Unify® DataServer: Developing a Database (データベースの開発)

UNIFY

Written by: Linda Costello
Contributions by: Anna Carlile

The Unify Corporation Publications Department values and appreciates any comments you may have concerning our products or this document. Please address comments to:

Accell/DataServer Product Manager
Unify Corporation
3927 Lennane Drive
Sacramento, CA 95834−1922
(800) 24−UNIFY
(916) 928−6400
FAX (916) 928−6406

Part Number: 7809−01

目次

このマニュアルについて

このマニュアルは Unify DataServer を使って新しいデータベースを開発するためのガイドとして書かれました。

マニュアル内のステップは一般的なステップです。**追加ヘルプ**に各ステップに関連する詳細な情報を含む Unify DataServer マニュアルをリストしています。データベースとアプリケーションの設計について完全に説明した、他の Unify DataServer マニュアルはありませんが、専門の書店でこれらのプロセスの解説本がたくさん入手できです。また、付録の「正規化したテーブル」でも、よいデータベース設計の基本の 1 つである、正規化したテーブルについて概説しています。

マニュアル内のステップは 2 つのカテゴリに分かれています。1 つめののカテゴリは、データベース管理者がデータベースを作成するときのステップです。2 つめは、アプリケーション開発者がデータベースを変更してアプリケーションを作成するときのステップです。

データベース管理者がアプリケーション開発者を兼任するような場合でも、データベース開発のプロセスを、この 2 つの観点から考察することが重要です。

なお、このマニュアルでは以下の絵表示を使用します :

追加ヘルプ
追加ヘルプによって、記述されているトピックスについてのより多くの情報が示されている場所を知ることができます。 ■

参考文献

RDBMS を設計して管理したり、トランザクションやロック、復旧を管理することについての追加情報として、次の書籍を推奨します：

Bernstein, Philip A., Vassos Hadzilacos, and Nathan Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley Publishing Company, 1987.

Chamberlin, Donald D. “Relational Data-Base Management Systems.” *Computing Surveys 8* (March 1976).

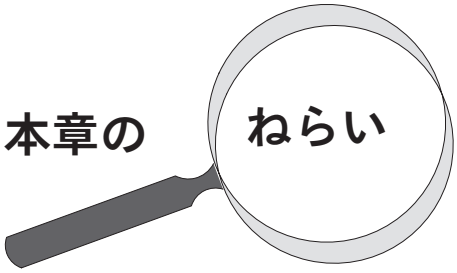
Date, C.J. *An Introduction to Database Systems*. 3rd ed. Addison-Wesley Publishing Company, 1982.

Martin, James. *Computer Data-Base Organization*. 2nd ed. Prentice-Hall, Inc., 1975.

Salzberg, Betty. *An Introduction to Data Base Design*. Academic Press, 1986.

Salzberg, Betty. “Third Normal Form Made Easy.” *SIGMOD Record,* Vol 15, No. 4 (December 1986).

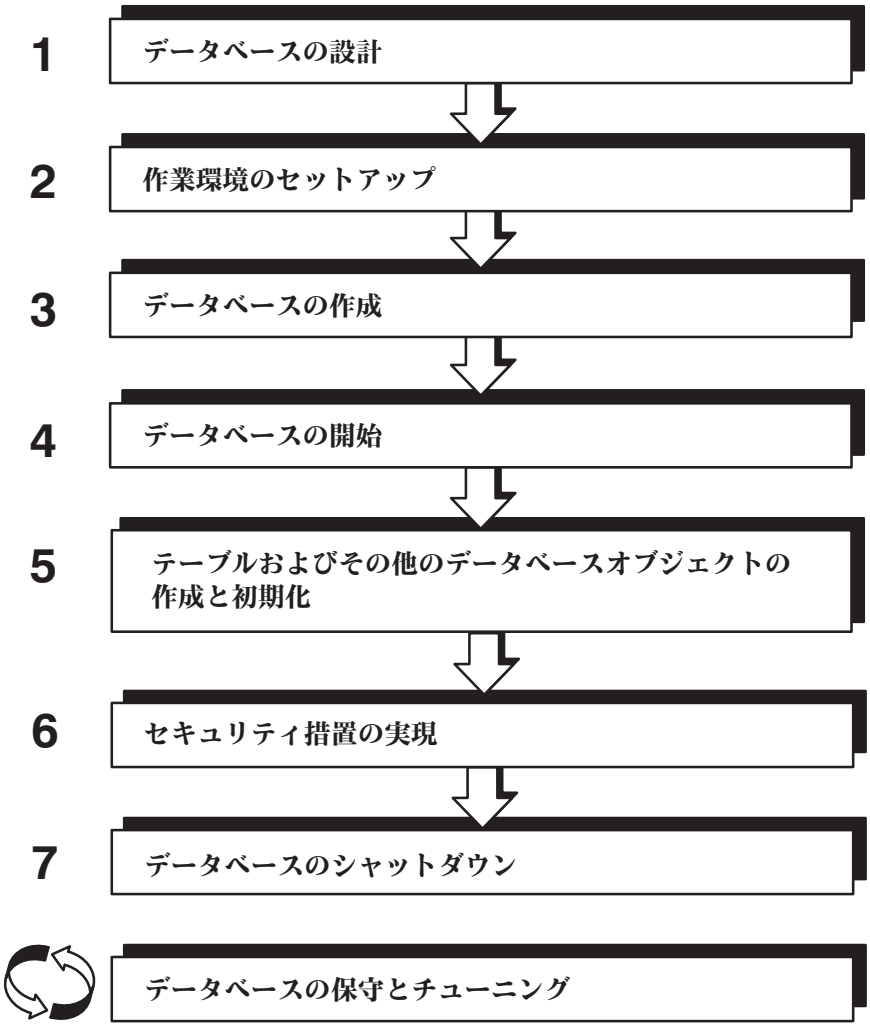
DBA のタスク



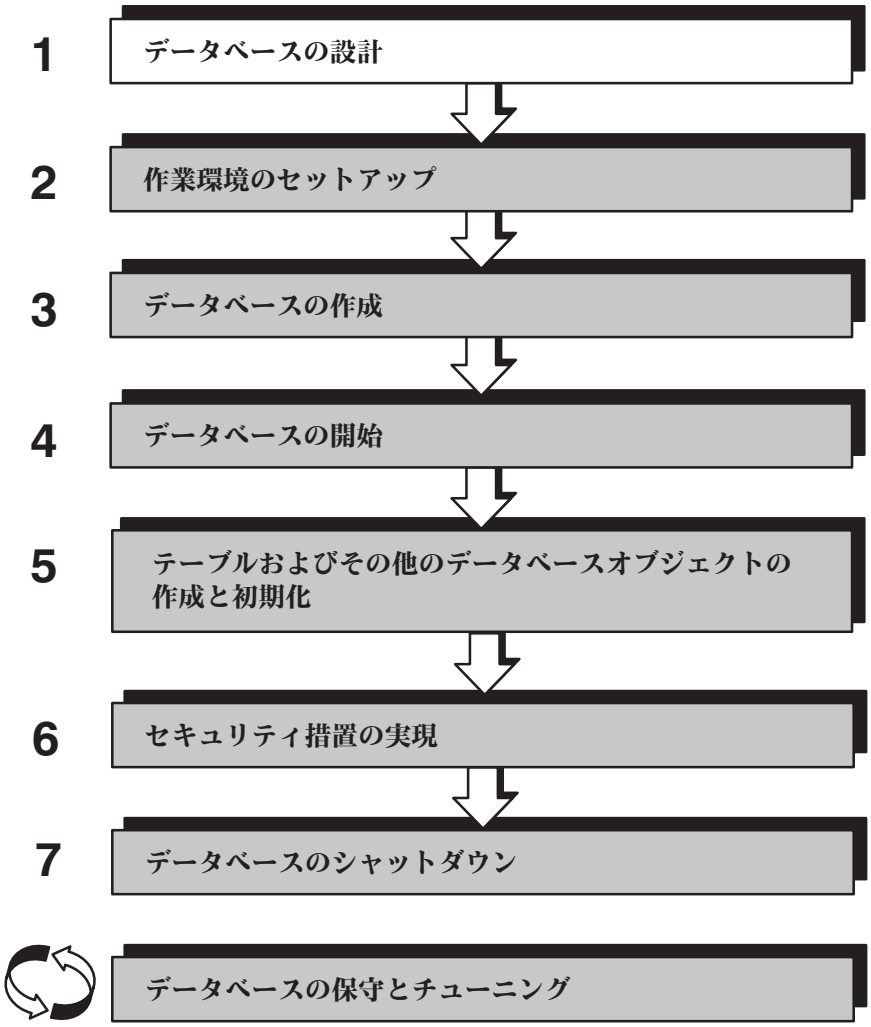
このセクションでは、データベースを開発、保守するためにデータベース管理者（DBA : DataBase Administrator ）が実行するタスクを説明します。DBA とは、DBA 権限を付与された Unify DataServer ユーザをいいます。本章で述べるタスクの大半は実行に DBA 権限が必要です。

タスクは、以下のダイアグラムに示すとおりです。最初の7つのタスクは、数字で示される順序で実行されなければなりません。8 番目のタスクは、データベースを使う間は繰り返し実行されます。

データベースの開発 : DBA のタスク



データベースの開発 : ステップ 1



ステップ 1 : データベースの設計

データベースを開発する最初のステップは、データベースの設計です。

データベースをうまく設計するには、実世界の状況をモデルにし、迅速で能率的なデータの検索と更新が可能で、しかも設計変更が容易にできるようにしなければなりません。またデータベースにはどんな情報が格納され、どんな情報グループがまとめられるかを決定しなければなりません。

データベースの設計は、データベースのパフォーマンスに最大の影響を与えます。必ず適切なガイドラインを研究して、データベース設計のときにはそのガイドラインに従ってください。

データベース設計には 2 つの側面があります。論理的設計と物理的設計です。

論理的設計

データベースの論理的設計とは、データベースのテーブル設計で、テーブル内部およびテーブル間の関係が含まれます。

論理的設計の最も重要な要素は、正規化されたテーブルを設計することです。正規化されたテーブルは、更新が最も能率的です。

論理的設計にはまた、それらのテーブルに関連するアクセス方式も重要です。アクセス方式は、そのテーブルのデータ検索方法を決定します。

Unify DataServer は、5 種類のアクセス方式を提供します。各々のテーブルで実行頻度が最も多い問い合わせのタイプに基づいて適切なアクセス方式を選択することが重要です。

追加ヘルプ

項目	参照先
正規化されたテーブルの設計	本マニュアルの付録A
適切なアクセス方式の選択	『*Unify DataServer: Managing a Database*』の「アクセス方式」

物理的設計

Unify DataServer　データベースの物理的設計も、パフォーマンスに大きく影響します。物理的設計には、データベースファイルが置かれる格納領域と使用されるファイルのタイプが含まれます。ファイルには、データベースからのデータか、データベース管理に関連するデータが含まれます。

Unify DataServer データベースファイルは、次の 3 つのファイルタイプの 1 つを選択して格納することができます :

- 通常ファイル（regular files）
- プリアロケートファイル （preallocated files）
- ローデバイス （raw devices）

ファイルタイプの長所と限界をよく理解した上で、ファイルタイプを選択してください。データベースファイルによっては、データベースを作成した後でファイルタイプを変更することができない場合があります。

データベースは単一のファイルにするには大きすぎることが多いので、Unify DataServer はデータベースのデータをボリュームと呼ばれる複数ファイルに格納します。

必要なボリュームの数は、データベースのサイズによります。作成中のデータベースが小さい（ 100 メガバイト以下）場合には、格納に必要な量はおそらく 1 ボリュームを超えることはありません。

しかしその場合でも、I/O の競合を減らすために複数のボリュームを使うことがあります。

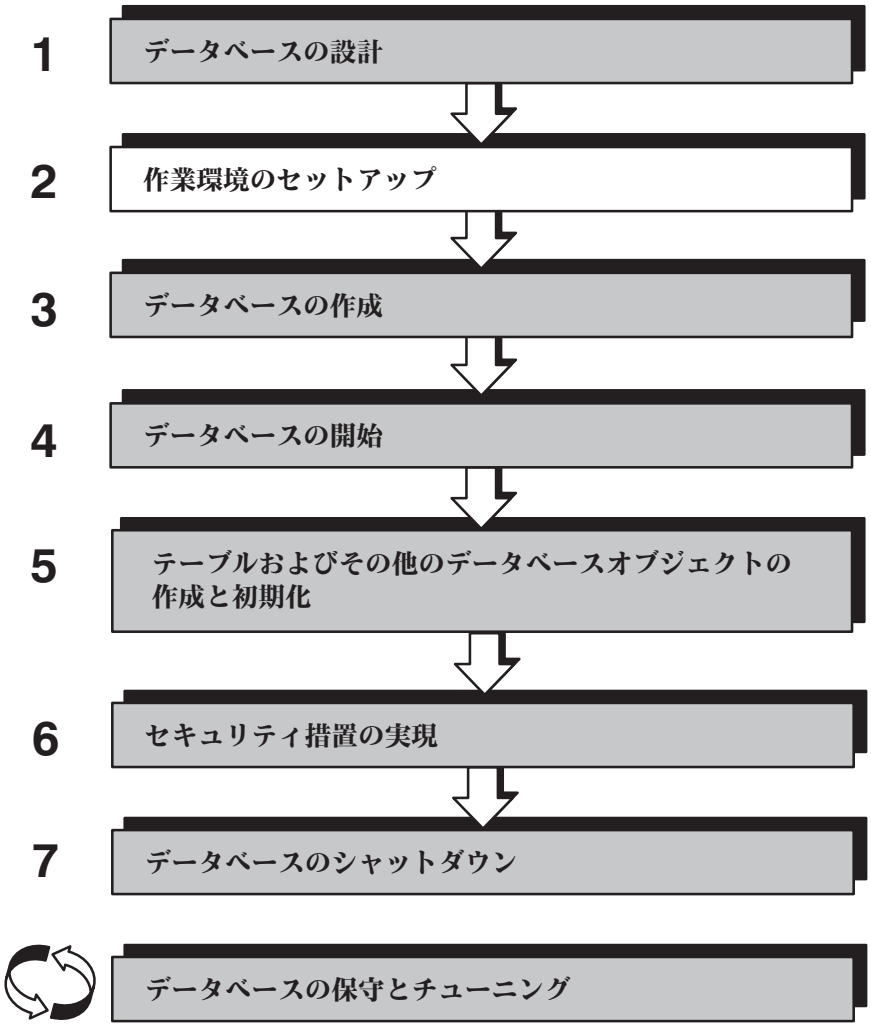
物理的設計を行う場合に考えなければならない点が、他にも 2 つあります :

- 新しいデータベースがシステムリソースにあたえる影響
- プログラム障害、システム障害またはメディア障害からの復旧方法

追加ヘルプ

項目	参照先
Unify DataServer によって作成されたファイルとそのデフォルト名	『*Unify DataServer: Managing a Database*』の「Unify DataServer ファイル」
通常ファイル、プリアロケートファイル、またはデバイスを使用する場合	『*Unify DataServer: Managing a Database*』の「適切なファイルタイプの選択」
データベースが使用する格納領域	『*Unify DataServer: Managing a Database*』の「格納領域」
物理的設計がシステム障害からの復旧にあたえる影響	『*Unify DataServer: Managing a Database*』の「データベースの復旧」の章

データベースの開発 : ステップ 2



ステップ 2 : 作業環境のセットアップ

データベースを設計したら、作業環境をセットアップしなければなりません。作業環境とは、使用中のオペレーティングシステムで Unify DataServer を能率的に実行することを可能にするものです。

作業環境の 2 つの重要な要素は、データベースファイルの物理的ストレージと、コンフィギュレーション変数の設定です。

データベースは、複数のデータベースファイルに格納されます。データベースファイルの位置はたいてい 1 つのディレクトリです。適切な空き領域を割り当てるために、データベースがどの程度の大きさになるのか予想しなければなりません。

デバイスに格納されるファイルについては、使用する前にデバイスの準備をしなければなりません。

作業環境の大半は、さまざまな値に設定可能な 100 以上のコンフィギュレーション変数によって制御されます。これらの変数にはたいてい適切なデフォルト値がありますが、次のコンフィギュレーション変数は、ユーザが設定しなければなりません :

