックス更新なし
非共有モード	インデックス更新あり

データ更新を行う以下のSQL文については、更新対象の列に設定されているインデックスに対して更新が行われます。したがって、そのようなインデックスに対する占有モードは非共有モードになります。一方、更新対象でない列に設定されているインデックスに対しては、更新は行われないので、そのようなインデックスに対する占有モードは共有モードになります。

- UPDATE文:探索
- UPDATE文:位置づけ

データ削除および挿入を行う以下のSQL文については、更新対象の表に張られているインデックスに対して更新が行われます。したがって、そのようなインデックスに対する占有モードは非共有モードになります。

- DELETE文:探索
- DELETE文:位置づけ
- INSERT文

SET TRANSACTION文での指定

SET TRANSACTION文での指定については、“I.3 トランザクションモード”を参照してください。

SQL文における占有モード指定での指定

SQL文に占有モード指定を指定することにより、SQL文が読み込んだ資源(データおよびインデックス)の占有モードと占有期間を柔軟に変更することができます。占有モード指定がある場合、SET TRANSACTION文で指定された内容とは無関係に動作します。占有モード指定と資源の占有について以下に示します。

表I.4 占有モード指定と資源の占有

占有モード指定	占有のしかたと占有期間	他のトランザクションとの待ち関係	読込み水準
EXCLUSIVE LOCK	非共有モードで、トランザクション終了まで資源を占有する。	当該トランザクションが終了するまで、同じ資源を参照／更新する他のトランザクションを待たせる。	COMMIT済みの行を読み込む。当該SQL文で読み込んだ行は、トランザクション終了まで他のトランザクションに更新されることがないため、一度読み込んだ行は、他のトランザクションによって更新されないことが保証される。
SHARE LOCK	共有モードで、トランザクション終了まで資源を占有する。	SQL文の実行後は、当該トランザクションが終了するまで、同じ資源を更新する他のトランザクションを待たせる。	
FREE LOCK (占有単位に行単位を指定した場合)	資源を占有しない。	当該SQL文を実行しても、他のトランザクションを待たせない。	COMMIT済みの行を読み込む。当該SQL文で読み込んだ行は、他のトランザクションに更新されることがあるため、同一トランザクションで再検索すると最新の結果を検索することができる。
FREE LOCK (占有単位に行単位を指定していない場合)	共有モードで、SQL文終了(カーソルならば、OPEN文終了時)まで資源を占有する。	当該SQL文の実行中は、同じ資源を更新する他のトランザクションを待たせる。当該SQL文の実行が終了すると、同じ資源を更新する他のトランザクションを実行することができる。	

占有モード指定	占有のしかたと占有期間	他のトランザクションとの待ち関係	読込み水準
NO LOCK	資源を占有しない。	当該SQL文を実行しても、他のトランザクションを待たせない。	他のトランザクションでCOMMITされていない更新中の行を読みことができる。

占有モード指定は、資源を読み込むときのモードです。読み込んだ資源の更新時には、占有モード指定での指定に関係なくEXCLUSIVE LOCKになります。

占有モード指定を指定したSQL文の待ち関係

占有モード指定を指定したSQL文が参照した資源に対する待ち関係を以下に示します。

更新した部分に関してはEXCLUSIVE LOCK指定がされたとみなして表を参照してください。

表I.5 占有モード指定と待ち関係

		先行トランザクション				
	占有モード指定	EXCLUSIVE LOCK	SHARE LOCK	FREE LOCK 占有単位に行単位を指定した場合	FREE LOCK 占有単位に行単位を指定していない場合	NO LOCK
後続トランザクション	EXCLUSIVE LOCK	×	×	○(注)	△	○
	SHARE LOCK	×	○	○	○	○
	FREE LOCK 占有単位に行単位を指定した場合	○(注)	○	○	○	○
	FREE LOCK 占有単位に行単位を指定していない場合	×	○	○	○	○
	NO LOCK	○	○	○	○	○

