静岡大学　　博士論文

拡張可能型データベース管理システムの研究開発

２０１１年　１月

大学院　自然科学系教育部

情報科学専攻

土田 正士

概 要

データベースシステムの適用分野が拡大しつつあり，業務システムで扱うデータ量が増大している．また取り扱うデータの質が変化し，稼動形態の多様化も要求されている．さらに，業務システムでも，画像，音声，動画などマルチメディアデータを取り込んだ情報システムの構築が不可欠となりつつある．マルチメディアデータは，データ量が大容量なだけではなく，メディア操作及び格納形式が多様化かつ順次拡張されることに特徴がある．

本論文では，利用者及び市場からの要件，それらを取り巻く動向から，適用分野を明示する．これらの課題に対処するアプローチとして，並列問合せ処理及びオブジェクト指向概念に基づく拡張可能型データベース管理システムを開発し，実システムに適用することにより，アプローチの有効性を確認した．

データ量の増大に対して，並列データベースシステムの開発の根幹をなす問合せ処理の並列化方式を示す．その問合せ処理の並列化方式は，並列 SQL 文を入力し，その記述の解析処理によって並列性を導き，具体的に並列に実行するための処理手順の表現方法，問合せ変換方法，並列実行環境など並列化に伴う技術からなる．本論文の目的は，SQL の問合せ処理に，パイプライン並列及びデータ並列の考え方を適用した問合せ処理機構について示すことである．具体的には，負荷平準化を目的とするフロータブルサーバ方式を適用し，パイプライン並列の効果により従来と比較して処理時間比で最大約 24 %削減され，実システムへの適用が可能であることを示した．

また，フロータブルサーバ方式を好適に活用するために，変数を含む SQL 文を解析処理する段階と，実際にその SQL の処理手順を実行する問合せの実行処理時に変数値に従い最適化する段階からなる，2 段階最適化方式を開発した．この 2 段階最適化方式の負荷評価を行い，処理時間比で問合せ処理実行時間の高々約 3 %以下であり，実システムへの適用が可能であることを示した．

さらに，マルチメディア情報からなるデータの質の変化に対して，検索及び格納などの操作に対応するライブラリ群を，自由に追加，登録できるプラグイン方式を開発した．このプラグイン方式により，耐久性 (Durability) の実現と高速性の確保，かつそれらの両立，及び ORDBMS(オブジェクト指向リレーショナルデータベース管理システム) 製品での SGML 文書，文字列，XML への構造検索を始めとして，類似画像検索，空間検索として実システムに適用することのより，組込みが容易であることを確認した．

XML データベースの応用事例として，電子申請システム，ワークフローシステムなどで Web 化が進展している．本論文では，Web フォーム基盤のアーキテクチャとして機能配置方法，記述方法，及びデータ連携・交換方法を開発し，それらを実システムに適用することで，電子申請システム，ワークフローシステムでの有効性を確認した．

目 次

図目次

表目次

第1章　緒言

データベースシステムの適用分野が拡大しつつあり，基幹系システムから情報系システムに広がることでデータ量が増大し，また取り扱うデータの質が変化し，さらに稼動形態の多様化も要求されている．ここでは，利用者及び市場からの要件，それらを取り巻く動向から，適用分野を明示し，課題に対処するアプローチを示す．

1.1.　動機と目的

IT システムの構成要素の中でもデータベースシステムは根幹をなす非常に重要な部分である．そのデータベースシステムは，昨今「データ量や処理量の増加への対応」,「データの多様化」,「データ処理特性の広がり」という動向への対応が進んでいる．図 1.1 は，データベースシステムの適用動向を示す．

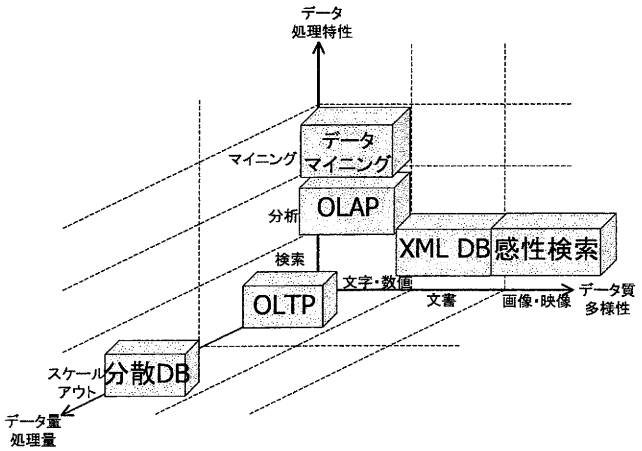


図 1.1: データベースシステムの適用動向

まず,「データ量や処理量の増加への対応」については，量の増加に対してスケーラビリティの考え方が重要である．その例としては，オンライントランザクション処理（OLTP）及び分散データベースがある [3].

また,「データ処理特性の広がり」については，検索から分析，さらにはマイニングまで利活用が進んでいる．その例としては，オンライン解析処理（OLAP）及びデータマイニングがある [2].

さらに,「データの多様化」については，文字・数値のみならず，文書や画像・映像もデータベース応用の対象として広がっている．その例としては，XML データベース及び感性検索がある [4].

1.1.1. 業務システムへの適用

業務システムを取り巻くシステム状況は，時代と共に変化を続けている．即ち，業務システムで必要とされるデータ量が，CPU 性能，ディスク容量の伸びを上回り，トランザクション量に比例して増大しつつあり，データベース容量の増大に対してスケーラブルなデータベースシステムが要求されている．また，数万ユーザ数を超える同時実行ユーザへの対応が要求され，PB 級単位のデータへのアクセスを行う検索トランザクションの出現に対しても表のサイズに依存しない応答時間の保証が必要とされてきている．

そのような業務システムは，基幹系システムと情報系システムに区分することができる．図 1.2 は，基幹系システム及び情報系システムの位置付けを示す.

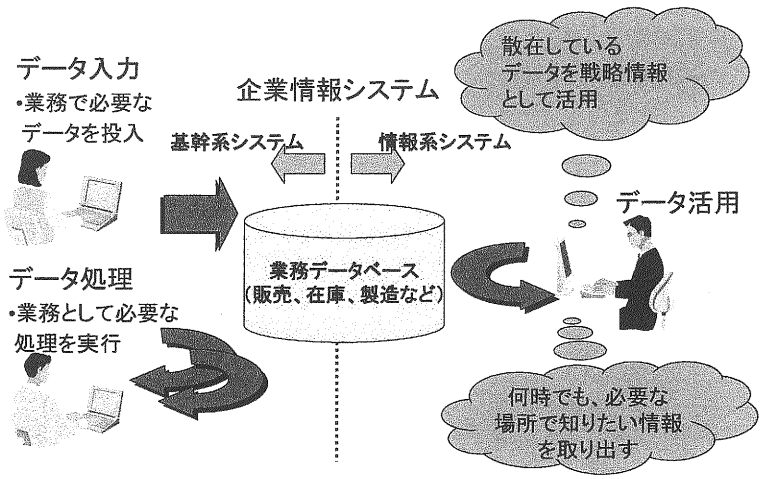


図 1.2: 基幹系システムと情報系システム

基幹系システムは，販売業務，経理業務，在庫管理業務など日々の企業活動を支える基幹的な業務を処理するためのシステムである．この基幹系システムでは，業務で必要なデータを投入し，業務で必要な処理を行うなど一時も止まることなく，また誤りを発生させてはならない．企業活動の根幹を担うシステムであるため，企業にとって重要な資産であるデータベースは，安全保護の観点での万全の対策が必須とされる．このように，高度な信頼性，耐障害性，及び可用性が必要となる特性をミッション・クリティカルと呼び，OLTP が活用されている.

一方，情報系システムは，企業活動において，計画業務，企画業務など経営活動を支えるシステムであり，経営管理システム，意思決定支援システムなども含まれる．基幹系システムとは違い，情報系システムが存在しなくても企業活動を行うことは可能ではあるが，日々の業務の効率化や高度な経営上の判断を支援する目的で戦略的にデータを活用することがで

きる．また，このようなデータは予め見方を決めることができずに，発見的に知りたい情報を取り出すこともできねばならない．このために，OLAP やデータマイニングが活用されている．

基幹系システム

銀行のオンライン端末や航空会社の航空券予約端末などから発生する取引のような同時多発的なトランザクション処理要求を，オンラインでサービスするアプリケーションプログラムの実行形態を OLTP(OnLine Transaction Processing) と呼ぶ [60]．図 1.3 は，OLTP システムの機能構成を示す．OLTP では，それぞれの端末からの処理要求に正しい結果を返すために，ACID 特性 [60] を実現するためのソフトウェア，TP(Transaction Processing) モニタが利用されている．Web システムを始めとしてクライアント/サーバシステムが広まってきており，TP モニタを活用する OLTP システムが注目されている．

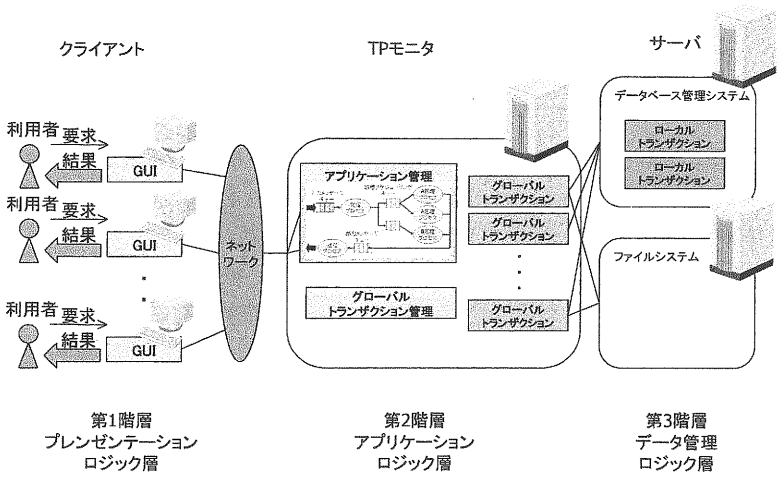


図 1.3: OLTP システム

この例は，3 階層クライアント/サーバシステム構成であり，第 3 層ではデータベースサーバ，ファイルサーバなど異種のサービスが混在してデータを提供している．ファイルシステムではトランザクション管理が実現されていないので，TP モニタが代わりに ACID 特性を実現することとなる．また，データベース管理システムが扱うトランザクションと TP モニタが扱うトランザクションを区別して，前者をローカルトランザクション，後者をグローバルトランザクションと呼ぶ．

ここで，トランザクションについて説明する．図 1.4 は，トランザクションの概念を示す．

トランザクションとは何かを，銀行で A さんの口座から B さんの口座に金額 100 を送金するケースで考える．これは，A さんの口座から指定された金額 100 を引き落とし，引き落とした金額 100 を B さんの口座に振り込むという手順となる．この場合，A さんの口座から引き落とし B さんの口座に振り込みを行う前に，何かの障害で処理が中断してしまい，A さんの口座から引き落としてしまい，一方で振込先の B さんの口座には入金されずにどこ

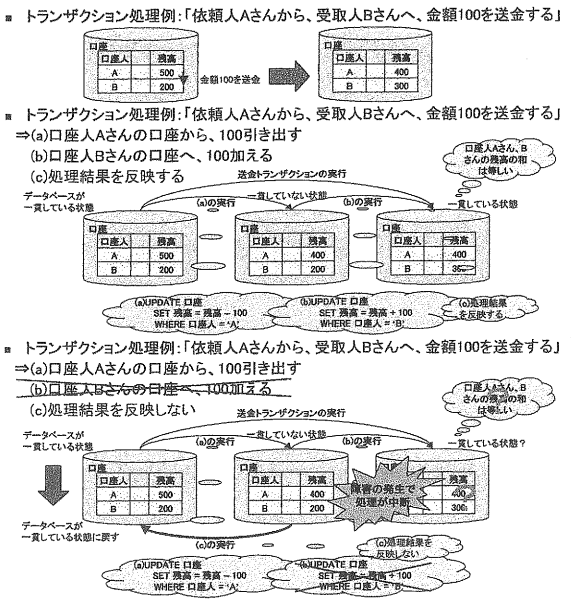


図 1.4: トランザクション

かに消えてしまう，という状況はけっして発生してはならない．B さんの口座への振り込みができないならば，最初の A さんの口座の引き落としもされなかったことにしておき，元々の送金処理自体が全くなかったことにする必要がある．

このように，引き落としと振り込みが一つの処理として実行された結果がデータベースに反映されるのか，あるいは全く結果が反映されないようにするかのいずれかであり，中途半端な処理状態を残さない処理単位をトランザクション呼ぶ．また，銀行の残高の総和は不変というデータベースの整合性，即ち送金処理の前後で残高の総和 700 は保たれている．

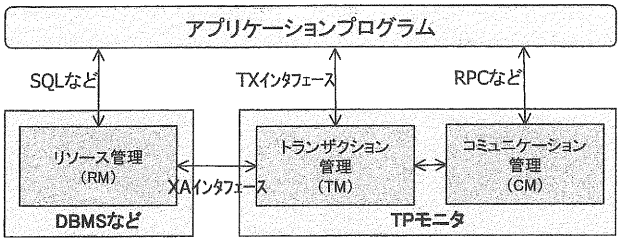
このトランザクションを，分散環境に適合させる分散トランザクション処理モデルも実現されている．図 1.5 は，分散トランザクション処理モデルの例を示す．

分散トランザクションとは，異種のサービスが分散してデータベースを提供するシステムにおいて，データの整合性を保証するために，各サービスに対応するローカルトランザクションを一つのグローバルトランザクションとして取り扱い，このグローバルトランザクションを整合性の単位として ACID 特性を実現するものである．

この分散トランザクション処理の内容を記述したものに，X/OPEN が規定する分散トランザクション処理 (Distributed Transaction Processing：DTP) モデルがある [5][6]．この DTP モデルは，分散したアプリケーションプログラムとリソース管理，トランザクション管理，及びコミュニケーション管理の間の機能分担とインタフェースを規定する．

リソース管理 (Resource Manager：RM) は，データベース及びファイルなど共有の資源を管理する．この RM は，トランザクション管理 (Transaction Manager：TM) との間に XA インタフェースが存在する．この XA インタフェースは，TM が各 RM とのトランザクション制御を行うために利用される．具体的には，XA インタフェースは，各アプリケーションとのトランザクションの開始，終了の指示，及び複数のローカルトランザクションを扱うための 2 相コミットプロトコルの制御を含む．通常は，アプリケーションプログラムから陽に使われることはなく，TM 及び RM 間の制御方法として隠されている．

一方，TM は，分散トランザクション，即ちグローバルトランザクション及び各 RM で管



凡例）
リソース管理:Resource Manager
トランザクション管理:Transaction Manager
コミュニケーション管理:Communication Manager
DBMS:Database Management System
TPモニタ:Transaction Processing Monitor
RPC:Remote Procedure Call

図 1.5: 分散トランザクション処理モデル

理されるローカルトランザクションを管理し，TP モニタがこの機能を実現している．TM はアプリケーションプログラムと TX インタフェースで接続する．具体的には，アプリケーションプログラムは，DTP サービスを呼び出すことでグローバルトランザクションの開始，終了の指示が行われる．この他に，コミュニケーション管理 (Communication Manager：CM) は，アプリケーションプログラム間での RPC(Remote Procedure Call) 通信サービスなどの機能が提供されている．

昨今，この基幹系システムは，数万ユーザ数を超える同時実行ユーザへの対応，及び PB 級単位のデータへのアクセス要求のようなデータ量，アクセス処理量に対応する必要性が高まってきている．

情報系システム

どのデータベースも何らかの意図を持って情報を保持しているが，基幹系データベースとは対極的な情報系データベースと呼ばれるものがある．基幹系データベースには，受発注処理，在庫処理，発送処理という日常の活動を支える生のデータが含まれるが，一方で情報系データベースは，マーケティングや生産管理など意思決定を支援する目的で管理されている．その情報系データベースは，基幹系データベースのデータから抽出・変換されて構築されるが，いくつかの点で強化が図られている．

図 1.6 の例では，各部門の視点で管理される顧客及び商品テーブルを部門横断で併合して，顧客分析及び商品分析テーブルとして構築している．これによって，全社レベルで統合的に事実を把握することが可能になる．また，地域毎の商品売上推移傾向を分析するために多次

元データモデルや顧客購買パターンを把握するために販売履歴テーブルを構築することもある.

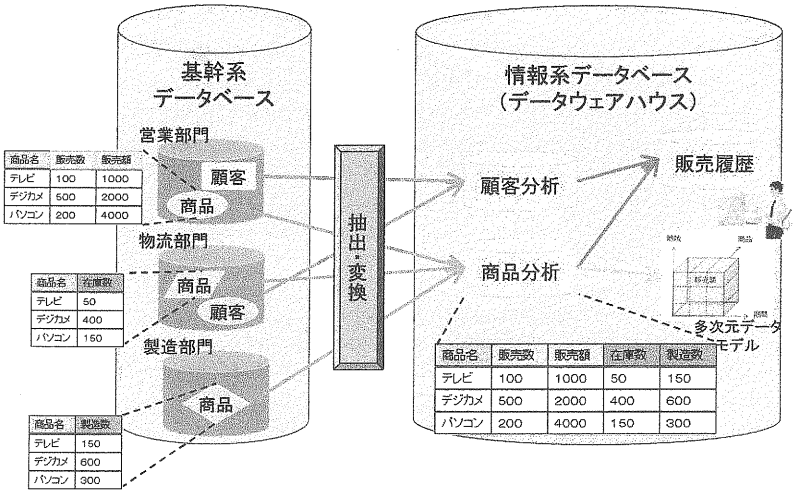


図 1.6: データウェアハウス

このように意思決定を支援する分析処理に適した形に整えることがある．この情報系データベースの器をデータの倉庫，即ちデータウェアハウスと呼ぶ [1].

1. OLAP システム

   企業や市場で何が起こっているかを知るための方法を利用者に提供できることは重要である．ここで，期間，地域，商品というデータ項目をひとつの次元軸とし，これら複数の次元軸にまたがって行う分析手法を多次元分析と呼ぶ．図 1.7 の例は，立方格子の多次元データモデルを示している．オンライン分析処理（OLAP：OnLine Analytical Processing）とは，この多次元分析を対話的に実行するために使われるツールとデータベースの組み合わせによって，分析者が分析処理をオンラインで実行し，エンドユーザが直接的にデータベースを検索及び集計し，簡単に分析する機能を提供することで，業務上の課題やその解決方法を発見する分析型のアプリケーションの処理形態を指す.

   OLAP では，3 次元以上の多次元データベースから 2 次元のクロス集計表としてデータを取り出し，ドリルダウン，スライシング，ダイシングという 3 つのデータ分析操作によって，データの多角的な分析を行う．ドリルダウン操作は，2009 年を月毎に詳細化してきめ細かに傾向を把握できる．スライシング操作は，特定商品「パソコン」に絞り込んだ販売額推移が把握できる．ダイシング操作は，地域軸及び期間軸の組合せから，商品軸及び期間軸の組合せに分析軸を変更した傾向の把握ができる.

   また，この OLAP の基本要件が 12 のルールとして知られている [2]．さらに，OLAP には，構成の違いから M-OLAP(Multi-demensional OLAP) 及び R-OLAP(Relational OLAP) の 2 種類が存在する.

   M-OLAP は，部門横断で顧客あるいは商品のような視点での事実の情報から要約情報

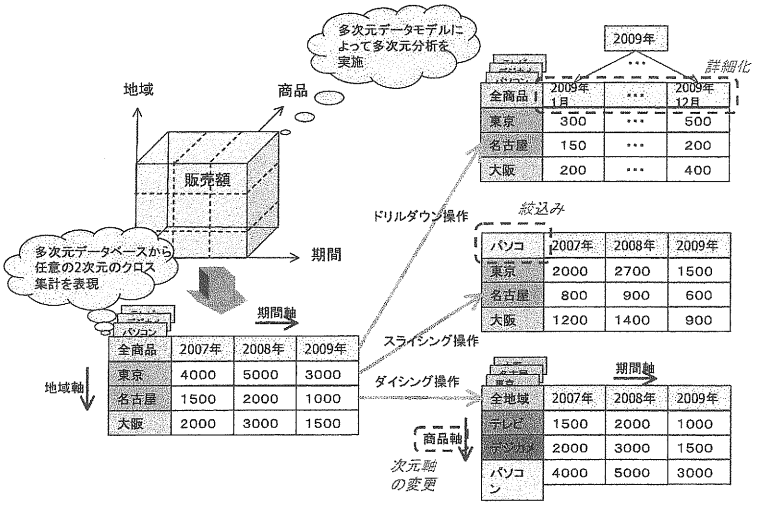


図 1.7: OLAP

をデータウェアハウスに作成し，ドリルダウン操作，スライシング操作，ダイシング操作の処理要求に対して，対話的に処理する形態を指す．次元毎の要約情報を予め集計した結果を多次元データベースとして専用エンジンを使い格納しているので，各種操作に対して高速な処理が実現できる．反面，事前に集約計算を実施し，多次元データベースに格納しておく必要がある．

一方，R-OLAP は，リレーショナル DBMS を格納手段として利用して，処理要求の結果を多次元の軸に基づき表示を行う処理形態である．M-OLAP とは異なり，多次元データベースのための専用エンジンを必要とはしないが，性能を考慮して要約情報を作成するなどの工夫が必要である．この要約情報をデータの市場，即ちデータマート (Data Mart) とも呼ぶ．

このデータマートを作成するためには，多数表からなるジョイン処理が必要とされ，その高速化が要求されている．

2. データマイニング

データマイニングは，データベース探索などとも呼ばれ，基幹系データベースから新しい情報を探し出すことで，規則，関連性や傾向を見い出す作業を指す．この作業によって，ビジネスに掛かる費用を大幅に削減し，過去の販売履歴の中から新たな特徴を見出して新しい収入源を発見することができる．

また，既存顧客とのビジネスの中で収集されたデータ間の関係を調査し，これらの事実の間に新たな規則関係がないかどうかを探る．そのためには，過去の履歴データと，最新のデータの集まりを提供するデータウェアハウスが必要である．

図 1.8 の例では，基幹系データベースから抽出・変換されて構築された顧客分析及び商品分析テーブルから，販売履歴テーブルを作成し，販売傾向パターンを導き出している．

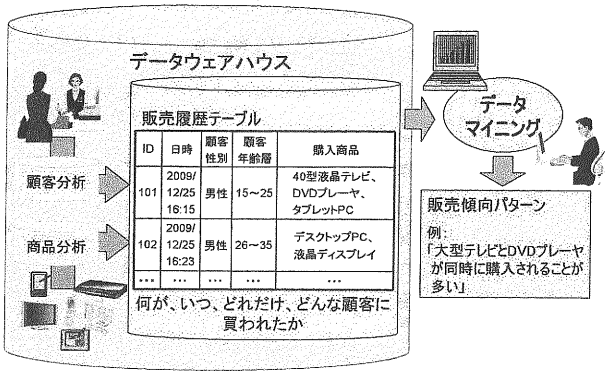


図 1.8: データマイニング

1.1.2. アプリケーション分野の拡大

オフィス環境のデータベースシステムでは，テキストを中心に画像，図形，音声，文字数値などを含む文書データを扱う．また，1 トランザクション当たり数百 KB～数 MB でデータ量を取り扱う．さらに，平坦化されたレコードをアクセスする処理パターンから，構造が動的に変更され，また任意の階層を検索する処理パターンに対処する必要性がある．このようなアプリケーション分野の拡大に対して，対処がなされている．

マルチメディアデータベース

テキストに加えて画像，音声などを総称してマルチメディアデータと呼ぶ．そのマルチメディアデータに対して，文字数値と同様に検索を行うためにマルチメディアデータベースシステムが提供されている．図 1.9 の例では,「いちご」に似た色の写真で商品を検索できると便利である．

まず，検索の対象となるマルチメディアデータの集まりから特徴ベクトルからなる特徴量を抽出する．色であれば RGB の三次元空間のように，マルチメディアデータの特徴量は多次元空間の点あるいは領域に対応する．また，この特徴量から元のマルチメディアデータを効率よく検索するための多次元インデクスを作成することがある [8]．マルチメディアデータを検索する場合，利用者が検索対象として例示する画像などキーとなるマルチメディアデータから特徴量を抽出し，多次元インデクスがあれば，そのキーの特徴量に近いデータ集合を検索する．さらに，キーと検索されたデータ集合の各要素との距離を計算して，距離が近い要素を類似性が高いとみなし，合わせてメタデータ属性も加味して，その類似性の高い順に利用者に検索結果を返す．将来的には，ある特徴量の組み合わせに対して例えば「鮮やかな」などの特定の概念，言葉に対応させることで，いわゆる感性検索が実現できる．

文字，数値のデータを検索するためには，B 木インデクス [7] が広く利用されているが，一方ここで示すマルチメディアデータを効率よく検索するためには，各メディアの特性に適合するインデクスを利用することが必要であり，このようなインデクスはマルチメディアデータベースシステムに適宜に拡張可能であることが要求されている [8][9][10].