- *SHMKEY*
- *DBNAME*
- *UNIFY*
- *PATH*

データベースにリモートからアクセスする場合には、追加のコンフィギュレーション変数を設定しなければなりません。

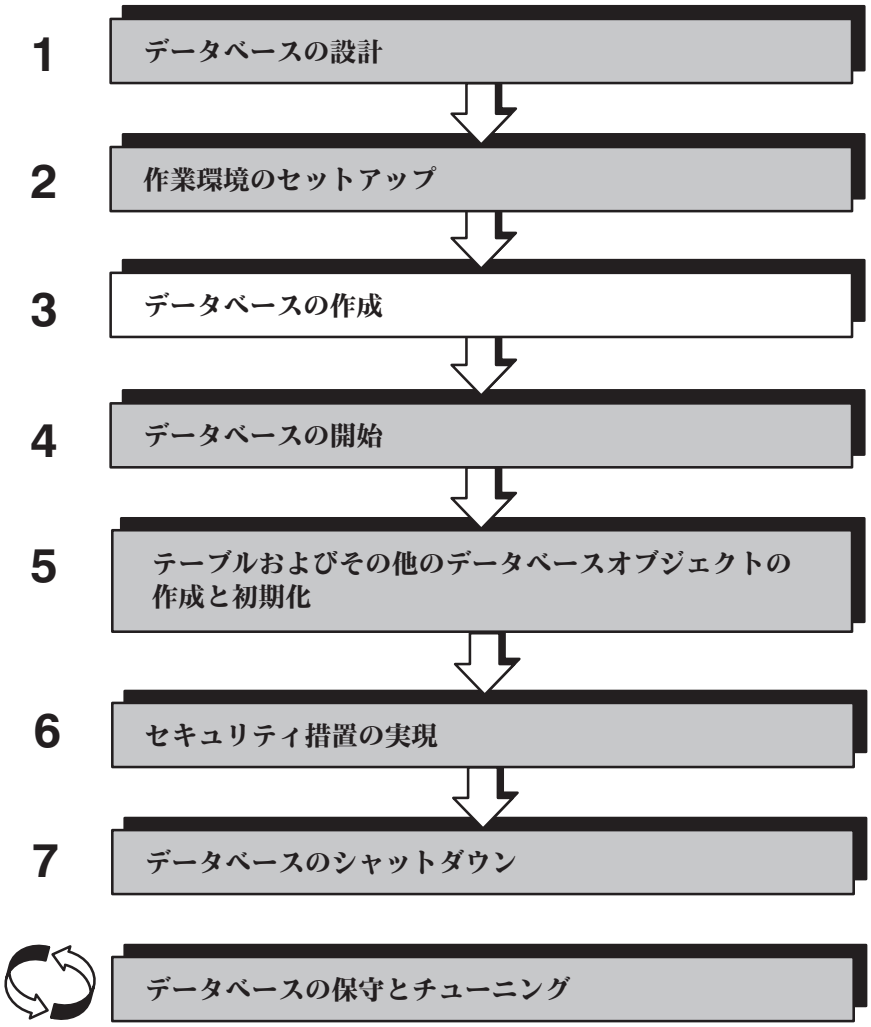
コンフィギュレーション変数が正しく設定されていることを確認するには、**config** ユーティリティを使ってコンフィギュレーション変数の値を表示します。

作業環境をセットアップする前に、Unify DataServer を使用中のシステムにインストールしておかなければなりません。

追加ヘルプ

項目	参照先
デバイスの作成と初期化	『*Unify DataServer: Managing a Database*』の「デバイスの作成と初期化」
コンフィギュレーション変数の設定と **config** ユーティリティの使用	『 *Unify DataServer: Managing a Database*』の「データベース環境のコンフィギュレーション」の章
config ユーティリティ構文	『*Unify DataServer: Configuration Variable and Utility Reference*』
リモートからのデータベースアクセス	『*Unify/Net Guide*』
データベースファイルのサイズの見積もり	『 *Unify DataServer: Managing a Database*』の「データベースファイルの管理」の章
Unify DataServer のインストール	『 *Installing ACCELL/SQL*』のマニュアル

データベースの開発 : ステップ 3



ステップ 3 : データベースの作成

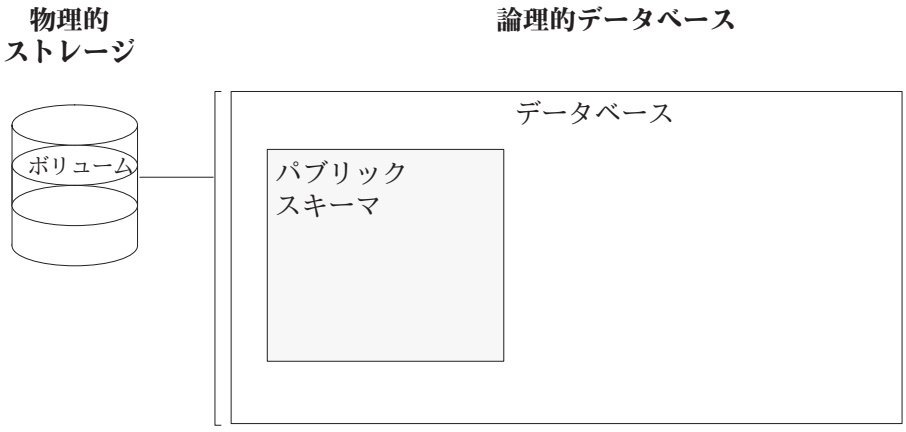
データベースの設計と作業環境のセットアップが終了したら、データベースおよびテーブルの作成を始めることができます。まず、データベースを作成します。しかし、データベースを作成する前に、ユーザ独自のコンフィギュレーションファイルを作成しておくと便利な場合もあります。

データベースを作成するときには、データベースオブジェクトのフォルダー、つまり格納場所を作成します。オブジェクトの 1 つであるスキーマは、テーブルをグループ化するために使用します。テーブルには、データベースに保存しようとするデータがすべて保持されます。

データベースを作成すると、 **PUBLIC** スキーマが必ず作成されます。**PUBLIC** スキーマは、すべてのユーザがアクセスできるスキーマです。**PUBLIC** スキーマは、ユーザがデータベースのセキュリティを管理するのに重要な役割を果たします。

また、データベースを作成するときには、物理的ストレージの 一番目のボリュームであるルートボリュームが作成されます。そのボリュームは、データベースのテーブルにデータを格納するために使用されます。ユーザがボリュームに直接アクセスすることはありません。その代わりにテーブルにアクセスします。Unify DataServer は、ユーザがテーブルに加えた変更に基づいてボリュームに適切な変更を加えます。

概念図 : データベースの作成後



以下のガイドラインに従って、データベースを作成してください：

☐ **準備**

データベースファイルをどこに作成するか決定します。そのディレクトリを作成してください。データベースに関連する約 15 個のファイルが、このディレクトリ内に作成されることになります。

データベースのルートボリュームや他のボリュームが、通常ファイルや、プリアロケートファイルではなく、デバイスに格納しなければならない場合には、デバイスを作成し初期化しなければなりません。

どのデータベースファイルもファイル名を変更する場合は、適切なコンフィギュレーション変数の設定を変更してください。変更を行わないと、デフォルトのファイル名でデータベースファイルが作成されます。

☐ **作成**

対話型 SQL/A 文の **CREATE DATABASE** を使用すれば、簡単にデータベースを作成できます。**creatdb** ユーティリティ、または RHLI 関数の **uadddb** を使うこともできます。

このステップのすべての方式は、データベース上の物理的ストレージの最初のボリューム、 volume 1 を作成します。volume 1 は、ルートボリュームとも呼ばれます。

☐ **確認**

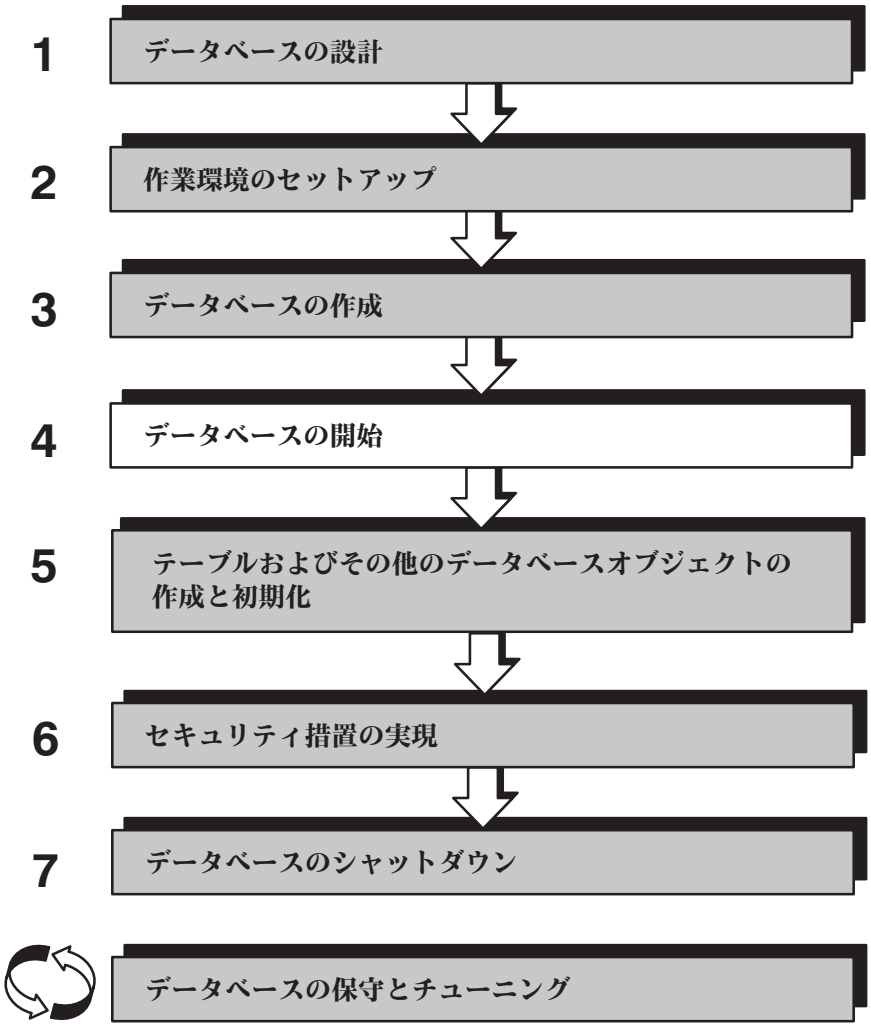
データベースが正確に作成できていることを確認するには、そのデータベースを含むディレクトリの内容を一覧表示します。そのディレクトリには、データベース用に作成された約 15 個のファイルが含まれています。

ユーザがコンフィギュレーション変数を使ってファイル名を変更していなければ、そのファイルには **bu**、**cfg**、**db**、**err** などデフォルトのサフィックスが付いています。

追加ヘルプ

項目	参照先
PUBLICスキーマ	『*Unify DataServer: Managing a Database*』の「パブリックスキーマとプライベートスキーマ」
ルートボリューム	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「ルートボリューム情報」
対話型 SQL/A によるデータベースの作成	『*Unify DataServer: Writing Interactive SQL/A Queries.*』の「データベースオブジェクトの作成と削除」の章
CREATE DATABASE 文	『*Unify DataServer: SQL/A Reference*』
creatdb ユーティリティ	『*Unify DataServer: Configuration Variable and Utility Reference*』
uadddb 関数	『*Unify DataServer: RHLI Reference*』
Unify DataServer デフォルトのファイル名と関連するコンフィギュレーション変数	『*Unify DataServer: Managing a Database*』の「Unify DataServer ファイル」
デバイスの作成と初期化	『*Unify DataServer: Managing a Database*』の「デバイスの作成と初期化」

データベースの開発 : ステップ 4



ステップ 4: データベースの開始

データベース作成後には、データベースのデータを格納するためのテーブルを作成します。しかし、テーブルを作成したりデータベースに変更を加えるには、データベースを開始しなければなりません。

データベースを開始すると、データベースのデーモン、つまりプロセスが開始され、データベースに変更を加えるために使用されるメモリのエリアが初期化されます。

□ 開始

データベースを開始するには **startdb** ユーティリティを使います。

startdb ユーティリティを使って、データベースデーモンのオーナとなります。デーモンのオーナとなることで、データベースの復旧やその他の操作が必要となる場合にデーモンを制御できます。

対話型 SQL/A セッションの開始（AUTOSTART コンフィギュレーション変数が TRUE に設定されている場合）、またはアプリケーションの最初の RHLI か埋め込み型 SQL/A 文によって、データベースは暗示的に開始されます。

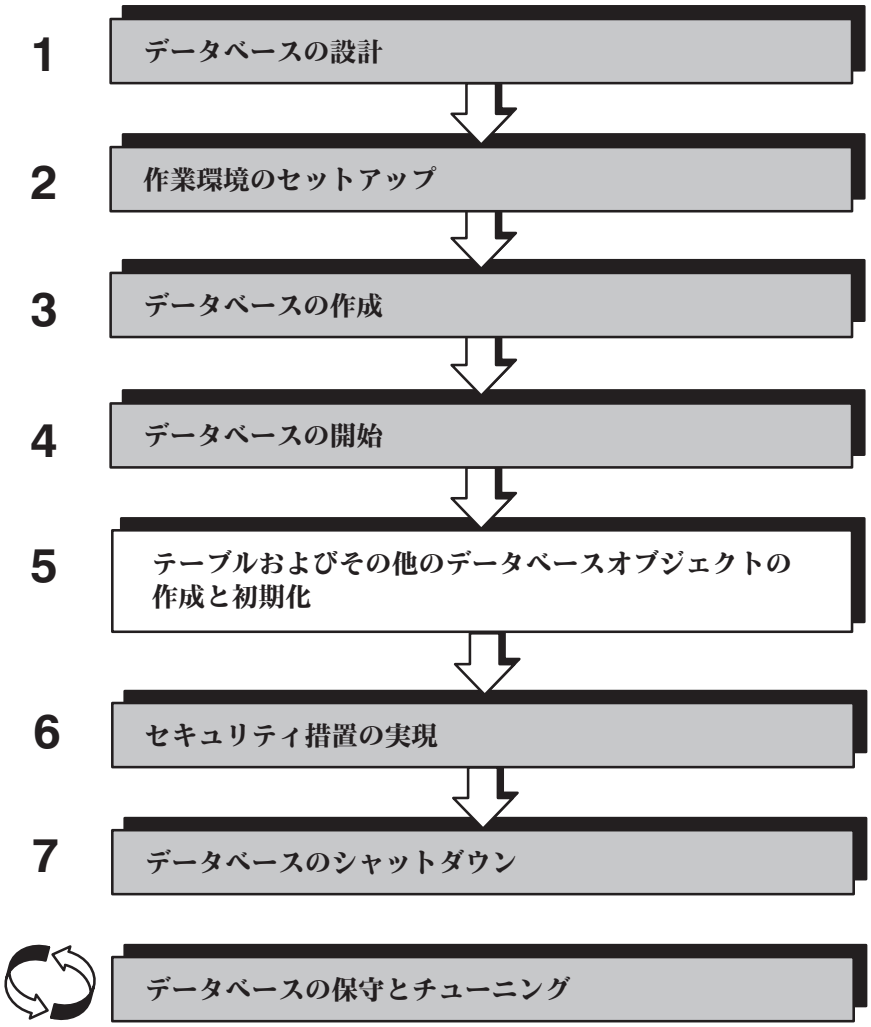
□ 確認

データベースが開始したことを確認するには、オペレーティングシステム･レベルでアクティブプロセスを一覧表示します。データベースプロセスには、**cmdmn**、 **cldmn**、 **lgdmn**、そして**fmdmn** の名前がついています。

追加ヘルプ

項目	参照先
startdb 構文と説明	『*Unify DataServer: Configuration Variable and Utility Reference*』
データベースデーモン	『 *Unify DataServer: Managing a Database*』の「アーキテクチャとトランザクション処理」の章
AUTOSTART コンフィギュレーション変数	『*Unify DataServer: Configuration Variable and Utility Reference*』
SQL コマンド	『*Unify DataServer: Configuration Variable and Utility Reference*』
アクティブプロセスの表示	使用しているオペレーティングシステムのマニュアル