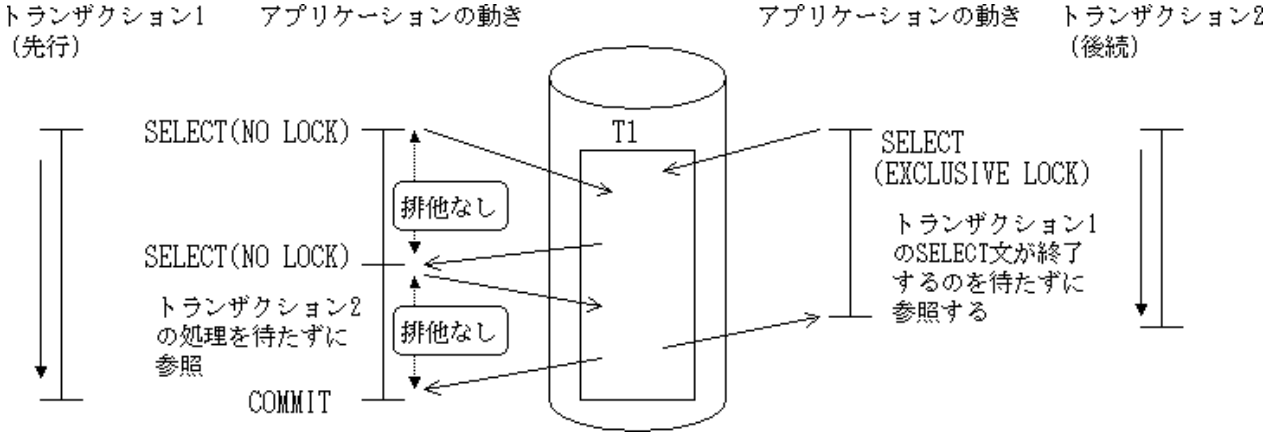
○:待つことなく動作する

注):FREE LOCK指定側が、更新可能性句にFOR UPDATEを指定したカーソルの場合は、EXCLUSIVE LOCKと同じ動作になります

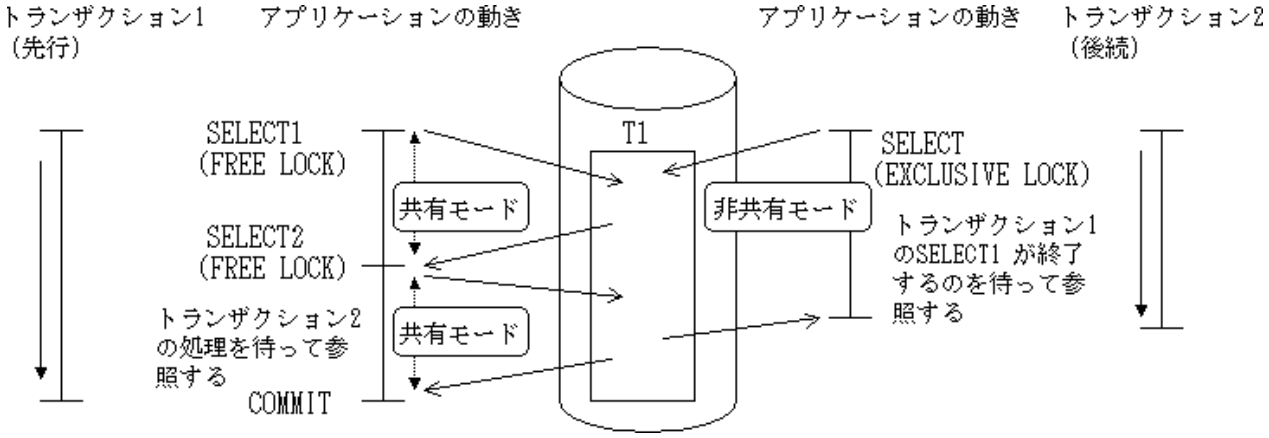
△:先行トランザクションのSQL文の終了を待つ

×:先行トランザクションの終了を待つ

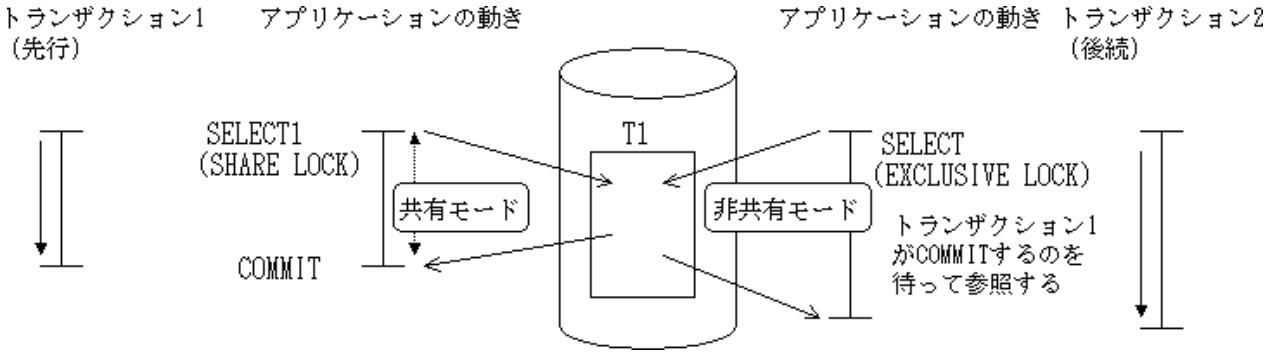
図I.11 ○:待つことなく動作する場合



図I.12 △:先行トランザクションのSQL文の終了を待つ場合



図I.13 ×:先行トランザクションの終了を待つ場合



カーソル宣言およびSET TRANSACTION文における指定とSQL文の占有モード指定における指定との対応関係

SQL文の占有モード指定を省略した場合、独立性水準、トランザクションアクセスモードおよび更新可能性句を組み合わせることによって、占有モード指定を指定したときと同等の資源の占有のしかたを設定できます。以下に、SET TRANSACTION文および更新可能性句での指定と、それに対応する占有モード指定を示します。

占有モード指定があるSQL文と、占有モード指定を省略したSQL文との待ち関係については、以下の表を利用して同等の占有モード指定を確認した後、“表I.5 占有モード指定と待ち関係”を参照してください。

- 占有モード指定を省略したSQL文がINSERT文、UPDATE文:位置づけおよびDELETE文:位置づけの場合

SET TRANSACTION文における指定		対応する占有モード指定
独立性水準	トランザクション アクセスモード	
SERIALIZABLE REPEATABLE READ READ COMMITTED READ UNCOMMITTED	READ WRITE	EXCLUSIVE LOCK

- 占有モード指定を省略したSQL文がUPDATE文:探索およびDELETE文:探索の場合

SET TRANSACTION文における指定		対応する占有モード指定
独立性水準	トランザクション アクセスモード	
SERIALIZABLE REPEATABLE READ READ COMMITTED	READ WRITE	SHARE LOCK (更新対象行を検索するとき) EXCLUSIVE LOCK (更新対象行を更新するとき)
READ UNCOMMITTED		NO LOCK (更新対象行を検索するとき) EXCLUSIVE LOCK (更新対象行を更新するとき)

ただし、クライアント用の動作環境ファイルの実行パラメタUSQL_LOCKにEXを指定した場合は、更新対象行を検索するときから、EXCLUSIVE LOCKで資源を占有します。

- 占有モード指定を省略したSQL文が、単一行SELECT文の場合

SET TRANSACTION文における指定		対応する占有モード指定
独立性水準	トランザクション アクセスモード	
SERIALIZABLE REPEATABLE READ	READ ONLY READ WRITE	SHARE LOCK
READ COMMITTED	READ ONLY READ WRITE	FREE LOCK
READ UNCOMMITTED	READ ONLY READ WRITE	NO LOCK

- 占有モード指定を省略したSQL文が、カーソル宣言の場合

SET TRANSACTION文における指定		カーソル宣言の更新可能性句	対応する占有モード指定
独立性水準	トランザクションアクセスモード		
SERIALIZABLE REPEATABLE READ	READ ONLY	FOR READ ONLY または 省略	SHARE LOCK
	READ WRITE	FOR UPDATE	EXCLUSIVE LOCK
		FOR READ ONLY または 省略	SHARE LOCK
READ COMMITTED	READ ONLY	FOR READ ONLY または 省略	FREE LOCK
	READ WRITE	FOR UPDATE	EXCLUSIVE LOCK
		FOR READ ONLY	FREE LOCK
		省略	FREE LOCK （占有単位を行単位に指定した場合） SHARE LOCK （占有単位を行単位に指定していない場合）
READ UNCOMMITTED	READ ONLY	FOR READ ONLY または 省略	NO LOCK
	READ WRITE	FOR UPDATE	EXCLUSIVE LOCK
		FOR READ ONLY または 省略	NO LOCK

SQL文単位の排他制御の例

SET TRANSACTION文によりトランザクションモードを変更する場合、一度トランザクションを終了させる必要があります。しかし、SQL文に占有モード指定を指定することにより、トランザクションを終了することなく、SQL文が読み込んだデータベース資源の占有状態を変更することができます。以下に、デッドロック抑止を目的とした占有モード指定の利用例を示します。

```
EXEC SQL SELECT * FROM  T1                                    …… (1)
         WHERE C1 = :H1
         WITH OPTION LOCK_MODE(EXCLUSIVE LOCK);

EXEC SQL UPDATE T1                                            …… (2)
     SET C2 = :H2  WHERE C1 = :H1;

EXEC SQL COMMIT WORK;                                         …… (3)
```

(1) デッドロック抑止を目的として、単一行SELECT文にEXCLUSIVE LOCKを指定します。参照時から資源を非共有モードで占有します。

(2) 単一行SELECT文によって位置づけられた行の列を更新します。

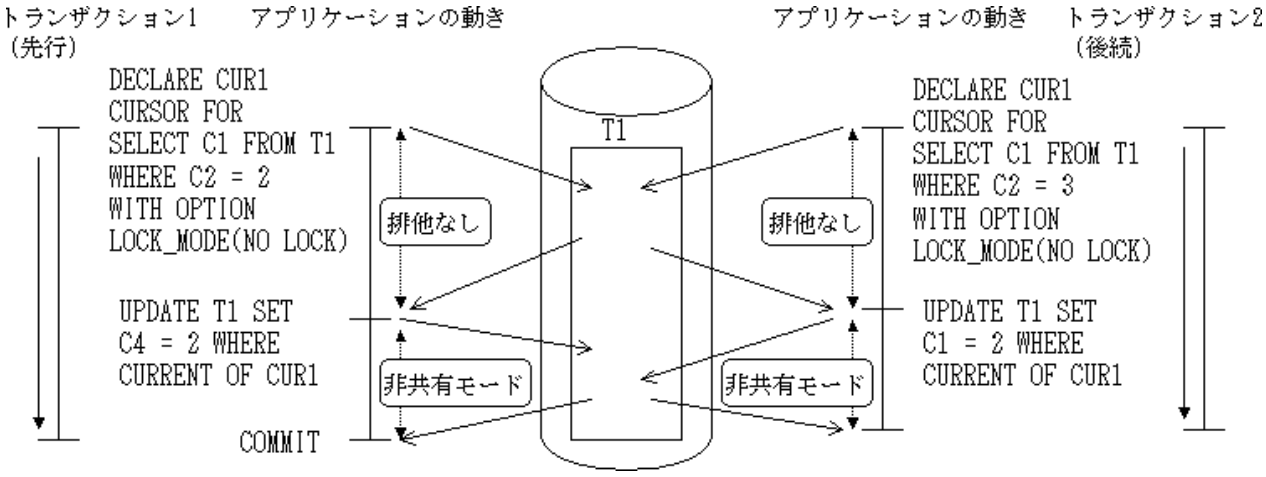
(3) COMMIT文によりトランザクションを終了します。

SQL文での占有モード指定の使用例

アプリケーションを多重実行する場合、更新処理があると非共有モードで資源を占有する必要があります。このため、他のアプリケーションが先に資源を占有している場合、そのアプリケーションのトランザクションが終了するまで、後続のアプリケーションは処理を待たされます。しかし、同一のレコードを同時に更新することがなければ、排他なしで参照し、実際に更新するレコードのみ排他制御を行えばよいので、システム全体の処理スピードを向上できます。