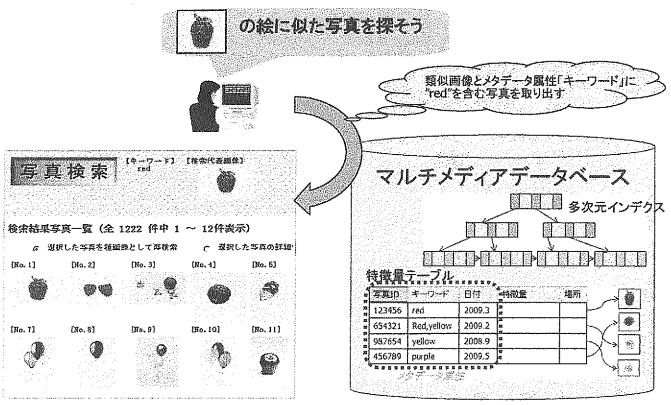


図 1.9: マルチメディアデータベース

XML データベース

Web 基盤上でのデータ交換の活用に留まらず，データ処理及び格納の活用に広まっている [4]．具体的には，XML データは，医療，財務関係の報告書類や，官公庁の公式文書，出版物など XML 形式での作成が標準となり，またインターネット上で種々の情報提供が進んでいる．

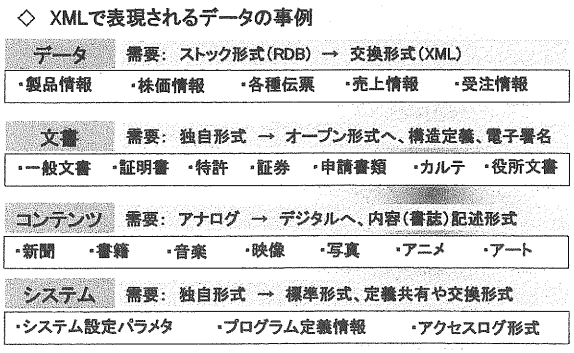


図 1.10: XML の適用分野

この XML データベースの応用システムでは，インターネットを介して流通，交換が行われ，現在では蓄積，検索まで広がりつつある．XML はタグで囲まれた要素の階層構造 (木構造) を利用して，柔軟にデータ項目を表現することができるので，利用形態への進化に対応して，情報源を横断する検索やデータ構造を変更することが可能である．

XML が活用されるアプリケーションの情報提供の用途毎に事例を，図 1.10 に示す．デー

タ，文書を始めとして，コンテンツやシステム間での標準的な交換形式として採用されつつある．

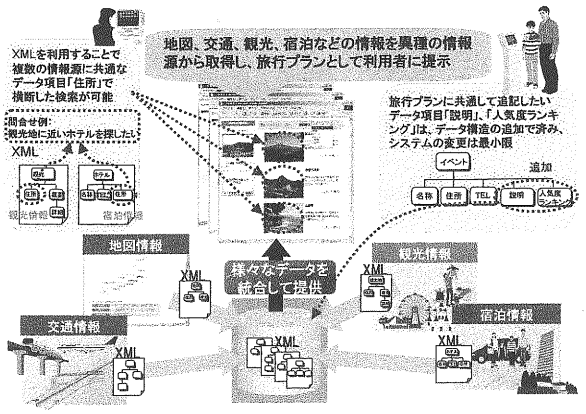


図 1.11: XML データベースシステム

図 1.11 の例では，「観光地に近いホテルを探したい」問合せに対して，データ項目「住所」に着目して情報源を横断して検索することができる．また，旅行プランにデータ項目「人気度ランキング」を追記することには，データ構造の追加で済ませることができる．

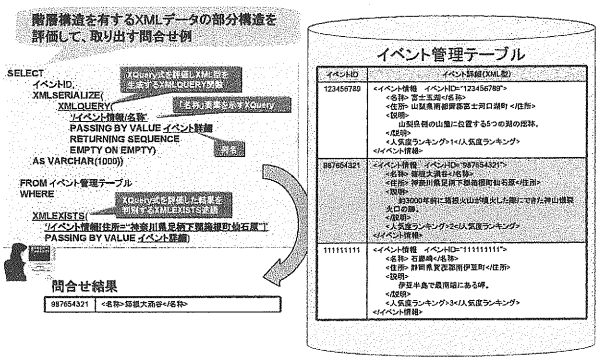


図 1.12: XML データベースシステムの例

また，この XML データベースの問合せ事例について，図 1.12 に示す．XML データベースは，XML データを格納するためのデータベースであるが，XML データだけではなく，リレーショナルデータモデルで表現されたデータも併用できる XML データベースシステムが主流になりつつある．

この例では，リレーショナルデータベースのイベント管理テーブルのイベント詳細列に XML データ型のデータが格納されている．「神奈川県足柄下郡箱根町仙石原に所在する観光地イベントの名称を探したい」問合せに対して，データ項目「住所」に着目して XML データの部分構造を指定することで条件を指定している．また，取り出すデータ項目「名称」も XML データの部分構造を指定して，イベント ID と供に問合せ結果を取得することができる．

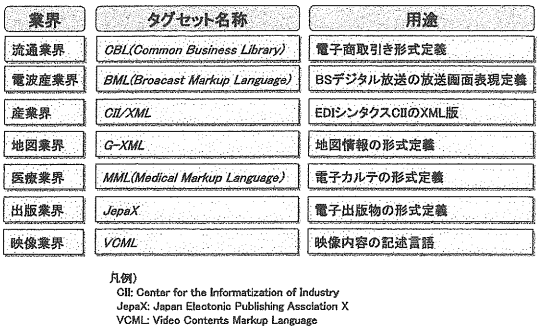


図 1.13: XML タグセットの例

さらに，XML は，異なるアプリケーション間でのデータ交換形式として，広範囲の業界でタグセットの標準化が進んでいる．図 1.13 は，そのタグセット例を示す．XML をデータ交換形式として利用することの利便性が認識されるのに従い，Web 上で交換される申請書類のような文書，書籍や映像のように流通するコンテンツ，及びアクセスログを記録するようなシステムにおいて幅広く活用される場面が益々増えてくる状況にある．

1.2. 関連システム動向

1.2.1. データベース管理システム

データベース管理システム（DBMS）は，データベースを管理，運用するための専用のソフトウェアである [3]．図 1.14 で示す，DBMS が提供する機能について解説する．

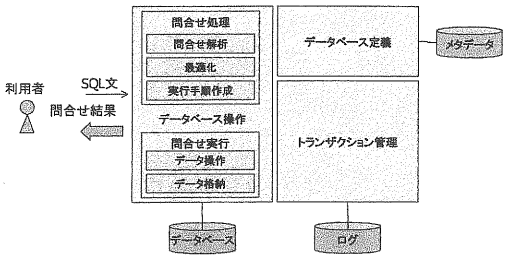


図 1.14: データベース管理システム

まず，データベース定義とは，DBMS が管理するデータベースに関するデータ，即ちメタデータを管理する機能である．メタデータは，データベースに存在する表及び列，インデク

スの有無などの情報を管理する．このメタデータは，データベース操作の際にアクセスされ，表に関する定義情報を提供する．また，データベース操作とは，データベースに対して利用者が発行したSQL文を構文的，意味的に解析して，データベースを操作することで，問合せ結果を返す機能である．さらに，トランザクション管理とは，データベースの整合性を保つためにデータベースの読み書きの単位であるトランザクションを管理する機能である．このために，トランザクションで読み書きされたデータベースの更新情報は，ログとして取得されている．

データベース操作

1. 問合せ処理

   利用者によって発行されたSQL文を実行した結果を返すデータベース操作は，問合せ処理と問合せ実行から成り立つ．図1.15は，問合せ処理の構成を示す．問合せ処理は，問合せとして発行されるSQL文を入力として，次の3つの処理が実行される．

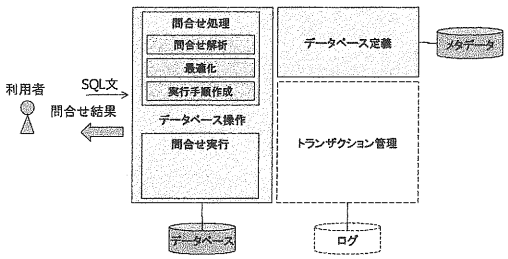


図 1.15: 問合せ処理

まず，問合せ解析はSQL文の構文を解析して，構文的な妥当性を確認する．次に，最適化は，複数の処理方法の候補を作成し，その中から問合せの処理時間が最も短くなる処理方法を選択する．その際に，データベース定義を介してメタデータを参照しながら表が定義されているかなど意味的な妥当性も検証する．さらに，実行手順作成は，最適化で選択された処理方法の中で，問合せの処理時間が最も短くする実行手順を作成する．また，問合せ実行は，問合せ処理で決定された実行手順に基づき，データベースをアクセスしながら，問合せ結果を利用者に戻す．

次に，図1.16を用いてデータベース操作の流れについて説明する．利用者が発行するSQL文は，「何を取り出すか（What）」が表現されているだけなので，「如何に取り出すか（How）」はDBMSで決定する必要がある．即ち，DBMSはWhatからHowへの変換を行うことになる．一般的に，Howによって処理時間が左右されるので，この変換処理が重要である．

まず，問合せ解析はSQL文が構文的に妥当であるか確認するとともに，構文要素からなる解析木を作成する．次に，最適化は，問合せの処理方法を表現する処理手順の候

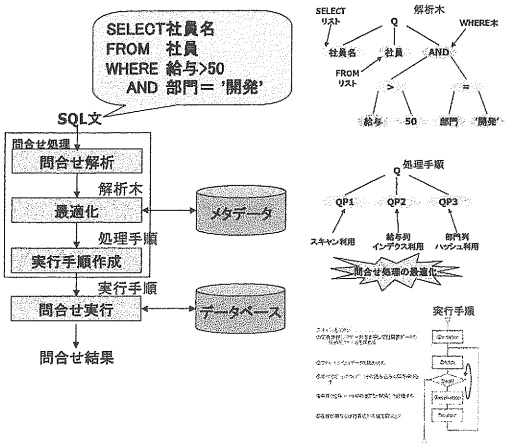


図 1.16: データベース操作の流れ

補（QP1,QP2,QP3）を作成する．各処理手順は，インデクスの利用有無も考慮して作成されている．その際に，メタデータを参照しながらデータベースに関する種々の統計量（行数など）から処理手順毎に処理時間を推定している．さらに，最適な処理手順に基づき，データへのアクセス方法をフローで表現する実行手順を作成する．問合せ実行にて，実行手順のフローを辿りながら，データベースをアクセスし，問合せ結果を利用者に戻す．

その中で最適化は，問合せの処理方法を表現する処理手順の候補を作成するが，その際に生成される各処理手順は，インデクスの利用有無を考慮して作成されている．このインデクスの有効性について，図 1.17 を用いて考察する．

インデクスは，データベースへのアクセスを高速化する手段である．しかし，データ値が重複している場合，必ずしも検索の効率が向上しない場合が存在する．ここで，ひとつのデータ項目当たりの平均的な検索効率を算出して，インデクスの有効性を評価する．前提として，全行数が 10,000 件，データ値数が 100 個とし，各データ値毎に対応する行数設定されている．また，データベースを格納するページ群には，1 ページ当り 50 行が格納されている．ここで，等号条件で検索を行うことを考える．単純化のために，データベースのページ読み込み回数で評価する．

インデクスを利用しなければ，全行数 10000 件のデータはスキャン法を利用することで，逐次的に取り出されることになり，ページ読み込み回数は同じである．即ち，200 回 (=10,000/50) のページ読み込み回数となる．

一方，インデクス法を利用する場合，異なるデータ値を識別するための処理効率は向上するが，同じデータ値の中では重複する行数分だけ逐次的にデータベースのページを読み込む処理となる．従って，インデクスのキー値の分布によって，ページを読み込む回数が決まることになる．データ値として 100 個が，ばらつきが無く各行数が 100 件の場合と，ばらつきが有り各行数が各々1〜500 件の場合を考える．

まず，データの分布にばらつきが無い場合，各データ値に対応する行数 100 件分の読み込み処理が行われる．即ち，100 回の読み込みが行われる．

次に，データの分布にばらつきが有る場合，例えばキー値「AAA」が検索されると対

データ値にばらつきが無い場合：

データ値	行数
AAA	100
：	：
AAJ	100
BBA	100
：	：
BBJ	100
CCA	100
：	：
CCJ	100
DDA	100
：	：
DDJ	100
EEA	100
：	：
EEJ	100
FFA	100
：	：
FFJ	100
GGA	100
：	：
GGJ	100
HHH	100
：	：
HHJ	100
IIA	100
：	：
IIJ	100
JJA	100
：	：
JJJ	100

データ値にばらつきが有る場合：

データ値	行数
AAA	1
：	：
AAJ	1
BBA	2
：	：
BBJ	2
CCA	4
：	：
CCJ	4
DDA	8
：	：
DDJ	8
EEA	15
：	：
EEJ	15
FFA	30
：	：
FFJ	30
GGA	60
：	：
GGJ	60
HHH	130
：	：
HHJ	130
IIA	250
：	：
IIJ	250
JJA	500
：	：
JJJ	500

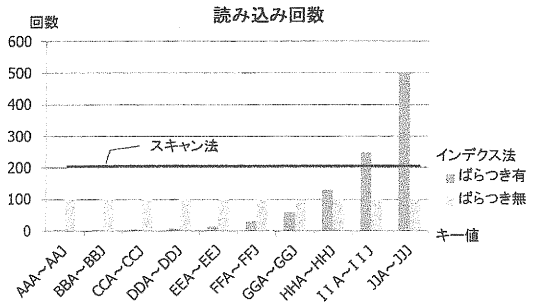


図 1.17: インデクスの性能特性

応する行数1件分，即ち1回の読み込み処理が行われる．また，キー値「JJJ」が検索されると対応する行数500件分，即ち500回の読み込み処理が行われる．

これらの評価から，等号条件で検索処理を行う場合，必ずしもインデクス法が必ずしも有効ではないことが分かる．データの分布にばらつきが無い場合，スキャン法の200回の読み込み回数と比較して，インデクス法は100回の読み込み回数となり，インデクス法が優位となる．一方，データの分布にばらつきが有る場合，インデクス法でキー値「IIJ」～「JJJ」を読み込むと，例えばキー値「IIJ」では250回，キー値「JJJ」では500回と，いずれも読み込み回数はスキャン法の200回を上回る．

また最適化は，2つの表間で突き合わせを行う結合処理の処理手順候補を作成する．一般に，結合処理の高速化も重要な課題である．その結合処理には，下記の3つのアルゴリズムが代表的である．ここで，表R及び表Sとする．図1.18は，ネストループ結合法の実行手順の例を示す．

- ネストループ結合法 (nested loop join method)
  表R及びSの各行同士のすべての組合せを確認する．表Rの1行に対して，ループ制御構造の内側になる表Sを繰り返し読み出すことで実行される．
- ソートマージ結合法 (sort-merge join method)
  表R及び表Sがその結合列上で順序付けされていることが前提であり，それらの表R及び表Sを交互に辿り行同士を結合する．
- ハッシュ結合法 (hash join method)

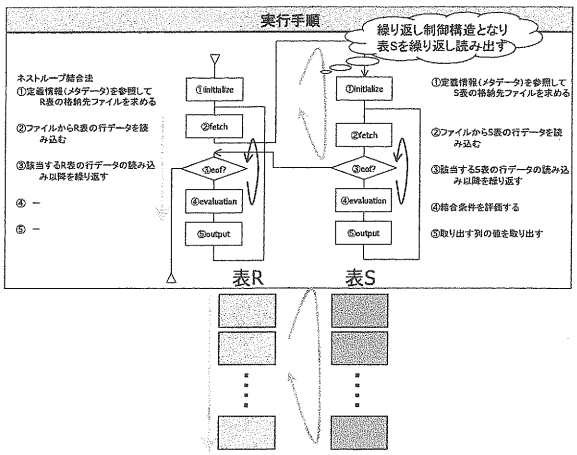


図 1.18: ネストループ結合法の実行手順の例

まず，表 R の各行を，結合する列にハッシュ関数を適用してハッシュ表 R' を作成する．次に，表 S の各行毎に，同じハッシュ関数を適用して，ハッシュ表 R' をアクセスして結合する．

これら 3 つの代表的な結合処理のアルゴリズムを示したが，結合処理に関与する 2 つの表の格納方法に従って，並列性を見い出すことで更に高速性を引き出すことも必要とされている．

2. 問合せ実行

問合せ実行では，データベースをアクセスしながら，問合せ結果を利用者に戻す．図 1.19 は，問合せ実行の構成を示す．

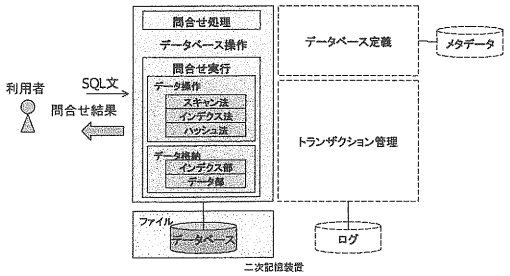


図 1.19: 問合せ実行

問合せ実行は，データへのアクセスのために種々のアクセス法を実現するデータ操作と，すべてのデータを格納するデータ格納から構成される．データベースは，システムのトラブル時にも影響されないため，主記憶装置ではなく二次記憶装置に格納され

ている．データベースのすべてのデータは，ファイルを介してその二次記憶装置に格納されているデータ群を仮想化し，利用者に論理的に分かり易い形式で提供される．データ操作では，取り出すデータにある規則性を持たせてデータへのアクセスを利便化する試みがなされている．これをアクセス法と呼び，スキャン法，インデクス法，ハッシュ法が提供されている．一方，データ格納には，データへのアクセスを高速化するインデクス部及びデータを管理するデータ部がある．

次に，アクセス法と格納方法の関係について，図 1.20 に示す．

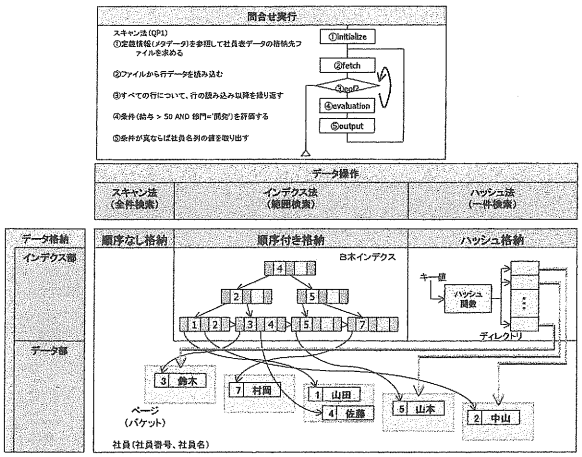


図 1.20: アクセス法と格納方法の関係

問合せ実行では，実行手順のフローを辿りながら，データベースをアクセスする．その際には，データ操作で実行するアクセス法は，データ格納で提供する格納方法に基づいて実現されている．まず，条件式が指定されていない全件検索の場合，データ操作ではスキャン法が選択される．また，条件式で範囲を取り出す範囲検索の場合，インデクス法が選択される．さらに，条件式で特定のキー値だけを取り出す一件検索の場合，ハッシュ法が選択される．

データ格納では，データ部が基本的にデータベースが管理するすべてのデータを格納する．しかし，データへのアクセスを高速化する目的から，インデクス部には各アクセス法に向いたデータ構造を用意することで，利便性を図っている．具体的には，B 木を活用した順序付き格納，ハッシュ構造を活用したハッシュ格納が提供されている．

- スキャン法の動作

  スキャン法は，データベース中の行を同定するアクセス法であり，逐次検索あるいは線形検索とも呼ばれる．スキャン法は，例えばデータベースに格納する行を順次読み出す際に利用される．図 1.21 の例では，(3，鈴木)，(7，村岡) が連続的に読み出され，最後に (2，中山) が読まれて終了する．

- インデクス法の動作

  インデクス法は，データベース中の行群をあるキー値の順序で同定する方法として，利用されている．インデクス法としては，B 木インデクスが広く実用に供されている [7]．ある表を構成する適当な列にインデクスを定義することができる．そのインデクスは，2 つの項目から構成される．1 つの項目はキー値であり，もう

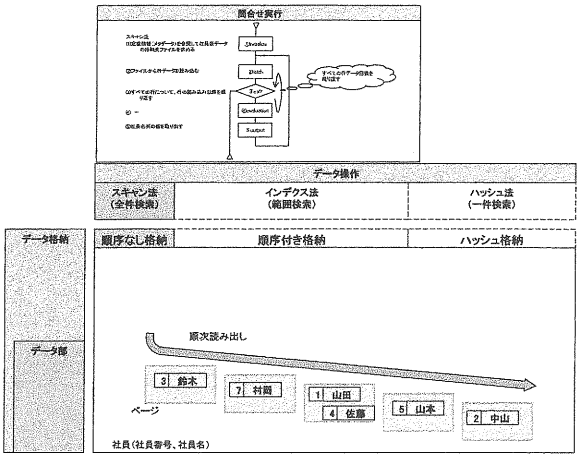


図 1.21: スキャン法

1 つの項目はそのキー値の格納位置を示す．図 1.22 の例では，社員（社員番号，社員名）からなる表の社員番号列にインデクスを付与している．また例では，(5，山本）が読み出され，次に（7，村岡）が読まれて終了する．

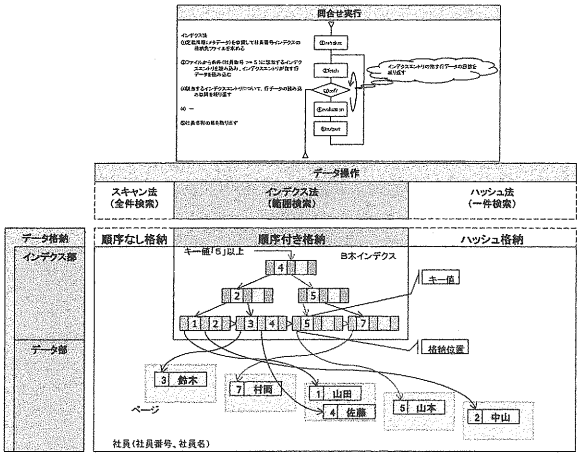


図 1.22: インデクス法

- ハッシュ法の動作

  ハッシュ法は，データベースの行群をある順序性でアクセスすることを前提とはしない [11][12][13]．図 1.23 の例では，社員（社員番号，社員名）からなる表の社員番号列にハッシュ格納を対応させている．行の特定の列値（キー値）にハッシュ関数を適用して，行群が格納されているバケットを指し，そのバケット中での該当する行の格納位置を同定する．この場合，キー値として「5」が指定されると，キー値「5」に対するハッシュ関数の評価値が算出されて，該当するバケットからレコードが特定できる．また，ハッシュ関数は，格納するレコード群が特定のバケットに集中しないために均一性とランダム性の性質を有することが望まれる．

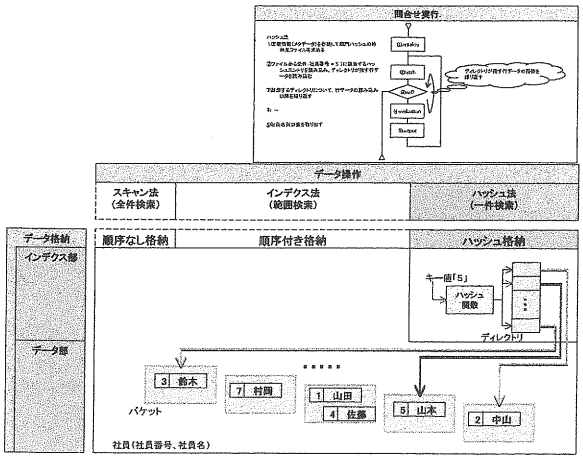


図 1.23: ハッシュ法

この問合せ実行では，文字，数値のデータを検索するために広く利用されている，B木インデクスをインデクス法の事例として示している．一方で，マルチメディアデータを効率よく検索するためには，各メディアの特性に適合するインデクスを利用することが必要であるが，このようなインデクスは問合せ実行を拡張し，実装することが要求されている．

トランザクション管理

トランザクション管理とは，データベースの整合性を保つためにデータベースの読み書きの単位であるトランザクションを管理する機能である [60]．具体的には，同時実行制御及び障害回復からなる．図 1.24 は，トランザクション管理の構成を示す．

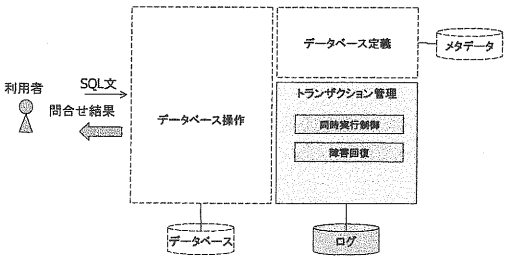


図 1.24: トランザクション管理

同時実行制御は，並行に複数のトランザクションが同一のデータをアクセスする場合に不都合が起きないように制御する．

また，障害回復は，トランザクションを回復の単位とし，どのような障害が発生しようともデータベースの内容が失われないように制御する．そのために，データベースの更新情報を記録しているログを用いる [3].