データベースの開発 : ステップ 5



ステップ 5 : テーブルおよびその他のデータベースオブジェクトの作成と初期化

データベース用のテーブルを設計し、作業環境のセットアップとデータベースの作成を行った後、テーブルとその他のデータベースオブジェクトを作成することができます。

最も重要なデータベースオブジェクトはテーブルです。テーブルにはデータの行と列が含まれています。スキーマ、ビュー、そしてボリュームといったその他のオブジェクトは、テーブルとともに動作するために作成されます。

すべてのデータベースオブジェクトには、識別名と ID ナンバーが付いています。SQL/A では、オブジェクト名で照会します。RHLI では、オブジェクト名または ID で照会できます。

以下の表は、すべてのデータベースオブジェクトを記載し、各タイプのデータベースオブジェクトの作成が必要な場合を説明しています。

データベースオブジェクト

データベースオブジェクト	説明	オブジェクトを作成する場合
プライベートスキーマ	テーブルと関連する許可権の集合	1 つ以上のテーブルへのアクセスを制御する場合
テーブル	関連するデータの列と行	すべての場合
ビュー	1 つ以上のテーブルから作成される仮想テーブル	データグループへのアクセスを制限または簡素化する場合
ボリューム	ストレージユニット	データベースの格納容量がカレントのボリューム割り当てを超える場合
アクセス方式	Unify DataServer が行にアクセスする方法	デフォルトのシーケンシャル·アクセス方式が適切でない場合には、デーブル上で最も頻繁に実行されるタイプの問い合わせに基づいたアクセス方式を選択してください。

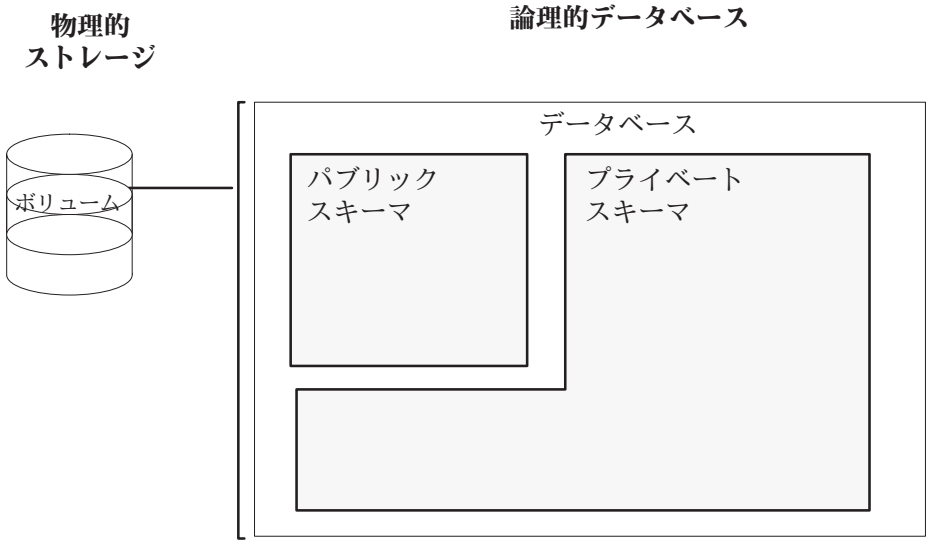
任意のプライベートスキーマ

データベースには、 1 つ以上のスキーマを含むことができます。各スキーマにはそのスキーマ独自のテーブルセットが含まれています。

スキーマは論理的データベースと考えることができます。スキーマは 1 つのアプリケーションに関連付けられたすべてのデータを含むことができます。

スキーマを作成して、同じようなアクセス許可権を持つユーザをグループ化することができます。たとえば、事務員用のスキーマ、管理者用のスキーマ、そしてプログラマ用のスキーマを作成できます。あるスキーマへ特権を割り当てることにより、そのスキーマへのアクセス権を持つすべてのユーザにその特権を割り当てることができます。

概念図 : プライベートスキーマ作成後



プライベートスキーマは、作成する必要がありません。データベース作成時に自動的に作成された PUBLIC スキーマを使用できます。

どのユーザもパブリックスキーマにアクセスできます。しかし、プライベートスキーマには、アクセス権を持つユーザしかアクセスできません。

プライベートスキーマを作成するには、以下の手順に従ってください：

☐ **作成**　SQL/A 文の **CREATE SCHEMA** を使用すれば、プライベートスキーマを簡単に作成できます。RHLI 関数の **uaddath** を使うこともできます。

☐ **確認**　プライベートスキーマが作成できたことを確認するには、SQL/A の **DISPLAY MODE** コマンドを使ってカレントのスキーマ名を表示できます。データディクショナリ･テーブルの **SYS.ACCESSIBLE_SCHEMAS** に問い合わせを行うこともできます。

☐ **変更**　スキーマに含まれるオブジェクトを変更するには、SQL/A の **ALTER SCHEMA** 文を使用します。

追加ヘルプ

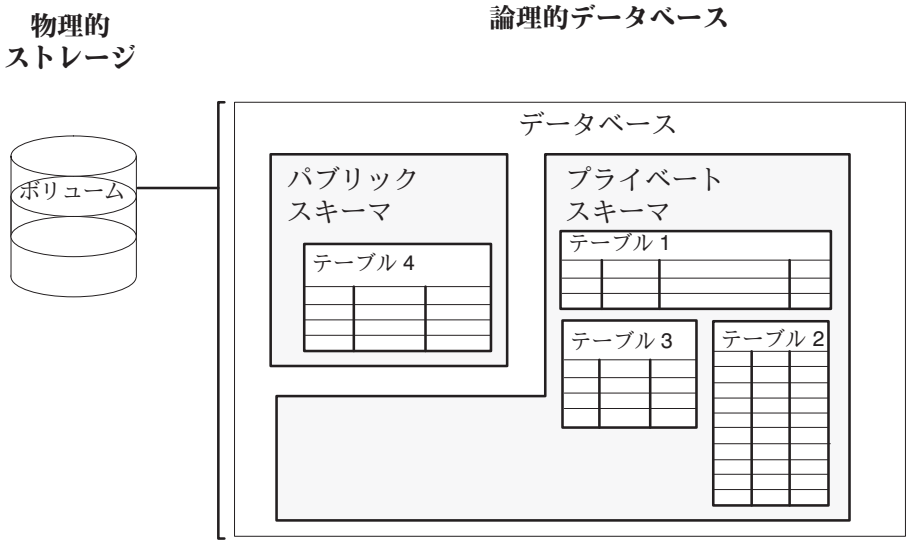
項目	参照先
CREATE SCHEMA 文、**DISPLAY MODE** コマンド、**ALTER SCHEMA** 文の構文	『*Unify DataServer: SQL/A Reference*』
uaddath 関数	『*Unify DataServer: RHLI Reference*』
SQL/A によるスキーマ作成	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「データベースオブジェクトの作成と削除」の章
パブリックスキーマとプライベートスキーマ	『*Unify DataServer: Managing a Database*』224ページ「パブリックスキーマとプライベートスキーマ」
スキーマへのアクセス許可	『*Unify DataServer: Managing a Database*』223ページ「スキーマアクセス」

テーブル

テーブルは、データの列と行の集合です。テーブルというのは、データを格納、検索、または変更するためにアクセスするものですから、データベースに最低 1 つは作成しなければなりません。

テーブルには 1 つ以上の行があります。各行には 1 つ以上の列があります。テーブルを作成すると、テーブルの各行には列が作成されます。

概念図 : 4 つのテーブルの作成後



テーブルを設計し、データベースを作成した後で、テーブルを作成します。テーブルを作成するには、以下の手順に従ってください :

□ 準備

大きなテーブル（ 100 メガバイト以上）の場合は、データベースを格納するために追加のボリュームを作成しなければならないことがあります。この章の後半で説明する「ボリューム」を参照してください。

作成する各テーブルは、既存のスキーマに属していなければなりません。テーブルを含むスキーマを決定し、そのスキーマをカレントスキーマとしてください。

テーブルを格納する特定のボリュームがある場合は、そのテーブルの作成時に、選択されたボリューム（preferred volume）としてそのボリュームを指定できます。

☐ **作成**　SQL/A 文の **CREATE TABLE** を使って、簡単にテーブルを作成することができます。RHLI 関数の **uaddtbl** も使用できます。

☐ **確認**　以下の方法のいずれかで、テーブルが正しく作成されているかどうかが確認できます :

- SQL/A **TABLES** コマンドを使って有効なテーブル名をすべて表示します。
- SQL/A **COLUMNS** コマンドを使って、テーブル内にあるすべての列の名前とデータタイプをチェックします。
- データディクショナリ･テーブルの **SYS.ACCESSIBLE_TABLES** と **SYS.ACCESSIBLE_COLUMNS** に問い合わせを行います。
- **schlst** または **schempt** ユーティリティを使って、列とテーブル･データディクショナリ情報を検索します。

☐ **変更**　テーブルと関連付けられたボリューム、または列の名前を変更するには、SQL/A 文の **ALTER TABLE** を使用します。テーブル名を変更するには、SQL/A 文の **RENAME TABLE** を使用します。

追加ヘルプ

項目	参照先
SQL/A でのテーブル作成	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「データベースオブジェクトの作成と削除」
CREATE TABLE 文、**ALTER TABLE** 文、**RENAME TABLE** 文	『*Unify DataServer: SQL/A Reference*』
uaddtbl 関数	『*Unify DataServer: RHLI Reference*』
データ辞書情報	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「データディクショナリの使用方法」
schlst または**schempt** ユーティリティ	『*Unify DataServer: Configuration Variable and Utility Reference*』
テーブルを格納するボリュームの指定	『*Unify DataServer: Writing Interactive SQL/A Queries*』154ページ「優先格納ボリューム」
スキーマのカレント指定	『*Unify DataServer: SQL/A Reference*』の **SET CURRENT SCHEMA** 『*Unify DataServer: RHLI Reference*』の **ufchath** function

アクセス方式

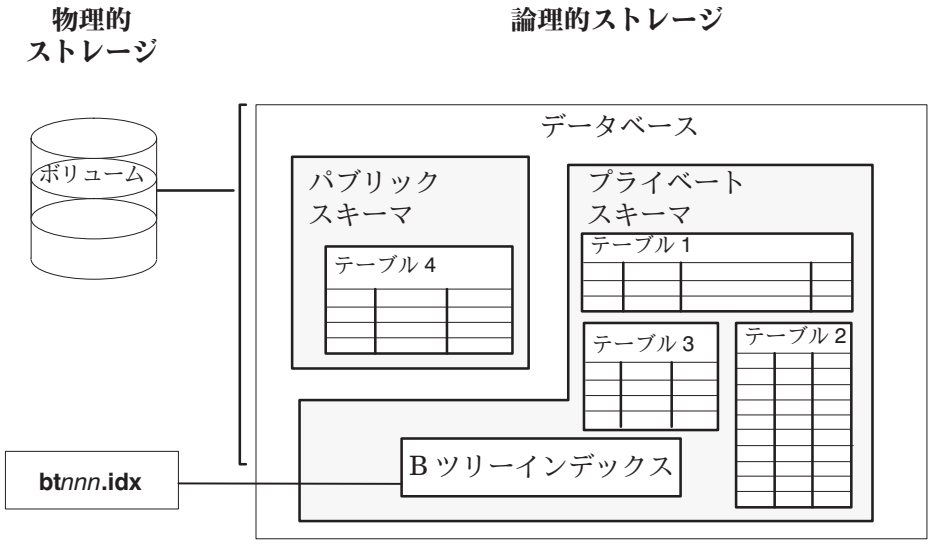
テーブルを作成後、通常はテーブルのデータへのアクセス方式を作成します。ただし、アクセス方式はいつでも作成できます。

アクセス方式というのは、Unify DataServer がテーブルの特定の行や複数の行を検索する方法です。 Unify DataServer には次の5つのアクセス方式があります。ハッシュテーブル、B ツリー、リンク、シーケンシャルおよびダイレクトです。

テーブルは、 1 つ以上のアクセス方式を持つことができます。

テーブル用にアクセス方式を作成する必要はありません。アクセス方式を作成しない場合は、シーケンシャルアクセスが使用されます。しかし、このアクセス方式が適応するのは、限られたテーブルだけです。

概念図 : アクセス方式作成後



テーブル用のアクセス方式の作成方法は、以下のとおりです :

☐ **準備** テーブルの設計と有効なアクセス方式を調査し、適切なアクセス方式を選択してください。

☐ **作成** SQL/A 文の **CREATE BTREE INDEX**、**CREATE HASH INDEX**、または **CREATE LINK INDEX** を使って、インデックスアクセス方式を割り当てることができます。

ダイレクトアクセス·テーブルを作成するには、**CREATE TABLE** 文の **DIRECT** オプションを使用します。

RHLI 関数の **uaddbt**、**uaddhsh**、または **uaddlnk** を使うか、**DBTABLE** 構造体の **DIRECT** オプションを使って、アクセス方式を割り当てることもできます。

☐ **確認** インデックスがあるかどうかを確認するには、適切なデータディクショナリ·テーブルに問い合わせを行います。

問い合わせを実行し **BEGIN EXPLAIN** と **EXPLAIN** 文を使って、使用されているアクセス方式を確認することができます。

☐ **変更** 既存の方式を削除し新しい方式を作成して、テーブルのアクセス方式を変更することができます。

SQL/A のインデックスを削除するには、**DROP BTREE INDEX**、**DROP HASH INDEX**、**DROP LINK INDEX** 文を使用します。

RHLI のインデックスを削除するには、**udrpbt**、**udrphsh**、または **udrplnk** 関数を使用します。

追加ヘルプ

項目	参照先
アクセス方式の選択	『*Unify DataServer: Managing a Database*』
CREATE/DROP BTREE INDEX、**CREATE/DROP HASH INDEX**、または **CREATE/DROP LINK INDEX** 文	『*Unify DataServer: SQL/A Reference*』
ハッシュアルゴリズムの選択とチューニング	『*Unify DataServer: Managing a Database*』の「アクセス方式の選択」の章
ダイレクトアクセス列	『*Unify DataServer: Writing Interactive SQL/A Queries*』150ページ「ダイレクトアクセス列」
データディクショナリ情報	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「データディクショナリの使用方法」の章
すべての RHLI 関数と構造体	『*Unify DataServer: RHLI Reference*』

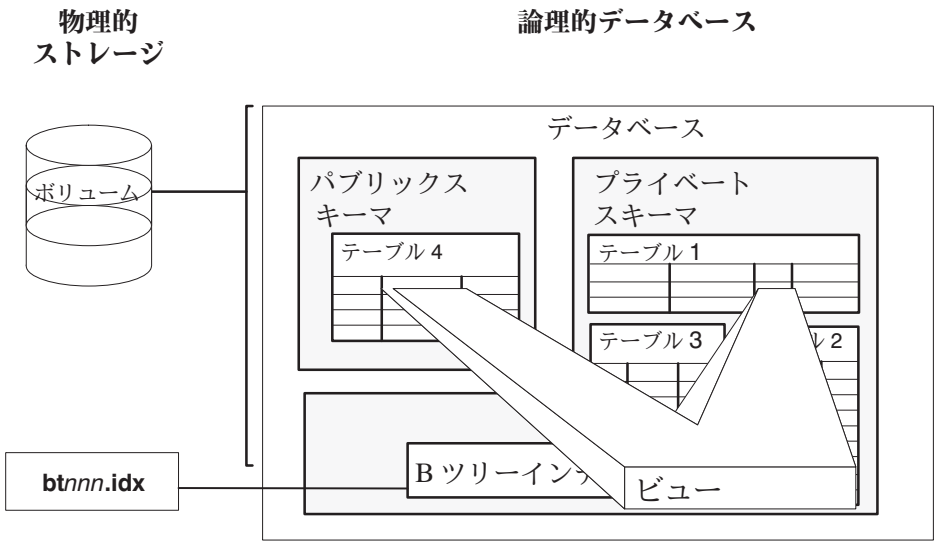
ビュー

テーブルと適切なアクセス方式を作成した後、ビューを作成することができます。

ビューは仮想テーブルで、通常のテーブルと同じ方法でアクセスできます。ビューは 1 つ以上の既存のテーブルから作成されているため、テーブルを変更するとビューも変更されます。ビューを作るために使用されるテーブルは、ベーステーブルと呼ばれます。