お互い同一のレコードを同時に更新しない場合は、レコード位置づけ処理において他のアプリケーションのトランザクションを待たないように、カーソル宣言の占有モード指定にNO LOCKを指定します。

図I.14 SQL文での占有モード指定



占有待ちの有無

トランザクション間で資源(行、表全体、インデックス全体)に対して競合が発生した場合、その時点で占有待ちに入る(資源が解放されるまで待つ)場合と、アプリケーションに復帰する場合があります。

Symfoware/RDBによる選択

占有待ちを行います。

動作環境ファイルの実行パラメタでの選択方法

アプリケーションの動作環境ファイルの実行パラメタISOLATION_WAITで、以下に示すように指定します。

表I.6 動作環境ファイルの実行パラメタでの占有待ちの有無の選択方法

占有待ちの有無	選択方法
占有待ちになる	WAITを指定した場合、または省略した場合
アプリケーションに復帰する	REJECTを指定した場合

排他制御に関するパラメタの相互関係

排他制御に関する各パラメタの指定と、占有の単位および占有のふるまいについて、以下に示します。

表I.7 排他制御に関するパラメタの相互関係

R_LOCKの値	SET TRANSACTION文の指定	占有モード指定の指定	占有のふるまい	占有の単位
YES	○	○	占有モード指定での指定に従う	行単位
		-	SET TRANSACTION文の独立性水準に従う。ただし、独立性水準にSERIALIZABLEを指定した場合は、REPEATABLE READになる	
	-	○	占有モード指定での指定に従う	
		-	独立性水準は、REPEATABLE READになる	
NO	○	○	占有モード指定での指定に従う	表またはインデックス単位
		-	SET TRANSACTION文の独立性水準に従う。ただし、独立性水準にREPEATABLE READを指定した場合は、SERIALIZABLEになる	
	-	○	占有モード指定での指定に従う	
		-	独立性水準は、SERIALIZABLEになる	

○：指定可能

×：指定できません

－：指定を省略

I.8　資源の競合が起きた場合の制御

資源の競合が起きた場合に、どのタイミングまで占有待ちとなるのかは、トランザクションモードの指定によって異なります。以下に、トランザクションモードの指定と資源の占有について示します。

表I.8 占有の単位が行の場合

			先行トランザクション											
			独立性水準	SERIALIZABLE REPEATABLE READ				READ COMMITTED				READ UNCOMMITTED			
			SQL文の種類	SELECT	INSERT	UPDATE	DELETE	SELECT	INSERT	UPDATE	DELETE	SELECT	INSERT	UPDATE	DELETE
後続トランザクション	独立性水準	SQL文の種類													
	SERIALIZABLE REPEATABLE READ READ COMMITTED	SELECT		○	×	×	×	○	×	×	×	○	×	×	×
		INSERT		○	○	○	○	○	○	○	○	○	○	○	○
		UPDATE		×	×	×	×	○	×	×	×	○	×	×	×
		DELETE		×	×	×	×	○	×	×	×	○	×	×	×
	READ UNCOMMITTED	SELECT		○	○	○	○	○	○	○	○	○	○	○	○
		INSERT		○	○	○	○	○	○	○	○	○	○	○	○
		UPDATE		×	×	×	×	○	×	×	×	○	×	×	×
		DELETE		×	×	×	×	○	×	×	×	○	×	×	×

○：待つことなく動作する

×：トランザクションの終了を待つ

先行または後続のトランザクションがカーソルのOPEN文を実行した場合は、カーソル宣言の可能性句の指定によって占有待ちの制御が異なります。

- 更新可能性句にREAD ONLYを指定する、または指定を省略した場合は、SELECT文と同じです。
- 更新可能性句にFOR UPDATEを指定した場合は、UPDATE文と同じです。

表I.9 占有の単位が行単位ではない場合

			先行トランザクション											
			独立性水準	SERIALIZABLE REPEATABLE READ				READ COMMITTED				READ UNCOMMITTED			
			SQL文の種類	SELECT	INSERT	UPDATE	DELETE	SELECT	INSERT	UPDATE	DELETE	SELECT	INSERT	UPDATE	DELETE
後続トランザクション	独立性水準	SQL文の種類													
	SERIALIZABLE REPEATABLE READ READ COMMITTED	SELECT		○	×	×	×	○	×	×	×	○	×	×	×
		INSERT		×	×	×	×	△	×	×	×	○	×	×	×
		UPDATE		×	×	×	×	△	×	×	×	○	×	×	×
		DELETE		×	×	×	×	△	×	×	×	○	×	×	×
	READ UNCOMMITTED	SELECT		○	○	○	○	○	○	○	○	○	○	○	○
		INSERT		×	×	×	×	△	×	×	×	○	×	×	×
		UPDATE		×	×	×	×	△	×	×	×	○	×	×	×
		DELETE		×	×	×	×	△	×	×	×	○	×	×	×

○：待つことなく動作する

△：先行トランザクションのSQL文が以下のいずれかの場合は、先行トランザクションのSQL文の終了を待つ

- 単一行SELECT文
- 更新可能性句に、FOR READ ONLYを指定したカーソルのOPEN文
- トランザクションアクセスモードが読込み専用モードのときのOPEN文

  先行トランザクションのSQL文が上記以外の場合は、トランザクションの終了を待つ

×：トランザクションの終了を待つ

先行または後続のトランザクションがカーソルのOPEN文を実行した場合は、カーソル宣言の更新可能性句の指定によって占有待ちの制御が異なります。

- 更新可能性句にREAD ONLYを指定する、または指定を省略した場合は、SELECT文と同じです。
- 更新可能性句にFOR UPDATEを指定した場合は、UPDATE文と同じです。

I.9　行単位の排他を使用する場合の注意事項

行単位の排他を使用する場合に、設計時点で注意しなければならない事項について説明します。

SQL文のデータ操作文に関する注意事項

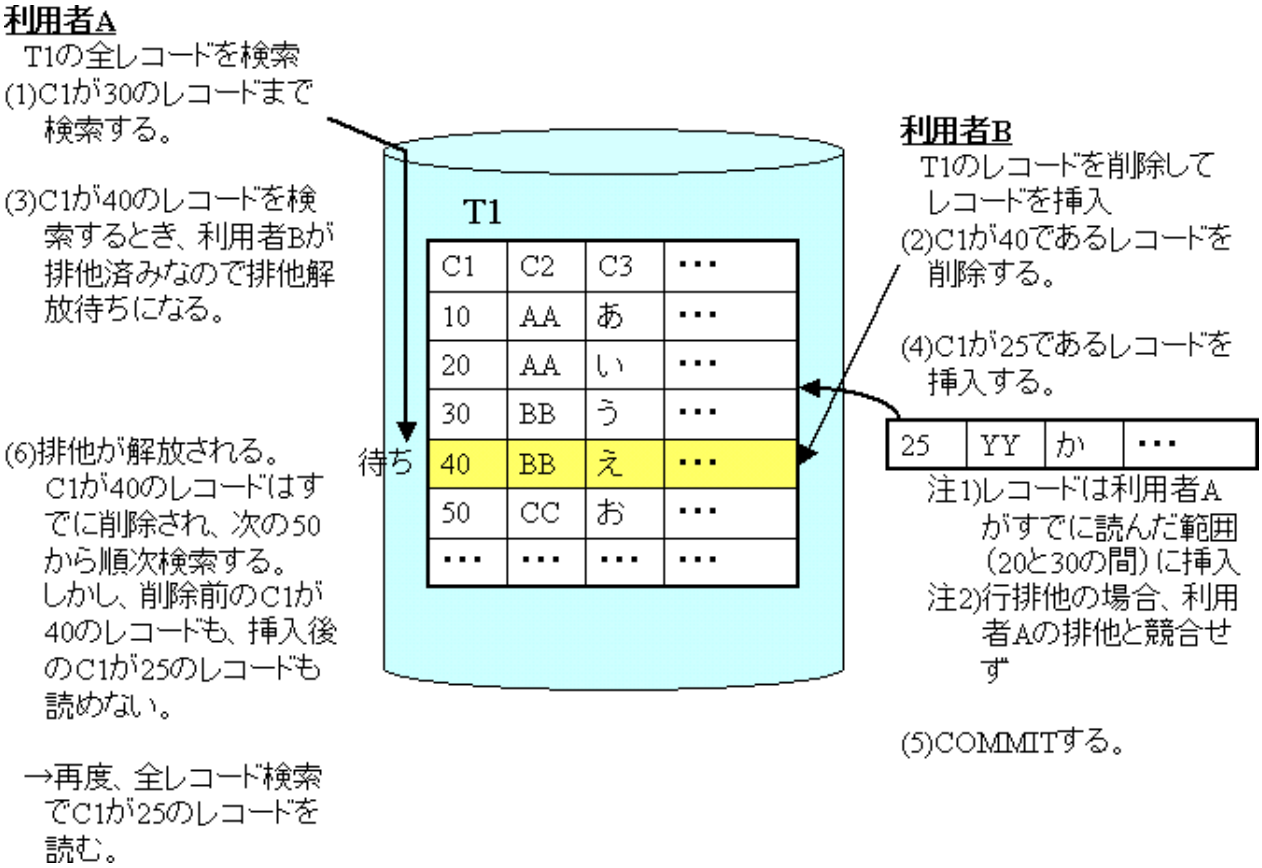
行単位の排他を使用する場合は、そのトランザクションがアクセスしていない行は占有しません。競合する別のトランザクションでは、最初のトランザクションがアクセスしていない行を更新したり、新たな行を追加したりすることができます。このため、以下のような現象が起こることがあります。構築する業務において、この現象が発生すると不都合な場合は、行単位の排他を使用しないでください。