データベースの回復に利用するログとは，データベースの読み書き動作などの作用を記録する時系列順に記録するデータである．ログを利用してデータベースの回復処理を行うので，二次記憶装置などの安定記憶に永続的に記録される．多くの場合，二重化され格納される．ログは，以下の構成要素からなる．

- ログ通番 (log sequence number)
- トランザクション ID(transaction ID)
- スキーマ情報（表 ID，行 ID，列 ID など）
- 更新値情報（旧値，新値など）

図 1.25 は，並行するトランザクション T1 及び T2 が実行された場合，取得されるログの一例を示す．

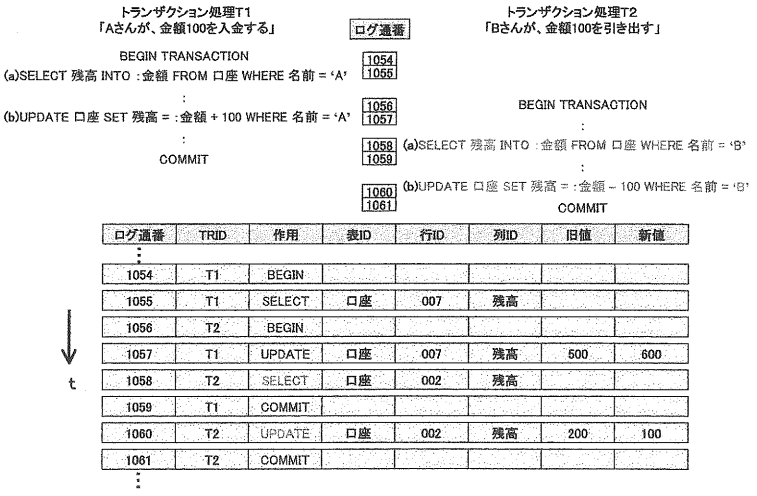


図 1.25: ログ構造

1. ACID 特性

   データベースの整合性を保つためには，4 つの処理特性，即ち ACID 特性を有する必要がある [60]．ここで，A（Atomicity）は原子性，C（Consistency）は一貫性，I（Isolation）は隔離性，D（Durability）は耐久性を示す．

   まず，原子性は，複数の処理が一つの処理として分けることができない単位を示している．即ち，引き落としと振り込みが一つの処理として実行された結果がデータベースに反映されるのか，あるいは全く結果が反映されないようにするかのいずれかであ

り，中途半端な処理状態を残さない処理特性を指している．即ち，トランザクションは二者択一の実行の単位になる．図 1.26 の例では，銀行で A さんの口座から B さんの口座に金額 100 を送金することを一つの操作として完結していることを示する．トランザクションの処理結果を反映することを COMMIT 処理と呼び，またトランザクションの処理結果を反映しないことを ROLLBACK 処理と呼ぶ．

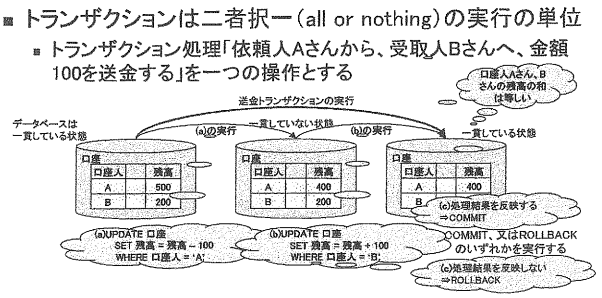


図 1.26: 原子性の例

次に，一貫性は，例えば銀行の残高の総和は不変というデータベースの一貫性，この場合は送金処理の前後で残高の総和が 700 に保たれているような処理特性を指している．即ち，トランザクションはデータベースの一貫性を維持する単位になる．図 1.27 の例では，口座人 A さんの残高 500 と口座人 B さんの残高 200 の総和 700 は，送金トランザクションを実行した後でも，口座人 A さんの残高 400 と口座人 B さんの残高 300 の総和 700 と保たれている．

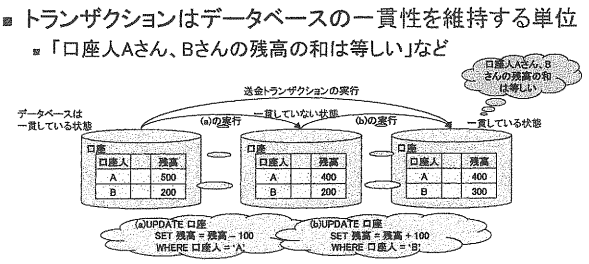


図 1.27: 一貫性の例

さらに，図 1.28 に示す隔離性（isolation）の例では，A さんが口座に 100 入金するために，残高 500 を読み込み，振り込み金額 100 を加えて残高を 600 に変更する．一方で，入金する前に残高 500 を読み出し，引き出し金額 100 を差し引き残高を 400 に変

更してしまうと，A さんの口座から 100 入消えてしまう，という状況が発生している．このような状況は決して発生してはならない．即ち，隔離性は，各々のトランザクション同士で影響を受けずに，あたかもそれらのトランザクションが独立に処理されたのと同じ結果になる処理特性を指している．

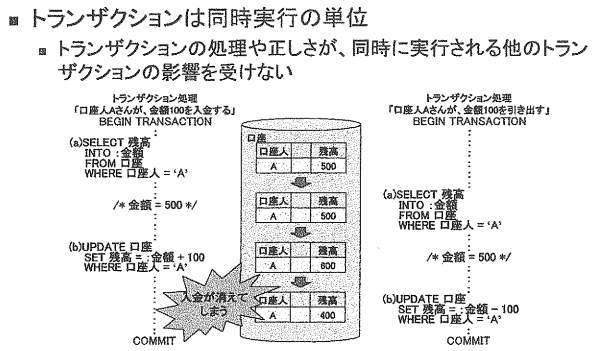


図 1.28: 隔離性の例

最後に，図 1.29 に示す耐久性（durability）の例では，A さんの口座から引き落とし B さんの口座に振り込みを行う前に，何かの障害で処理が中断してしまい，A さんの口座の残高が不明となるような状況が発生している．この場合，B さんの口座への振り込みができなかったので，最初の A さんの口座の引き落としもされなかったことにしておき，元々の送金処理自体が全くなかったことにする必要がある．即ち，耐久性は，どのような障害が発生しようともデータベースの内容が失われるようなことが発生しない処理特性を指しており，またトランザクションは障害回復の単位になる．

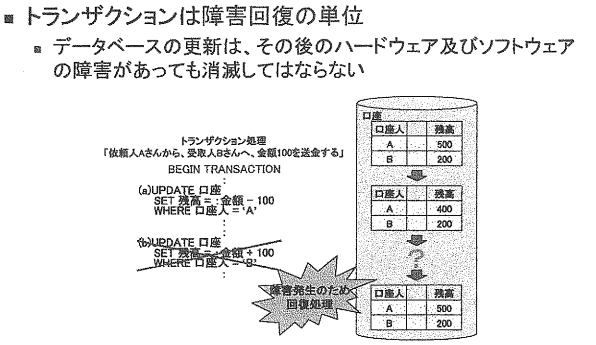


図 1.29: 耐久性の例

2. 同時実行制御

同時実行制御の必要性について示す．

一般的に，二次記憶装置への読み書きなどの入出力動作を伴うトランザクションが入出力待ちの時に CPU を有効に利用するために，複数のトランザクションは並行に実行されている．しかし，各トランザクションで無秩序にデータベースを読み書きすることで種々の異状が発生する．図 1.30 は，このような異状が発生する同時実行制御の例を示す．

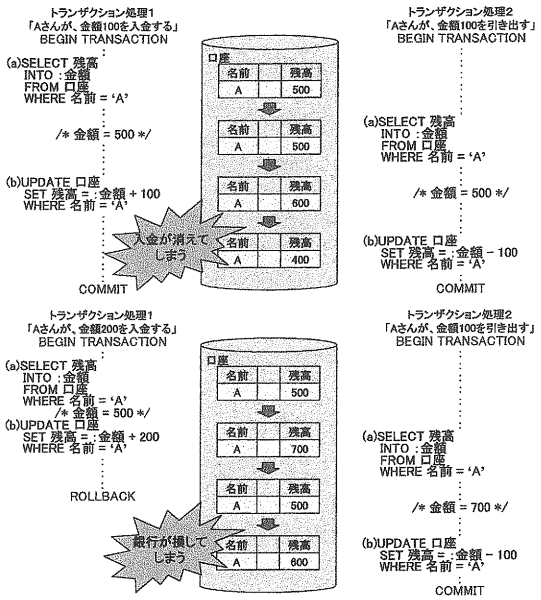


図 1.30: 同時実行制御の例

まず，A さんの入金処理が完了する前に処理途中の残高 500 を読み出し，引き出し金額 100 を差し引き残高を 400 に変更してしまうと，A さんの口座から入金 100 が消えてしまうという状況が発生している．

また，A さんの入金処理が途中で取り止められ残高を 500 に戻しているのにもかかわらず，処理途中の残高 700 を読み出してしまうために，銀行が損をする状況も発生する．いずれの事例も，本来は入金処理を実行してから引き出し処理を実行するように直列な順序で実行されるとデータベースの一貫性は保たれる．同時実行制御は，各トランザクションを並行に処理されたとしても，その結果が，直列に実行された結果と同じとなるように制御する．

この同時実行制御を実現するためにロック方式が実装されている [3]．図 1.31 は，ロック方式による同時実行制御の実装例を示す．

A さんの入金処理が完了する前に処理途中の残高 500 を読み出し，引き出し金額 100 を差し引き残高を 400 に変更してしまうと，A さんの口座から入金 100 が消えてしまうという状況が発生している．この状況は，同じ A さんの口座を無秩序に読み書きすることが原因である．この問題を解決するためには，同じ A さんの口座をアクセスするトランザクション間で何らかの順序制御が必要となる．この順序制御の手法がロック方式である．

ロック方式では，A さんの口座の読み書き前にロックし，既に他のトランザクションでロックされていればロックができずに，アンロックされるまで待つことになる．こ

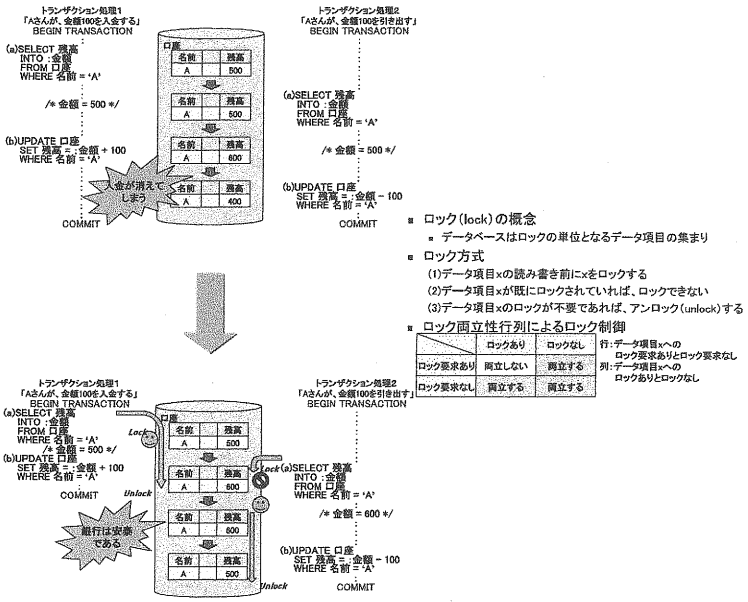


図 1.31: 同時実行制御の実装

れは，ロック両立性行列によって表現できる．

データベースの一貫性を保つために，入金処理を実行してから引き出し処理を実行するように，あたかも直列な順序で実行させるような制御を実現している．先行する入金処理が確保するロックを COMMIT 処理で解放するまで，競合する引き出し処理を待たせている．これによって，データベースの一貫性は保たれている．

3. 障害回復

障害回復の基本概念として，障害種別は 3 つに分類できる．

まず，トランザクション障害は主にアプリケーションが異常終了するような場合である．システム障害は，DBMS や OS の障害，あるいはハードウェア障害で主記憶上のデータが保証できないような場合である．

メディア障害は，二次記憶装置が破壊した場合である．トランザクションが障害に遭った際には，COMMIT 処理を実行し完遂するか，ROLLBACK 処理を実行し直前の一貫性のある状態に戻す必要がある．前者を REDO，後者を UNDO と呼ぶ．

また，このようなトランザクションの耐久性を実現するために，3 つの障害種別に対して，障害回復は 4 つに分類できる．図 1.32 は，障害種別の中でトランザクション障害及びメディア障害に対する障害回復について示す．

トランザクション障害が発生した場合，書き込みによる変更はすべて取り消す必要がある．メディア障害が発生した場合，データベースは修復が不可能になる．このような状況で一貫性のあるデータベースを再生するためには，事前に取得されているデータベースの保管用アーカイブを基にして，それ以後メディア障害が発生に至る間に COMMIT 処理したすべてのトランザクションを REDO する．

次に，システム障害が発生した場合の障害回復について，図 1.33 に示す．

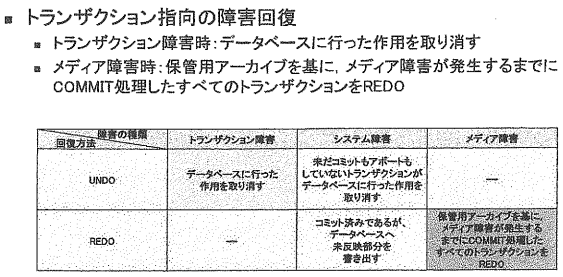


図 1.32: トランザクション障害及びメディア障害に対する障害回復

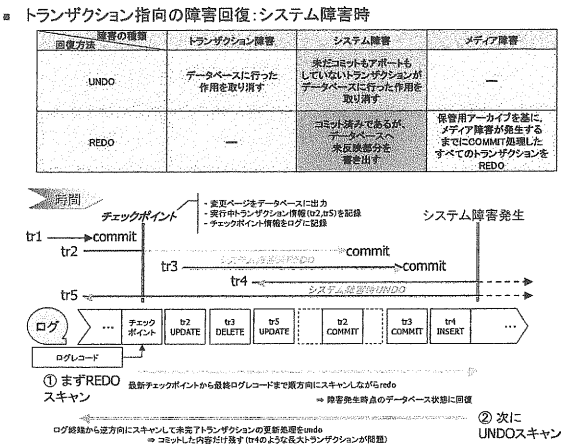


図 1.33: システム障害に対する障害回復

システム障害が発生した時点で COMMIT 処理が実行されていた場合，データベースへの書き込みがすべて行われていない可能性があるので，そのようなトランザクションは DBMS が再起動した際に REDO する．また，システム障害が発生していた時点で COMMIT 処理が実行されていない場合，そのようなトランザクションは DBMS が再起動した際に UNDO する．

このような障害回復は，トランザクションの開始終了，またトランザクションがデータベースに対して行った読み書きの一連の作用などのすべての情報を時系列に記録することで実現できる．この情報をログと呼ぶ．

ログ方式による障害回復の機構の根幹をなす，WAL(write ahead log) プロトコル [3] について図 1.34 を用いて説明する．

まず，一般のアプリケーションプログラムがファイルを利用する構成について示す．通常は，アプリケーションプログラムがファイルに書き出すタイミングと二次記憶装置に書き出すタイミングとは同期せずに，OS がシステムの効率化を図り遅延書きを実行している．この場合，システム障害が発生すると，ファイルに書き込んだ内容が失

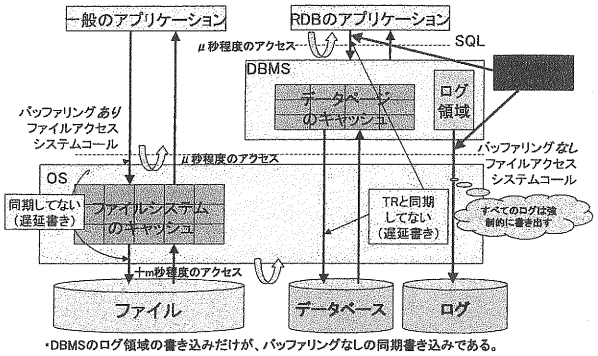


図 1.34: WAL プロトコル

われてしまう．

一方，DBMS の構成について示す．DBMS は二次記憶装置上にデータベース及びログを記録する．また，DBMS は主記憶装置上にデータベースの写しであるキャッシュ，データベースへの操作を記録するログ領域を管理している．データベースへの更新が行われると，更新したデータを持つデータベースのキャッシュに書き込みを行うのと並行して，ログ領域にもデータベースへの操作状況を反映する．この場合，ログの二次記憶装置への書き込み動作がトランザクション処理と同期しているが，データベースのキャッシュの二次記憶装置への書き込みはトランザクション処理とは同期していない．これによって，データベースの障害回復の手掛かりとなるログ情報が失われることを防いでいる．

そこで，確実にデータベースが回復可能とするため，WAL プロトコルが採用されている．WAL プロトコルは，下記の手順からなる

- トランザクションはコミットする前にそのすべてのデータベース更新のログを書き出す
- ログは安定記憶に強制的に（同期）書き込む

さらに，障害回復を効率良く行う手法として，図 1.35 に示すチェックポイント法 [3] も併用されている．チェックポイント時点では，実行中のトランザクションで書き込まれた変更ページをデータベースに書き出し，チェックポイント情報をログに記録する．これによって，チェックポイント以前に終了しているトランザクションについては，DBMS が再起動した際に回復処理を行う必要がなくなる利点がある．

チェックポイント法とは，二次記憶装置上のデータベースへ更新操作を反映した整合性のある状態を指す．チェックポイント法の動作は，以下である．

- 実行中のトランザクションを一時停止
- データベースのキャッシュをデータベースに強制的に書き出し
- チェックポイントのログレコードをログに書き出し

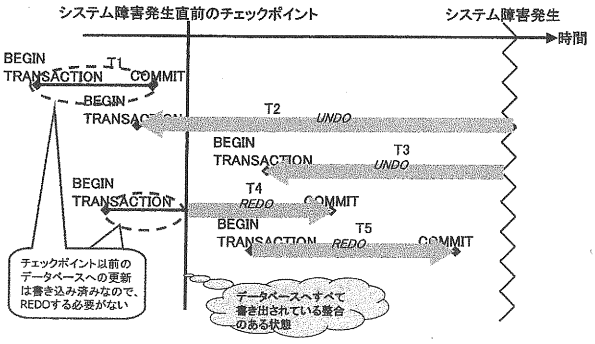


図 1.35: チェックポイント法

- 停止中のトランザクションを再開

チェックポイントとシステム障害が発生する時間の関係から，チェックポイント法の効果を示す．

5 つの並行するトランザクション処理 T1〜T5 があったとすると，チェクポイントが取得されていると，データベースの更新ページが二次記憶装置上のデータベースに強制的に書き出されているので，チェックポイント取得以前にコミット処理が完了したトランザクション T1 は障害回復の対象にする必要がない．また，チェックポイント取得時点でトランザクションが実行中で，システム障害が発生する以前にコミット処理が完了したトランザクション T4 については，チェックポイント以前の作用については REDO する必要がない．

一方で，もしチェックポイントが取得されないと，T1 あるいは T4 のようなトランザクションもシステム障害から回復対象とする範囲として処理する必要があり，回復処理に要する時間が長くなることが考えられる．

1.2.2. 分散データベースシステム

データベース管理システムはひとつのコンピュータ上で一極に集中するシステムであるので，その DBMS が管理するデータベースシステムが何らかの災害に見舞われれば壊滅的な被害を受けることになる．また，ユーザの要求も一極に集中するので，負荷が処理能力の限界を超えてしまうと，データベースシステムの機能が果たせなくなる．さらに，データベースシステムで支援できる利用者数，データ量にも限界があるので，DBMS のスケーラビリティを達成するにも限界がある．

そこで，災害のリスクを分散し，また負荷も分散させ，さらにスケーラビリティを達成するために，分散データベースシステムの必要性が認識された．図 1.36 は，分散データベースシステムの要件を示す．