ビューを使えば、選択されたデータへ簡単にアクセスできるようになります。たとえば、ビューは 2 つ以上のテーブルを接合（Join）して作成できます。またビューを使えば、特定の列や行へのアクセスを制限することもできます。つまり、選択されたデータの特定の列や行だけが、そのビューの一部となるからです。ビューのアクセス方式は、制御することができます。

概念図 : ビューの作成後



ビューを作成方法するには、以下のガイドラインに従ってください :

☐ **準備** ビューの作成前に、ビューを形成するのに使用されるベーステーブルがなければなりません。また、ビューで使用されているテーブルの部分に対する、アクセス許可権を持っていなくてはなりません。

☐ **作成** ビューを作成するには、SQL/A 文の **CREATE VIEW** を使用します。RHLI を使ってビューを作成したり、ビューにアクセスしたりすることはできません。

☐ **確認** ビューがあるかどうかを確認するには、テーブルと同様に問い合わせを行います。データディクショナリ･テーブルに問い合わせを行うか、**schlst** ユーティリティを使うこともできます。

追加ヘルプ

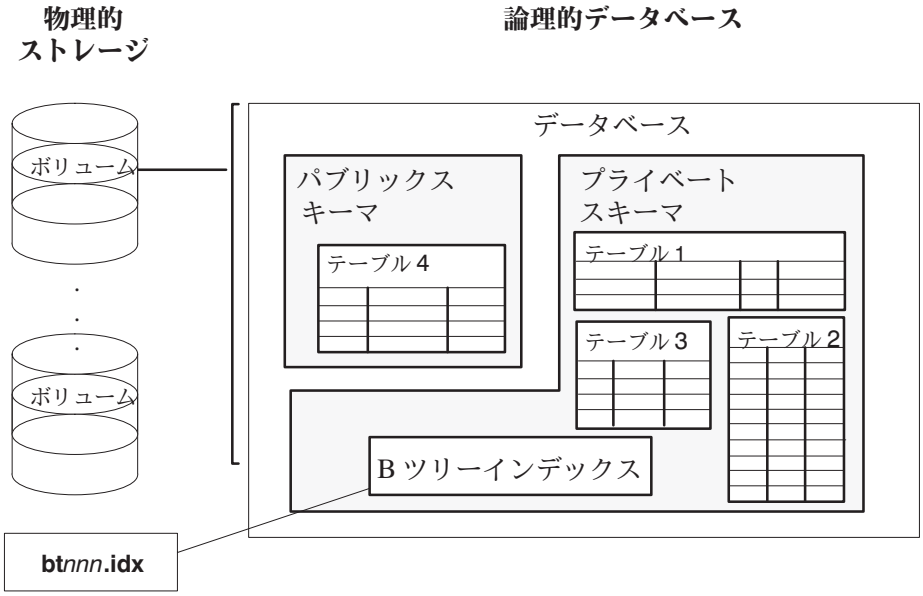
項目	参照先
CREATE VIEW 文	『*Unify DataServer: SQL/A Reference*』
SQL/A でのビュー作成	『 *Unify DataServer: Writing Interactive SQL/A Queries*』の「データベースオブジェクトの作成と削除」
データディクショナリ･テーブル	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「データディクショナリの使用方法」
schlst ユーティリティ	『*Unify DataServer: Configuration Variable and Utility Reference*』

追加ボリューム

ボリュームは、データベース情報を格納するディスクスペースの割り当てです。データベースには、1 つ以上のボリュームがあります。1 つのボリュームに複数のテーブルを含むことができますし、1 つのテーブルを複数のボリュームに含めることもできます。

データベースの 1 番目のボリュームは、そのデータベースが作成されたときに作成されます。大きなデータベースの場合、そのデータベースを格納するために追加ボリュームを作成しなければなりません。

概念図 : 追加ボリュームの作成後



追加ボリュームを作成するには、以下の一般的手順に従ってください :

☐ **準備** ボリュームのタイプがデバイスの場合は、**mkvol** ユーティリティを使ってそのデバイスを初期化してください。

☐ **作成** 追加ボリュームは、SQL/A 文の **CREATE VOLUME** を使って、簡単に作成できます。RHLI 関数の **uaddvol** を使用することもできます。

☐ **確認** ボリュームがあるかどうかを確認するには、データディクショナリ･テーブルに問い合わせを行います。

追加ヘルプ

項目	参照先
SQL/A でのボリューム作成	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「データベースオブジェクトの作成と削除」の章
データディクショナリ	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「データディクショナリの使用方法」の章
CREATE VOLUME 構文	『*Unify DataServer: SQL/A Reference*』
uaddvol 関数	『*Unify DataServer: RHLI Reference*』
ボリュームの一般的説明	『*Unify DataServer: Managing a Database*』194ページ「データベースボリューム」
mkvolユーティリティを使ったデバイスの作成と初期化	『*Unify DataServer: Managing a Database*』199ページ「デバイスの作成と初期化」

テーブルへのデータのロード

テーブル作成後、データをテーブルにロードできます。

テーブルにデータをロードすると、テーブルのデータ値は初期化されます。データベースのテーブルを初期化するには、以下の手順に従ってください：

☐ **ロード** **dbld** か **pdbld** ユーティリティを使って、能率的に大量のデータをデータベーステーブルにロードできます。

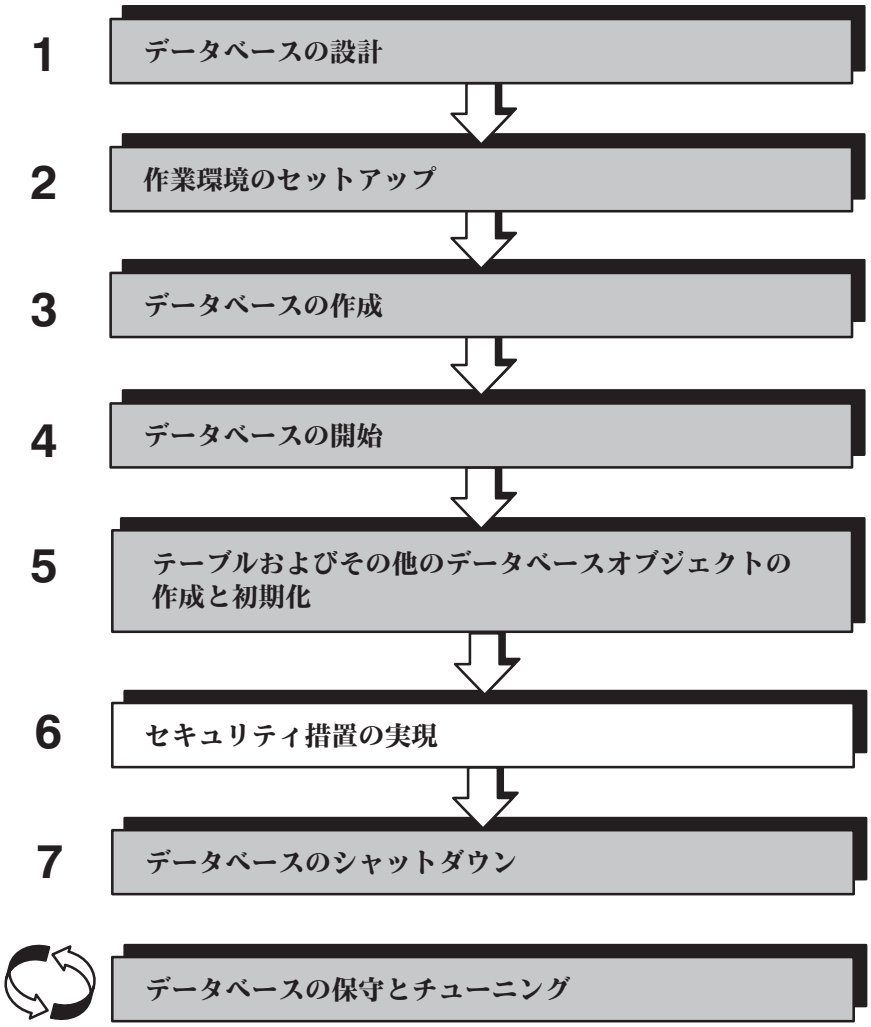
☐ **確認** データ整合性サブシステム（DIS）を使って、ロードされているデータ値の整合性をチェックできます。

また、ロードされているデータがリンクインデックスの列、またはダイレクトアクセス列に適用される場合には、Unify DataServer はその入力値を確認します。

追加ヘルプ

項目	参照先
dbld または **pdbld** の構文と説明	『*Unify DataServer: Configuration Variable and Utility Reference*』
データ整合性サブシステム	『*Unify DataServer: Configuration Variable and Utility Reference*』の**disc**

データベースの開発 : ステップ 6



ステップ 6 : セキュリティ措置の実現

データベーステーブルを作成し初期化をした後、セキュリティ措置の実現が必要となります。

データベースセキュリティをセットアップし、データベースにユーザを追加するには、ユーザがアクセスできるデータベースの範囲、およびユーザが行える操作の種類を決定しなければなりません。

Unify DataServer データベースは、セキュリティにとって重要な 2 層の論理的データベースオブジェクトで構成されています。それはデータベースとスキーマです。

セキュリティは、以下の 3 つの分野で説明することができます :

- データベースとスキーマへのアクセス : ユーザがどのデータベースまたはスキーマにアクセスできるか
- 権限 : ユーザがどの文を使用できるか
- スキーマ特権 : スキーマがどのデータベースオブジェクトにアクセスできるか

データベースとスキーマへのアクセス

どのユーザが、データベースまたはデータベース内のプライベートスキーマにアクセスできるのかを指定します。どのユーザもパブリックスキーマ内の情報にはアクセスできます。

データベースまたはスキーマへのユーザアクセスを制御するには、以下の手順に従ってください:

☐ **許可**

GRANT ACCESS ON DATABASE 文、または RHLI 関数の **uaddusr** を指定して、データベースへのアクセスを許可します。

SQL/A 文の **GRANT ACCESS ON SCHEMA**、または RHLI 関数の **uaddprm**を使って、プライベートスキーマへのアクセスを許可します。

☐ **確認**

SYS.DATABASE_USERS および **SYS.ACCESSIBLE_SCHEMAS** データディクショナリ･テーブルに問い合わせを行えば、ユーザのアクセス権を確認できます。

☐ **変更**

SQL/A 文の **REVOKE ACCESS ON DATABASE**、または RHLI 関数の **udrpusr** を使って、データベースへのユーザアクセス権を削除できます。

SQL/A 文の **REVOKE ACCESS ON SCHEMA**、あるいは RHLI 関数の **udrpprm** を使って、スキーマへのユーザアクセス権を削除できます。

追加ヘルプ

項目	参照先
GRANT ACCESS ON DATABASE または **REVOKE ACCESS ON DATABASE** 文	『*Unify DataServer: SQL/A Reference*』
データディクショナリ･テーブル	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「データディクショナリの使用方法」の章
uaddusr または **uaddprm** 関数	『*Unify DataServer: RHLI Reference*』

権限

権限は、ユーザに対して許可される一連の特権です。ユーザによるデータベースオブジェクトへのアクセス、作成、修正、削除は、ユーザ特権により制御します。ユーザまたはユーザグループの権限レベルを制御するには、以下の手順に従ってください：

権限には 2 つのレベルがあります。DBA とスキーマです。

☐ **許可**

SQL/A の **GRANT DBA** 文 または RHLI 関数の **uaddprv** を指定して、DBA 権限を許可します。SQL/A の **GRANT SCHEMA** 文、あるいは RHLI 関数の **uaddprv** を指定して、スキーマ権限を許可します。

☐ **確認**

どのユーザが DBA 権限を許可されているかは、データディクショナリ･テーブル **SYS.USER_PRIVILEGES** に問い合わせを行えば確認できます。

☐ **変更**

REVOKE DBA FROM または **REVOKE SCHEMA FROM** を指定して、ユーザの権限を削除できます。

追加ヘルプ

項目	参照先
GRANT DBA または **GRANT SCHEMA** 文	『*Unify DataServer: SQL/A Reference*』
SQL/A での権限の制御	『 *Unify DataServer: Writing Interactive SQL/A Queries*』の「データベースオブジェクト･アクセスの制御」
uaddprv または **udrpprv** 関数	『*Unify DataServer: RHLI Reference*』

スキーマ特権

スキーマ特権は、スキーマに対して許可される特権です。スキーマのどのユーザも、そのスキーマに許可された特権を使用できます。これらの特権は、以下のようなものです：

- 他のスキーマ、およびそのスキーマに属するすべてのオブジェクトへのアクセス
- 他のスキーマにあるテーブルへのアクセス
- 他のスキーマのテーブルにある列へのアクセス

特定のオブジェクトへの特権を許可するときには、そのオブジェクトに対してどの操作を許可するかを指定します。スキーマのユーザは、他のスキーマではスキーマ、テーブル、または列に対し、指定された操作のみを行うことができます。

スキーマ特権を制御するには、以下の手順に従ってください：

☐ **許可**

SQL/A の **GRANT SCHEMA PRIVILEGE** または **GRANT TABLE PRIVILEGE** 文を使って、スキーマ特権を許可します。

☐ **確認**

データディクショナリ･テーブルの **SYS.SCHEMA_PRIVILEGES**、**SYS.TABLE_PRIVILEGES** および **SYS.COLUMN_PRIVILEGES** に問い合わせを行えば、どのスキーマ特権があるのかを確認できます。

☐ **変更**

SQL/A 文の **REVOKE SCHEMA PRIVILEGE**、**REVOKE TABLE PRIVILEGE**、または RHLI 関数の **udrpprv** を使って、スキーマ特権を削除できます。

追加ヘルプ

項目	参照先
GRANT SCHEMA PRIVILEGE または **GRANT TABLE PRIVILEGE**、**REVOKE SCHEMA PRIVILEGE** または **REVOKE TABLE PRIVILEGE**	『*Unify DataServer: SQL/A Reference*』
uaddprv または **udrpprv** 関数	『*Unify DataServer: RHLI Reference*』

他のレベルのセキュリティ

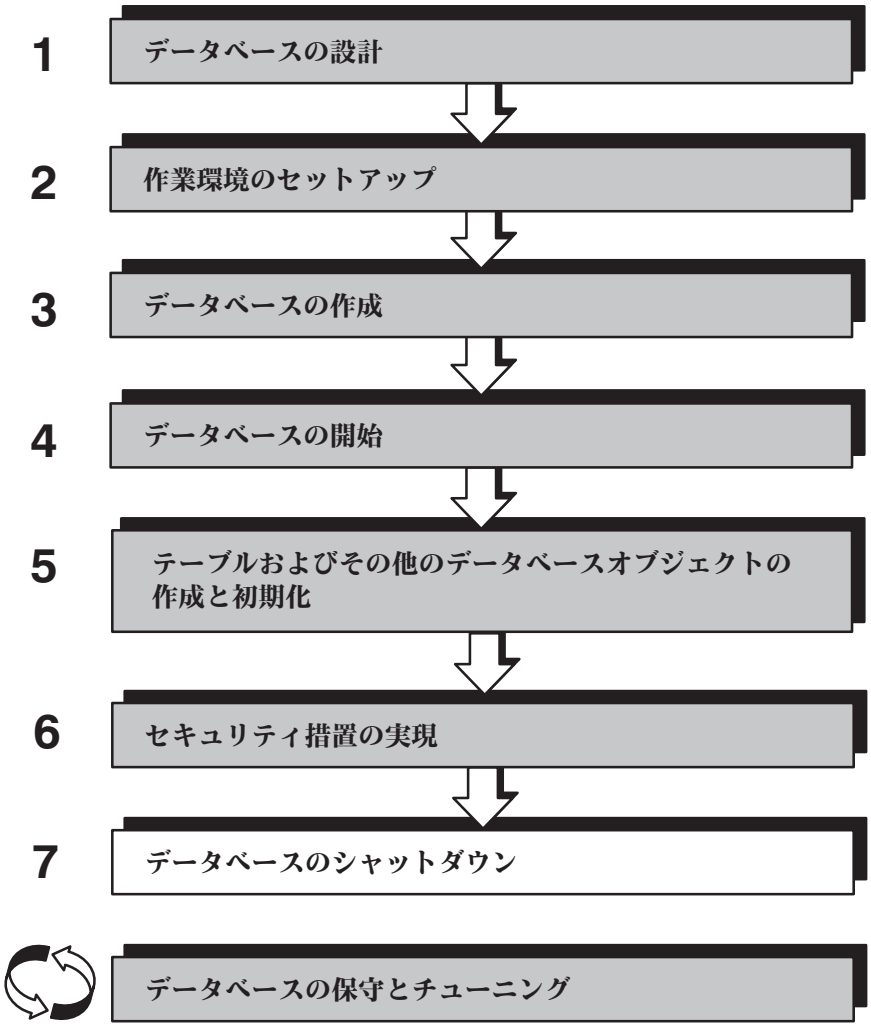
セキュリティレベルを実現できる他の分野には、以下のものがあります：

- データベースを含むファイルに許可権を設定して、データベースへアクセスできるユーザを制御できます。
- ビューを使って仮想テーブルを作成し、そのビューへのアクセスを制御できます。
- プライベートスキーマにテーブルその他のオブジェクトを置いて、そのスキーマへのアクセスを制御できます。