- 同一トランザクション内で、同一のSQL文を繰り返し実行した場合に、異なる集合が検索されることがあります。例えば、
  - ー カーソルで検索する行が増えることがあります。
  - ー 一度目は成功した単一行SELECT文が、例外(基数違反)となることがあります。
  - ー 集合関数の結果が変化することがあります。
  - ー 探索条件に副問合せを指定した問合せの検索結果が変化することがあります。
- 行単位の排他では、INSERT文は他のトランザクションの終了を待たずに実行できます。このため、あるトランザクションで行を削除して、新たな行を挿入する操作を行っているとき、競合する別のトランザクションでは、まれに削除された行も挿入された行も参照できない場合があります。

  例

  利用者AがT1の全レコードを検索します。これと同時に、利用者Bが、C1が40であるレコードを削除して、C1が25であるレコードを挿入します。

  図I.15 SQL文のデータ操作文に関する注意事例



- 行単位の排他を使用して更新系の処理を行うトランザクションに対し、同じ資源を参照するSQL文を排他なしで実行した場合、以下の現象が発生する場合があります。排他なしとは、独立性水準にREAD UNCOMMITTEDを指定したトランザクション、または、占有モードにNO LOCKを指定したSQL文を指します。
  - ー UPDATE文を実行中に排他なしでデータ操作のSQL文を実行した場合
    データ操作のSQL文は、更新前の行も更新後の行も参照できない場合があります。

－ ROLLBACK文を実行中に排他なしでデータ操作のSQL文を実行した場合
データ操作のSQL文は、更新前の行も更新後の行も参照できない場合があります。

・ 以下のSQL文は、独立性水準がREAD COMMITTEDであるか、または、占有モードをFREE LOCKにした場合には、各行の読み込み時の最新のコミット済みデータを参照します。

－ 単一行SELECT文

－ 更新可能性句に、FOR READ ONLYを指定したカーソル、または更新可能性句を省略したカーソルのOPEN文

このため、上記のSQL文実行中に更新業務がコミットした場合、次に示すような現象が起こることがあります。

－ SQL文の検索結果に、ある行については更新業務が更新する前の行、別の行については更新業務が更新した後の行が含まれる場合があります。

－ SQL文の検索結果に、ある行の更新前後の行がともに含まれる場合や、いずれも含まれない場合があります。

また、上記の事象にともなって、以下のようにデータの整合性がとれなくなります。

－ 他のトランザクションが更新した行を上書きにより更新してしまう場合がある。

－ 他のトランザクションが削除した行についてUPDATE文:位置づけやDELETE文:位置づけを行うと、対象行が存在しないのでエラーになる。

－ 他のトランザクションが削除した行についてUPDATE文:探索やDELETE文:探索を行うと、対象行が存在しないのでデータなしになる。

このような場合には、更新トランザクションについて、以下に示す対処を行ってください。

－ カーソルの更新可能性句にFOR UPDATEを指定する。

－ 占有モード指定にSHARE LOCK、またはEXCLUSIVE LOCKを指定する。

・ 更新系の処理を行う場合、内部的に更新対象行を特定するための検索処理が行われるため、占有モードが下記のとおり推移します。

1. 更新対象行の検索時
検索する範囲を共有モードで占有
2. 更新対象行(検索結果)のデータの更新時
更新対象行を非共有モードで占有

このため、更新対象行のデータを更新する際に、更新対象行が別のトランザクションから共有モードで占有され、データ更新できないといった事象が発生する場合があります。この場合には、占有モードを以下のように指定することで、この現象を回避することが可能です。

－ 動作環境ファイルのパラメタの設定やイルシデーションロックを用いて、内部的な検索を行うときから非共有モードでの占有とする。

－ SELECT文によって更新する行を検索し、その後、データを更新する場合には、SELECT文にFOR UPDATE句を指定し、あらかじめ、非共有モードでの占有とする。

占有モードを指定する方法は、“I.7.1 排他の属性と選択方法”を参照してください。

COMMIT文およびROLLBACK文に関する注意事項

行単位の排他では、トランザクション中に行った更新内容はCOMMIT文実行時にまとめてインデックスへ反映します。このため、トランザクション中に行った挿入、インデックスキー値更新および削除の件数に比例してCOMMIT文の処理時間も長くなります。

COMMIT時のエラー処理に関する注意事項

COMMIT文実行時に、以下のエラーが発生する可能性があるため、COMMIT文もエラー処理の対象としてください。

・ テンポラリログ領域不足

・ インデックスの容量不足

・ デッドロック

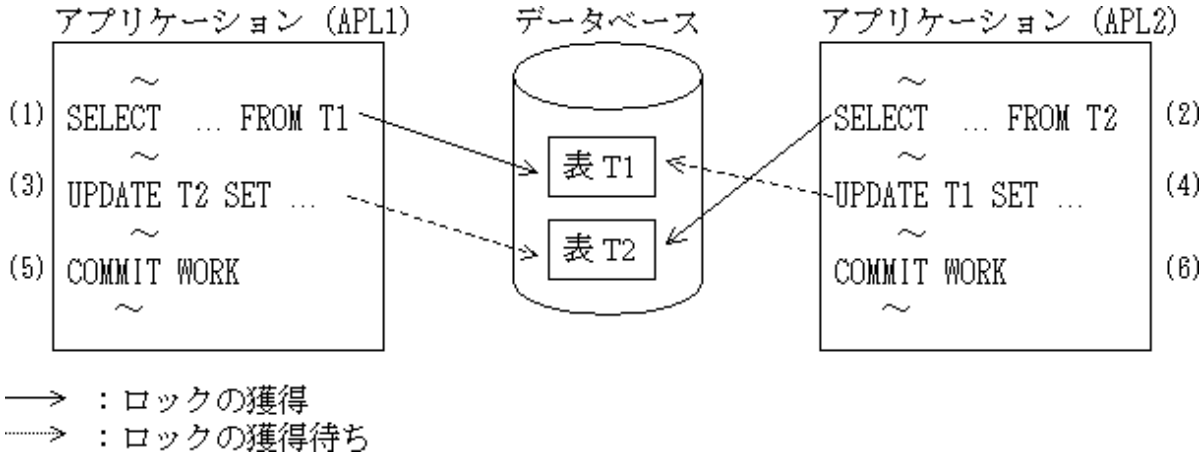
領域の再利用に関する注意事項

行単位の排他を使用する場合、DELETE文またはUPDATE文を実行することによって発生する削除領域は、トランザクション完了後に再利用されます。そのため、同一トランザクション内で大量の件数を削除し、続いて挿入を行うような場合は、削除後に一度COMMIT文を実行してトランザクションを区切ることで、領域を有効に使用することができます。