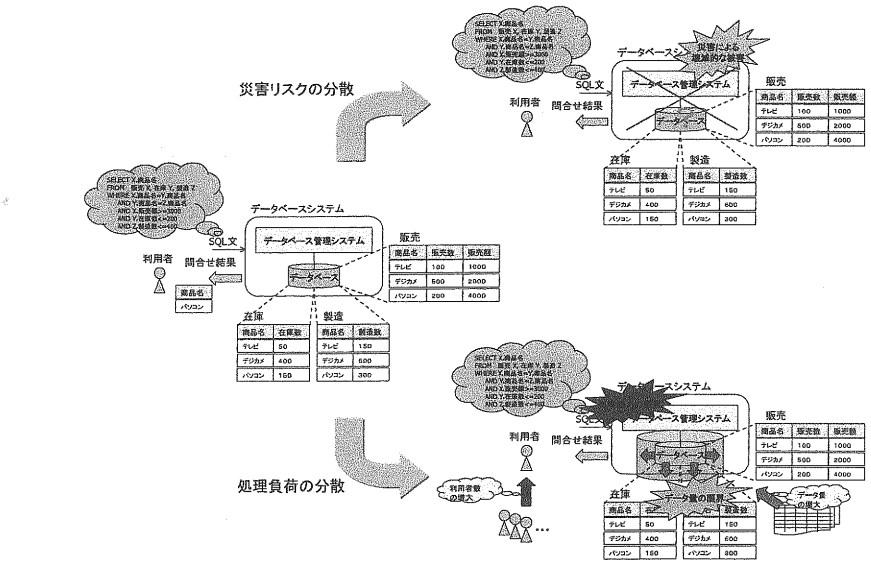


図 1.36: 分散データベースシステムの要件

分散データベースシステムの構成

この分散データベースシステムは，ひとつのコンピュータ上で相異なる DBMS が動作する場合も含めて，地理的にあるいは論理的に分散している複数の DBMS がネットワークを介して，ユーザから単一の DBMS であるかのようにみえる DBMS から構成されている．

図 1.37 の例では，東京，名古屋，大阪と地理的に異なる 3 つの部門で DBMS が動作しており，これらがネットワークで接続されて，東京営業部門では販売 (商品名，販売数，販売額)，名古屋物流部門では在庫 (商品名，在庫数)，大阪製造部門では製造 (商品名，製造数) が管理されている．この分散データベースシステムに対して，「販売額が 3000 以上，在庫数が 200 以下，製造数が 400 以下の商品名を求める」ために，例のような問合せが発行される．

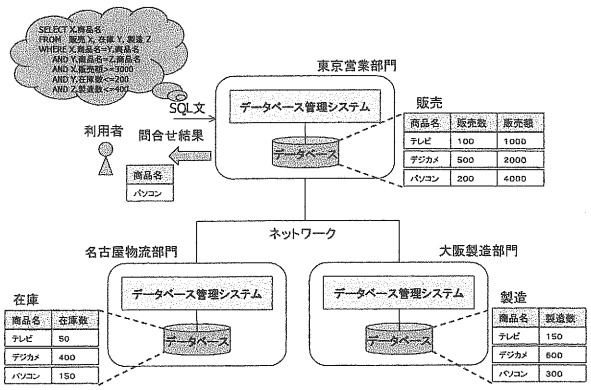


図 1.37: 分散データベースシステムの例

並列処理システムでの実装

ここでは，並列処理によるスケーラビリティを達成するためのDBMSがどのようにスレッドを使用しているか，そのスレッドを持つプロセスがどのように実現されているか，どのプロセスがどのプログラムに対応するのかを述べる．DBMSがUAPからの要求を受けてトランザクション処理を行っている場合のスレッド，プロセス，プログラムの構成を，図1.38に示す．DBMSでは，通常のトランザクション処理はすべてスレッドによって行われる．

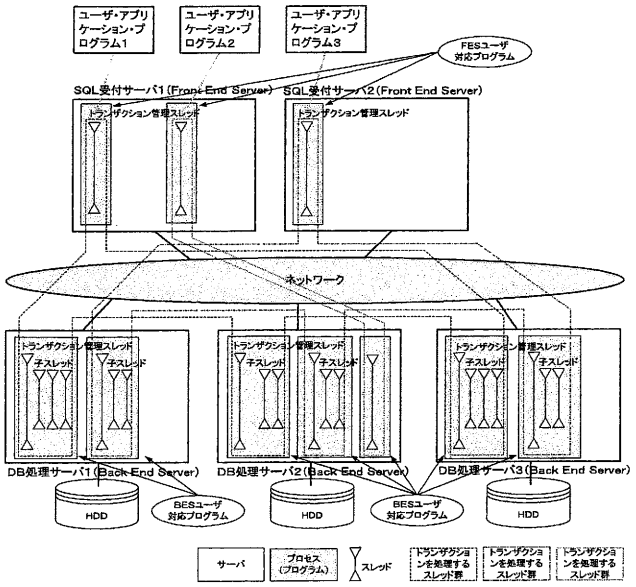


図 1.38: スレッド構造

UAPのトランザクションに1対1に対応してSQL要求受付サーバにスレッドが割り当てられる．このスレッドはUAPのトランザクションを管理するスレッドで，トランザクション管理スレッドと呼ばれる．このトランザクション管理スレッドに対応してDB処理サーバにトランザクション管理スレッドが割り当てられる．DB処理サーバのトランザクション管理スレッドは，SQL要求受付サーバのトランザクション管理スレッドに対して各DB処理サーバに最大1個が割り当てられる．どのDB処理サーバにトランザクション管理スレッドが割り当てられるかはUAPの発行するSQL文に依存する．

例えば，検索対象の表が3つのDB処理サーバに分割されていて，検索条件を満足する行が3つのDB処理サーバ全てに存在する可能性がある場合，3つのDB処理サーバそれぞれにトランザクション管理スレッドが割り当てられる．検索対象の表が3つのDB処理サーバに分割されていてるが，検索条件を満足する行が存在する可能性が1つのDB処理サーバだけに絞られる場合，該当するDB処理サーバだけにトランザクション管理スレッドが割り当てられる．また，INSERT文では1行を挿入すべきDB処理サーバが一意に定まるので，該当するDB処理サーバだけにトランザクション管理スレッドが割り当てられる．DB処理サーバのトランザクション管理スレッドはSQL文の処理のために子スレッドを生成して処理を実行する場合がある．それは，以下の場合である．

- 最適化処理がSQL文を並列に実行することができる複数の処理単位に分割できると判断した

- UAP がカーソルをオープンした

UAP が複数のカーソルを同時にオープンしている場合は，少なくともその数の子スレッドが存在することになる．逆にこれらの場合に当てはまらない時は，子スレッドは生成しない．例えば，INSERT 文では子スレッドを生成しない．トランザクション管理スレッドとプロセスは 1 対 1 に対応しているが，1 プロセスにトランザクション管理スレッドを複数個割り当てることも可能である．

1.3. 研究の経過

図 1.39 は，本論文での研究開発の対象分野を示す．以下では，拡張可能型データベース管理システムの研究開発に関する経緯について概観する．

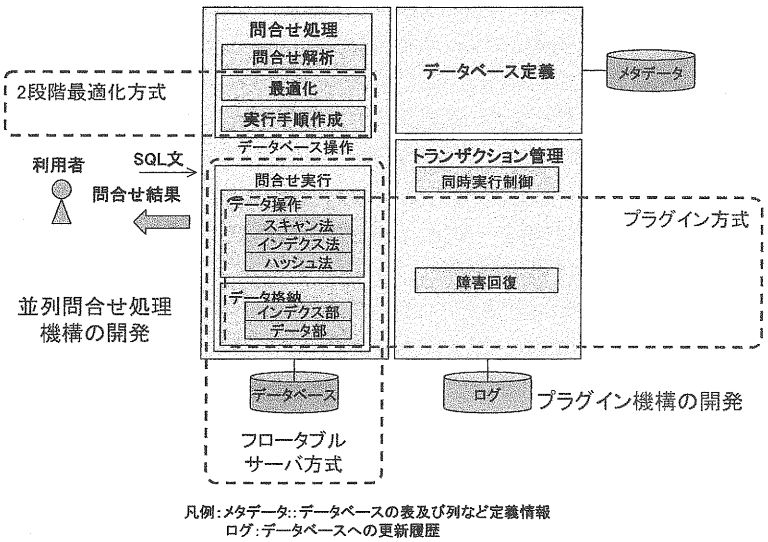


図 1.39: 研究開発の対象分野

1.3.1. 大規模データ処理向けスケーラブルアーキテクチャにおける並列問合せ処理機構の開発

実世界においてデータベース化が望まれるデータ量は増加する一方である．大量なデータに対して DBMS(Database Management System) での処理性能はデータ量に比例しない応答時間が望まれている．しかし，大量のデータを逐次的に処理していたのでは限界がある．そこで，データベース処理の並列化が必要とされている．最近のハードウェア基盤の進展によって，並列システムが構築できる環境が揃ってきていると考えられる．

データ処理に並列処理を適用する実践は実応用システムで行われている．例えば，MapReduce が大規模データ解析に活用されつつある [45][46]．利用者が key/value ペアを処理する Map と Reduce と呼ぶ 2 つの関数によって，大規模なデータ処理を記述する．具体的には，分

散ファイルからの入力を，key にハッシュ関数を適用して分割し，各ノードに格納する Map と，それらを対象にデータ処理を行う Reduce からなる [15][21]．しかし，MapReduce で提案されているデータ処理フレームワークは，SQL を対象にする並列処理システムとして既に並列 DBMS[40][41][42] に内在するものであり，著者は負荷平準化 [16][17][18][19][20] 及び分割データ操作 [22] に関する提案を行っている．

SQL ではデータ操作が入力，出力ともに同じく表と呼ばれる形式で閉じた演算で表現され，多種類のデータ操作をパイプライン的につなぎ，各々の操作を並列に実行するパイプライン並列化が可能である．また，表は行と呼ぶデータの集まりであるため，データを均等の量に分割して，1 つの操作を各々分割されたデータに対して並列に実行するデータ並列化が容易に実現される．さらに，SQL に対する並列化技術が確立されてきた．

この論文で提案する並列化技術は，1.2.2 節に述べるようにマルチプロセス，マルチスレッドによるソフトウェアによる記述レベルである [25]．具体的には，並列システムにおけるハードウェアの Shared-Nothing 方式を活用するキーレンジ分割，ハッシュ分割などデータ分割方式は，並列データベースシステムに適合する．この並列処理システムでは，SQL で表への参照手段を記述しないため，SQL の問合せ処理，中でも最適化処理で並列処理に向く参照手段を作成することで，並列データベースシステムに適用することができる．

このような背景から，並列データベースシステムの開発の根幹をなす問合せ処理の並列化方式が開発された．SQL 文を入力し，その記述の解析処理によって並列性を導き，具体的に並列に実行するための処理手順の表現方法，問合せ変換方法，並列実行環境など並列化に伴う技術からなる．本論文の目的は，SQL の問合せ処理に，パイプライン並列及びデータ並列の考え方を適用した問合せ処理機構について示すことである．

具体的には，負荷平準化を目的とするフロータブルサーバ方式は，DB を格納していないプロセサに処理の一部を分担させ，ジョイン処理にデータ並列及びパイプライン並列を適用することで，処理時間及び効果を評価する．また，フロータブルサーバ方式を好適に活用するために，変数を含む SQL 文を解析処理する段階と，実際にその SQL の処理手順を実行する問合せの実行処理時に変数値に従い最適化する段階からなる，2 段階の最適化方式 [14] を評価した．この 2 段階からなる最適化方式は，部分計算法 [23][24] の「関数 f が 2 つのパラメタ k，u を持ち，既知のパラメタ k(known parameter) と未知のパラメタ u(Unknown parameter) とした場合，関数 f の計算中で k について実行可能な部分を実行し，u の値が与えられないと実行不可能な部分はそのまま残しておき，関数 f の計算を進めておく」を，SQL 文中に含まれる変数を未知のパラメタ u，定数部分を既知のパラメタ k として，SQL 文の最適化方式に応用したものである．

1.3.2. マルチメディアデータベースシステムにおけるプラグイン機構の開発

インターネットに代表されるネットワークコンピューティングにおいては，利用者に親しみを与える情報提供形態が重要である．そのため，画像や音声，動画といったマルチメディアデータを取り込んだ情報システムの構築が必要不可欠となりつつある．

マルチメディア情報は，データ量が大容量なだけではなく，メディア操作や格納形式が多様化かつ順次拡張していくことに特徴がある．これに対応するマルチメディアデータ処理系として，オブジェクト・リレーショナルデータベース (ORDB) が期待されている．ORDB は，RDB モデルをベースとし，オブジェクト指向モデルを取り入れた DB である．一方，国際標準化機構 ISO では，データベース言語 SQL の標準仕様を策定している．現時点では，

SQL2008 が最新であり，利用者定義型（以下では，UDT(User-Defined Type) と呼ぶ），型継承などのオブジェクト指向モデルを取り入れた拡張が規定されている．これにより，複雑な構造を持つマルチメディアデータをデータベースに容易に取り込めるようにする枠組みとして活用が可能である．

ORDB では，データに対する操作機能を利用者定義のルーチンによって提供する．ここではプラグイン (plug-in) と呼ばれる機構によって，ルーチンを実装するモジュール (以下では，プラグインモジュールと呼ぶ) を ORDBMS に組み込む．これにより，DBMS 自体を変更することなく，各種のメディアデータを扱う先進的な機能を実装するプラグインモジュールを，容易に DB システムに取り込むことができる．また，このプラグインモジュールは DBMS と独立して開発することができる．即ち，この方式によれば，例えば構造化文書の構造指定検索などの高度な機能を持ち，また n-gram 方式による高速な全文検索機能をプラグインとして開発し，ORDB に組込むことで，拡張可能なアーキテクチャが実現できる．さらに，文書に限らず画像や地図データなどを管理する機能を持つプラグインモジュール群を組込めば，多様にシステムを拡張することもできるようになる．

そのために，ORDB にプラグイン機構を開発し，また Shared Nothing 方式の並列 DB に適用することで，マルチメディア情報管理システムの基盤となる ORDBMS を開発した．本論文の研究範囲は，プラグイン機構である．具体的には，プラグイン機構に関して，プラグイン IDL（Interface Definition Language）と PPI（Plug-in Programming Interface）を開発した．

1.3.3. XML 応用システムにおける Web フォーム基盤アーキテクチャの開発

Web アプリケーションで構築される業務システムは，顧客からの要望に合わせて，短い周期で継続的に拡張や改善が行われている．表示層には Web ブラウザを利用し，HTTP や HTML などを解釈できればよいため，PC だけではなく PDA や携帯電話などでもオンライン情報の閲覧，オンラインショッピング，銀行の振込みなどのサービスが利用できるようになってきている．また，アプリケーション層には JSP（JavaServer Pages），Servlet，EJB（Enterprise JavaBeans）などの JavaTM 関連技術を利用し，Web フレームワーク（MVC モデル）及びコンポーネント部品の機能配置及び記述方法に基き，各々の役割分担に従うことでシステム構築が可能である． したがって，表示層及びアプリケーション層での再利用性が高く，顧客の要望に素早く対応できる Web アプリケーション開発への期待が益々高まってきている．

一方，インターネット及びイントラネットを利用した Web システムでは入力画面及び出力帳票開発の容易化，入出力データの XML 基盤への対応などが要望されている．政府主導による IT 化推進，自治体や民間企業の電子申請システム，ワークフローシステムなどで Web 化が急速に進められている．このように，電子帳票，電子フォーム市場での Web 化の要望が高まってきている．

これらの背景を踏まえて，Web アプリケーションでの電子フォームシステム・アーキテクチャのコンポーネント，機能構成，及びコンポーネント間インタフェースを開発し，また上記のアーキテクチャを Web フォーム基盤ミドルウェアとして実現した．具体的には，Web フォーム基盤アーキテクチャとして，Web アプリケーションフレームワークを支える機能配置方法，記述方法，及びデータ連携・交換方法を開発した．ここでは，携帯電話などのモバイル機器を始めとして，RFID などユビキタス基盤を支える情報機器への対応も視野に幅

広く適用が期待されている Web フォーム基盤のアーキテクチャを提案し，それらを実装し様々な応用事例に適用した結果を示す．

1.4. 本論文の構成

本論文では，データベースシステム適用分野の拡大及び稼動形態の多様化によってもたらされる利用者からの要求，及び業務システムでも，画像，音声，動画などマルチメディアデータを取り込んだ情報システムの構築が不可欠となりつつある状況を踏まえ，それらを取り巻く動向から，適用分野を明示する．これらのデータ量が大容量なだけではなく，メディア操作及び格納形式が多様化かつ順次拡張される課題に対処するアプローチとして，並列問合せ処理及びオブジェクト指向概念に基づく拡張可能型データベース管理システムについて示す．

第 2 章では，データ量の増大に対して，並列データベースシステムの開発の根幹をなす問合せ処理の並列化方式を示す．SQL 文を入力し，その記述の解析処理によって並列性を導き，具体的に並列に実行するための処理手順の表現方法，問合せ変換方法，並列実行環境など並列化に伴う技術からなる．本論文の目的は，SQL の問合せ処理に，パイプライン並列及びデータ並列の考え方を適用した問合せ処理機構について示すことである．具体的には，負荷平準化を目的とするフロータブルサーバ方式の処理時間及び効果を評価する．また，フロータブルサーバ方式を好適に活用するために，変数を含む SQL 文を解析処理する段階と，実際にその SQL の処理手順を実行する問合せの実行処理時に変数値に従い最適化する段階からなる，2 段階の最適化方式を評価した．

第 3 章では，データの質の変化に対して，提案する ORDBMS(オブジェクト指向リレーショナルデータベース管理システム) では，それぞれのマルチメディア情報に対して，検索及び格納などの操作に対応するライブラリ群を，自由に追加，登録できるプラグイン機構を示す．プラグインとしては，SGML 文書，文字列，XML への構造検索を始めとして，類似画像検索，空間検索を実装した．

第 4 章では，XML データベースの応用事例として，電子申請システム，ワークフローシステムなどで Web 化が進展している状況を踏まえ，Web フォーム基盤のアーキテクチャとして機能配置方法，記述方法，及びデータ連携・交換方法を提案し，それを実装し電子申請システム，ワークフローシステムに適用した事例を示す．

第 5 章では，本研究開発の成果及び今後の課題について纏める．

第2章　並列問合せ処理機構の開発

並列データベースシステムの開発の根幹をなす問合せ処理の並列化方式を示す．SQL 文を入力し，その記述の解析処理によって並列性を導き，具体的に並列に実行するための処理手順の表現方法，問合せ変換方法，並列実行環境など並列化に伴う技術からなる．本論文の目的は，SQL の問合せ処理に，パイプライン並列及びデータ並列の考え方を適用した問合せ処理機構について示すことである．

具体的には，負荷平準化を目的とするフロータブルサーバ方式の処理時間及び効果を評価する．また，フロータブルサーバ方式を好適に活用するために，変数を含む SQL 文を解析処理する段階と，実際にその SQL の処理手順を実行する問合せの実行処理時に変数値に従い最適化する段階からなる，2 段階の最適化方式を評価した．

2.1.　はじめに

実世界においてデータベース化が望まれるデータ量は増加する一方である．大量なデータに対して DBMS(Database Management System) での処理性能はデータ量に比例しない応答時間が望まれている．しかし，大量のデータを逐次的に処理していたのでは限界がある．そこで，データベース処理の並列化が必要とされている．

最近のハードウェア基盤の進展によって，並列システムが構築できる環境が揃ってきていると考えられる．データ処理に並列処理を適用する実践は実応用システムで行われている．中でも SQL を対象にする並列処理システムは様々なアプローチが行われてきている [40][41][42]．SQL ではデータ操作が入力，出力ともに同じく表と呼ばれる形式で閉じた演算で表現され，多種類のデータ操作をパイプライン的につなぎ，各々の操作を並列に実行するパイプライン並列化が可能である．また，表は行と呼ぶデータの集まりであるため，データを均等の量に分割して，1 つの操作を各々分割されたデータに対して並列に実行するデータ並列化が容易に実現される．さらに，SQL に対する並列化技術が確立されてきた．具体的には，並列システムにおけるハードウェアの Shared-Nothing 方式を活用するキーレンジ分割，ハッシュ分割などデータ分割方式は，並列データベースシステムに適合する．この並列処理システムでは，SQL で表への参照手段を記述しないため，SQL の問合せ処理，中でも最適化処理で並列処理に向く参照手段を作成することで，並列データベースシステムに適用することができる．

このような背景から，並列データベースシステムの開発の根幹をなす問合せ処理の並列化方式が開発された．SQL 文を入力し，その記述の解析処理によって並列性を導き，具体的に並列に実行するための処理手順の表現方法，問合せ変換方法，並列実行環境など並列化に伴う技術からなる．本論文の目的は，SQL の問合せ処理に，パイプライン並列及びデータ並列の考え方を適用した問合せ処理機構について示すことである．

具体的には，負荷平準化を目的とするフロータブルサーバ方式は，DB を格納していないプロセサに処理の一部を分担させ，ジョイン処理にデータ並列及びパイプライン並列を適用

することで，処理時間及び効果を評価する．また，フロータブルサーバ方式を好適に活用するために，変数を含むSQL文を解析処理する段階と，実際にそのSQLの処理手順を実行する問合せの実行処理時に変数値に従い最適化する段階からなる，2段階の最適化方式を評価した．

以下では，2.2節で課題について述べ，2.3節で並列問合せ処理機構のアプローチについて解説し，2.4節で具体的な並列問合せ処理機構の実装による適用評価を示し，2.5節で関連研究を述べ，2.6節で纏める．

2.2. 課題

2.2.1. 処理負荷の平準化

並列データベースシステムにおいて，並列処理のDBMSのアーキテクチャは3つの方式[1]に分類できる [39]．

本論文の並列データベースシステムは，プロセサ数及びHDD数が多く，かつ大規模なDBMSで更新処理の並列化を実現し易いので，SN方式のアーキテクチャを採用した．

ただし，SN方式の課題は，特定のデータにアクセスするプロセサが固定されてしまうので，複数あるプロセサの処理負荷の平準化を図るのが難しいことである．この課題を解決するために，負荷平準化を目的とする，フロータブルサーバ方式を実現する [27][28][32]．このフロータブルサーバ方式は，プロセサ間での処理負荷の平準化のために，DBを格納していないプロセサに処理の一部を分担させる方法である．従って，フロータブルサーバ方式の目的は，ジョイン処理やソート処理などの負荷をDBを格納しない専用のサーバ（プロセッサも異なる）に分散させることによってDBをアクセスするサーバの負荷を軽減させることである．

2.2.2. 並列問合せ処理

本論文で提案するDBMSは，フロータブルサーバ方式を好適に活用するために2段階からなるSQLの最適化方式を採用している [30][31]．即ち，UAP(User Application Program)が発行したSQL文を後述するSQL受付サーバが受け取って解析処理[2]する段階と，実際にそのSQLの処理手順を実行する問合せの実行処理時に最適化する段階からなる．UAPで書かれる多くのSQL文には変数が含まれており，これらの変数の値は問合せ実行時に決まるので，SQLをどのような処理手順で実行すれば応答時間を最短にできるかは，この値に依存する．従って，SQL文の問合せ解析時には必ずしも最適な処理方法を選択できない．特に，大規模な表に対するジョイン処理及びソート処理は，処理手順の選択を間違うと大幅に性能が低下することを意味する．しかも，変数の値が決まった問合せの実行処理時点でSQL文を改めて最適化処理を含めて解析処理するのでは，解析処理の負荷が問題となる．2段階の最適化方式は，これらの課題を解決するものである．

具体的には，最適な処理手順を選択するのにコストに基づく方式を採用している．まず，問合せ処理を構成する個々の処理単位の処理時間（コスト）を見積もる．SQL文の解析結