追加ヘルプ

項目	参照先
データベースを含む Unify DataServer ファイル	『*Unify DataServer: Managing a Database*』の「データベースファイルの管理」の章
オペレーティングシステム･ファイル許可権	使用しているオペレーティングシステムのマニュアル
ビュー プライベートスキーマ	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「データベースオブジェクトの作成と削除」の章

データベースの開発 : ステップ 7



ステップ 7: データベースのシャットダウン

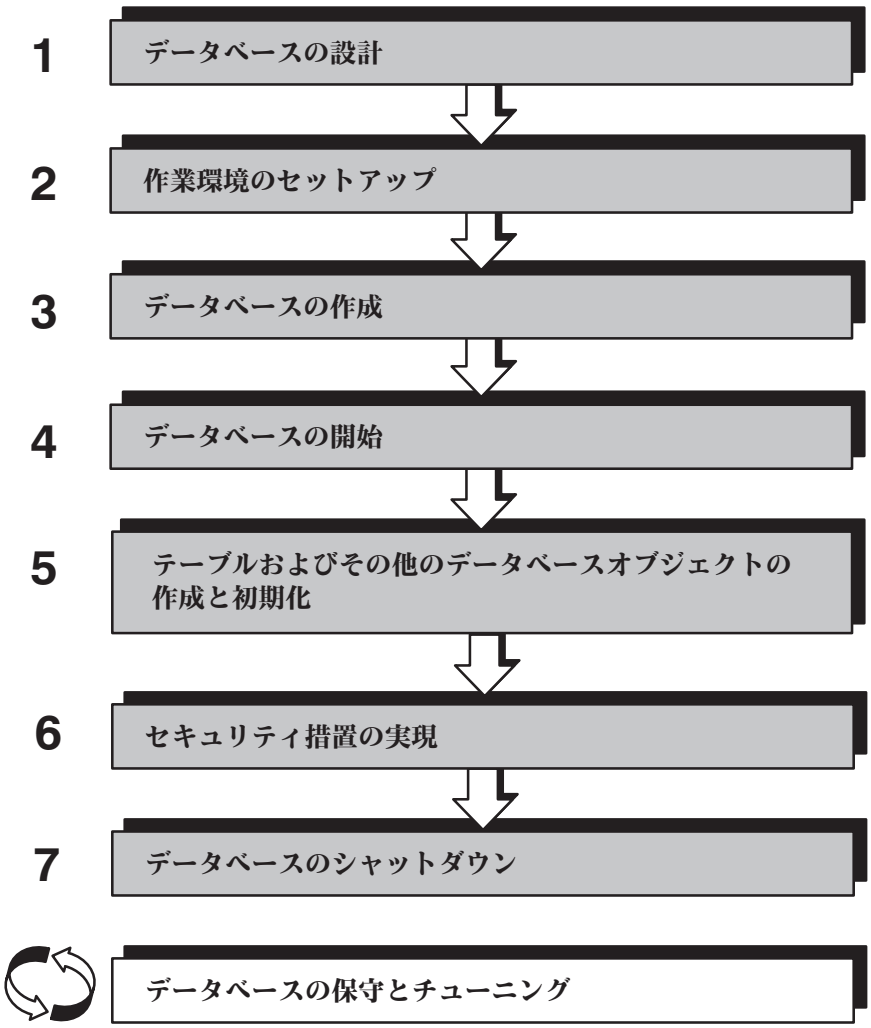
システムの再起動、あるいは何らかの復旧操作を行うために、データベースをオフラインにする必要がある場合は、**shutdb** ユーティリティを使ってデータベースをシャットダウンします。

shutdb ユーティリティは、データベースデーモンを終了させ、データベースの同期をはかり、共有メモリを削除します。これらのタスクを実行するために、シャットダウンが行われる時点であらゆるユーザプロセスは、データベースにアクセスできなくなります。

追加ヘルプ

項目	参照先
データベースのシャットダウン	『*Unify DataServer: Managing a Database*』の「データベースのシャットダウン」
shutdb ユーティリティ	『*Unify DataServer: Configuration Variable and Utility Reference*』
データベースデーモン	『*Unify DataServer: Managing a Database*』の「アーキテクチャとトランザクション処理」の章

データベースの開発 : 保守とチューニングのサイクル



データベースの保守とチューニング

データベースの操作を行っている間、そのデータベースのパフォーマンスと整合性を監視する必要があります。

整合性の保護

システム、メディア、あるいはプログラムエラーが起こった場合には、DBAとしてデータベースの整合性を保護するよう注意しなければなりません。

システム障害またはメディア障害の復旧を確実に行うには、以下の手順に従ってください：

- トランザクションログを使用
- 物理ログを使用
- ジャーナルを使用
- 物理ログ、トランザクションログ、ジャーナル、およびデータベースのバックアップメディアを、データベースが格納されているデバイスと異なるデバイスに配置してください。同時に、複数の異なるデバイスが壊れることはあまりありません。
- 頻繁にデータベースのバックアップを行ってください。

追加ヘルプ

項目	参照先
整合性の保護	『*Unify DataServer: Managing a Database*』の「データ消失の防止」の章
トランザクションログ	『*Unify DataServer: Managing a Database*』の「トランザクションとロックの管理」の章

データベースのチューニング

データベースのチューニングには、Unify DataServer 製品を使ってデータベースの集計表を表示したり、その集計表に基づいたデータベースの機能を調整したりすることが含まれます。

シングルユーザシステムのパフォーマンスを分析することから、データベースのチューニングを開始します。シングルユーザシステムについて、以下の項目を分析します :

- 処理能力と期待値の比較
- リアル、システム、ユーザ、およびアイドルタイム
- システムデーモン·パフォーマンス
- CPU タイム

シングルユーザパフォーマンスのベースラインがわかれば、マルチユーザパフォーマンスに取りかかれます。理論的な最大処理能力は、シングルタスクユーザ、およびシステムタイムに基づいて決まります。

マルチユーザシステムについて、以下の項目を分析します :

- システムデーモン·パフォーマンス
- CPU バウンドとディスクバウンドの比較
- アイドルタイム
- ロック
- CPU 使用状態
- ページング

データベースの集計表を表示するために、以下のことができます :

- データディクショナリ·テーブルを確認することによって、データベースの情報を表示できます。
- 以下に記載するどの Unify DataServer ユーティリティも利用できます。

RHLIでは、検索する情報のタイプに応じて、各種の情報検索関数を使って、情報を得ることができます。

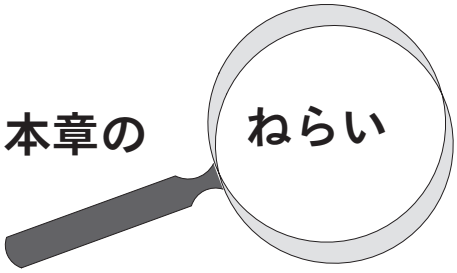
追加ヘルプ

収集に関する項目	参照先
B ツリー集計表	『*Unify DataServer: Configuration Variable and Utility Reference*』の**btstats**
コンフィギュレーション変数のカレント設定	『*Unify DataServer: Configuration Variable and Utility Reference*』の**config**
データベース用データ定義	『*Unify DataServer: Configuration Variable and Utility Reference*』の**dumpdd**
ハッシュツリーインデックス集計表	『*Unify DataServer: Configuration Variable and Utility Reference*』の**htstats**
ロック情報	『*Unify DataServer: Configuration Variable and Utility Reference*』の**lmshow**
リンクインデックス集計表	『*Unify DataServer: Configuration Variable and Utility Reference*』の**lnkstats**
トランザクションログ情報	『*Unify DataServer: Configuration Variable and Utility Reference*』の**prtlghd**
データディクショナリ情報	『*Unify DataServer: Configuration Variable and Utility Reference*』の**schlst** または **schempt**
共有メモリ集計表	『*Unify DataServer: Configuration Variable and Utility Reference*』の**shmmap**
テーブル集計表	『*Unify DataServer: Configuration Variable and Utility Reference*』の**tblstats**
物理ログ、トランザクションログ、共有メモリ、およびキャッシュに関連したパフォーマンス情報	『*Unify DataServer: Configuration Variable and Utility Reference*』の**uperf**
ボリューム集計表	『*Unify DataServer: Configuration Variable and Utility Reference*』の**volstats**

追加ヘルプ

チューニングに関する項目	参照先
チューニングの一般的情報	『*Unify DataServer: Managing a Database*』の Part III
共有メモリのクリーンアップ	『*Unify DataServer: Configuration Variable and Utility Reference*』の **shmclean**
データベースの同期	『*Unify DataServer: Configuration Variable and Utility Reference*』の **syncdb**
RHLI でのデータベース情報検索	『*Unify DataServer: RHLI Reference*』の **uinfdb** 関数

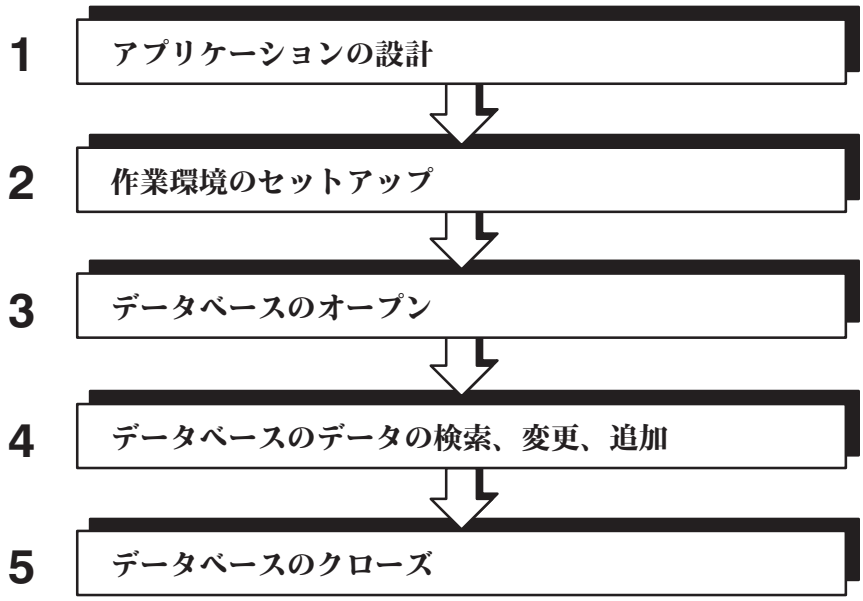
アプリケーション開発者のタスク



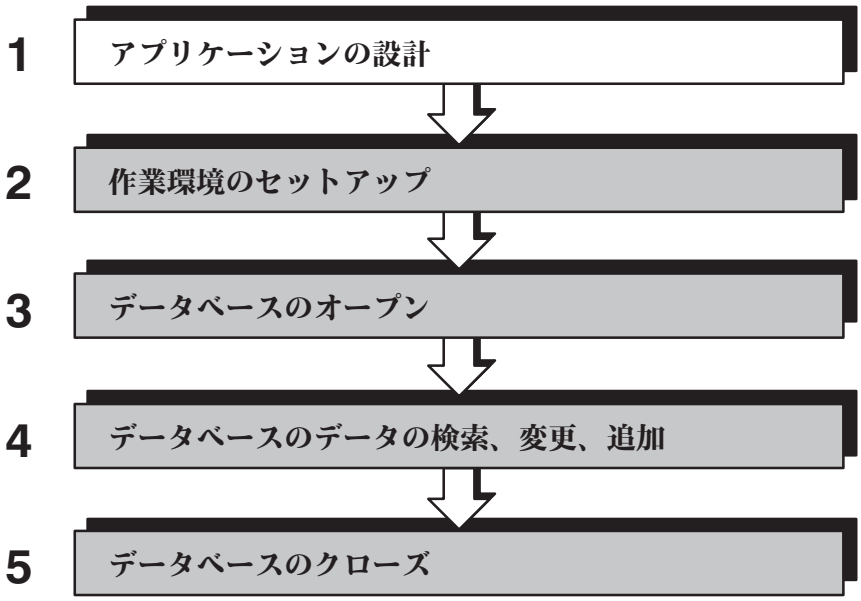
このセクションでは、アプリケーション開発者がデータベースを開発するときのタスクを説明します。以下の図表に手順を示します。

第 1 ステップでは、アプリケーションを設計します。その後のステップでは、実際にアプリケーションが実行するタスクを説明します。

データベースの開発：
アプリケーション開発者のステップ



アプリケーションの開発 : ステップ 1



ステップ 1: アプリケーションの設計

アプリケーション設計の第 1 ステップは、どのインターフェースを採用するかを決定することです。
以下のインターフェースがアプリケーションに使用できます :

- ACCELL/SQL
- Unify Vision
- 対話型 SQL/A
- 埋め込み型 SQL/A
- RHLI

ACCELL/SQL はオープンシステムのアプリケーション開発ツールで、他のデータベース管理システムと互換性があります。ACCELL/SQL は扱いやすく、「空欄を埋める」形式で作成するフォームを提供しています。コマンドはファンクションキーを押すか、マウスのクリックによって実行されます。

Unify Vision はグラフィカルなクライアント/サーバ型アプリケーション開発ツールで、 複雑なアプリケーションの配布や管理を高度に自動化して行う機能を持ち、扱いやすく、クロスプラットフォーム環境で使用できます。

対話型 SQL/A は、Unify DataServer データベースのデータへ直接アクセスしたり操作したりするには最も簡単な方法です。SQL/A 文を対話型 SQL/A セッションで指定することにより、データベースへのアクセス、変更ができます。

埋め込み型 SQL/Aでは、C プログラムで SQL/A 文とコマンドを使用することができます。

RHLI はデータベースのアクセスには最高のパフォーマンスを発揮します。RHLI は、アプリケーションがデータベースへのアクセスや操作をするために呼び出す関数のライブラリです。

RHLI または埋め込み型 SQL/A インターフェースでアプリケーションを書くには、C のプログラミング技術が必要です。

作成するアプリケーションは使用する言語のプログラミング標準に従わなければなりません。例えば、RHLI または埋め込み型 SQL/A プログラムは C プログラムのガイドラインに従わなければなりません。

どのインターフェースの場合でも、アプリケーションの作成には次の 2 つの重要なことに注意してください。それはトランザクション管理およびエラー処理です。

追加ヘルプ

項目	参照先
対話型 SQL/A	『*Unify DataServer: Writing Interactive SQL/A Queries*』
埋め込み型 SQL/A	『*Unify DataServer: Embedding SQL/A Into C*』
ACCELL/SQL	『*ACCELL/SQL: Developing an Application*』
RHLI	『*Unify DataServer: RHLI Reference*』

トランザクション管理

マルチユーザデータベースは、ダイナミックなデータベースで、複数のユーザが同時にデータの追加、削除、変更を行うことができます。

データへの同時アクセスの許可や、データの整合性を保つために、データベース管理システムは効果的なロックシステムを持たなければなりません。このシステムは、一方のユーザが他のユーザよってアクセスまたは操作されているデータに干渉することを防ぎます。