I.10　デッドロックの対処方法

アプリケーションがデータベースをアクセスする場合の排他制御は、トランザクション単位に行われます。したがって、いくつかのアプリケーションを並行して実行した場合、データ操作の順序によっては、以下に示すようにデッドロックが発生します。このため、アプリケーションには、デッドロックの対処が必要になる場合があります。

図I.16 デッドロックの発生例



以下の順序で処理が行われるとデッドロックが発生します。

(1) APL1が表T1に対して参照ロックを獲得します（(5)のCOMMITまでロックを保持します）。

(2) APL2が表T2に対して参照ロックを獲得します（(6)のCOMMITまでロックを保持します）。

(3) APL1は表T1の参照を終え、表T2を更新するために表T2に対して更新ロックを獲得しようとしますが、表T2にはAPL2が参照ロックを獲得しているため待たされます（表T1はAPL1がロックを獲得したままとなります）。

(4) APL2は表T2の参照を終え、表T1を更新するために表T1に対して更新ロックを獲得しようとしますが、表T1にはAPL1が参照ロックを獲得しているため待たされます（表T2はAPL2がロックを獲得したままとなります）。

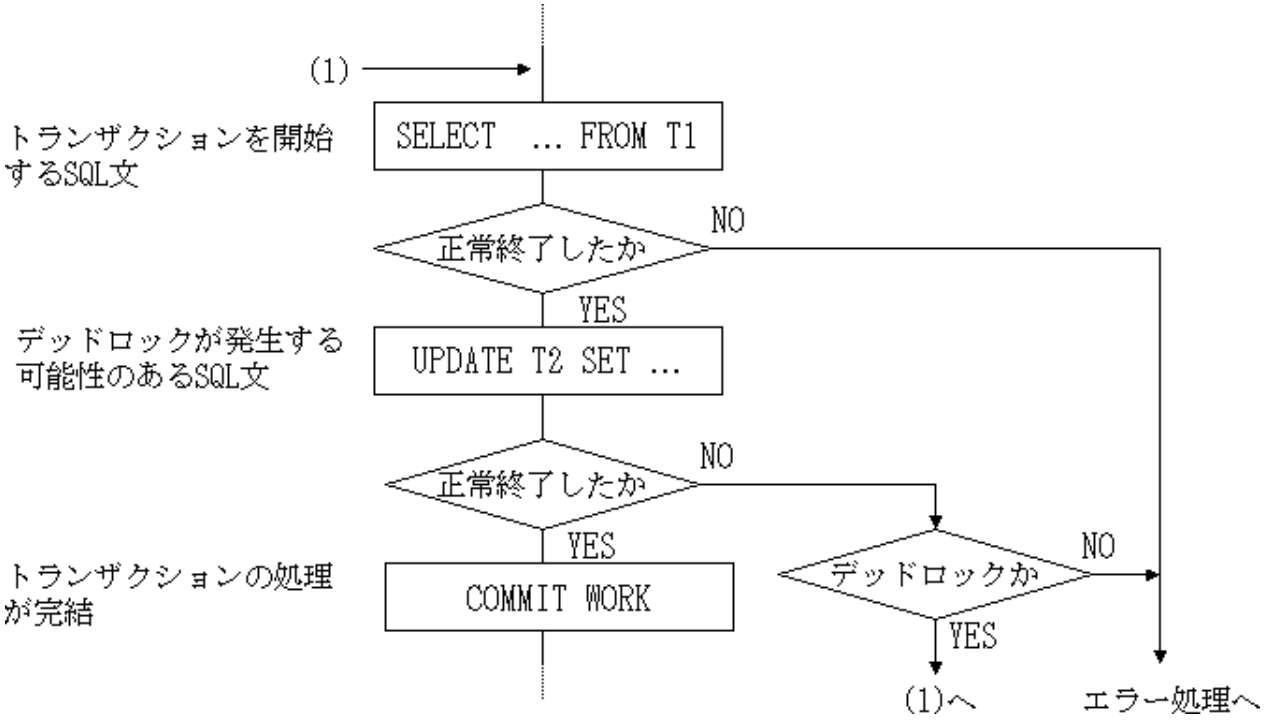
APL1とAPL2は、互いに相手方のトランザクションが終了するのを待たされる状態になります。ところが、APL1の(5)のCOMMIT、およびAPL2の(6)のCOMMITが実行されないために、どちらのアプリケーションも実行できなくなります。この状態がデッドロックです。

デッドロックはSymfoware/RDBが自動的に検出します。Symfoware/RDBがデッドロックを検出すると、複数のトランザクションのうちの1つのトランザクションを取り消して、アプリケーションにエラー復帰します。このとき、SQLSTATEには40001が設定されます。

デッドロックの発生を通知されたアプリケーションでは、一般にデッドロックが発生したトランザクションの再実行を試みます。トランザクションを再実行するには、トランザクションを開始した文の直前に制御を移します。ただし、SQL文で使用するホスト変数をトランザクションの処理途中で変更している場合には、その値をトランザクションの開始時点に戻す必要があります。

以下に、“図I.16 デッドロックの発生例”のAPL1が、トランザクションの再実行を行う場合の処理の流れを示します。

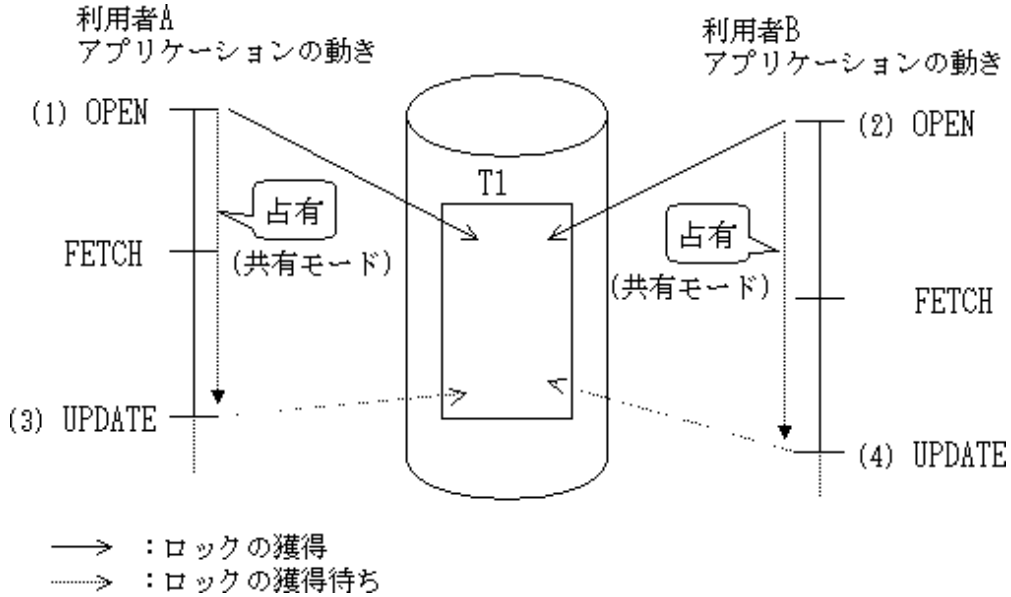
図I.17 デッドロックに対処する処理の例



カーソル操作での注意事項

カーソル宣言の更新可能性句を省略しそのカーソルが更新可能な場合、そのカーソルで更新を行うことができます。この場合、以下に示す操作を行うと、デッドロックが起きることがあります。