[1]すべてのプロセサが主記憶とHDD(Hard Disk Drive)を共用するSE(Shared-Everything)方式，HDDだけを共用するSD(Shared-Disk)方式，及びプロセサ間で主記憶もHDDも共有しないSN(Shared-Nothing)方式からなる．

[2]SQL文を実行形式あるいはそれに近い処理手順に変換する最適化処理も含まれる．

果と，アクセス対象となる表のデータの分布状態などの統計情報をもとに，問合せの解析時点で全体のコストが最小になるような処理手順候補の組み合わせを選ぶ．これを最適な処理手順の候補群とする．最終的に決定される処理手順の最適性は，コストモデルに実際の情報システムを反映させることができるかどうかに依存している．なお，この方法は，なるべく負荷をかけずに統計情報を精度よく得ることが必要になる．提案DBMSは，データの分布情報などをディクショナリ[3]で一元的に管理しているので，迅速にコストを評価できる．この論文では，問合せ最適化によって生成される処理手順の最適性に影響を与えると考えられる，変数を含む問合せの最適化方式を議論する．また，本論文で提案する最適化方式は実適用されている．

2.3. アプローチ

2.3.1. フロータブルサーバ方式

分散マルチサーバ構成

並列データベースシステムを実現するために，DBMSを複数のソフトウェア機能，即ち用途別のサーバ[4]に分割している．プロセサと主記憶，HDDを1つの単位としたノード[5]に対して，サーバを複数割り当てる．これを分散マルチサーバ構成と呼ぶ．また，サーバ間はRPC(Remote rocedure call)[6]により連携する．他のプロセサのサーバにデータを転送し，リモートで他のプロセサのサーバを起動する．こうしてプロセサ間の処理負荷を調整し，障害発生時の可用性を高めることもできる．

DBMSを構成する代表的なサーバには，以下のものがある．

- DB処理サーバ（Back End Server，BESと呼ぶ）
- SQL受付サーバ（Front End Server，FESと呼ぶ）
- フロータブルサーバ（BESで代用する）
- ディクショナリサーバ（DSと呼ぶ）

各サーバの機能について，図2.1示す．

1. DB処理サーバ(BES)

   DB処理サーバは，各ノードに複数割り当てる．DB処理サーバは，そのノードのHDDに分割して格納してある個々の表（分割表[7]）に1対1で対応する．特定の分割表に対するデータベース処理を専任的に行う．入出力処理だけでなく，データの取り出し及び並べ替えの処理も行う．

[3]データベースの論理的な構造や内部状態を管理する仕組み

[4]ここでいうサーバは，ＤＢＭＳの機能単位．一つのサーバは同じ機能をもつ複数のプロセスからなる．一つのプロセスはさらに複数のスレッドをもつ．

[5]ここでいうノードは，並列コンピュータを構成する単位．プロセサを基本とする．SN方式では，プロセサ，主記憶，HDDの3要素をまとめてノードとする．SD方式では，プロセサと主記憶の組み合わせがノードとなる．SE方式にはノードという概念はない．

[6]遠隔手続き呼び出し．プロセス間で通信するための機構の1つで，呼び出し側のプロセスは，データとその処理内容を呼び出される側のプロセスに転送する．呼び出される側は指示された処理を実行する．

[7]DBMSでは，1つの表（テーブル）を複数のHDDに分割して格納する．分割された個々の表を分割表と呼ぶ．個々の表を分割して複数のHDDに分配する方法として，キーレンジ分割とハッシュ分割の両方に対応する

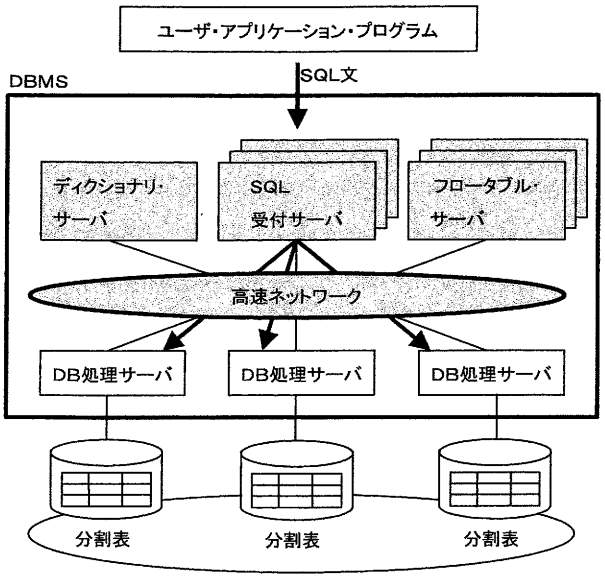


図 2.1: 分散マルチサーバ構成

2. SQL 受付サーバ (FES)

   SQL 受付サーバは，UAP から受け取った SQL 文を解釈して DB 処理サーバに処理手順を割り当てる．処理手順に必要なデータの存在するノードを調べ，そのノードの DB 処理サーバに処理要求を指示する．DB 処理サーバが問合せ処理を終えたら結果を受け取り，UAP に渡す．

3. フロータブルサーバ

   フロータブルサーバは，特定の分割表を割り当てていない DB 処理サーバである．プロセサの処理負荷の平準化を図る場合に利用する．例えば，大量のデータをジョイン処理する場合，プロセサの処理負荷が重いマージ処理などを受け持つ．DB を持たない別のノードのフロータブルサーバにマージ処理を任せることにより，通常の DB 処理サーバの処理負荷を減らすことで，特定のノードに集中した負荷を分散できる．どの処理をフロータブルサーバに割り当てるかは SQL 受付サーバで決める．フロータブルサーバは，データベース処理の負荷を考慮してノードに割り当てておく．システム性能を平準化する目的では，ハッシュ分割によるデータの分散配置を行う．即ち，フロータブルサーバへのデータ配布は，ハッシュ手法に基づく．

4. ディクショナリサーバ

   ディクショナリサーバは，システムおよび分割表などのデータベースの構成情報を管理する．

また，上記以外には，DBMS はトランザクション処理に対応するためのログ機能[8]，トランザクション管理機能，タイマー機能，プロセス管理機能などのシステム管理機能を装備し

[8]データベースに対して，どの時間にどのような処理が行なわれたかを示す記録．障害発生時の回復処理などで利用する．

ている．さらに，システムの対故障性を高めるため，通常は一方の通常系システムだけを使用し，他方の待機系システムは，通常系が故障したときに備える，二重化システムを実現している．

フロータブルサーバを用いた負荷平準化

SQLの実行を単に並列化しても，プロセサの処理負荷が平準化されなければ応答時間は短縮しない．この課題に対して，プロセサの処理負荷の平準化が難しいSN方式は不利とされていたが，フロータブルサーバを導入することで，動的にプロセサの負荷平準化が図れるようにした．即ち，フロータブルサーバに負荷の重い処理を割り当てることで効果を高めた．

具体的には，通常はDB処理サーバが受け持つソート処理とデータのマージ処理を，別のノードのフロータブルサーバに移す．その結果，ノード間の通信が増えるが，そうした負荷よりも全体の応答時間の短縮を優先させた．また，個々のフロータブルサーバの処理負荷についても平準化できるようにしている．DB処理サーバの処理するデータがある程度均一になっていたとしても，DB処理サーバが出力する中間結果のデータ量は偏りが生じる場合がある．SQLで指定された条件式によって，DB処理サーバごとに処理結果が異なるためである．これに対処するために，ハッシュ手法を使ってフロータブルサーバに中間結果のデータを均等に分配する．こうしてフロータブルサーバの処理負荷を均等化する．

フロータブルサーバを導入することの利点は2つである．

- システム性能要件の変更に対するチューニングの容易化

  データの移動を伴うデータベースの再構成をしなくても，フロータブルサーバの追記でチューニングができる．処理すべきデータの増大に対しても，フロータブルサーバを追加することで対応できる．

- トランザクション処理[9]と大量データを一括検索する問合せ処理[10]の共存

  トランザクション処理は，一定の応答時間を保証する必要がある．しかし，ジョイン処理などの高負荷な問合せ処理をトランザクション処理と並行して実行すると，トランザクション処理の応答性能に悪影響を及ぼす可能性がある．この場合，問合せ処理の特性に合わせて，ソート処理に要する時間がデータ入力時間に対して一定の比率になるようにフロータブルサーバの割り当て数を調整し，問合せ実行時に割り当て数だけフロータブルサーバを割り当てる．ただし，ソート処理に要する時間がデータ入力時間を同じにして，問合せ処理時間を最小化する方法も考えられるが，並行してジョイン処理が実行されることを考慮する必要がある．システム内で設定するフロータブルサーバ数から按分することで，フロータブルサーバを専有させない方法も併用することとしている．これにより，処理負荷を別のプロセサに分散できるので，トランザクション処理の性能を一定以上に保証できる．

また，このフロータブルサーバによる負荷の平準化を図るために，割り当て方式を設定できるようにしている．

---

[9]比較的少量のデータを更新し，一定時間以内の応答が求められる処理
[10]大量のデータを一括で全件検索する処理

1. フロータブルサーバ専用方式

   データ入力処理を行う BES に，ソート処理など負荷が掛かる処理が割り当てられず，それらの処理を専用に実行するフロータブルサーバのために BES が割り当てられる．データ入力処理とソート処理など負荷が掛かる処理が，オーバラップすることで，データ並列及びパイプライン並列の効果が期待できる．

2. フロータブルサーバ対象限定方式

   データ入力処理を行う BES に対象を限定して，ソート処理など負荷が掛かる処理を実行するフロータブルサーバが割り当てられる．データ入力処理とソート処理など負荷が掛かる処理が，オーバラップできないので，データ並列の効果は期待できるが，一方でパイプライン並列の効果は期待できない．

3. フロータブルサーバ対象拡大方式

   データ入力処理を行う BES に，ソート処理など負荷が掛かる処理を実行するフロータブルサーバの BES にも対象が拡大されて割り当てられる．

フロータブルサーバ方式によるジョイン性能の改善

フロータブルサーバ方式によるジョイン性能の改善例を示す．DBMS で大量のソート処理を含むジョイン処理に多大な時間を要するので，できるだけジョイン処理を並列化して応答時間を短縮する必要がある．前述したように，ジョイン処理にデータ並列及びパイプライン並列を適用する．図 2.2 にその方法を示す．ジョイン処理は 5 つの処理に分けられる．

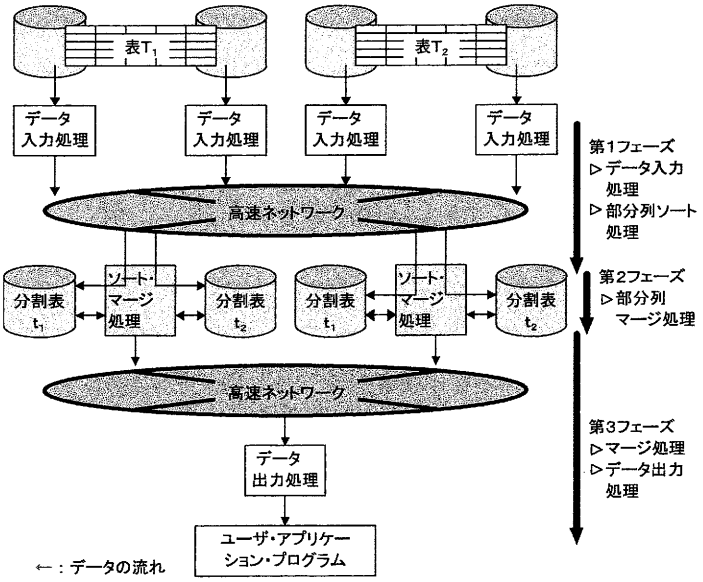


図 2.2: 並列ジョイン処理

- データを HDD から読み出す処理（データ入力処理）

- 部分列ソート処理[11]
- 部分列マージ処理[12]
- データのマージ処理
- データを UAP に送り出す処理（データ出力処理）

この5つの処理にパイプライン並列化を適用する．この結果，ジョイン処理は3つのフェーズで処理が行われる．第1フェーズでは，データ入力処理及び部分列ソート処理の間にパイプライン並列化を適用する．次の第2フェーズでは，第1フェーズの結果で得られた部分列に対してマージ処理を行う．ただし，第2フェーズの前後で一旦パイプライン処理は途切れる．最後の第3フェーズでは，データのマージ処理及びデータ出力処理の間にパイプライン並列化を適用する．

2.3.2. 2段階最適化方式

処理手順の並列化表現

ここでは，処理手順の表現形式について，図 2.3 に示す．

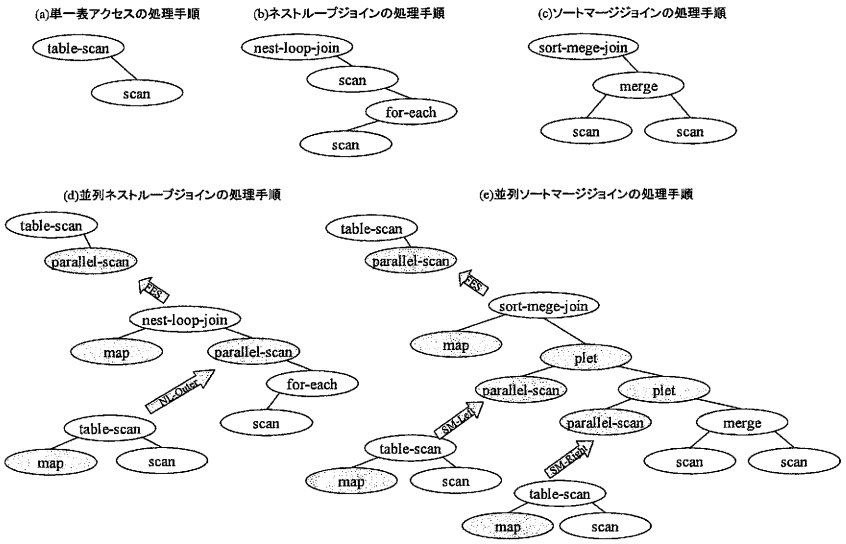


図 2.3: 処理手順の並列化

まず，逐次処理のための処理手順の表現形式を示す．

(a) は単一表へのアクセスの処理手順である．scan ノードには，インデクススキャンを利用するのか，テーブルスキャンを利用するのか指定されている．

[11]ハッシュ分割された表に対してソート処理を行うこと．

[12]部分列ソート処理によって生成した複数の部分列に対してマージ処理を行うこと．最終的に一つの部分表を作成する．

(b), (c) はいずれもジョイン処理の処理手順である．(b) はネストループジョインの処理手順であり，foreach ノードより上に位置する scan ノードが，最初にアクセスされる外側表へのアクセスを示し，下に位置する scan ノードが繰り返しアクセスされる内側表へのアクセスを示す．これらの scan ノードには，インデクススキャンを利用するのか，テーブルスキャンを利用するのか指定されているが，ネストループジョインの処理要件から上に位置する scan ノードには選択条件に出現する列に付与されるインデクスへのインデクススキャン，及び下に位置する scan ノードにはジョイン条件に出現する列に付与されるインデクスへのインデクススキャンが設定されることが想定される．(c) はソートマージジョインの処理手順であり，merge ノードより左右に位置する scan ノードがそれぞれの表へのアクセスを示す．この場合，ソートマージジョインの処理要件からそれぞれの表の scan ノードから出力されるデータはソート済みであることが想定される．

次に，問合せ処理を並列化する上で，並列手順の表現方法を規定する必要がある．(d), (e) はいずれも並列ジョイン処理の処理手順である．

(d) は並列ネストループジョイン (以下では，PNL(Parallel nest-loop join) と略記する) の処理手順である．複数のサーバからのデータを受け取る parallel-scan ノードを導入する．(d) では，外側表からのデータを，PNL 処理の parallel-scan ノードで受け取ることを表現する．また，データ転送先のサーバに分散して配布する記述のために，map ノードを導入する．この map ノードは，外側表からのデータを，PNL 処理の parallel-scan ノードでハッシュ分割して受け取る場合と，PNL 処理の結果のデータを FES の parallel-scan ノードに集約して受け取る場合に表現されている．これらの，parallel-scan ノード及び map ノードを利用するデータ転送処理は，パイプライン的に行われる．

(e) は並列ソートマージジョイン (以下では，PSM(Parallel sort-merge join) と略記する) の処理手順である．この場合，PNL と同様に，parallel-scan ノード及び map ノードを利用するデータ転送処理は，パイプライン的に行われるが，部分列に対するマージ処理で一旦パイプライン処理は途切れる．例えば，集合演算のためのソート処理，及びジョイン処理は複数の表からデータを受け取り，複数の中間結果を作成する必要がある．各表のデータ読み出しは別々のサーバから並列に送られてくるのに対して，plet ノードによって中間結果の作成処理 (図 2.3 では，部分列に対するマージ処理に該当) を記述する．受け取り側の処理を各表毎に並列化するために，各々中間結果の作成処理に対して，3 つの子スレッドが生成され，その中で 2 つの子スレッドは plet ノードの部分列のマージ処理が実行され，また残りの 1 つの子スレッドは merge ノード以下の処理が実行される．

2 段階最適化方式での処理手順候補の作成

SQL 受付サーバが実行する最適化処理の手順を，図 2.4 に示す．

まず，UAP から受け取った SQL 文の問合せ解析処理時に，問合せ処理のコストを評価する．この時点で唯一最適と評価できる処理手順が決まればその処理手順を [30]，それ以外であれば代替案含めて複数の処理手順を作成する [31]．問合せ解析処理時に作成した処理手順は SQL 受付サーバが保存する．SQL 文の問合せ解析処理は最初の 1 回だけでよい．同じ SQL 文を何回も実行するときは，2 回目以降は問合せ解析処理を省略できる．具体的には問合せ解析処理時には，まず UAP から渡された SQL 文の構文を解析して並列化する．そのあと，複数の並列化した処理手順候補を含めて出力する．処理手順は，プロセサ間のデータ転送，同期処理，他のプロセサの処理の呼び出しを表現する．図 2.3 の (d), (e) に示すよう

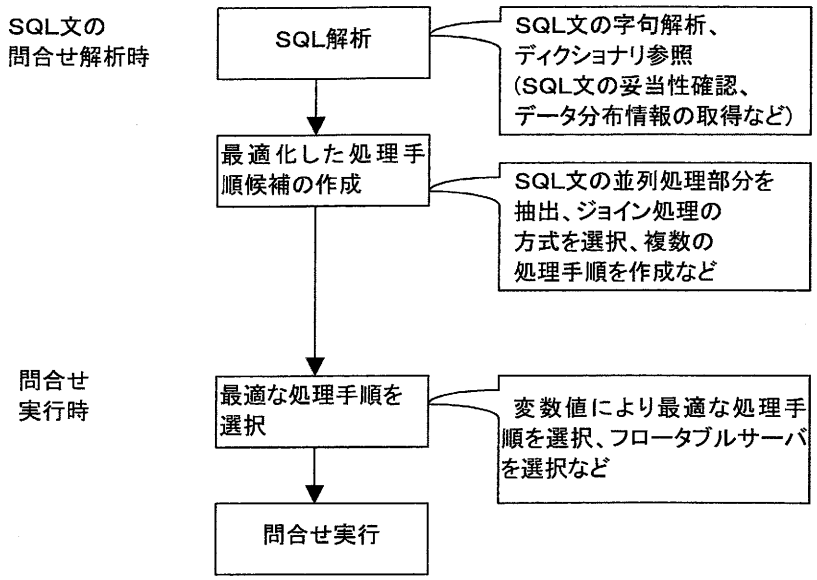


図 2.4: 2 段階最適化方式

に，並列化した処理同士の呼び出し関係は RPC を用いたデータ転送によって記述する．作成した個々の処理手順の相違点は，使用するジョイン方式（PNL，PSM のどちらを使うか），データ取り出し処理にインデクスを利用するか否か，サーバ間でデータをハッシュ手法で配布するか集約して配布するかの点である．この処理手順を保存する．フロータブルサーバを割り当てる数は，SQL 受付サーバが問合せ解析処理時に決める．この際に，ソート処理に要する時間がデータ入力時間に対して一定の比率になるようにフロータブルサーバの割り当て数を調整する．

次に，問合せ実行時に，確定した変数値とディクショナリで管理されているデータ分布の最新情報を基に，問合せ解析処理時に作成した処理手順候補の中から最適な処理手順を選択する．この時点で始めてフロータブルサーバを確保する．問合せ解析処理時に生成される処理手順を確認することで，問合せ解析処理時に決めた数では足りない場合，データベース管理者がフロータブルサーバの割り当て方式を変更することで，問合せ実行時にフロータブルサーバの割り当て数を変更できるようにしておくこともできる [29]．どのフロータブルサーバにデータを割り振って処理を実行するかは，確保したフロータブルサーバの中からランダムに選択する．これは，特定のフロータブルサーバに負荷が集中しないようにするためである．

この 2 段階最適化方式を適用して，問合せ実行時に PNL 及び PSM を処理手順として選択する場合，問合せ解析処理時に作成される処理手順候補を図 2.5 に示す．ここでは，前述した処理手順の並列化表現に加えて，select-procedure ノードが導入される．select-procedure ノードが指すデータ構造の左側に選択条件が設定される．この選択条件は，問合せの中に出現する選択条件である．この選択条件に，問合せ実行時に確定した変数値とデータ分布の最新情報を基に選択率を算出し閾値と比較することで，問合せ解析処理時に作成した処理手順候補の中から最適な処理手順を選択する．図 2.5 では，3 つの select-procedure ノードが出現するが，いずれも同様に算出が行われる．選択率が閾値以内であれば，PNL の外側表のアクセスでデータ数が絞り込まれるので，(b) の PNL 用外側表アクセスの処理手順，及

び (d) の PNL 用内側表アクセスの処理手順が，各々選択されて，結果的に (b)，(d)，及び (f) によって PNL 処理が実行される．一方，選択率が閾値を上回れば，PNL の外側表のアクセスでデータ数が絞り込まれないので，(a) の PSM 用左側表アクセスの処理手順，(c) の PSM 用右側表アクセスの処理手順，及び (e) の PSM 用結合処理の処理手順が，各々選択されて，結果的に (a)，(c)，(e)，及び (f) によって PSM 処理が実行される．

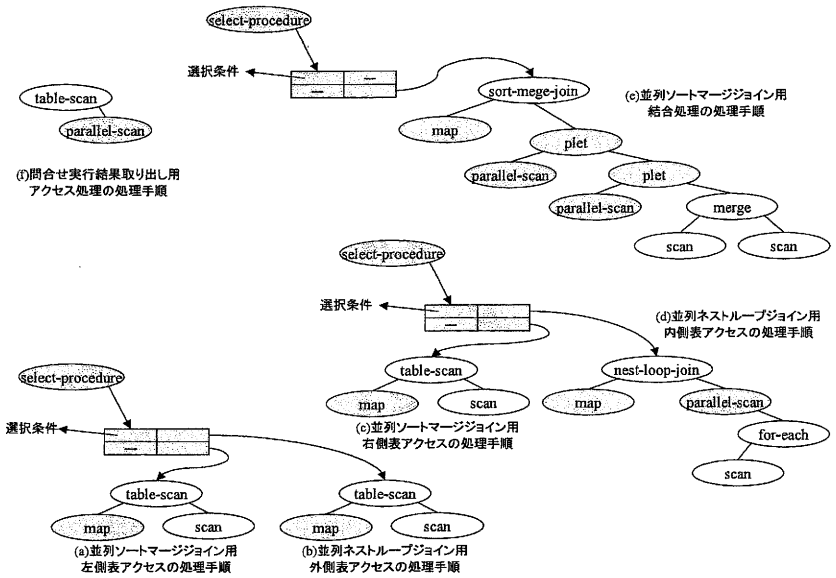


図 2.5: 処理手順の作成例

このように問合せ解析処理時に作成した処理手順候補の中から最適な処理手順を選択するための基準として，並列結合処理で絞り込んだデータを受け取る場合，及びインデクスを利用してデータを取り出す場合を想定して，処理時間の算出式を作成し，選択率，行数，及び行長などを変化させることで，各並列ジョイン方式の特性を抽出した [31]．この評価結果に従い，以下のことを導いた．