例えば、2 人のユーザ A と B が同時に同じ行を読み出している場合、ユーザ A が知らないうちに、ユーザ B が値を変更したり行を削除してデータベースを更新することがあります。ユーザ A は更新された行を再読み込みしないかぎり、変更を知ることはできません。

この複雑な問題の効果的な解決策は、トランザクションとロックを使用することです。このセクションでは、トランザクションとロックが、異なるレベルの同時データアクセスをどう管理するかを説明します。トランザクション対応の操作の実行は、データベースアプリケーションの設計、特に同時アクセスをサポートするシステムで大きな役割を果たします。

実例

簡単な銀行業務の演算に関するトランザクションを使用した実例：
$100 を普通預金から当座預金に振り替えると仮定します。

このトランザクションを完了するために銀行が従うべき手続きは以下のようになります。普通預金には $100 の借方が記入され、当座預金には $100 の貸方が記入されます。預金の貸方借方双方の操作は 1 つのトランザクションと考えられます。当然、銀行の帳簿の残高を合わせるには、2 つの操作がともに成功していなければなりません。

このトランザクションを処理するコンピュータが、トランザクションをアボートするような問題に遭遇したらどうなるでしょうか？トランザクションはさまざまな理由によってアボートされることがあります。普通預金が $100 未満であったり、当座預金の口座番号が間違っていたり、停電が起きたりなどです。

トランザクションがアボートされた場合、貸方借方の勘定残高のようなトランザクションを行っている間になされたすべてのデータベースの変更は取り消され、整合状態に復帰します。

はじめの処理は問題なく実行され、後の処理がアボートされた場合、両方の処理が 1 つのトランザクションの一部でなければ、普通預金から $100 の借方を失うことになります。

アプリケーションによるトランザクションの管理方法

トランザクションを管理するために、アプリケーションの中でロックの獲得や開放を制御する異なるトランザクションレベルを指定することができます。トランザクション管理のチューニングに関しては、いくつかのコンフィギュレーション変数があります。

すべてのデータベース操作は、同時実行やデータ復旧を可能にし、データベースの整合性を維持するために、トランザクションの文脈の中で実行されます。

COMMIT WORK 文、**OPEN DATABASE** 文、**CONNECT** 文のいずれから始まる SQL/A 文を使用しても、トランザクションを開始することができます。

RHLI を使用してトランザクションを開始するには、RHLI 関数の **ubegtx** を使用してください。 RHLI では 各トランザクションを明示的に開始および終了しなければなりません。

SQL/A 文の **COMMIT WORK** または、RHLI 関数の **ucmttx** または **ucbgtx** を使用してトランザクションをコミットすることができます。セッションまたはアプリケーションを通常に終了する場合、トランザクションは SQL/A で暗示的にコミットされます。SQL/A および RHLI で暗示的にコミットされるトランザクションに文をひとまとめにして処理することができます。

SQL/A 文の **ROLLBACK WORK** または RHLI 関数の **uabttx** を使用してトランザクションをロールバックすることができます。セッションまたはアプリケーションが異常終了した場合は、トランザクションは SQL/A で暗示的にロールバックされます。

エラー処理

アプリケーションのエラー処理をするときは、各データベースの文や関数が正常に機能しているかを確認してください。SQL/A スクリプトでは、**WHENEVER** 文を使用してください。埋め込み型 SQL/A では、すべての文の後にステータス変数 **SQLCODE** をチェックしてください。RHLI では、エラーが起きるかどうか判断するために、各関数の戻り値をチェックしてください。

アプリケーションがエラーを検出したら、カレントトランザクションをロールバックするか、アボートしてください。部分的に成功した操作を無効にするという方法により、データベースは常に整合性のある状態に保たれます。

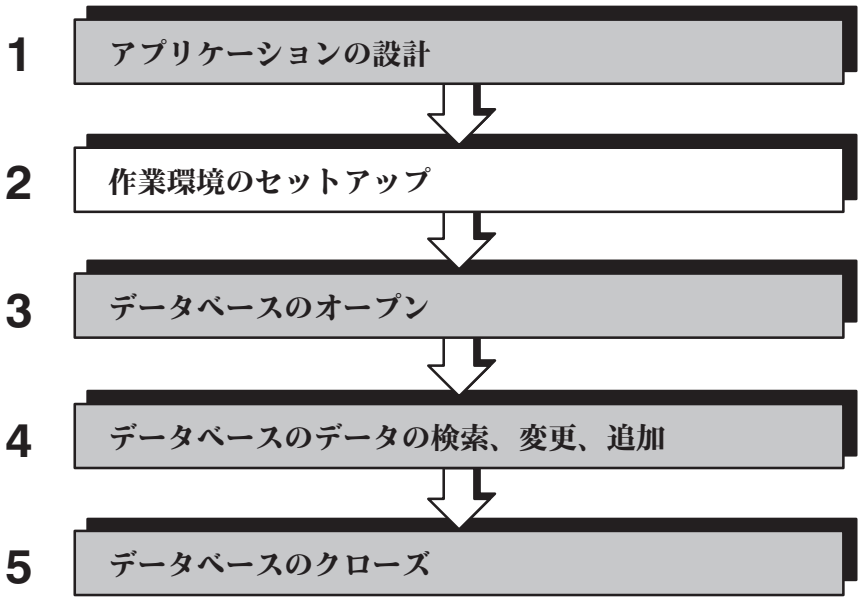
データベースのエラー･ログファイル *dbname*.**err** を参照し、ステータスイベントまたはシステムエラー･メッセージをチェックすることができます。

埋め込み型 SQL/A アプリケーションのデバッグに、対話型デバッガを使用することができます。

追加ヘルプ

項目	参照先
Unify DataServer のトランザクション管理	『*Unify DataServer: Managing a Database*』の「トランザクションとロックの管理」の章
SQL/A でのトランザクション管理	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「トランザクションとロックの管理」の章
対話型 SQL/A スクリプトでのエラー処理	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「SQL/A 文の入力」
埋め込み型 SQL/A でのエラー処理	『*Unify DataServer: Embedding SQL/A Into C*』の「エラーと条件処理」

アプリケーションの開発 : ステップ 2



ステップ 2 : 作業環境のセットアップ

作成するアプリケーションの特定の要求のために、どのコンフィギュレーション変数を調整する必要があるかを決定してください。詳しくは『 *Unify DataServer: Configuration Variable and Utility Reference*』にあるコンフィギュレーション変数の説明を参照してください。

例えば、*SQLPMEM*、*SQLSELCNT*、*SQLSMEM*、*TIMEM* などは作成するアプリケーションの必要によって設定が異なるコンフィギュレーション変数です。

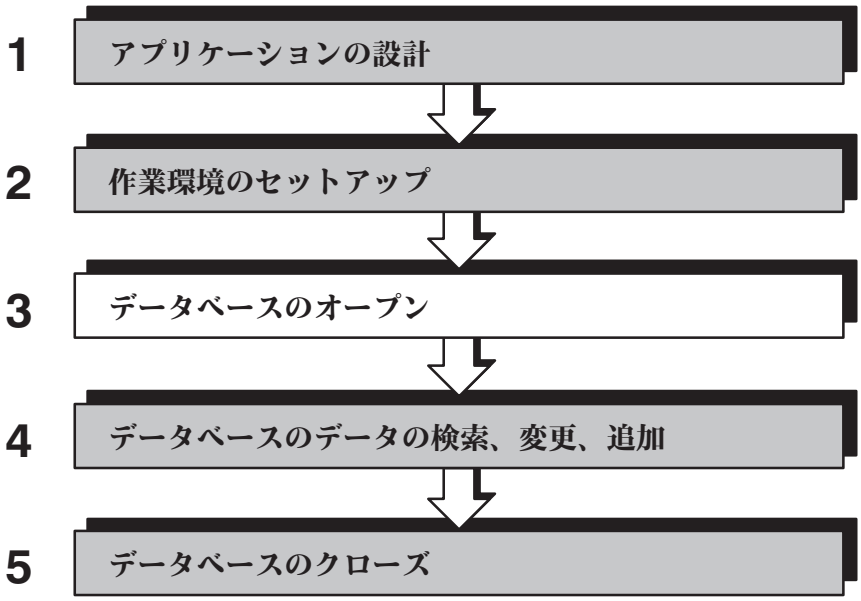
コンフィギュレーション変数の設定を確認するには、**config** ユーティリティを使用してください。

RHLI 関数にアクセスするアプリケーションの場合、必ずアプリケーションがインクルードファイル **rhlierr.h** および **rhli.h** にアクセスできるようにしてください。

追加ヘルプ

項目	参照先
config ユーティリティ	『*Unify DataServer: Configuration Variable and Utility Reference*』
コンフィギュレーション変数の設定方法	『 *Unify DataServer: Managing a Database*』の「データベースアプリケーションの構成」の章
RHLI インクルードファイル	『*Unify DataServer: RHLI Reference*』
すべてのコンフィギュレーション変数	『*Unify DataServer: Configuration Variable and Utility Reference*』

アプリケーションの開発 : ステップ 3



ステップ 3 : データベースのオープン

作成したアプリケーションでデータベースの問い合せまたは変更を行う前に、データベースをオープンしなければなりません。

既存のデータベースをオープンするには以下の一般ガイドラインに従ってください。

☐ **準備**

作成したアプリケーションの文や関数に完全なデータベース名が指定されていない場合、以下のコンフィギュレーション変数がオープンすべきデフォルトのデータベースを決定します。それは、*DBNAME*、*DBPATH*、*DBHOST*、*DBUSER* です。正しいデータベースがオープンしていることを確かめるには、コンフィギュレーション変数の値をチェックしてください。

データベースは、オープンされる前に開始されていなければなりません。データベースは、コンフィギュレーション変数 *AUTOSTART* または *OWNERSTART* に拒否されなければ、データベースをオープンするアプリケーションによって暗示的に開始されます。

☐ **オープン**

対話型 SQL/A または埋め込み型 SQL/A を使用している場合、明示的にデータベースをオープンする必要はありません。対話型 SQL/A セッションが開始するとき、または埋め込み型 SQL/A 文が実行されるときに、デフォルトのデータベースがオープンされます。対話型 SQL/A または埋め込み型 SQL/A のデータベースを明示的にオープンするには、**CONNECT** 文または **OPEN** 文を使用してください。

RHLI を使用している場合は、明示的にデータベースをオープンしなければなりません。データベースをオープンするには、**uopndb** 関数を呼び出してください。

☐ **確認**

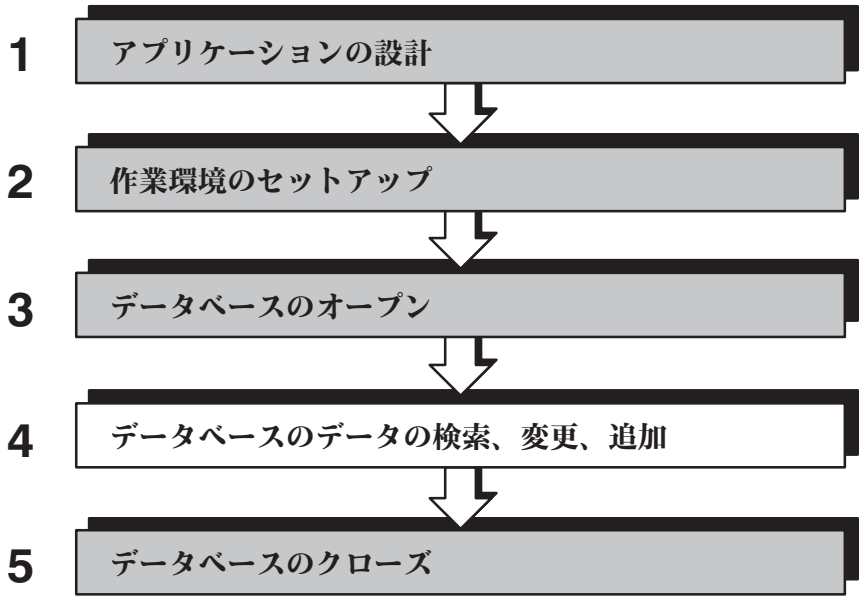
データベースがオープンされたかを確認するには、アクセス権を持つデータベースのテーブルに問い合わせを行ってください。

追加ヘルプ

項目	参照先
CONNECT 文または **OPEN** 文	『*Unify DataServer: SQL/A Reference*』
uopndb 関数	『*Unify DataServer: RHLI Reference*』

項目	参照先
コンフィギュレーション変数	『*Unify DataServer: Configuration Variable and Utility Reference*』
データベースの開始	このマニュアルの始めの方にある「 DBA のタスク」の**ステップ** 4

アプリケーションの開発 : ステップ 4



ステップ 4: データベースのデータの検索、変更、追加

ステップ 1 でアプリケーションを設計したときに、どのタスクをアプリケーションが実行するかを決定しています。タスクは以下のうちの 1 つ以上です:

- テーブルに新データを追加する
- テーブルからデータを検索する
- テーブルの既存のデータを変更する
- テーブルの既存のデータを削除する

アプリケーションが実行する他のすべてのタスクは、これらのタスクに比べると副次的です。上記の 4 つのタスクは、データベースを変更したり、データベースにアクセスしたりする唯一の方法です。

データの追加

作成したアプリケーションは、データベース内のテーブルに新データを追加することができます。データを追加するためには、行を挿入してください。行の中でヌルが禁止されている列はすべて挿入されなければなりません。ヌルが許可されている他の列は、この時に挿入される必要はありません。

列に値を指定しないで行を挿入した場合、SQL/A は値の指定されていない列にデフォルト値を設定します。

データをテーブルに大量にロードするために、DBA は **dbld** ユーティリティを使用します。

☐ 準備

ファイルからデータを挿入したい場合、ファイルにデータフォーマットを準備してください。ファイルはアスキーまたはバイナリ形式が可能です。

挿入する値が、必ず列の順序に対応するようにしてください。

☐ 挿入

SQL/A では **INSERT** 文を使用してください。RHLI では **uinsrow** 関数を使用してください。

☐ 確認

データが挿入されたかどうかをチェックする簡単な方法は、テーブルに問い合わせを行うことです。有効なデータを挿入させる他の方法は、データ整合性サブシステムを使用することです。

また、列の定義がユニーク、ダイレクトアクセス、1 次キーになっているものは、Unify DataServer によってユニークであると確認されます。

最後に、挿入された列がリンクインデックスのメンバである場合、Unify DataServer はその値を容認された子値であることを確認します。

追加ヘルプ

項目	参照先
対話型 SQL/A での行の挿入	『*Unify DataServer: Writing Interactive SQL/A Queries*』の95ページ「テーブルへの行挿入」
埋め込み型 SQL/A での行の挿入	『*Unify DataServer: Embedding SQL/A Into C*』の30ページ「テーブルへの行挿入」
RHLI での行の挿入	『*Unify DataServer: RHLI Reference*』の **uinsrow**
データ整合サブシステム	『*Unify DataServer: Configuration Variable and Utility Reference*』の **disc**
リンクインデックス･アクセス方式	『*Unify DataServer: Managing a Database*』の「アクセス方式」

データベースへの問い合わせ

作成したアプリケーションがデータベースに問い合わせを行うことがよくあります。データベースへ問い合わせを行うときは、特定の条件を基準に行を検索してください。

□ 準備

検索したい行を決定します。つまり、検索の基準になる条件を決定します。

例えば、ある値に等しい列を持っているすべての行という条件などが考えられます。

□ 検索

対話型 SQL/A または埋め込み型 SQL/A の **SELECT** 文を使用して、テーブルやビューから行を検索することができます。**SELECT** 文の **WHERE** 句で、検索する行に必要な条件を指定します。

埋め込み型 SQL/A で 2 つ以上の行を選択する場合は、カーソルを使用してください。カーソルで選択された各行に一度にアクセスすることができます。

行を選択すると、 Unify DataServer は 1 つのアクセス方式を使用して行を検出します。SQL/A では Unify DataServer が行を検索するために最適なアクセス方式を選択します。RHLI の場合は、どのアクセス方式を使用するか決定してください。

RHLI アプリケーションで行を検索するには、スキャンまたは順序アクセスを行ってください。スキャンには以下の関数が関係します :