図I.18 カーソル操作でデッドロックが起きる例



(1) 利用者Aは、更新可能性句を省略したカーソルのOPEN文を実行します。これにより、表T1で参照した部分に対して共有モードで占有します。

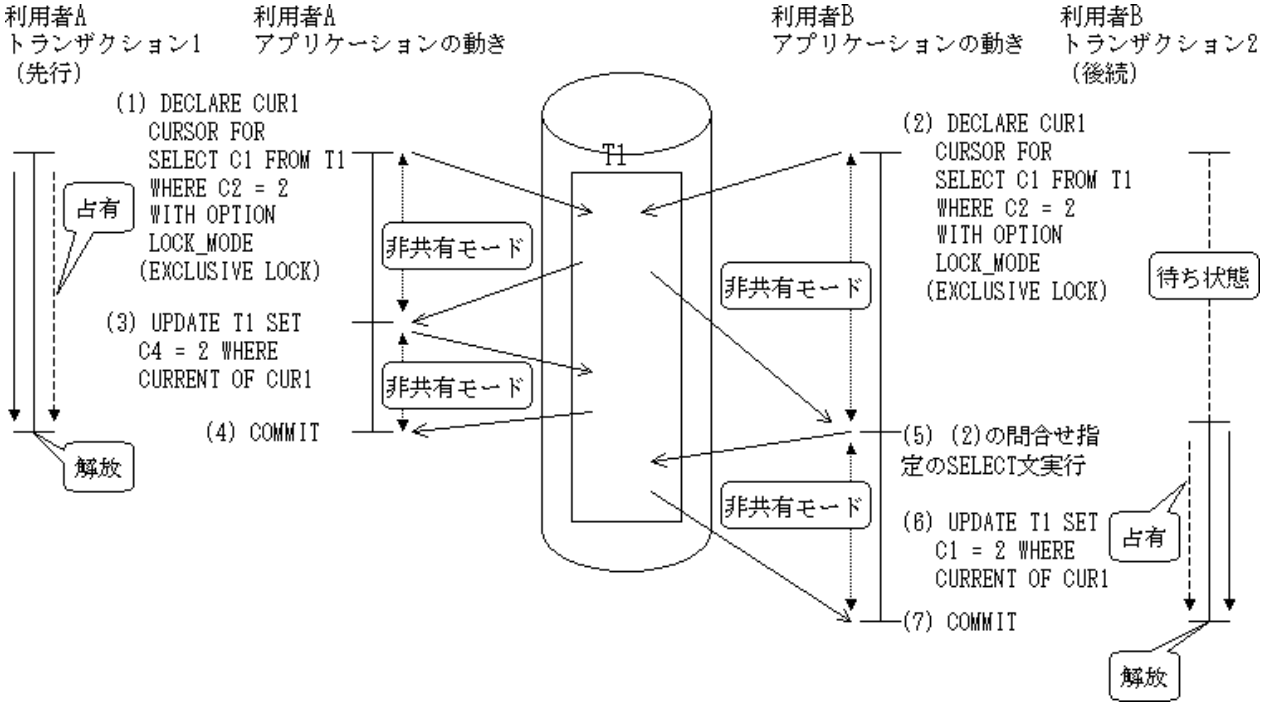
(2) 利用者Bも同じく、更新可能性句を省略したカーソルのOPEN文を実行します。これにより、表T1で参照した部分に対して共有モードで占有します。

(3) 利用者Aが表T1で参照した部分を更新するために、非共有モードで占有しようとしますが、表T1は利用者Bが共有モードで占有しているため待たされます(表T1は利用者Aが共有モードで占有したままとなります)。

(4) 利用者Bが表T1で参照した部分を更新するために、更新ロックを獲得しようとしますが、表T1は利用者Aが共有モードで占有しているため待たされます(表T1は利用者Aが共有モードで占有したままとなります)。

カーソル操作でデッドロックを起こさないため、更新系のSQL文を実行する場合には、カーソル宣言の占有モード指定に、EXCLUSIVE LOCKを指定してください。

図I.19 EXCLUSIVE LOCKを利用したデッドロックの回避例



(1) 利用者Aは、EXCLUSIVE LOCKで参照し、COMMIT(ROLLBACK)終了するまで資源を占有します。

(2) 利用者Bは、EXCLUSIVE LOCKで参照を行うため、利用者Aが資源を解放するまで参照を行えず、待ち状態となります。

(3) 利用者Aは、他のアプリケーションに待たされることなく更新処理を行います。

(4) 利用者AがCOMMIT(ROLLBACK)します。利用者Aの資源は解放されます。

(5) 利用者Aが資源を解放したので、利用者Bの参照が実行されます。この時、資源を占有します。

(6) 利用者Bの更新が実行されます。

(7) 利用者BがCOMMIT(ROLLBACK)します。

I.11 トランザクション実行時間の設定

アプリケーションがSQL文を実行すると、データベースサーバにメモリ資源や排他資源を獲得します。この資源を獲得したまま、アプリケーションに何らかの異常が発生した場合、獲得した資源が回収できなくなることがあります。

このような問題を防止するには、動作環境ファイルの実行パラメタTRAN_TIME_LIMITを使用します。TRAN_TIME_LIMITに1つのトランザクションの最大実行可能時間を指定すると、トランザクションが指定時間を超過した場合は、コネクションを切断し、獲得した資源を回収します。当パラメタは、クライアント用の動作環境ファイルまたはサーバ用の動作環境ファイルに指定することができます。

動作環境ファイルの詳細については、“2.1.1 動作環境ファイル”を参照してください。

索 引

[]

||.....179

[記号]

%.....172
*.....102
_.....174

[A]

ABSOLUTE指定.....134
ACCESS_PLAN.....62,310
ALL集合関数.....103
ANDの真理表.....160
ASC.....126
ASSIST指定でSQL文の最初の1件の検索を高速化する.....84
ASSIST指定でアクセスモデルを指定する.....72
ASSIST指定の種類.....74
ASSIST指定の使用方法.....74
ASSIST指定の注意事項.....79
ASSIST要素の優先順位.....79
AVG関数.....102

[B]

BETWEEN述語.....166

[C]

CASE式を使用してデータを操作する.....190
CAST指定.....189
CLOSE文.....131
COLUMNS.....254
COMMIT時のエラー処理に関する注意事項.....28,420
COMMIT文およびROLLBACK文に関する注意事項…28,420
COMMIT文によるトランザクションの終了.....5,396
CONNECT文.....37,38,57
COUNT.....102
COUNT関数.....103
CREATE TRIGGER文.....208
CURRENT_DATE.....109,115
CURRENT_TIME.....109,115
CURRENT_TIMESTAMP.....109,115

[D]

DATABASE.....249
DEFAULT.....107,115
DEFAULT_CONNECTION.....55
DEFAULT VALUES.....108
DELETEエレメント.....392
DELETE文.....118,136
DELETE文位置づけ.....378
DELETE文探索.....378
DESC.....126
DISCONNECT文.....37,39
DISTINCT.....98
DISTINCT集合関数.....103
DIV_TRACE_FILE.....62
DSI情報.....283
DSO_LOCK.....7,15,17

[E]

ESCAPE.....175
EXISTS述語.....170

[F]

FETCH JOINエレメント.....389
FETCH文.....131
FIRST_ROWS.....84
FIRST指定.....133
FROM句.....138

[G]

GROUP BY句.....149
GROUPINGエレメント.....394

[H]

HAVING句.....155

[I]

INDEX ALL SCAN.....387
INDEX KEY SCAN.....384
INDEX TABLE SCAN.....387
INSERTエレメント.....391
INSERT文.....48,106
IN述語.....167
ISOLATION_WAIT.....23,415

[J]

JOIN_ORDER.....310
JOIN_RULE.....76,310

[K]

KEY_COLUMN_USAGE.....259

[L]

LAST指定.....133
LEADING_TABLE.....76
LEFT.....141
LIKE述語.....171
LIMITを使用して結果件数を限定する.....200

[M]

MAX関数.....103
MERGE JOINエレメント.....388
MIN関数.....103

[N]

NESTED LOOP JOINエレメント.....390
NEXT指定.....133
NOT NULL制約.....107,115
NOTの真理表.....160
NULL.....106,114
NULL述語.....165
NULL値かどうかを調べる.....165
NULL値にする.....106,114
NULL値を含むデータを取り出す.....96,133

[O]
OPEN文....131
ORDER BY句....126
ORの真理表....160
OUTPUTエレメント....393

[P]
PARALLEL....194
PARALLEL_SCAN....194
PARAMETERS....278
PRECEDENCE(1)を指定したSEQUENTIAL構造の特徴と注意事項....31
PRIOR指定....133
PROC_COLS....266
PROCS....263