- PNL 処理の処理時間は，選択列の選択率に比例する．一方で，PSM 処理の片側表の選択率を変化させ処理時間を対数で表しても，10 万件規模以上の行数の場合，曲率が小さい．具体的には，PNL 処理の処理時間を選択率の 1 次式で近似し，また PSM 処理の処理時間を選択率の低次式近似することで，問合せ実行時に決定する選択率の閾値を決める．

- 行数が数倍程度に変化しても，PNL 処理及び PSM 処理を切り分けるための評価では PSM 処理の処理時間の曲率は小さく，閾値となる選択率は 1 つでほぼ一定の値である．

- インデクススキャン及びテーブルスキャンを選択するデータ取り出し処理の場合，選択率の閾値は結合方式に関わらず一定値である．

次に，並列ジョイン方式の選択方法について示す．

1. 問合せ解析処理時の最適化処理

問合せの条件で指定する列のインデクスの有無から，並列ジョイン方式の候補を作成する．具体的には，結合条件の列にインデクスが存在しなければPSM処理だけを，片側の表でも結合条件の列にインデクスが存在すればPNL処理及びPSM処理を候補とする．

次に，問合せ実行時に並列ジョイン処理を決定する判別式を作成する．結合する表をT1及びT2，表T1に関する選択条件の選択率をx，表T2に関する選択条件の選択率をy，並列ジョイン処理時間の見積りに使用する関数h，u，v，wとする．

PNL処理は，最外側表（最も早く取り出される対象の表を指す）が選択条件によって十分に絞ることができ，かつその内側表の結合列にインデクスが存在する場合に適用できる．従って，PNL処理時間は最外側表の選択条件によって絞られる行数に比例する．この最外側表（ここでは仮にT1とする）は，選択率が小さいxが選ばれる．また，PNL処理は内側表の結合列にインデクスが存在すれば，連続的に適用することができる．その場合，外側表の選択率と，内側表の結合行数比（外側表の1行に対して，内側表の何行が対応するかの比率）を掛け合わすことで，結果として生成される中間表の選択率とする．この中間表の選択率によって，中間表の行数が絞られる表を順番に選択する．

PNL処理の処理時間は，以下とする．関数hは，最外側表T1の選択率xを用いて処理時間が導かれる．

PNL処理の処理時間

＝ h(行数，選択率，インデクス段数，結合行数比，IO性能，データ分割数)

＝ n*x

（ただし，最外側表T1の選択率x以外は，問合せ解析処理時に決定される定数nとする）

次に，PSM処理は各2表のPSM処理時間が小さい順番で行う．これは，他の表のデータ取り出し時間の間に，なるべくデータ取り出し時間が短くなると期待できる表同士でジョイン処理を進めておく期待がある．基本的には，ジョイン処理が可能な2つの表のデータ読み出し時間の最大値が最小となる2表を見付けて，PSM処理を行い，その中間表を交えて再びデータ読み出し時間が最小となる2表を見付け出す．これを，中間表が1表になるまで繰り返す．

PSM処理の処理時間は，以下とする．関数uは，表T1及び表T2のデータ取り出し時間，突き合わせ時間，及び問合せ結果転送時間からなる．また，関数v，wは，表T1及び表T2のNウェイマージ時間からなる．

PSM処理の処理時間

＝ u(行数，選択率，射影行長，CPU性能，IO性能，データ分割数)

＋ v(行数，選択率，射影行長，CPU性能，IO性能，データ分割数)

* log(w(行数，選択率，射影行長，IO性能，ソートウェイ数，データ分割数))

選択率以外のパラメタは問合せ解析処理時に決定されるので，logを低次多項式で近似した式s(x，y)と置く．

以上から，s(x，y) <＝ n*xを予め解いておき判別式とする．

また，片側の表 T1 の選択条件だけに変数が含まれる場合，T2 の選択率も問合せ解析処理時に決定される．問合せの実行時に決定するパラメタを片側表の選択条件の選択率の 1 つだけにする場合，各式を示すグラフの交点は 1 つなので，閾値 C が決まる．

最後に，その選択条件に出現する列のデータ分布情報から，選択率の取り得る最大値及び最小値を取得し，最大値が閾値 C より小さければ PNL 処理，最小値が閾値 C より大きければ PSM 処理に処理手順を絞ることができる．

2. 問合せ実行処理時の最適化処理

問合せ実行処理の最適化処理で，確定した変数値とディクショナリで管理されるデータ分布の最新情報を基に，選択率を取得し，その選択率が閾値 C より小さい場合は PNL 処理を，閾値 C より大きい場合は PSM 処理を選択することで，最適な処理手順を決める．

2.4. 適用評価

2.4.1. フロータブルサーバ導入及びスキャン法選択の効果

フロータブサーバ導入及びスキャン法選択の効果を検証するために，TPC-C ベンチマーク [49] で規定される，注文表及び注文明細表を利用する．表データは，ハッシュ分割し，各 BES に均等に配置する．問合せは，注文明細表に条件を付与して，絞り込んだ結果の件数を得る処理を評価する．

次に，システム評価環境について示す．ハードウェア環境は，以下の緒元である．

- モデル名：日立 BS320
- CPU：Intel Xeon X5570 QUAD 2.93GHz × 2[13]
- メモリ：48GB
- ストレージ：日立 内蔵 SAS300GBx2 RAID1
- ネットワーク：ブレード間接続 1Gbps

また，ソフトウェア環境は以下の緒元であり，このシステム評価環境にソフトウェアを配置した．

- DBMS：HiRDB V08-05(64bit 版)
- OS：CentOS 5.4 (x64)
- BES を 1〜8 つ割り当てる．それらの内，フロータブルサーバに 1〜4 つの BES を割り当てる．

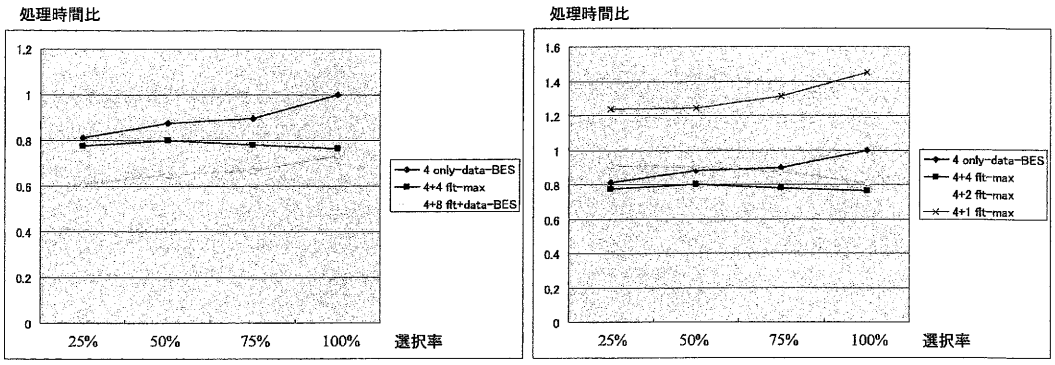
さらに，データベース緒元は以下の緒元である．

- 表のレコード数：注文表 (480,000 件)，注文明細表 (4,819,276 件)

[13]Intel Xeon は，アメリカ合衆国およびその他の国における Intel Corporation の商標です．

- データベース容量：注文表 (24.4MB)，注文明細表 (400.6MB)
- 表の分割方法：ハッシュ分割し，4 つの BES に均等に配置

フロータブルサーバ導入の効果について，図 2.6 に示す．なお，問合せの実行に当たり，問合せ解析処理のための処理時間及び表データを読み出す時間の影響を避けるために，2 回目以降の問合せの処理時間を計測する．



(a)フロータブルサーバ割り当て方式による処理性能　　(b)フロータブルサーバ割り当てBES数による処理性能

凡例：4 only-data-BES，選択率100%の処理性能を1とする処理時間比を評価

・問合せ

```
select count(*) from orders join order_line
                              on (o_id,o_d_id,w_id) = (ol_o_id,ol_d_id,ol_w_id)
                 where ol_w_id <= :w_id
```

・4 only-data-BES
4つのデータ入力処理を実行するBESに，フロータブルサーバが割り当てられるフロータブルサーバ対象限定方式

・4+n flt-max
4つのデータ入力処理を実行するBES以外で，フロータブルサーバに専用のBESがn個割り当てられるフロータブルサーバ専用方式

・4+8 flt+data-BES
4つのデータ入力処理を行うBESに，フロータブルサーバに専用のBESにも対象が拡大されて割り当てられるフロータブルサーバ対象拡大方式

図 2.6: フロータブルサーバの評価

1. フロータブルサーバ割り当て方式による処理性能

   図 2.6(a) は，フロータブルサーバ割り当て方式による性能特性を示す．

   4 つの BES に注文表及び注文明細表を均等に配置して，各々フロータブルサーバ割り当ての 3 方式によるパイプライン並列の効果を PSM 処理の処理時間で評価する．UAP で問合せ実行結果を受け取る時間の影響を避けるために，問合せを満たす行数 (SQL 文で select 節に count(*) を指定) だけを問合せ実行処理結果とするので，データのマージ処理及びデータ出力処理のオーバラップによる，双方の処理間でパイプライン並列の効果はない．即ち，データ入力処理及び部分列のソート処理がオーバラップによるパイプライン並列の効果が評価できる．

   フロータブルサーバを割り当てないフロータブルサーバ対象限定方式の場合，注文表及び注文明細表を格納する 4 つの BES を利用して，ソート処理及びマージ処理を実行する．このフロータブルサーバ対象限定方式では，注文表及び注文明細表のデータ入力処理とソート処理及びマージ処理が，オーバラップできないので，データ並列の効

果は期待できるが，一方でパイプライン並列の効果は期待できない．また，フロータブルサーバ専用方式の場合，注文表及び注文明細表の表データを格納する4つのBESとは異なるBESを利用してソート処理及びマージ処理を実行する．このフロータブルサーバ専用方式では，注文表及び注文明細表のデータ入力処理とソート処理及びマージ処理が，オーバラップすることで，データ並列及びパイプライン並列の効果が期待できる．さらに，フロータブルサーバ対象拡大方式の場合，4つのデータ入力処理を行うBESに，フロータブルサーバに専用のBESにも対象が拡大されて割り当てられ，すべてのBESでソート処理及びマージ処理を実行する．

その結果，フロータブルサーバ専用方式は，データ並列の効果だけ期待できるフロータブルサーバ対象限定方式と比較して，選択率が100%の場合では処理時間が約24%削減されており，データ入力処理及び部分列のソート処理がオーバラップしていることに起因するパイプライン並列の効果が分かる．即ち，従来までのソート処理など負荷が掛かる処理に専用のサーバ(フロータブルサーバ)を割り当てないフロータブルサーバ対象限定方式と比較して，フロータブルサーバ専用方式が優位であることを示している．

また，フロータブルサーバ対象拡大方式は，他の2方式が4つのBESでソート処理及びマージ処理を実行するのに対して，計8つのBESでソート処理及びマージ処理を実行するので，選択率が25%の場合では処理時間が約17%削減されている．ただし，フロータブルサーバ対象限定方式と同様に，4つのBESではデータ入力処理及び部分列のソート処理がオーバラップしないことから，データ量が増加するにつれて処理時間の削減効果が相殺される．

2. フロータブルサーバ割り当てBES数による処理性能

図2.6(b)は，フロータブルサーバ割り当てBES数による性能特性を示す．

フロータブルサーバ専用方式で，フロータブルサーバの割り当てBES数によって処理時間が短縮されることが期待できる．これは，データ入力処理及び部分列のソート処理がオーバラップしている処理間でパイプライン並列の効果が認められることに起因するが，逆にソート処理及びマージ処理に割り当てるBES数によって処理性能の優劣が決まる．

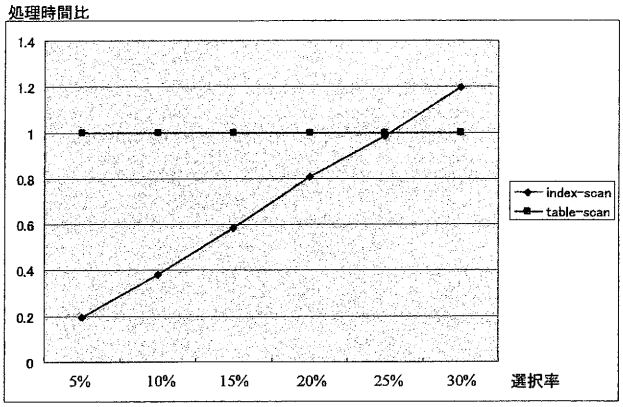
その結果，フロータブルサーバ専用方式の場合，フロータブルサーバの割り当てBES数が1から4に増えるにつれて処理時間が短縮されることが分かる．これは，データ入力処理及び部分列のソート処理がオーバラップしていることの処理間でのパイプライン並列，及びフロータブルサーバの割り当てBES数の増加によるデータ並列の両効果である．

一方，フロータブルサーバ対象限定方式の場合，データ入力処理及び部分列のソート処理の間，データのマージ処理及びデータ出力処理の間で，各々オーバラップが行われないが，フロータブルサーバ専用方式でフロータブルサーバのために専用に割り当てるBES数によっては，処理時間が優位となる場合もある．これは，各処理間でのハイプライン並列の効果により処理時間が短縮される効果が，部分列のソート処理，部分列のマージ処理，及びデータのマージ処理のために利用される，フロータブルサーバを割り当てるBES数の増減によって決まる処理時間の大小に従って相殺されることに起因する．フロータブルサーバ対象限定方式と，フロータブルサーバ専用方式でフ

ロータブルサーバのために 2 つの BES 数を割り当てる場合を比較すると，選択率によって処理時間の優劣が決まる．これから，フロータブルサーバを導入する場合，フロータブルサーバに割り当てる BES 数が重要であることが分かる．

3. スキャン法の選択

図 2.7 は，2 段階最適化によるスキャン法の選択を示す．



インデクススキャン及びテーブルスキャンによる処理性能

凡例：4つのデータ入力処理を実行するBES に問合せを発行し，table-scanの処理性能を1とする処理時間比を評価

・問合せ

```
select count(*) from order_line
        where ol_w_id <= :w_id and ol_supply_w_id>= :supply_w_id
```

・index-scan

4つのデータ入力処理を実行するBESに対して，各々作成されたB木インデクスを経由して表データを検索する．なお，B木インデクスは，4列（ol_w_id, ol_d_id, ol_o_id, ol_I_id）の複数列インデクスである．

・table-scan

4つのデータ入力処理を実行するBESに対して，表データを直接検索する．

図 2.7: スキャン法の選択

4 つの BES に注文明細表を均等に配置して，選択率を変化させて適切なスキャン法を選択しているか評価する．UAP で問合せ実行結果を受け取る時間の影響を避けるために，問合せを満たす行数 (SQL 文で select 節に count(*) を指定) だけを問合せ実行処理結果とするので，テーブルスキャン法を選択する場合，表データをすべてアクセスする処理時間はほぼ一定である．

また，インデクススキャン法を選択する場合，各々BES で作成された B 木インデクスを経由して表データを検索するが，アクセスする表データは選択率に比例して増加するので，処理時間も増加する．

ここで示す評価環境において提案 DBMS で実装する 2 段階最適化方式では，選択率が 25%以下ではインデクススキャン法を，また選択率が 30%以上ではテーブルスキャン法を選択することを確認している．2 段階最適化方式を採用しない場合，問合せ解析時に決定されたスキャン法が実行されることになり，変数に代入された値によっては最適ではないスキャン法を実行する可能性がある．例えば，選択率が 5%であれば，約 5 倍の性能差が起こり得る．

2.4.2. 多数サーバ環境でのスケーラビリティ

多数サーバ環境では，データベース量の増加及びスループット性能を向上させるために，サーバ数を増やすことで対処することが考えられる．しかし，サーバ数を増やすことで，それらのサーバ上で実行される問合せ実行の起動時間及びコミット処理の遅れ，各サーバ間での問合せ実行結果の受け取り処理への影響を評価する必要がある．また，サーバ数を増やして，問合せ実行処理の性能が向上することの確認も必要である．

問合せ実行処理の内容について，図 2.8 に示す．

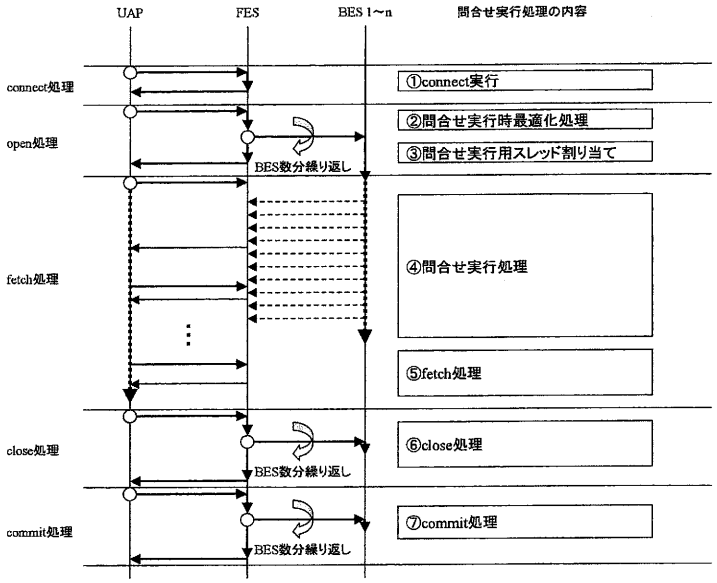


図 2.8: 問合せ実行処理

- connect 処理は，UAP と FES との接続関係を確立する．
- UAP でのカーソルの open 処理では，FES で変数を含む問合せであれば，変数値の値に従って最適な処理手順を選択する．
- FES で決定された処理手順に従って，問合せ実行処理に関与する BES に問合せ実行要求が発行され，その対象となる BES で問合せ実行用にスレッドが割り当てられる．
- 各 BES で実行手順に従って問合せ実行処理が行われる．
- UAP から発行される fetch 処理に従って，各 BES から問合せ処理結果が FES に転送され，また FES に転送された処理結果を UAP に転送する．
- UAP でのカーソルの close 処理では，問合せ実行処理に関与した各 BES に対して問合せ実行用に割り当てたスレッドを解放する．
- commit 処理は，問合せ実行処理に関与した各 BES に対して commit 処理を指示する．

次に，システム評価環境について示す．ハードウェア環境は，以下の緒元である．

- モデル名：日立 BladeSymphony BS1000A(16 ブレードシステム)
- CPU：Intel Xeon DP 3GHz × 2(ブレード毎)
- メモリ：4GB(ブレード毎)
- ストレージ：日立 SANRISE 9970
- ネットワーク：ブレード間接続 1Gbps

また，ソフトウェア環境は以下の緒元であり，このシステム評価環境にソフトウェアを配置した．

- DBMS：HiRDB V07-02(32bit 版) 及び TextSearch Plug-in V07-02(32bit 版)
- OS：Red Hat[14] Enterprise Linux[15] AS3
- UAP 及び FES は，各々1 つのブレードを専用に割り当てる．
- 1〜14 個のブレードに，BES を 1〜10 つずつ割り当てる．

さらに，データベース緒元は以下の緒元である．

- 表のレコード数：文献表 (200,000 件)
- レコード長：約 20KB(8 つの SGML テキスト列を含む 40 列から構成)
- データベース容量：文献表 (4GB)
- 表の分割方法：キーレンジ分割し，各々BES 数に応じて均等に配置

1. 問合せ実行の起動時間への影響

   14 個のブレードに 10 個までの BES を割り当て，最大で 140BES に対して open 処理を行って，問合せ実行の起動処理への影響を測った．この open 処理には，問合せ実行時最適化処理及び問合せ実行用スレッド割り当てが含まれているが，それらの処理が BES 数と open 時間に与える影響を評価した．

   その結果，FES における open 時間は，10BES の open 時間を 1 とした場合，20BES で 2.2，40BES で 2.9，80BES で 6.9，140BES で 13 となり，10〜140BES までの範囲では，ほぼ BES 数に比例することが分かった．

   また，close 処理及び commit 処理に関して，10〜140BES までの範囲では BES 数に比例する処理時間となることも確認した．

2. 問合せ実行結果の受け取り処理への影響

   14 個のブレードに 1 つの BES を割り当て，1，2，4，8，及び 14 の各 BES から問合せ実行の結果を FES に転送し，また FES では問合せ実行結果の受け取り処理を行い，UAP から発行された fetch 処理に対して，問合せ実行結果を転送する時間を評価する．

[14]Red Hat は，米国およびその他の国で Red Hat, Inc. の登録商標若しくは商標です．
[15]Linux は，Linus Torvalds 氏の日本およびその他の国における登録商標または商標です．

この場合，問合せ結果1件当たりのfetch処理時間に換算して，影響を評価した．ただし，UAPとFES間での転送時間が影響を及ぼさないように，1回のfetch処理で100行単位で転送する．

その結果，1つのBESでは1件当たりのfetch処理時間を8とすると，2BESで4.9，4BESで4.1，8BESで4.3，14BESで4.6となり，4BES以上では1件当たりのfetch処理時間がほぼ一定になることを確認した．1BES及び2BESでは，FESでBESからの問合せ実行結果の転送を待っているために，1件当たりのfetch処理時間が長くなっている．

3. 問合せ実行処理の応答時間

問合せ処理として，各BESでの処理負荷が高い全文検索処理を設定した．具体的には，8つのSGMLテキスト列に対する全文検索条件contains関数の論理和，及び他の列に対する検索条件を指定している．全文検索処理の応答時間は，検索する対象のデータベース量に比例するので，本システムの総データベース量を一定にして，BES数を2，4，8，14と変化させ，並列度と応答時間の関係を評価した．各BESに格納されるデータベース量毎に，UAPから見てopen処理直後からfetch処理が行われるまでの時間を応答時間とした．また，UAPで問合せ実行結果を受け取る時間の影響を避けるために，問合せを満たす行数(SQL文でselect節にcount(*)を指定)だけを問合せ実行処理結果とする．

その結果，1つの問合せ当たりの応答時間は，2BESで4.1，4BESで2.1，8BESで1.5，14BESで1.2と，2〜14BESまでの範囲ではBES数に比例する．

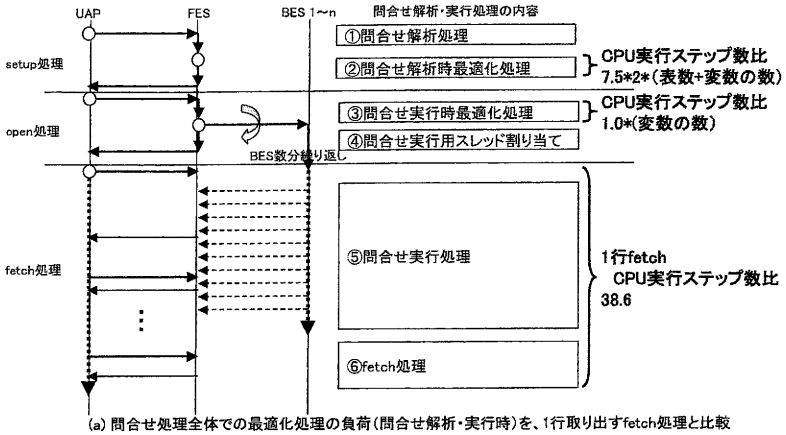
2.4.3. 2段階最適化方式の評価

2段階最適化方式を実装するDBMSのCPU実行ステップ数を取得し，SQL文の問合せ解析処理及び問合せ実行処理の負荷を評価する．SQL文の問合せ解析処理は，1回目のopen処理だけで実行されるが，2回目のopen処理では実行されない．また，SQL文の問合せ実行処理は，1回目以降のopen処理で実行される．図2.9は，2段階最適化方式の評価結果である．

2段階最適化処理の負荷評価

まず，図2.9(a)で，問合せ解析処理及び問合せ実行処理の内容を示す．問合せ実行処理の最適化処理で，確定した変数値とディクショナリで管理されるデータ分布の最新情報を基に，最適な処理手順を決めるためのCPU実行ステップ数比を1.0とする．また，この処理は，問合せ解析処理で処理手順を選択するために利用される，高々変数の数回だけ繰り返される．

一方で，問合せ解析処理の全体のCPU実行ステップ数比は81.2であり，特に問合せ解析処理の最適化処理で，ジョイン処理の選択及び処理手順候補を作成するためのCPU実行ステップ数比は7.5である．また，この処理は，FESの処理手順で1回，PNL処理のために問合せに出現する表数回，PSM処理のために問合せに出現する（表数-1）回，単一表の処理手順候補の毎に，高々問合せ変数の2倍の回数だけ繰り返される．即ち，繰り返される回数は，2*(表数の数＋変数の数)となる．



(a) 問合せ処理全体での最適化処理の負荷(問合せ解析・実行時)を、1行取り出すfetch処理と比較

	従来方式	2段階最適化方式
問合せ解析時最適化処理	7.5*(表数+1)	7.5*2*(表数+変数の数)
問合せ実行時最適化処理	0	1.0*(変数の数)
1行fetch処理	38.6	

(b) 従来方式と2段階最適化方式での最適化処理の負荷見積り

注記): 負荷はいずれもCPU実行ステップ数比とする.