1. **ualcscn** 関数を呼び出し、スキャン情報構造体を割り当ててください。
2. **uinstbl** 関数を呼び出し、どのテーブルにスキャンが行われるかを示してください。
3. **uinsprj** 関数を呼び出し、プロジェクションに追加する列を示してください。
4. **uinsand** 関数を呼び出し、検索基準構造体を割り当ててください。
5. **uinsor** 関数を呼び出し、個々の検索基準を指定してください。
6. 検索された行を整列させたい場合は、**uinssrt** 関数を呼び出し、ソートキーおよびソート順序を指定してください。
7. **UDBVAL** 列構造体を獲得したい場合は、 **uinfscn** 関数を呼び出してください。
8. **ubegscn** 関数を呼び出し、スキャンを開始してください。
9. **ufchscn** 関数を呼び出し、スキャンから行を取り出してください。
10. **uendscn** 関数を呼び出し、スキャンを終了してください。
11. **ufrescn** 関数を使用して、スキャン情報構造体の割り当てを解除してください。

RHLI ではすべての問い合わせを行うことはできません。SQL/A を使用すれば、すべての問い合わせを行うことができます。

追加ヘルプ

項目	参照先
対話型 SQL/A での行の選択	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「行と列の選択」の章
埋め込み型 SQL/A での行の選択	『*Unify DataServer: Embedding SQL/A Into C*』の「埋め込み型アプリケーションの書き方」の章
すべての RHLI 関数	『*Unify DataServer: RHLI Reference*』

行の削除

データベースからデータを削除するには、行を削除してください。どの行が削除されるかを識別するために、条件を指定してください。条件を満たすすべての行が削除されます。

SQL/A文の **DELETE** または RHLI 関数の **udelrow** を使用して、テーブルから行を削除することができます。

追加ヘルプ

項目	参照先
対話型 SQL/A での行の削除	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「行の挿入、更新、削除」の章
埋め込み型 SQL/A での行の削除	『*Unify DataServer: Embedding SQL/A Into C*』の「埋め込み型アプリケーションの書き方」の章
すべての RHLI 関数	『*Unify DataServer: RHLI Reference*』

行の更新

データベースのデータを更新するには、行を更新してください。どの行が更新されるかを識別するために、条件を指定してください。条件を満たすすべての行が更新されます。

SQL/A 文の **UPDATE**、RHLI 関数の **uupdrow** または **dbld** ユーティリティを使用して、テーブルの行を更新することができます。

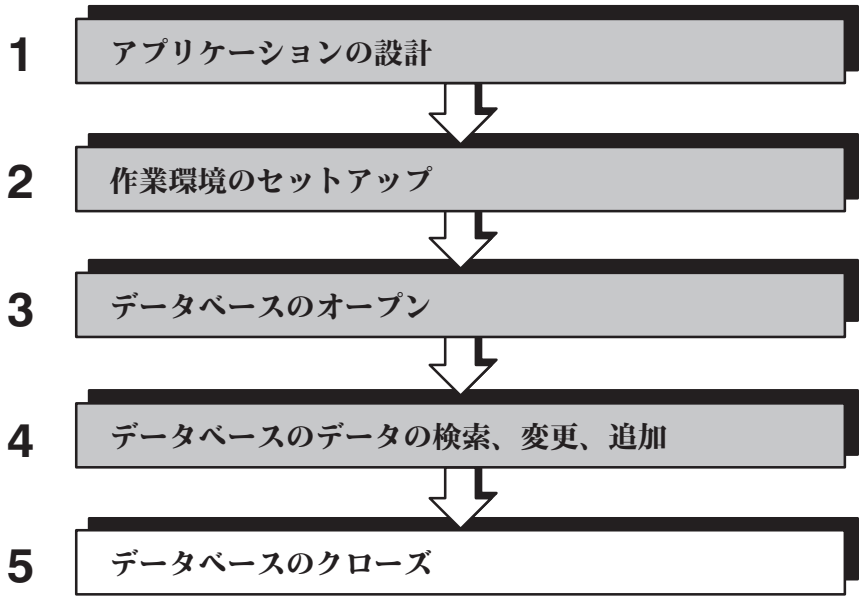
ダイレクト、ユニーク、1 次キーの属性を持つ列の場合、SQL/A は変更された行のデータの値がユニークであることを確認します。

DIS を使用して、列に設定される値を制御することができます。

追加ヘルプ

項目	参照先
対話型 SQL/A での行の更新	『*Unify DataServer: Writing Interactive SQL/A Queries* 』の「行の挿入、更新、削除」の章
埋め込み型 SQL/A での行の更新	『*Unify DataServer: Embedding SQL/A Into C* 』の「埋め込み型アプリケーションの書き方」の章
すべての RHLI 関数	『*Unify DataServer: RHLI Reference*』
データ整合サブシステム	『*Unify DataServer: Configuration Variable and Utility Reference*』の **disc**

アプリケーションの開発：ステップ 5



ステップ 5 : データベースのクローズ

作成したアプリケーションがデータベースの変更やトランザクションのコミットを完了したら、アプリケーションはデータベースをクローズしなければなりません。

□ 準備

データベースをクローズする前にすべてのトランザクションが終了していることを確認してください。通常、操作が問題なく完了していれば、トランザクションをコミットします。問題があった場合は、ロールバックします。

□ クローズ

SQL/A 文の **CLOSE DATABASE**、**DISCONNECT**、**END**、または RHLI 関数の **uclsdb** を使用してデータベースをクローズすることができます。

追加ヘルプ

項目	参照先
CLOSE DATABASE、**DISCONNECT**、**END**	『*Unify DataServer: SQL/A Reference*』
すべての RHLI 関数	『*Unify DataServer: RHLI Reference*』
SQL/Aでのトランザクションの終了	『*Unify DataServer: Writing Interactive SQL/A Queries*』の「トランザクションとロックの管理」の章

付録

付録 A:　　正規化したテーブル

正規化したテーブルを開発することは、実際には、一連のデータのセットを改善することです。 できるだけキー列を定義し、列間の従属を決定することによって、また不必要な冗長性を排除することによってテーブルを改善します。

このセクションは、正規化したテーブル内の 3 つのステップについて述べています。:

1. データベースに格納する要素の予備的なリストを作成する。
2. 正規化したテーブル内の要素に編成する。
3. データベースの図形表示（連係図）を作成する。

ステップ 1: 要素の予備的なリストの作成

正規化したテーブルを作成する前に、データベースに格納する要素の予備的なリストを作成します。まず、データベースを利用する人たちと相談し、彼らのニーズを満たすようにします。ユーザのニーズにもどついて、データベースに格納したい要素または列の予備リストを作成します。

ユーザがレポート内で参照する必要のある情報について考えてください。そして、すべてのレポート項目が別個の要素であるか、あるいは、他の要素に基づいた単なる計算であるかを考えてください。

使用する定義と共通の用語のリストを作ってください。 たとえば、もしある人たちがpartsを項目と呼び、そしてある人たちがpartsを製品と呼ぶなら、ユーザは、データベースでpartsを参照するために使用する用語について同意しておかなくてはなりません。

それぞれの要素が表す情報のタイプについて考えます。 —そのデータは、名前、日付、価格、数量、その他どのようなタイプに該当しますか。

SQL ベースのリレーショナル・データベースは、テーブルの集合を含み、それぞれ関連のある列の名称のついたグループからなります。それぞれの列は名称、長さ、および CHARACTER , NUMERIC ,または DATE のようなデータ・タイプをもっています。

たとえば、在庫管理アプリケーションのデータベースを設計する場合、製造業者と商品項目のテーブルを作成するでしょう。製造業者のテーブルには名称、住所、電話番号、ID 番号、担当者などの列が含まれます。

ステップ 2: 正規化したテーブルの作成

1 組のテーブルを作成し、それぞれのテーブルはデータベースに格納する項目を記述した列の集合であると仮定してください。それらのテーブルがよく定義されているか、すなわちデータベース操作が簡単に実行できるように設定されているか考えてみます。特に、テーブルの定義は新しいデータの追加や既存の変更と削除が他に影響を与えずに行えますか？ 将来アプリケーションが発展したときに変更に順応できる設計ですか？

正規化（*normalization*）の形式理論は、よく定義されたテーブルの設計上の問題を処理することです。正規化は、第 1 正規形から第 3 正規形、およびさらに高度なボイス・コッド（*Boyce-Codd*）正規形までの厳密な一連の正規形（*Normal Form*）に基礎を置いています。

正規形が上にいくほどデータベース操作に関するその特性は向上します。テーブルが第 3 正規形（3NF と略します）にあるように定義された形式は、第 2 正規形（2NF と略します）にも入り、その結果第 1 正規形 （1NF と略します）にも入っていることを意味します。

次の「基本概念」では、データベース設計が正規化されていないときに起こりうるいくつかの問題について述べます。

基本概念

テーブルが繰り返しグループを持たない場合は、テーブルは 1NF（第 1 正規形式） にあります。繰り返しグループとは行、各行ごとに単一の値でなく、親子集合の値の入った列をいいます。リレーショナル・データベースでは、繰り返しグループを含むことはできません。

たとえば、次のテーブルは 1NF ではありません。

1NF でないテーブル

manufacturer_ID	part_ID
01 02	01−111, 01−222, 01−333 02−444, 02−555

上記のテーブルは、製造業者番号 01 が 3 種類の部品を作り、一方、製造業者番号 02 は 2 種類の部品を作っていることを表しています。これは **part_ID** が繰り返しグループであることを意味します。

この状態の共通の変更は、**part_ID** ごとに 1 つの独立した列を設けることです。このテーブルのデータでは、4 種類以上の部品を製造する業者はありませんから、以下のようにテーブルに 4 個の部品列を設けることができます。

1NF でないテーブルの解決方法

manufacturer_ID	part_ID1	part_ID2	part_ID3	part_ID4
01 02	01−111 02−444	01−222 02−555	01−333	

しかし、このテーブルはまだ第 1 正規形ではありません。後になってある製造業者が 4 種類以上の部品を製造した場合は、それをデータベースで表す方法がありません。

この問題を解決するには、テーブルに多数のpart_ID（たとえば 100 。こんなに多数作成する製造業者はないと考えられますが。）を入れる方法があります。しかし、これでは 2, 3 部品しか製造しない平均的な製造業者に対しては格納領域が無駄になります。

この解決策は、以下のようにテーブルを再配置して 1NF にします。

1NF のテーブル

manufacturer_ID	part_ID
01	01−111
01	01−222
01	01−333
02	02−444
02	02−555

これでテーブルは、各行の各列が 1 つの値しかもたない厳密な表構造になりました。テーブルがこの形式になると、リレーショナル・データベースの最低条件は満たすことになります。

なぜ 1NF より上位の正規形が望ましいのでしょうか？この質問に答えるためには、3NF ではないデータベース設計を検討して、挿入、削除、更新操作でどんな問題があるか調べてみましょう。

予備的データベース設計から 1NF へ

倉庫アプリケーションの予備設計をすると仮定します。倉庫は販売用の多くの商品を保管し、各商品は特定の製造業者が製造し、シリアル番号で識別されています。各商品ごとのその製造業者、シリアル番号、および販売価格が必要になります。また、製造業者ごとにそのID 番号、住所、出荷状況が必要になります。

出荷状況は商品の製造業者から入手することがどの程度困難か、あるいは容易かを表します。製造業者が小さくて離れた場所にある場合は、この業者から大量の出荷を迅速に行うのが困難なため status は低い数字になります。反対に、業者が大きな交通の便のよい場所にある場合は、大量の製品を迅速に出荷することが容易なため、status は高い数字になります。したがって、status 列は製造業者の所在地に従属しています。

この予備設計は、次の図に示すような列をもつテーブルを含みます。

データベース設計 1

manfitem	
manf_ID	製造業者の ID (識別) 番号
city	製造業者の所在地
status	製造業者からの相対的出荷の容易度
ser_no	商品のシリアル番号
price	商品の価格

この形式のテーブルは、繰り返しグループを含まないので 1NF の規準を満たしています。テーブルの行の例、**manfitem**（1NF）を**データベース設計 1** に示します。

テーブル名の後につけた（**1NF**）は、テーブルが第 1 正規形であることを示します。

設計 1では、各商品が 1 つの製造業者しかもちませんが、各製造業者は複数の商品を作ることができます。したがって、**manf_ID** が 1 次キーの論理的選択になります。しかし、設計 1 では、**manf_ID** がテーブル中の行をユニークに表すものではないので、 1 次キーにはできません。ある製造業者が作る商品ごとに同じ **manf_ID** が現れます。

データベース設計 1、 manfitem テーブルとデータ値

manfitem（1NF）, key =（**manf_ID** + **ser_no**）

manf_ID	ser_no	status	city	price
0001	101	10	Lynn	3.00
0001	102	10	Lynn	2.00
0001	103	10	Lynn	4.00
0001	104	10	Lynn	2.00
0001	105	10	Lynn	1.00
0001	106	10	Lynn	1.00
0002	101	20	Reston	3.00
0002	102	20	Reston	4.00
0003	102	20	Reston	2.00
0004	102	10	Lynn	2.00
0004	104	10	Lynn	3.00
0004	105	10	Lynn	4.00

manfitem テーブルの 1 次キーは、**manf_ID** と**ser_no** の 2 つの列を組み合わせです。なぜなら、製造業者が重複して同じシリアル番号を供給するからです。ユニークなキーとしては、**manf_ID** と**ser_no** 列の組み合わせを使わなければなりません。

設計 2 が良いリレーショナル・データベースかどうかを判断するには、その設計が追加、削除、変更の基本的な 3 つの操作に与える影響を考えます。

追加

新しい製造業者を追加するとき、その業者が供給する商品が解らなくても追加できますか？ 1 次キーは、**manf_ID** と**ser_no** の両方の列の値が必要なためこの設計では追加できません。

新しい製造業者の行を追加するには、製造業者ID とシリアル番号の両方が必要なため、少なくとも 1 つの商品を供給するまで新しい業者は追加できません。

たとえば、Houston にある製造業者 0008 は、少なくとも 1 つの商品のシリアル番号が解るまで追加できません。

削除
: ある製造業者の唯一の商品を削除し、その業者の情報を残しておけますか？この設計では商品情報を削除すれば、製造業者ID 、所在地、出荷状況も削除されてしまいます。

  たとえば、製造業者 0003 が供給する商品番号102 を削除すると、その製造業者 0003 がReston にあるという情報も失われます。

変更
: 製造業者の所在地を他の場所に変更できますか？その製造業者の所在地は多くの行に現れるので簡単ではありません。これは、行の変更時に起こる問題点です。

  たとえば、製造業者 0001 が Lynn からRevere に移転すると、製造業者 0001 の行をすべて見つけてLynn をRevere に変更しなくてはなりません。 1 行でも忘れると製造業者0001 は LynnとRevere の両方に存在することになります。

設計 1 の問題点は、あまりに多くの異なった種類の情報が互いに従属することです。たとえば、製造業者情報は商品情報に従属します。製造業者情報は商品の行に影響を与えずに変更でき、商品情報は製造業者の行に影響を与えずに変更できるようにしなければなりません。

使いやすいリレーショナル・データベースであるためには、これらの従属性を排除しなければなりません。

1NF から 2NF へ

設計 1 から第 2 正規形に変換するには、設計 1 のノン・キー列間の従属性を排除します。最初のステップは、単一のテーブルmanfitem（1NF）を 2 つのテーブル manf（2NF）とitem（3NF）に置き換えることです。次の図に manfitemをmanf とitem に分割したテーブルを示します。

データベース設計 2, manf と item テーブル

manf(2NF), key =(**manf_ID**)

manf_ID	city	status
0001	Lynn	10
0002	Reston	20
0003	Reston	20
0004	Lynn	10
0005	San Diego	30

item(3NF), key =(**manf_ID** + **ser_no)**

manf_ID	ser_no	price
0001	101	3.00
0001	102	2.00
0001	103	4.00
0001	104	2.00
0001	105	1.00
0001	106	1.00
0002	101	3.00
0002	102	4.00
0003	102	2.00
0004	102	2.00
0004	104	3.00
0004	105	4.00