[R]
rdbadjrcvコマンド....318
rdbaldicコマンド....318
rdbalidxコマンド....333
rdbalmdsiコマンド....334
rdbauditコマンド....333
rdbchksanityコマンド....335
rdbcninfコマンド....335
rdbconbfコマンド....334
rdbcrbfコマンド....334
rdbcrdicコマンド....318
rdbddlexコマンド....215,218,333
rdbdisbfコマンド....334
rdbdmpdicコマンド....318
rdbdmpコマンド....334
rdbdrbfコマンド....334
rdbdvinfコマンド....334
rdbexecsqlコマンド....333
rdbexspcコマンド....318
rdbfmtコマンド....211,334
rdbgcdicコマンド....318
rdbgcdsiコマンド....334
RDBII_ASSISTTABLE....282
RDBII_COLUMN....253
RDBII_COLUMN_USAGE....258
RDBII_COMMENT....259
RDBII_CONSTRAINT....256
RDBII_DATABASE....248
RDBII_DESCRIPTION....257
RDBII_DSI_STATUS....283
RDBII_INDEX....260
RDBII_INDEX_EX....261
RDBII_PARAMETER....278
RDBII_PRIVILEGES....268
RDBII_PROC....262
RDBII_PROC_COL....265
RDBII_PROC_EX....264
RDBII_ROLE....279
RDBII_ROLE_GRANT....282
RDBII_ROLE_PRIV....280
RDBII_SCHEMA....249
RDBII_SEQUENCE....274
RDBII_TABLE....250
RDBII_TABLE_EX....251
RDBII_TABLE_USAGE....257
RDBII_TRIG_COLUMNS....273
RDBII_TRIGGER....270
RDBII_USER....277
rdbinfコマンド....334
rdbinhコマンド....334
rdbloggroupコマンド....335
rdblogコマンド....334
rdbmrdmpコマンド....334
rdbmrinfコマンド....335
rdbmrrcvコマンド....318
RDBNAME....43
rdbpmtコマンド....334
rdbprdicコマンド....318,334
rdbprtbfコマンド....334
rdbprtmsgコマンド....335
rdbprtコマンド....334
rdbpsコマンド....334
rdbrcvadtコマンド....318
rdbrcvdicコマンド....318
rdbrcvコマンド....318
rdbrlsコマンド....334
rdbrtrコマンド....334
rdbsaloaderコマンド....334
rdbsarコマンド....334
rdbscldirコマンド....318
rdbsloaderコマンド....211,334
rdbspcinfコマンド....334
rdbsuloaderコマンド....334
rdbtermコマンド....334
rdbudsiコマンド....334
rdbunladt コマンド....334
rdbunlsqlコマンド....333
rdbunlコマンド....334
rdbupsコマンド....334
rdbuptコマンド....52,333
rdbxarcvコマンド....318
READ COMMITTED....9,401
READ ONLY....7,399
READ UNCOMMITTED....10,404
READ WRITE....7,399
RELATIVE指定....134
REPEATABLE READ....9,400
RIGHT....141
R_LOCK....7,15,399
ROLE_GRANTS....282
ROLE_PRIVILEGES....281
ROLES....279
ROLLBACK文によるトランザクションの終了....5,397
ROUTINE_SNAP....61
ROW_ID....193
ROW_ID検索....358
ROWNUMを使用してデータ操作する....200
RSC_PRIVILEGES....269

[S]
SCANエレメント....381
SCHEMATA....249
SCROLL....133
SEQUENCES....275
SERIALIZABLE....8,400
SERVER_ENV_FILE....312
SERVER_SPEC....56
SET SESSION AUTHORIZATION文....197
SET TRANSACTION文....15,408
SORT_MEM_SIZE....59
SQL_SNAP....61
SQLSTATE値....336
SQLSTATE値とトランザクションの関係....340
SQL_TRACE....63,311
SQLサーバ名....57
SQL制御文....214
SQL手続き文....214
SQL手続き文の実行情報をファイルに出力する....61
SQLの使用方法....89
SQL文の簡易実行情報....282
SQL文の条件式には明示的に条件を記述する....87
SQL文の情報をファイルに出力する....61
SQL文の処理結果異常とトランザクション....13,406
SQL文の共通規則....90
SS_RATE....311
SUBSTRING....179
SUM関数....103
Symfoware/RDBの定量制限....348

[T]
TABLE ALL SCAN....382
TABLE_CONSTRAINTS....256
TABLE KEY SCAN....381
TABLE PARALLEL SCAN....383
TABLES....251
Textアダプタの定量制限....356
TID....381
TID_SORT....311
TID_UNION....311
TIDユニオンマージ....362
TRAN_SPEC....13,406
TRAN_TIME_LIMIT....57,58
TRIGGERED_COLUMNS....273
TRIGGER EXECUTIONエレメント....394
TRIGGERS....272
TRIM関数....179

[U]
UNION....146
UNIONによる集合和の順序....147
UPDATEエレメント....393
UPDATE文....113,134
UPDATE文位置づけ....377
UPDATE文探索....376
USE_INDEX....75
USERS....277
USQL_LOCK....311

[W]
WAIT_TIME....58
WHERE句....139
WORK_ALLOC_SPACESIZE....59
WORK_MEM_SIZE....60
WORK_PATH....60,315,316

[X]
XMLアダプタの定量制限....357
XMLデータの文字コード....227

[あ]
アクセスプラン....65
アクセスプランの詳細....358
アクセスモデル....65
アクセスモデルとは....65
アクセスモデルの確認....80
アクセスモデルの指定....68
アクセスモデルの種類....358
アクセスモデルの選択....67
アクセスモード....7,399
値式....104,168,169
値の集合と比較する....169
アプリケーションおよびRDBコマンド間の排他....318
アプリケーション開発時の性能チューニング....66
アプリケーション開発における基礎知識....1
アプリケーション実行時のメモリ見積り....316
アプリケーションの異常終了とトランザクション....14,407
アプリケーションの開発....50
アプリケーションの終了によるトランザクションの終了....6,398
アプリケーションの性能....65
アプリケーションの設計に必要な概念....2
アプリケーションの接続先や通信に関する実行パラメタ....55
アプリケーションの動作環境....50
一意性制約の定義情報....257
イルシデーションロック....15,408
イルシデーションロック(elucidation lock)....15
インデックス検索....81,359
インデックス検索と表データ取得....360
インデックス定義拡張情報....261
インデックス定義情報....260
インデックスに対する占有....17,18,409
インデックスを定義する....68
エスケープ文字....175
エレメント....66
エレメントとは....66
エレメントの種類....381
演算式....100
演算の優先順序....100
オプティマイザ....65

[か]
会社表....92
概数を使用するときの注意事項....48
カタログについて....48
カーソル宣言....48
カーソル操作での注意....30,40,422
カーソルにより取り出す行の順序....125

カーソルの位置づけ.....131
カーソルのオープン.....131
カーソルの概念.....120
カーソルのクローズ.....131
カーソルを移動する.....133
カーソルを使用したデータ操作の順序.....121
カーソルを使用するデータ操作の方法.....120
カーソルを宣言する.....123
行値構成子を使用してデータを操作する.....206
行識別子.....193
行数を求める.....101
行単位の排他を使用する場合の注意事項.....26,418
共有モード.....17,409
共用ソート作業域.....315
クライアント用の動作環境ファイル.....306
クライアント用の動作環境ファイルの作成.....54
クライントとサーバの接続方法.....36
クラスタキー検索.....359
クラスタキー値.....382
クラスタサービス名.....45
グループ化関数.....149
グループ化して操作する.....155
グループ化列.....149
グループ表.....148
グループ表の集計結果を再集計する.....158
結果のデータ型.....100,104
権限情報.....268
検索対象がソートテーブルである場合.....383
検索対象がデータベースの表である場合.....382
検索対象がワークテーブルである場合.....383
限定述語.....169
限定値リスト.....167
合計値.....101
降順ソート.....126
更新可能カーソル.....129,134,136
更新可能性句.....131
更新系アクセスモデル.....375
更新する.....134
更新操作での注意.....41
更新対象がインデックスである場合.....393
更新対象がデータベースの表である場合.....393
コネクション.....2,37,38
コネクション情報.....292
コネクションの接続.....37
コネクションの切断.....37
コネクションの動作環境のパラメタ設定情報(一括出力).....286
コネクションの動作環境のパラメタ設定情報(順序付き出力)288
固有ソート作業域.....315

[さ]