図 2.9: 2 段階最適化方式の評価

図 2.6 に示す問合せで試算すると，表数の数が 2，変数の数が 1 である．各々の CPU 実行ステップ数比は，問合せ実行処理の最適化処理では 1.0*(変数の数=1)=1.0 となり，問合せ解析処理の最適化処理では 7.5*2*(表数の数=2 +変数の数=1)=45.0 となる．また，比較のために，単一表から高々1 行を取り出すための fetch 処理に要する CPU 実行ステップ数比は 38.6 であり，問合せ実行処理の最適化処理は約 3%の負荷に相当する．適用評価の問合せは，ジョイン処理であるので，全体の負荷における割合がさらに小となる．

次に，問合せ解析処理の全体の CPU 実行ステップ数比は 81.2 に対して，問合せ解析処理の最適化処理は約 55%の負荷に相当する．2 段階最適化方式で，問合せ解析処理時に唯一最適と評価できる処理手順を決める場合，この負荷が削減できる効果がある．

従来方式との比較

図 2.9(b) は，コストモデルに基づき問合せ解析時最適化処理で唯一の処理手順を選択する従来方式 [26][44] と 2 段階最適化方式での最適化処理の負荷見積りを示す．

まず，問合せ解析時最適化処理の負荷は，従来方式と比較して 7.5*(表数-1 + 2*(変数の数)) ほど増加する．図 2.6 に示す問合せで試算すると，表数の数が 2，変数の数が 1 であるので，22.5 となる．単一表から高々1 行を取り出すための fetch 処理に要する CPU 実行ステップ数比は 38.6 と比較すると，約 58%の負荷増加に相当する．動的 SQL のような，問合せ解析処理と問合せ実行処理が連続動作する場合，単一表から高々1 行を取り出すだけで済む fetch 処理では，負荷増加が問題となることが考えられる．

また，問合せ実行時最適化処理の負荷は，従来方式と比較して 1.0*(変数の数) ほど増加する．図 2.6 に示す問合せで試算すると，変数の数が 1 であるので，1.0 となる．単一表から高々1 行を取り出すための fetch 処理に要する CPU 実行ステップ数比は 38.6 と比較すると，約 3%の負荷増加に相当する．取り出す行数が増加すれば，負荷増加の割合がさらに小とな

る．従って，問合せ実行時においては，負荷増加が高々約3%となり，実システムへの適用性が確認できる．

2.5. 関連動向

2.5.1. 並列問合せ機構について

パイプライン並列に関する問合せ機構については，Double pipelined hash join によって自明ではないブロッキングオペレータを除去する手法 [48] が提案されている．

MapReduce が大規模データ解析に活用されつつある [45][46]．利用者が key/value ペアを処理する Map と Reduce と呼ぶ2つの関数によって，大規模なデータ処理を記述する．具体的には，分散ファイルからの入力を，key にハッシュ関数を適用して分割し，各ノードに格納する Map と，それらを対象にデータ処理を行う Reduce からなる．しかし，MapReduce で提案されているデータ処理フレームワークは，既に並列 DBMS に内在するものである [40][41][42]．提案方式では，SQL 文で指定されたジョイン処理では，2つの表の結合処理を実行する前に，各表に指定される選択条件を適用して絞り込めるかどうかを問合せ実行時に判断する．もし絞り込める場合，選択条件を適用してからもう一方の表をアクセスする PNL 処理が実行される．絞り込めない場合，2つの表をアクセスして，同一のハッシュ関数を適用して，各ノードに分散格納，即ち Map を実行する．また，分散格納された2つの表同士で結合処理，即ち Reduce を実行する．SQL の並列データベースシステムに，フロータブルサーバ方式を併用し2段階最適化方式を適用するものは，他では議論されていない．

並列データベースシステムは，問合せの応答時間を短縮するために，複数の処理要素で実行を行いつつ，単一のシステムイメージで振舞う．並列性の利点は，データ処理の対象が互いに独立に分割されていて，しかもパイプラン操作によって実行することによって得られる．XPRS では，2段階で最適化処理を行う [43]．最初の段階で，ジョイン処理の順序及びソート処理など処理手順を決定し，また第2段階で，通信処理を加味し，スケジューリング方法を提示するが，提案する方式とは異なる．

2.5.2. 問合せ最適化について

問合せ最適化の課題として，統計情報に基づいて処理手順を作成するのに必要とされる精度，正確さに限界があることである．問合せ最適化の役割は，問合せで参照する表に関連する各種統計情報に基づいて最適な処理手順を探すことである [26][44]．これら各種統計情報には，表，インデクス，あるいはそれらから導出される対象の，行数，列値の分布，格納状況などが存在する．また，問合せ最適化は実際の情報システムのストレージシステムでのシステム統計情報に依存する処理のコストモデルに基づいて，アクセスパス，結合方法，結合順序，問合せ変換を決定する．このコストモデルは処理手順の候補から最も効率が良いとされる処理手順を導くための用いられる．

しかし，コストモデルに基づく最適化に課題がある．その中で，正確とは言い難い統計情報に基づいて処理手順を予測することについて，その正確さには課題がある．例えば，列に存在する値が，極端な場合には数個の特定の値が非常に高い頻度で出現し，一方で他の値は低い頻度で分布するような場合が考えられる．この場合，処理性能が大幅に異なる処理手順

が値に依存して生成されることが分かっている．これに対処するために，ヒストグラムを用いて列の値に関する分布を正確に把握し，推定を精緻に行うことも考えられる [33][34][35].

また，例え精緻な分布を統計情報として管理しても，比較した値が定数ではなく問合せで出現する変数である状況で処理手順が作成されることもあるので，必ずしも最適な処理手順が作成されない．この変数は，実行される前に，どの値が束縛されるのか事前に知ることはできない．このために，特定の値のために処理手順を作成することはできない．実用上では，列の値が取り得る分布が精緻に分かっていても，役には立たず，実際には定数と比較する場合よりも，変数と比較する場合の方が多いことが分かっている．もし，変数が束縛されるまで処理手順の作成を遅延させると，変数が束縛される毎に動的SQLを利用することになり，毎回SQLの解析を行うことを意味する．この課題を解決する策として，問合せが初回に実行される場合，初回に設定される値に対して統計情報を利用して最適化処理によって処理手順を生成することも考えられるが，その値がその問合せを代表するという仮定には問題が残され，処理手順の最適性の課題は残る．この他に，変数を含む問合せの最適化方式が提案されている [36][37][38].

問合せ実行時まで情報が適用可能にならない限り，完全な処理手順の作成を遅らせることもなされている [36][37][38][47]．特に，choose-plan 操作を導入して，問合せ実行時に，複数の処理手順の選択肢から選択できる [36]．本論文では，問合せ解析時に列の統計情報から変数に代入される値に関わらず，最適な処理手順を選択できる点で異なる．また，並列データベースに適用している点でも異なる．

2.6. おわりに

並列データベースシステムにおいて，問合せ処理の並列化方式を提案した．具体的には，SQLの解析処理によって並列性を導き，具体的に並列に実行するためのパイプライン並列及びデータ並列の処理手順の表現方法，問合せ変換方法，並列実行環境など，実適用を伴う問合せ処理機構の実現方式及びその評価を示した．

負荷平準化を目的とするフロータブルサーバ方式は，プロセサ間での処理負荷の平準化のために，DBを格納していないプロセサに処理の一部を分担させる方法である．これによって，ジョイン処理にデータ並列及びパイプライン並列を適用することで，応答時間が改善される．また，サーバ数を増やすことにより各サーバの処理手順を起動するための処理時間及びスケーラビリティとの関係を評価した．さらに，フロータブルサーバ方式を好適に活用するために2段階からなるSQLの最適化方式を提案した．SQL文をSQL受付サーバが受け取って解析処理する段階と，実際にそのSQLの処理手順を実行する問合せの実行処理時に最適化する段階からなる．大規模な表に対するジョイン処理及びソート処理は，処理手順の選択を間違うと大幅に性能が低下する懸念があるが，2段階の最適化方式は，これらの課題を解決するものである．

第3章　プラグイン機構の開発

ネットワークコンピューティング時代には，画像，音声，動画などマルチメディアデータを取り込んだ情報システムの構築が不可欠となりつつある．マルチメディアデータは，データ量が大容量なだけではなく，メディア操作及び格納形式が多様化かつ順次拡張されることに特徴がある．従来までのRDB(リレーショナルデータベース)及びメディア毎に別サーバを有するシステム方式では，システム運用及びアプリケーション開発に多くの課題があった．

これらの課題に対して，提案するORDBMS(オブジェクト指向リレーショナルデータベース管理システム)では，それぞれのマルチメディア情報に対して，検索及び格納などの操作に対応するライブラリ群を，自由に追加，登録できるプラグイン機構を開発した．プラグインとしては，SGML文書，文字列，XMLへの構造検索を始めとして，類似画像検索，空間検索を実装した．

3.1.　はじめに

インターネットに代表されるネットワークコンピューティングにおいては，利用者に親しみを与える情報提供形態が重要である．そのため，画像や音声，動画といったマルチメディアデータを取り込んだ情報システムの構築が必要不可欠となりつつある．

マルチメディア情報は，データ量が大容量なだけではなく，メディア操作や格納形式が多様化かつ順次拡張していくことに特徴がある．これに対応するマルチメディアデータ処理系として，オブジェクト・リレーショナルデータベース(ORDB)が期待されている[50]．ORDBは，RDBモデルをベースとし，オブジェクト指向モデルを取り入れたDBである．

一方，国際標準化機構ISOでは，データベース言語SQLの標準仕様を策定している[51]．現時点では，SQL2008が最新であり，利用者定義型（以下では，UDT(User-Defined Type)と呼ぶ），型継承などのオブジェクト指向モデルを取り入れた拡張が規定されている．これにより，複雑な構造を持つマルチメディアデータをデータベースに容易に取り込めるようにする枠組みとして活用が可能である．

ORDBでは，データに対する操作機能を利用者定義のルーチンによって提供する．ここではプラグイン(plug-in)と呼ばれる機構によって，ルーチンを実装するモジュール(以下では，プラグインモジュールと呼ぶ)をORDBMSに組み込む．これにより，DBMS自体を変更することなく，各種のメディアデータを扱う先進的な機能を実装するプラグインモジュールを，容易にDBシステムに取り込むことができる．また，このプラグインモジュールはDBMSと独立して開発することができる．即ち，この方式によれば，例えば構造化文書の構造指定検索などの高度な機能を持ち，またn-gram方式による高速な全文検索機能をプラグインとして開発し，ORDBに組込むことで，拡張可能なアーキテクチャが実現できる．さらに，文書に限らず画像や地図データなどを管理する機能を持つプラグインモジュール群を組込めば，多様にシステムを拡張することもできるようになる．

そのために，ORDB にプラグイン機構を開発し，また Shared Nothing 方式の並列 DB に適用することで，マルチメディア情報管理システムの基盤となる ORDBMS を開発した [52][53][54][55][56]．開発した ORDBMS を適用したシステムの例を図 3.1 に示す．本論文の研究範囲は，プラグイン機構である．具体的には，プラグイン機構に関して，プラグイン IDL[1]（Interface Definition Language）と PPI[2]（Plug-in Programming Interface）を開発した [58][59]．

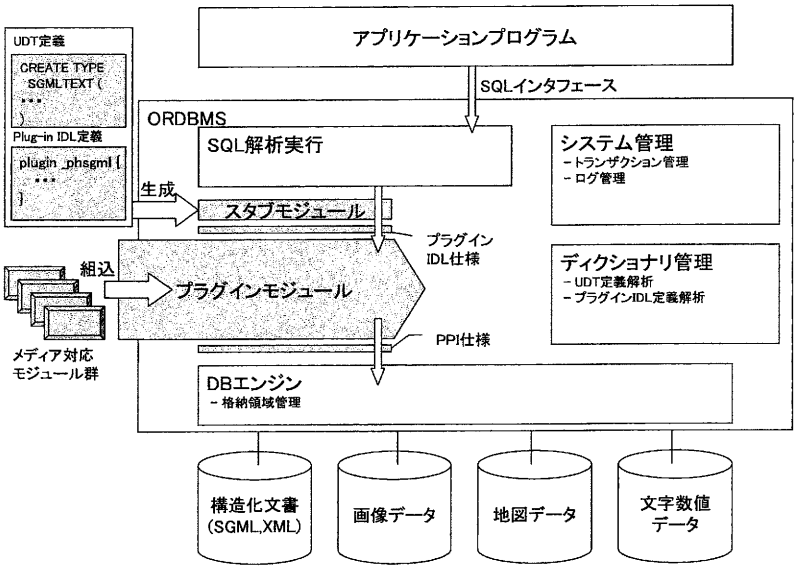


図 3.1: ORDBMS の構成

プラグイン機構について，マルチメディア情報システムへの要件分析と課題の検討を基にして，ORDBMS の開発及びその実装を通して適用性を確認した．

以下では，3.2 節で課題について述べ，3.3 節でプラグイン機構のアプローチについて解説し，3.4 節で具体的に実装したプラグイン事例による適用評価を示し，3.5 節で関連研究を述べ，3.6 節で纏める．

3.2. 課題

ORDBMS はデータに対する操作機能を利用者定義のルーチンによって提供するが，本章ではプラグイン機構によって，関数の実装モジュールを ORDBMS に組み込む場合の課題について述べる．

まず，ORDBMS の基本機能として，データベースの回復処理を実現することは信頼性を達成するための主要な要件である．プラグイン機構によってデータベースに対する操作群が提供されるが，拡張可能なアーキテクチャを実現するために，ORDBMS と独立に開発するので，この回復処理も組み込まれなくてはならない．ここで回復処理とは，ACID 特性の耐

[1]3.3.3 節で後述するが，データベースへの操作，トランザクション制御及びデータベース回復処理に対応した順序，タイミングの処理契機を規定する，プラグインの外部仕様を記述するための宣言的言語である．

[2]3.3.4 節で後述するが，データ操作，データ制御，ファイル操作，及びサービスに区分する関数インタフェースからなり，DB リソースへのアクセス手段を提供する．

久性 (Durability) を保証することを指す．また，回復処理はトランザクション及びログを基にして実装する必要もある [60]．ここで，この回復処理の実装に必要な，トランザクション情報，ログ情報，格納領域情報を DB リソースと呼ぶ．回復処理はこれらの DB リソースを利用して実現されるが，トランザクション制御及びデータベース回復処理に対応した順序，タイミングで処理を行わないと，データベースの破壊を招く可能性がある．従って，このような ORDBMS に組み込まれる実装モジュールの開発は，ORDBMS の実装に関する知識を理解しておく必要もあり，プログラミングの難易度が高いと考えられ，信頼性を確保するための施策が必要である．

一方で，ORDBMS はトランザクション処理での利用を想定しており，回復処理の性能も重要である．即ち，実装モジュールの起動及び実行に伴なうオーバヘッドは最小限に留めたい．

SQL[51] では，データに対する操作を行うルーチンの実装方式として，以下の二つを規定している．SQL で規定するルーチンには，関数及び手続きが含まれる．

- SQL 起動ルーチン (SQL-invoked routine)

  ルーチンの実装は，SQL 手続き文からなる．表，列，インデクスなど SQL 定義文で規定されるスキーマオブジェクトだけが，操作の対象である．

- 外部起動ルーチン (Externally-invoked routine)

  ルーチンの実装を，SQL 以外の C 言語などのプログラミング言語で記述する．ルーチンの実行では，その記述のコンパイル結果であるオブジェクトコードを実行する．

SQL 起動ルーチンは，SQL 文によりスキーマオブジェクトへのアクセスが可能で，SQL 手続きは計算完備である．しかし，回復処理の実装に必要な DB リソースはスキーマオブジェクトとして規定はされておらず，それら DB リソースへの操作記述，即ち表 3.1 に後述するロールバック及びロールフォワード処理の記述は制約されている．

一方，外部起動ルーチンは，C 言語などのプログラミング言語を利用して，領域格納手段としてファイルへのアクセスは可能であり，トランザクション情報及びログ情報からなる DB リソースへのアクセス及びデータベースの回復処理の記述が可能である．また，外部起動ルーチンが DBMS とは異なるアドレス空間で実行する方法 [66]，及び外部起動ルーチンが実行するアドレス空間を DBMS と同じアドレス空間にするのか，あるいは異なるアドレス空間にするのかを選択する方法 [50] があるが，トランザクション処理での性能を考えると，外部起動ルーチンが実行するアドレス空間と DBMS が実行するアドレス空間を同じとする必要がある．

以上，本研究の課題を纏める．

- 耐久性の保証

  DB リソースを利用した回復処理の実現

- 外部起動ルーチンの起動及び実行の高速化

  トランザクション処理での性能要求に応えること

- プラグインモジュールの組込みを簡単化

ORDBMS に組み込まれる外部起動ルーチンは，ORDBMS の実装に関する知識を理解して開発される必要があり，難易度が高いと考えられるプラグインモジュールの組込みを簡単にすること

本論文では，これらの課題を解決するために，DB リソースを利用した回復処理を実装するプラグインモジュールを ORDBMS に組み込み，またプラグインモジュールを外部起動ルーチンで呼び出す，プラグイン方式を採用した．このプラグインモジュールは，DB リソースを利用してデータベースへの操作，トランザクション制御及びデータベース回復処理に対応した順序，タイミングの処理契機に応じて処理の記述を行う．

プラグイン方式の特徴について纏める．

1. DB リソースへのアクセス及び回復手段

   プラグインモジュールの実装から DB リソースを引数とし，データベースへの操作，トランザクション制御及びデータベース回復処理に対応する関数群を容易に呼び出し，かつデータベースの回復処理を支援する PPI を提供する．

2. 容易なプラグインモジュールの組込み

   ORDBMS の基本機能であるデータベースへの操作，トランザクション制御及びデータベース回復処理に対応した順序，タイミングの処理契機を規定する，プラグインの外部仕様に基づくプラグイン IDL を開発する．このプラグイン IDL に従いプラグインの動作を記述する，プラグイン IDL 記述ファイルをプラグイン IDL コンパイラが解析し，ORDBMS からプラグインを呼び出すときに必要な定義データとスタブモジュールを自動生成する．また，プラグインを組込むツールを提供し，コマンドにより容易にプラグインを組込めるようにする．

3. プラグインモジュールの高速起動

   プラグインモジュール起動のオーバヘッドを少なくし，かつ ORDBMS の並列処理に合わせて並列実行するために，ORDBMS と同じアドレス空間で SQL 文の実行からなる DB 処理を行なうスレッドと同一スレッドでプラグインを実行する方式を採用した．

3.3. アプローチ

3.3.1. 概要

プラグイン機構を利用したデータ処理の概要を図 3.2 を用いて示す．

この例では，データベースに登録された SGML 文書からプラグインの機能を利用して全文検索を行う．SGML 文書への操作を実装するデータプラグインモジュール (SGML データプラグインモジュールと呼ぶ) と，n-gram 方式のインデクス処理系を実装するインデクスプラグインモジュール (n-gram インデクスプラグインモジュールと呼ぶ) を利用する[3]．

これらを，図 3.2 中の 1〜8 を用いて説明する．

[3]プラグインモジュールには，UDT への操作を実装するデータプラグインモジュールと，インデクス処理系を実装するインデクスプラグインモジュールの 2 種類からなる．詳細は，3.3.5 節に後述する．

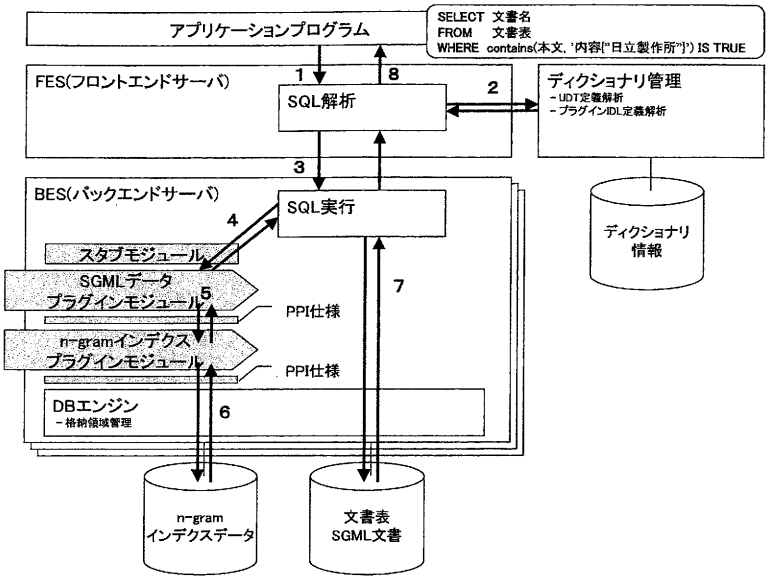


図 3.2: プラグインを利用した SGML 文書全文検索処理の例

1. アプリケーションから ORDBMS に，全文検索を要求する SQL を発行する．

   この例では，本文の内容に「日立製作所」という文字を含む文書の文書名を取得する SQL を発行する．

2. ORDBMS の FES(フロントエンドサーバ) がアプリケーションからの要求を受付ける．

   FES の SQL 解析部は，ディクショナリ管理と通信し，SQL を意味解析して DB 処理を行うための中間コードを生成する．なお，ディクショナリには，スキーマオブジェクト及び DB リソースに関連する定義情報が管理されている．SQL 文の解析評価でプラグインとして実装されているルーチン contains() が指定されていることを識別すると，SGML データプラグインモジュールを呼び出すための中間コードを生成する．また，インデクス定義でルーチン contains() に対応付けられている n-gram インデクスプラグインモジュールを呼び出す中間コードも生成する．