設計 2 は、設計 1 の **manf_ID** と**city** にあるいくつかの問題を解決しています。情報は変更されていませんがいくつかの改善が見られます。これは製造業者番号を両方のテーブルに入れたことにあります。「制御された冗長度」の導入は正規化の必要な結果です。

manfitem（1NF）テーブルを分割して得られる利点は次のとおりです。

- まだ商品を納入していない新しい製造業者でも追加できます。

  たとえば、製造業者 0005 はまだ商品を納入していませんが、その業者は San Diego にあるという情報を登録できます。

- 製造業者に関する情報を失うことなく商品の行が削除できます。

  たとえば、製造業者 0003 と商品シリアル番号102 の行を削除しても、データベースにはまだ製造業者 0003 が Reston にあるということを記録しています。

- 製造業者の所在地は設計に 1 回だけ現れます。製造業者の所在地を変更するには 1 行だけの更新ですむからです。

  たとえば、製造業者 0001 の所在地をLynn からRevere に変更するには、何行も修正する必要はありません。

テーブルが 2NF であるためには、すべてのノン・キー列は 1 次キーのすべての部分に従属しなければなりません。この定義の結果として、1NF にあって、かつ単一要素のキーを持つすべてのテーブルは 2NF にもあります。**manf**（2NF）と **item**（3NF）テーブルは、すべての列がそれぞれの 1 次キーの要素に従属するので、どちらも2NF にあります。

データベース設計者は、実世界の状況に関する知識に基づいてこの判断に責任があります。テーブルのすべての列が実際にキーのすべての部分に従属するかどうかを決定しなければなりません。

さらに詳細に見ると、設計 2 の **manf**（2NF）テーブルは、まだキーでない列の中で独立性が欠けていることがわかります。たとえば、 **status** 列はキー列 **manf_ID** でなく、キーでない列 **city** に従属しています。すなわち、製造業者は所在地を持ち、所在地は出荷状況をもっています。しかし、出荷状況は直接製造業者とリレーションがありません。

製造業者 ID 番号が変わっても、出荷状況は変わりません。しかし、製造業者の所在地が変わると、出荷状況も変わります。これが問題を起こします。

manf テーブルのノン・キー列間の従属制がデータの追加、削除、変更に次の問題を引き起こします。

追加
: 市の出荷状況を示す行は、その市に所在する製造業者を追加しなければ追加できません。新しい行を追加するには 1 次キーの値が必要です。

  たとえば、Houston にある製造業者を追加しない限り、Houston が出荷状況 25 であるという情報を追加することはできません。

削除
: ある特定の製造業者の行だけを削除しても、その業者が所在する市の出荷状況情報も削除することになります。

たとえば、製造業者 0005 の行を削除すると、San Diego が出荷状況 30 であるという情報も失います。

変更

市の出荷状況は 1 回以上現れます。市の出荷状況を変更すると、その市にあるすべての製造業者の出荷状況を変更しなければなりません。1 行でも忘れると、同じ市にある 2 つの製造業者が異なった出荷状況をもつことになります。

たとえば、Lynn の出荷状況を 10 から 20 に変更するには、製造業者 0001 と 0004 の出荷状況を変更しなければなりません。

2NF から 3NF へ

設計 2 の追加／削除／変更の問題点の解決方法は、manf（2NF）を2 つのテーブル manf（3NF）とcities（3NF）に分割することです。新しい manf テーブルの 1 次キーはmanf_ID、cities のキーは city です。新しいテーブルは以下の図に示します。

データベース設計 3, manf, cities, item テーブル

manf（3NF）, key = （**manf_ID**）

manf_ID	city
0001	Lynn
0002	Reston
0003	Reston
0004	Lynn
0005	San Diego

cities（3NF）, key = （**city**）

city	status
Lynn	10
Reston	20
San Diego	30

item（3NF）, key = （**manf_ID** + **ser_no**）

manf_ID	ser_no	price
0001	101	3.00
0001	102	2.00
0001	103	4.00
0001	104	2.00
0001	105	1.00
0001	106	1.00
0002	101	3.00
0002	102	4.00
0003	102	2.00
0004	102	2.00
0004	104	3.00
0004	105	4.00

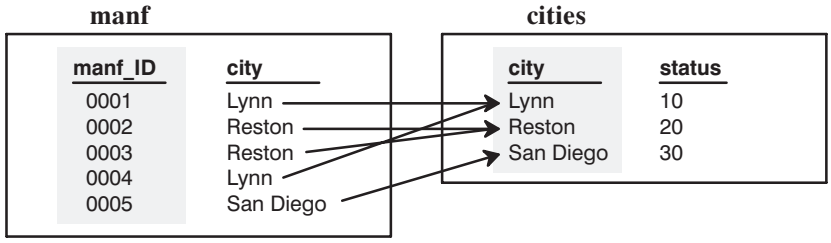
最後の 2 つのテーブルは3NF にあります。3NF でないテーブルの問題点を見てきたので、1NF と2NF は通常それ自体重要ではないことがわかります。それらは 3NF データベース設計への中間点を意味しています。

上記の例で正規化がどのようなもので、あるいはなぜ有益なのかがわかったはずです。正規化は、単一のデータベース・テーブルが実世界の単一の対象を記述することを明確にする形式化された過程です。

設計 3 の各テーブルは単一の対象についての情報をもっています。 **manf** テーブルは製造業者についての情報、**cities** テーブルは市の出荷状況についての情報、**item** テーブルは販売する商品についての情報をもっています。

また、設計 3 は**manf** の **city** と**cities** の **city** の間の連係をもっています。**manf** の **city** は、**cities** のキー列 **city** をポイント、または参照しています。この **manf** と **cities** の連係をとおして、製造業者の所在地について**cities** のすべての情報にアクセスできます。

データベース設計 3, manf, cities, item テーブル



1 つの市はいくつかの製造業者と連係される（参照される）ことができます。これは、1 対多の連係です。1 対多の連係は、リンク・インデックス・アクセス手法で説明します。

ステップ 3: 連係図の作成

データベース設計の改良が終われば、テーブルとテーブル間のつながりを表す連係図を作成します。この図はテーブルの定義を変更するものではなく、単に何ページかのリストの代わりに図形表示をしてテーブルの連係を明らかにするものです。

連係には 1対1、1対多、多対多があります。たとえば、設計 4 で、1 つの市はいくつかの製造業者の所在地で、1 つの製造業者はいくつかの商品を製造します。テーブル **cities** と**manf** は 1 対多の関係があり、テーブル **manf** と**item** は 1 対多の関係があります。

これらの連係は以下のように表せます。

連係図

連系図 では、矢印は 1 から始まり多を示します。このように連係は次の記号で表します。

→ 1 対多の連係

↔ 多対多の連係

— 1 対 1 の連係

多対多の連係の例は、学校の講義の時間割のデータベースです。各学生は複数の教官をもち、各教官は複数の学生をもちます。

連係図を作成する目的は、設計者とユーザ両者のデータベースに対する理解を明確にすることです。連係図は、データベースの主な要素とそれらの関連を明確に表現する簡単な方法です。

正規化したデータベースの設計: 手順説明

正規化したデータベース設計を作成するには、以下のステップに従ってください。

1. 列の予備リストを作成する
2. 予備テーブルを 1NF に変換する
3. 1NF テーブルを 2NF に変換する
4. 2NF テーブルを 3NF に変換する
5. 3NF テーブルを BCNF に変換する

上記のステップについては以下の項でさらに詳しく述べています。

☐ **Step 1** 列の予備リストの作成

列の予備リストを作成するには通常、そのアプリケーションの中心的な専門家の援助と、アプリケーションが現在行われている方法の記録資料によって作成されます。列のリストを作成することは、プロジェクトをとおして使用できる定義や共通の専門用語を準備することにもなります。

データベース設計のこの段階では、無理をしないことが大切です。できるだけ完全なリストを作成することがこのステップでの目標です。後のステップで設計を改良し、範囲を限定することができます。列のリストが揃えば、それらを予備テーブルにまとめます。

☐ **Step 2** 予備テーブルを 1NF テーブルに変換

各テーブルについて、どの列または複数の列が 1 次キーとして使用できるかを決定します。テーブル内の 1 つの行をユニークなものとして識別できる最少の列の組み合わせを探します。場合によっては、1 行をユニークなものとして識別するすべての列が必要になることもあります。

各テーブル内の各列について、行が 1 つ以上の値をもつかどうか調べます。答えがイエスの場合は、その列は繰り返しグループであり、テーブルを改良する必要があります。

たとえば、列 **A**, **B**, **C** および **D** を含むテーブル **T0** があります。これを 1NF に変換すると仮定します。テーブルは以下のテーブル **T0** のようになります。

T0

A	B	C	D

列 **A** と**B** がテーブル**T0** 内の行をユニークなものと識別する場合は、 テーブル **T0a** のように **A** と**B** を合成して 1 次キーに指定できます。1 次キーは☐ の部分です。

T0a

A	B	C	D

A と**B** の 1 つの値に対し、列 **D** は複数個の値があると仮定します。これは**D** が繰り返しグループであることを意味するので、テーブル**T0a** は 1NF ではありません。テーブル **T0a** と **T0b** のように 2 つの 1NF テーブルを作成すると繰り返しグループが除去できます。

T0a

A	B	C

T0b

A	B	D

新しいテーブル **T0b** にはキー列と繰り返しグループ **D** があります。**T0b** 内で、**D** の各値は独立した行にあります。**A** と**B** の同じ値は **D** のいくつかの異なる値をもつことができるので、**D** は 1 次キーの一部でなければなりません。

以下の **Employee** テーブルの例について考えてみます。

Employee

Emp_Number	Name	Address	Dependent

1 次キーとして最も良い選択は以下の図に示すように**Emp_Number** です。

Employee

Emp_Number	Name	Address	Dependent

Dependent 列は、**Emp_Number** の 1 つの値に対して複数個の値をもつので、**Employee** テーブルを以下のように 2 つのテーブルに分割しなければなりません。

Employee

Emp_Number	Name	Address

Emp_Dependent

Emp_Number	Dependent

Employee と**Emp_Dependent** テーブルはいずれも 1NF です。これで **Emp_Dependent** テーブルに年齢、性別、および続柄を加えることができます。

この正規化は、従業員の扶養家族という新しい、かなり重要なオブジェクトについて考えたものです。

☐ **Step 3** 1NF テーブルを2NF に変換

単一の列の 1 次キーをもつ 1NF テーブルは自動的に2NF にもなるので、このステップでは複合 1 次キーをもつテーブルについてだけ考える必要があります。2NF の定義により、2NF テーブルのすべてのノン・キー列は 1 次キーのすべての要素に従属しなければなりません。

例えば、以下のような一般的なテーブル **T1** について考えてみます。

T1

A	B	C	D	E

列 **C** の値は、列 **A** と**B** の両方の値に従属すると仮定します。すなわち、**C** は完全に複合キー **(A, B)** に関数的に従属します。

一方、**D** は列 **A** の値にだけに従属し、**E** は 列 **B** の値にだけ従属すると仮定します。これらの従属性は次のように表すことができます。

(A,B) → C
A → D
B → E

列 **D** と**E** は合成キー**(A, B)** に完全に関数的に従属しないので、テーブル **T1** は 2NF ではありません。**T1** を2NF に変換するには次の図のように分割します。

T1a

A	B	C

T1b

A	D

T1c

B	E

新しいテーブルの定義は従属にもとづいています。ノン・キー列は、それらが従属する複合キーの列と一緒に置かれています。

より具体的な例として **Product** テーブルの例について考えます。

Product

Supplier_ID	Part_ID	Supplier_Name	Description	Cost

エンド・ユーザと打ち合わせの結果、次のような従属が決まりました。

(Supplier_ID,Part_ID) → Cost
Supplier_ID → Supplier_Name
Part_ID → Description

Cost は**Part_ID** だけでなく、その部品（part）を購入する仕入れ先にも従属します。これが自然であり、正規化には直感的な感覚が含まれていることがわかります。

Product テーブルの従属を考えると、結果として2NF テーブルは以下のようになります。

Product

Supplier_ID	Part_ID	Cost

Supplier

Supplier_ID	Supplier_Name

Part

Part_ID	Part_Name

この時点で、新しいテーブルに追加するいくつかの列について考えられます。

☐ **Step 4** 2NF テーブルを3NF に変換

3NF となるテーブルには、ノン・キー列間の従属がありません。そのような従属があるかどうか判断するために、それぞれのノン・キー列の組み合わせを調べなければなりません。もしそのような従属をもつ列があれば、テーブルを分割する必要があります。

以下のような一般的なテーブル **T2** について考えます。

T2

A	B	C	D

列 **B**と**D** の間に次の従属があると仮定してください。

B → D

T2 を以下のように分割します。

T2a

A	B	C

T2b

B	D

2 つの新しいテーブル **T2a** と**T2b** は、前に**T2** テーブルに存在していた従属を分離します。

より具体的な例として、3NF でない 2NF テーブルの **Student** テーブルを以下に示します。

Student

Student_ID	Student_Name	Student_Address	. . .

Advisor_Faculty _ID	Advisor_Name	Advisor_Address

Student テーブルでは、教官の名前と住所は教官の学部 ID 番号に従属します。このテーブルを正規化するために、以下のように 2 つのテーブルに分割できます。

Student

Student_ID	Student_Name	Student_Address	Advisor_Faculty _ID

Faculty

Faculty_ID	Faculty_Name	Faculty_Address

Student テーブルを**Student** と**Faculty** に分割することにより、学部番号と学生番号を分離します。この分解によって情報がなんら失われていないことに注目して下さい。ある学生の教官が誰であるかを今までどおり判断することができます。

前の **Student** テーブルの定義では、教官に関する情報はすべて 1 つのテーブルに含まれていました。新しい定義では 2 つのテーブルにアクセスする必要があります。Unify DataServer はこのようなテーブルを結合するのに特に効率的な方法を提供します。

☐ **Step 5** 3NF テーブルをBCNF に変換

BCNF テーブルであるためには、テーブルの他の列の決定因である列はすべて候補キーでなければなりません。このため、テーブル内のすべての列の組み合わせについて、**C1 → C2** の従属を調べなければなりません。このような従属があれば、決定 **C1** はその行のユニークな識別子でなければなりません。

以下のような一般的なテーブル **T3** について考えます。

T3

A	B	C	D

C → D と仮定します。これは **T3** が 3NF でないことを意味します。 **C** がテーブルのユニークな識別子でない場合、**T3** は BCNF ではなく、以下の図のように分割しなければなりません。

T3a

A	B	C

T3b

C	D

3NF であって BCNF でないテーブルの例は微妙で、理解しにくいものです。ここでは、BCNF であるテーブルは 3NF でもありますが、その逆は必ずしも真でないことを知っていれば十分です。

上記の正規形を省略する場合

「正規化したデータベース設計の作成」で述べた手順は注意して使用してください。正規化は 1 つの手引きにすぎません。手引きに示された手順に固執する必要はありません。

以下の図の **Employee** テーブルの例で考えてみます。

Employee

Employee_ID	Name	Street	City	State	Zip_Code

State は**Zip_Code** だけに従属し、またある場合には **City** も**Zip_Code** に従属します。 このテーブルは 3NF ではないので、以下の図に示すように分割することがあります。

Employee

Employee_ID	Name	Street	City	Zip_Code

Zip_Code

Zip_Code	State

City が **Zip_Code** に機能的に従属しない場合は、この 2 つのテーブルは 3NF です。しかし、実際には次の 2 つの理由により、このような設計はあまり見ることはありません。

- ほとんどのアプリケーションは、4つの列 **Street**, **City**, **State**, および **Zip_Code** をグループとします。
- **Zip_Code** に対する **State** が変更されることはほとんどありません。また、その州に住む従業員がいなければ注意する必要はありません。4 つの列を一緒に格納して得る処理上の利益は、テーブルが 3NF でないために起こる追加／変更／削除に関する問題よりはるかに重要です。

Employee テーブルのノン・キー列間の別の相互従属は次のとおりです。

(Street, City, State) → Zip_Code

このように、テーブルが 3NF でない理由がいくつかあります。1NF はリレーショナル・データベースには必要とされますが、上位の正規形は任意選択であるということです。

索引

数字

A

C

D

O

P

S

U

あ

い

え

と

ひ

ふ

へ

ほ

れ

ろ