在庫管理データベースの各表の列の属性.....93
在庫表.....91
在庫表、発注表および会社表の関連.....94
最小値.....101
最大値.....101
作業用ソート領域.....59
作業用テーブル.....60
作業用テーブル域.....60
削除する.....136
削除対象がインデックスである場合.....393
削除対象がデータベースの表である場合.....392
サーバとの接続情報.....56
サーバ用の動作環境ファイル.....312
時間の設定.....109,115
資源の競合が起きた場合の制御.....24,416
資源の見積り.....315
時刻印の設定.....109,115
システムの動作環境のパラメタ設定情報(一括出力).....290
システムの動作環境のパラメタ設定情報(順序付き出力)....291
システム表.....242
システム表の概要.....242
システム表の参照.....243
システム表の詳細.....248
システム用の動作環境ファイル.....302
四則演算を行う.....117
四則演算をする.....99
実行制御ASSISTの使用方法.....84
実行パラメタの種類.....312
集合関数.....102,149
集合関数の種類と指定方法.....102
集合和.....146
集合和を求める.....146
順序情報.....274
順序を使用してデータを操作する.....198
ジョインASSISTの使用方法.....76
ジョイン系アクセスモデル.....81,365
条件を指定する.....139
昇順ソート.....126
使用する資源に関する実行パラメタ.....58
状態コード.....336
状態変数.....336
冗長な行を取り除いて取り出す.....98
省略値を使用する.....107,115
初期パラメタの設定.....55
数値のデータを操作する.....175
スキーマ.....90,98
スキーマ定義情報.....249
スキーマ名を省略する.....196
スナップファイル.....62
制約ASSISTの使用方法.....75
制約系アクセスモデル.....81,358
制約定義情報.....255
セクション.....66
セクションとは.....66
セション.....2,197
接続サーバ種別.....57
選択相手リスト.....95
選択リスト.....95
占有の資源.....15,318,330,408
占有の単位.....15,408
占有の強さ.....318,330
占有の範囲.....16,409
占有待ちの有無.....23,415
占有モード.....17,409

占有モード指定.....19,34,410
占有モードの指定.....17
相関名.....143
挿入対象がインデックスである場合.....391
挿入対象がソートテーブルである場合.....392
挿入対象がデータベースの表である場合.....391
挿入対象がワークテーブルである場合.....392
挿入値リストを指定したINSERT文.....379
ソート作業域の見積り.....315
外結合型.....141
その他の性能チューニング方法.....84
ソート.....126
ソートキー.....126
ソート作業域.....60
ソート指定.....127
ソートテーブルを利用してジョインする.....81

[た]

単一行SELECT文.....48,95
単項演算子.....100
探索条件を指定する.....159
単純なトリガ.....208
注釈情報.....259
チューニング.....50
通信データを暗号化する場合の設定.....47
通信方法.....57
月ごとに表をグループ化する.....153
ディクショナリ資源の占有.....318
定量制限.....348
デッドロックの対処方法.....29,421
デバッグおよび性能分析に関する実行パラメタ.....60
データ型を変換してデータを操作する.....189
データ資源名.....57
データ操作文.....214
データ操作文に関する注意事項.....27,48,419
データに対する占有.....17,409
データの取出し.....131
データベース資源の占有.....330
データベース定義情報.....248
データベースの定義での注意.....42
データベースの例.....91
データを更新する.....113
データを削除する.....118
データを追加する.....106
データを取り出す.....98
データを場合分けして表をグループ化する.....152
問合せ式.....124
問合せ指定.....110,124
問合せ指定の結果列に名前を付加する.....197
問合せ指定を指定したINSERT文.....380
同一資源に対する各機能の同時実行時の振る舞い.....331
動作環境ファイル.....50
動作環境ファイルの位置づけ.....51
動作環境ファイルの記述形式.....53
動作環境ファイルの種類.....50
動作環境ファイルの特徴.....52
動作環境ファイルのパラメタ一覧.....295
動作環境ファイルの優先順位.....53
導出表.....158
独立性水準.....7,399
トランザクション.....2,13,14,340,406,407
トランザクション管理文.....214
トランザクション実行時間の設定.....31,423
トランザクション制御の概要.....3,395
トランザクションとカーソル操作.....11,405
トランザクションと排他制御.....3
トランザクションの開始.....3,395
トランザクションの終了.....3,4,395,396
トランザクションモード.....6,398
トランザクションを超越するカーソル.....13,406
トリガ定義.....208
トリガ定義情報.....270
トリガ定義での注意事項.....211
トリガ列一覧情報.....273
トリガを使用したデータ操作の方法.....208
取出し相手リスト.....132

[な]

名前.....90
二項演算子.....100
日時を設定する.....109,115
任意文字指示子.....174
任意文字列指示子.....172
ネスティドループジョイン(実表を直接読み込んでジョインする)372
ネスティドループジョイン(実表を直接読み込んでジョインする)83
ネスティドループジョイン(ワークテーブルを利用してジョインする).....373

[は]

排他.....26,418
排他する資源.....318
排他制御.....3,14,395,407
排他の属性の選択方法.....14,408
発注表.....92
比較演算子を用いて比較する.....161
比較可能なデータ型.....162
比較述語.....161
非共有モード.....17,409
日付の設定.....109,115
日付のデータを操作する.....182
被トリガSQL文の実行でエラーの場合の注意事項.....341
標識変数.....96,106,114,133
表宣言.....196
表定義拡張情報.....251
表定義情報.....250
標的レコードを位置づける部分.....375
標的レコードを更新する部分.....375
表の全件検索.....363
表をグループ化する.....148
表を導出する.....138
ファイルディスクリプタ数の見積り.....317
ファンクションルーチンとは.....216
ファンクションルーチンの定義.....217
ファンクションルーチンのパラメタのデータ型と対応するデータ形式.....342

ファンクションルーチンの利用方法.....216
ファンクションルーチンを登録する.....218
ファンクションルーチンを利用するアプリケーションの作成の流れ.....217
フェイルオーバ運用.....45
フェッチジョイン(インデックスを利用してジョインする).....83,369
フェッチジョイン(クラスタキーを利用してジョインする).....371
フェッチジョイン(実表を直接読み込んでジョインする).....371
複数コネクション.....38
複数コネクションでSQL文を実行する場合の注意事項.....40
複数コネクションの接続.....38
複数コネクションの切断.....39
複数列から構成されるインデックスの検索範囲.....386
副問合せ.....131,167
副問合せを用いたIN述語.....168
副問合せを用いた比較述語.....164
副例外コード.....336
プロシジャルーチン定義文.....214
プロシジャルーチンとは.....213
プロシジャルーチンの定義.....214
プロシジャルーチンの利用方法.....213
プロシジャルーチンを登録する.....215
プロシジャルーチンを呼び出すトリガ.....208
プロシジャルーチンを利用するアプリケーションの作成の流れ213
ブール演算子.....159
平均値.....101
並列クエリ.....193
並列指定.....194
並列スキャン.....364
ホスト変数.....95,132
ホスト名.....57
ポート番号.....57

[ま]

待ち時間.....58
マルチRDB運用のコネクション.....42
マージジョイン(インデックスを利用してジョインする).....81,365
マージジョイン(インデックスを利用してジョインをして表データも取得する).....366
マージジョイン(ソートテーブルを利用してジョインする). .82,367
メモリの見積り式.....316
文字部分列関数.....179
文字列のデータを操作する.....179
文字列を表現する表情報.....257

[や]

ユーザパラメタ設定情報.....278
読込み専用カーソル.....129

[ら]

リモートアクセス時の注意事項.....44
領域の再利用に関する注意事項.....28,421
利用者情報.....276
利用者に対するロールの付与情報.....282
利用者を変更する.....197
ルーチン定義拡張情報.....264
ルーチン定義情報.....262
ルーチンパラメタ情報.....265
ルーチンを使用したデータ操作の方法.....213
例外コード.....336
レコードの格納アドレス.....382
列一覧情報.....258
列定義情報.....253
連結.....179
連結演算子.....179
論理演算子の指定方法.....159
論理演算子の評価順序.....161
ロール情報.....279
ロールに付与された権限の情報.....279
ロールバック.....13,406