3. FES が生成した中間コードを複数の BES(バックエンドサーバ) に送信し，並列に DB 処理を行うよう要求する．

   BES の SQL 実行制御部が，中間コードに従って DB 処理を行う．

4. BES は，条件判定処理において，中間コードに設定された情報を基に，プラグインモジュールを呼び出す．

   BES がプラグインモジュールを動的にロードし，BES の同一スレッドで実行する．SGML データプラグインモジュールの関数 contains() に対応するプラグインモジュールを起動する．呼び出しはスタブモジュールを介して行う．

5. SGML データプラグインモジュールが，PPI を通してインデクス利用要求を行う．

   PPI では，中間コードの情報を基に，n-gram インデクスプラグインモジュールを呼び出す．

6. n-gram インデクスプラグインモジュールは，PPI を用いて ORDBMS で管理されているファイルにアクセスし，インデクスデータを参照し，全文検索処理を行う．

7. 全文検索の処理結果に応じて，BES の SQL 実行制御は文書表から文書名を取得してまとめて FES に返す．

8. FES は複数の BES から受け取った処理結果をまとめてアプリケーションプログラムに返す．

この例において，本論文で開発した特長的な部分を以下に示す．

- UDT のルーチンをプラグインで実装

  UDT のルーチンをプラグイン方式による外部起動ルーチンとして実装した．プラグインで提供する機能を，UDT のルーチン呼び出しで提供できるので，アプリケーションと親和性の高いインタフェースが実現できる．

- 定義情報を基にしたプラグイン呼び出し

  ディクショナリにプラグインを呼び出すための定義情報を保持する．SQL 解析部は，この定義情報をもとに SQL を解析し，プラグインを呼び出すための中間コードを生成する．この定義情報を変更すれば，呼び出されるプラグインが変更される．これにより，プラグインの追加・変更・削除を柔軟に行うことができる．

- データプラグインモジュールとインデクスプラグインモジュールを分離

  データ操作処理及びインデクス処理に特化したプラグインモジュールを，それぞれデータプラグインモジュール，インデクスプラグインモジュールとする．これにより，n-gram 方式を実装する n-gram インデクスプラグインモジュールを，SGML データプラグインモジュール以外にも適用するなど，再利用を図ることができる．

- BES の処理と同一スレッドでプラグインを動的ロードして実行

  BES の処理スレッド上で動的ロードしてプラグインモジュールを実行する．これにより，モジュール起動のオーバヘッドが小さく，高速にモジュールを実行できる．さらに，複数の BES による並列処理で，プラグインモジュールも並列に実行することができる．

3.3.2. プラグインモジュールのルーチン定義

プラグインモジュールで提供する機能を SQL インタフェースで利用するために，UDT のルーチンを定義する．図 3.3 に，プラグインモジュールのルーチン定義例を示す．この例では SGML 構造化文書データを表現するために SGMLTEXT 型を定義する．

UDT の定義では，CREATE TYPE に続いて，型名を記述する．続いて，属性とルーチンの定義を記述する．

ルーチンの定義は，隠蔽レベル，ルーチン種別，ルーチン名，引数，戻り値の順に記述する．LANGUAGE C は C 言語で実装されたモジュールであることを示す．EXTERNAL NAME '...' は，外部起動ルーチンとしての実装であることを示す．プラグインモジュール

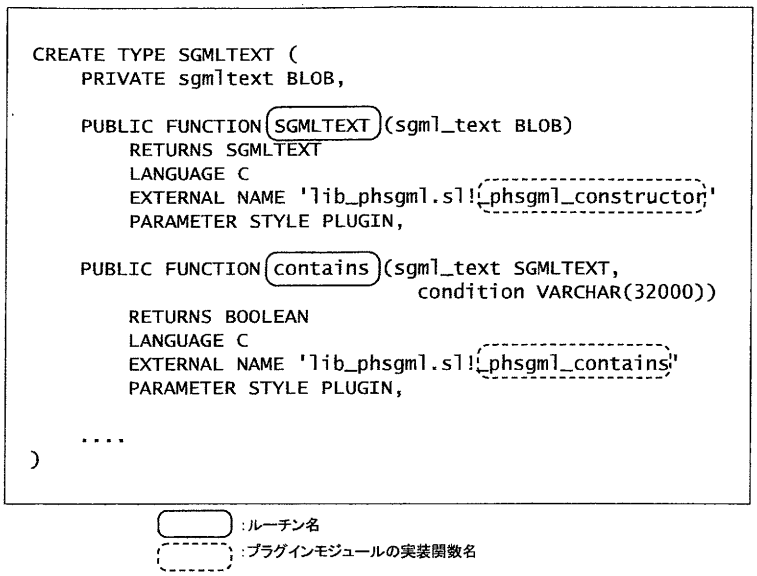


図 3.3: UDT のルーチンル定義例

は，共用ライブラリ名！実装関数名の形式で示す．PARAMETER STYLE PLUGIN は，プラグイン方式による実装であることを示す．

この定義は，ディクショナリに保持され，SQL 解析時にプラグインを呼び出すための情報として用いる．

3.3.3. プラグイン・インタフェース定義

プラグイン IDL 定義

プラグイン IDL は，ORDBMS の基本機能であるデータベースへの操作，トランザクション制御及びデータベース回復処理に対応した順序，タイミングの処理契機を規定する，プラグインの外部仕様を記述するための宣言的言語である．

プラグインモジュールは ORDBMS とは独立に開発される．プラグインが ORDBMS に組込まれたときに，そのプラグインの外部仕様を ORDBMS が認識する必要がある．プラグインの外部仕様を記述するための言語としてプラグイン IDL を開発した．プラグインモジュールの開発者は，プラグインの動作を記述するプラグイン IDL 記述ファイルを作成する．図 3.4 に，プラグイン IDL の記述例を示す．

キーワード plugin に続いて，プラグイン名，プラグイン ID，対応する UDT を指定する．続いて，{ } の中にプラグインモジュール内の実装関数の仕様を記述する．以下の記述が可能である．

1. 外部起動ルーチン呼び出しに現れない引数の指定

   プラグインの仕様定義では，ルーチンの引数には現れない引数の指定も可能である．例えば，ORDBMS の内部情報を保持した dbifb を指定できる．dbifb は，DB リソースをアクセスするために，PPI を利用するときに受け渡すことができ，PPI では dbifb

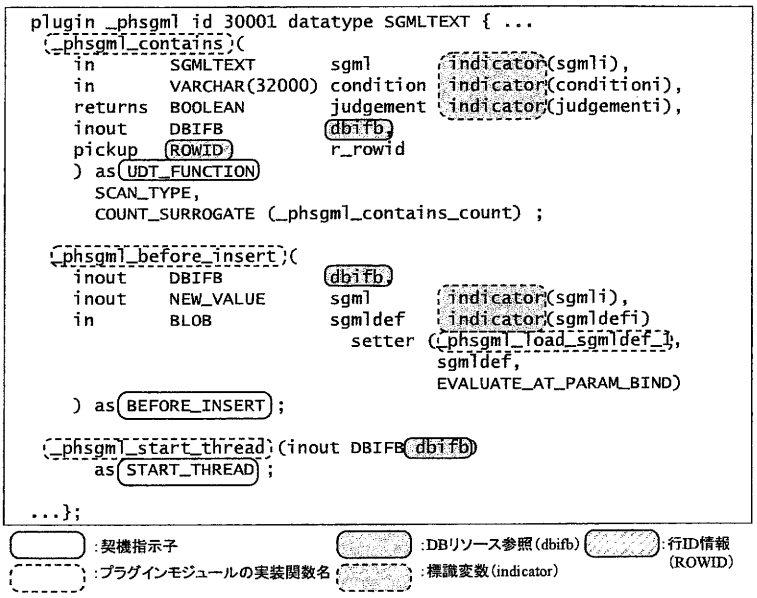


図 3.4: プラグイン IDL の記述例

の内部情報を参照して動作する．この他に，受け渡す引数がナル値であるかを示す標識変数 (indicator と記述)，行 ID 情報 (ROWID と記述) が指定できる．

2. プラグインモジュールを呼び出す契機の指定

   as に続いて，プラグインモジュールを呼び出す契機が指定できる．表 3.1 に示す種類の契機がある．表 3.1 にプラグインモジュールを呼び出す契機を示す．これらの中で，UDT 定義で指定されるルーチンのルーチン呼び出し契機以外では，行データの挿入，更新，及び削除のタイミングで呼び出されるデータ操作契機，トランザクション処理，コミット処理に関連するシステム制御契機，データベースの回復処理に関連する回復処理契機，及びインデクス処理に関連するインデクスメンテナンス契機からなりプラグインモジュールが呼び出される．これら呼び出す契機で提供する機能は，ORDBMS の実現のために必要とされる機能要件を満たすものである [50][60].

3. ORDBMS の処理と密接に連携した実行制御の指定

   以下の指定によって，プラグインモジュールを ORDBMS の処理と密接に連携して起動することができる．

   (a) スキャンタイプ処理指定（SCAN_TYPE）

   通常，インデクスを設定していない表に対しては，ORDBMS は表の行一つ一つに対して繰り返し処理を行う．このような処理形態をテーブルスキャンという [60]．テーブルスキャンでは，表の行一つ一つに対してルーチンを起動することになる．

   これに対し，インデクスを利用して検索条件を判定する場合は，ORDBMS はインデクス機能により選択された行に対して繰り返し処理を行う．このような処理形態をインデクススキャンと呼ぶ．

表 3.1: プラグイン実装の呼び出し契機

契機種別	処理内容	契機指示子	
ルーチン呼び出し[1]	UDT定義で指定したルーチンの呼び出し時に制御が渡る	UDT_FUNCTION	
		–	
データ操作[1]	行データの挿入，更新，及び削除などデータ操作の前後のタイミングで呼び出される	BEFORE_INSERT	AFTER_INSERT
		BEFORE_UPDATE	AFTER_UPDATE
		BEFORE_DELETE	AFTER_DELETE
		–	AFTER_ADD_COLUMN
		BEFORE_DROP_COLUMN	AFTER_DROP_COLUMN
		BEFORE_PURGE_TABLE	AFTER_PURGE_TABLE
システム制御[1)2)]	プロセス，スレッド，及びトランザクションの開始終了時点，コミット処理時点で呼び出される	START_PROCESS	TERMINATE_PROCESS
		START_THREAD	TERMINATE_THREAD
		BEGIN_TRANSACTION	PREPARE_COMMIT
		–	COMMIT
回復処理[1)2)]	ロールバック，ロールフォワードの開始終了時点，ロールバック処理時点，及びロールフォワード処理時点で呼び出される	START_ROLLBACK_PROCESS	TERMINATE_ROLLBACK_PROCESS
		START_ROLLBACK	TERMINATE_ROLLBACK
		ROLLBACK	–
		START_ROLLFORWARD_PROCESS	TERMINATE_ROLLFORWARD_PROCESS
		ROLLFORWARD	–
インデクスメンテナンス[2)]	インデクスの定義、初期化、作成，削除，及び遅延更新の各処理時点で呼び出され，またインデクスのエントリ登録，検索，更新前後，削除の各処理時点で呼び出される	DEFINE_INDEX	BUILD_INDEX
		INDEX_SEARCH	INDEX_COUNT
		INDEX_INSERT	–
		INDEX_BEFORE_UPDATE	INDEX_AFTER_UPDATE
		INDEX_DELETE	–
		DROP_INDEX	PURGE_INDEX
		INITIALIZE_INDEX	INDEX_MAINTENANCE_DEFERRED

注記
1) データプラグインモジュールで呼び出される
2) インデクスプラグインモジュールで呼び出される

プラグイン IDL で実装関数に SCAN_TYPE を指定すると，ORDBMS はインデクススキャンの処理形態をとり，その関数により選択された行について繰り返し処理を行うようになる．

(b) プラグインモジュールの実装関数同士での値の受け渡し（setter）

プラグインモジュールの実装関数同士での値の受け渡しを可能にする．

図 3.5(a) の例では，setter を用いて，実装関数 _phsgml_score() が _phsgml_contains_with_score() から値を受け取るように指示している（図中では破線で対応関係を示す）．

_phsgml_contains_with_score() は，n-gram インデクスプラグインモジュールを利用し，全文検索するとともに，その検索でのスコア情報 r_score を得る．_phsgml_score() は，そのスコア情報を引数 score_inf で受け取り，結果としてスコア値を返す．

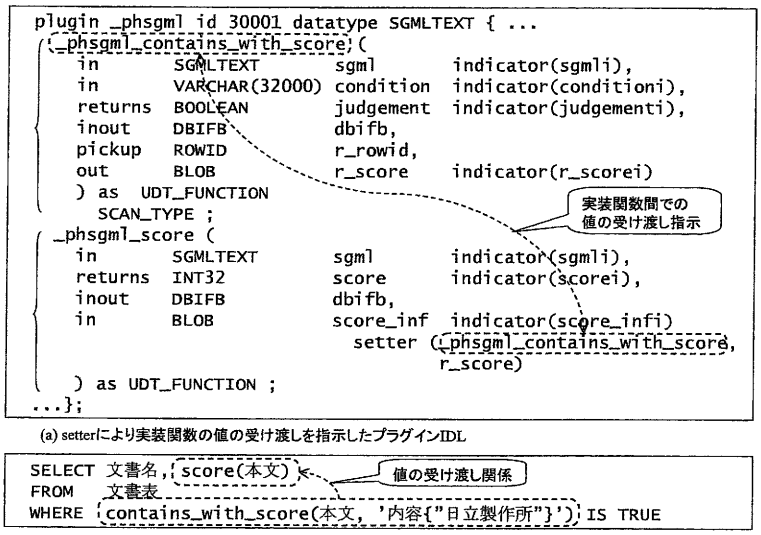
これを利用する SQL 文を図 3.5(b) に示す．ルーチン contains_with_score() の全文検索で絞り込んだ文書について，それぞれの文書名と全文検索でのスコア値を得る SQL である．

ルーチン contains_with_score() 自体は，WHERE 句の条件として指定される述語であり，真偽の値のみを返すだけで，スコア値を返すことができない．そこで，検索結果でスコア値を返すには，SELECT 句の射影項目にスコア値を返す指示が必要がある．setter を用いた実装関数 score() の仕様定義よってスコア値を返すようにすれば，SQL では直観的な記述となる．即ち，スコア値の受け渡しを SQL で指定する必要がない．

従って，setter 指定により ORDBMS がプラグイン実装関数間の値の受け渡しを支援することにより，このような SQL 文を効率良く処理することができる．

(c) カウントサロゲート関数（count_surrogate）

検索条件に合致したレコードの件数のみを返す実装関数を指定できる．件数の算



(a) setterにより実装関数の値の受け渡しを指示したプラグインIDL

(b) 実装関数間での値の受け渡しが行われるSQL
ルーチンscore( )の実装関数は、contains_with_score( )の実装関数の処理結果で得られるスコア値r_score
を受け取って、その結果を返す。

図 3.5: プラグイン実装関数同士での値の受け渡しの例

出に特化した関数を代わりに実行することにより，処理の高速化が図れる．

図 3.6(a) の例では，実装関数_phsgml_contains() に，カウントサロゲート関数として_phsgml_contains_count() を指定している．これに従い，図 3.6(b) のようなSQL を実行する場合，ORDBMS はルーチン contains() に対応付けられている実装関数_phsgml_contains() を起動する代わりに，_phsgml_contains_count() を起動する．_phsgml_contains_count() は，カウントサロゲート関数として，件数値を返す引数 r_count を持っている．ORDBMS はこの引数から得た値を件数値とする．

UDT，プラグイン IDL，及びプラグインモジュールの関係

プラグイン IDL 定義では，表 3.1 のプラグインモジュールを呼び出す契機で示すように，UDT 定義中でプラグインモジュールによる実装であることが指定されている外部起動ルーチンが起動されたときに，どのような引数を必要とするかを，それぞれの実装関数毎に指定する．また，UDT 定義のルーチンに対応した実装関数を呼び出すルーチン呼び出し契機以外では，行データの挿入，更新，及び削除のタイミングで呼び出されるデータ操作契機，トランザクション処理，コミット処理に関連するシステム制御契機，データベースの回復処理に関連する回復処理契機，及びインデクス処理に関連するインデクスメンテナンス契機の指定が可能である．

UDT で定義された SGMLTEXT 型は，利用者に対して contains()，contains_with_score()，score() というルーチンを提供する．これらのルーチンは，それぞれ，プラグインモジュール内の実装関数では，_phsgml_contains()，_phsgml_contains_with_score()，_phsgml_score によって実装されることが，SGMLTEXT 型の UDT 定義によって指定される．これらの実装関数が PPI を使用する際に必要な引数は，プラグイン IDL で指定が可能である．図 3.4 の記述例では，利用者によって呼び出されたルーチン contains(sgml,condition) は，対応する実

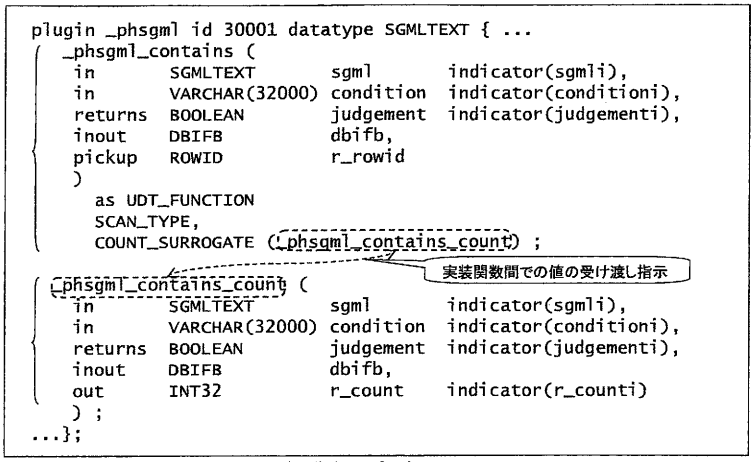


図 3.6: カウントサロゲート関数の例

装関数の呼び出し時には，_phsgml_contains(sgml, sgmli, condition, conditioni, judgement, judgementi, dbifb, r_rowid) として，6 個の引数が付け加えられる．これらは，実装関数の戻り値 (judgement)，引数の標識変数 (sgmli,conditioni,judgementi)，DB リソースへのアクセスを行うための引数 (dbifb)，検索結果として返される行識別子 (r_rowid) からなる．また，プラグインモジュールの初期化，終了処理やトランザクションとの同期処理をプラグインモジュールが行う必要がある場合に備えて，それらの契機でプラグインモジュールの関数を呼び出すような指定も可能になる．

さらに，データプラグインモジュールは，実装関数群からなるプラグインモジュール，それに対応する UDT 定義，及びプラグイン IDL から生成されるプラグインモジュール制御情報の 3 つから構成される．プラグインが ORDBMS に導入される場合，UDT 定義とプラグインモジュール制御情報がディクショナリに読み込まれ，一方でプラグインモジュールが UDT 定義の EXTRENAL NAME 句で指定される共用ライブラリの位置に組み込まれる．

プラグイン IDL コンパイラ

プラグイン IDL コンパイラは，プラグイン IDL の記述を解析し，ORDBMS が認識可能な定義情報や，プラグイン開発に必要なファイルなどを自動生成する．プラグイン IDL が生成するファイルを以下に示す．

1. プラグイン定義情報ファイル

   この定義情報を ORDBMS の意味解析・最適化が判断して，ORDBMS と密接に連携したプラグインの実行制御を行う．プラグイン登録コマンドを用いて，DB システムにプラグインを登録する場合，この定義情報をディクショナリに登録する．

2. スタブモジュールのソースファイル

ORDBMS からプラグインモジュールを呼び出す際の仲介をするスタブモジュールのソースコードを自動生成する．このソースをコンパイルしたオブジェクトをプラグインモジュールに加えておく．

3. プラグインモジュール実装ソースのテンプレートファイル

   プラグイン開発者は，このファイルをもとにプラグインを開発することができる．プラグインモジュールを作成する makefile も生成する．

図 3.7 に，プラグイン IDL と出力するファイルの概要を示す．

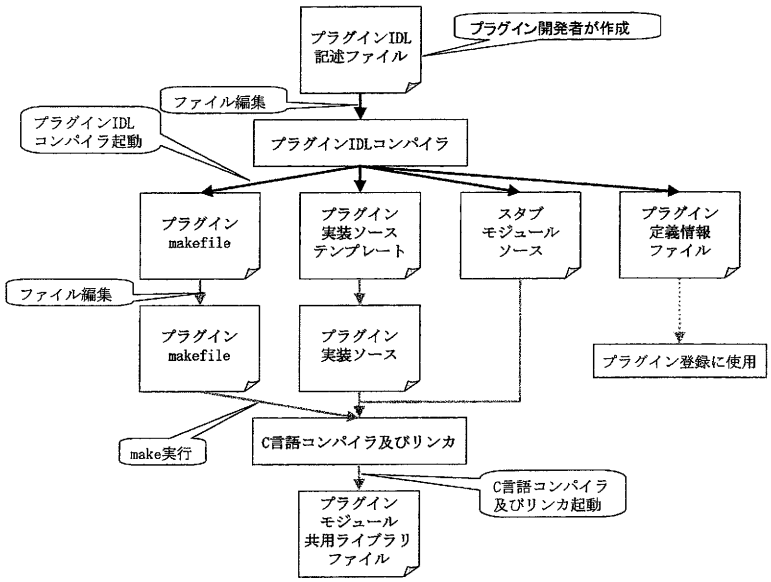


図 3.7: プラグイン IDL コンパイラと出力ファイルの概要

3.3.4. プラグイン・プログラミング・インタフェース

PPI は DB リソースへのアクセス手段を提供する．関数インタフェースで容易に ORDBMS のログベースの回復処理を呼び出すことができ，ACID 特性の耐久性が実現できる．具体的には，データ操作，データ制御，ファイル操作，及びサービスに区分する関数インタフェースを開発した．表 3.2 にその種類と機能を示す．これら PPI で提供する機能は，ORDBMS の実現のために必要とされる機能要件を満たすものである．

プラグインモジュールは，制御が渡されると DB リソースをアクセスしながら要求された処理を実行することができる．この DB リソースへのアクセス方法は PPI である．具体的には，C 関数呼び出しによるアクセスインタフェースであり，UDT インスタンス，データページ，及びファイルシステムへのアクセス，排他制御，プラグイン独自の回復を行うためのログ取得の操作が提供される．これらの操作を使用するためには，DB リソースへのアクセスを行うための引数が必要になる．この引数は利用者から与えられるものではなく，プラグインモジュールが起動される際に，ORDBMS から引数として取得されなければならない．

表 3.2: PPIの種類と機能

機能		処理内容
データ操作	検索処理の制御	インデクスプラグインモジュールによりインデクスが設定されて利用できる状態にあるか，1SQL内で初回の検索要求であるか否かを確認する．
	UDT値の操作[1)]	UDT値の生成，属性値の設定及び取得を行う．
	LOB値の操作[1)]	LOB値の生成，取得，及び更新を行う．
	インデクス検索[1)]	インデクスプラグインモジュールによりインデクスが張られていれば，インデクスの検索要求を発行する．
	インデクス更新情報の取得[2)]	表データが更新された場合，インデクスに関する更新情報を取得する．
データ制御	ログデータ取得	プラグインモジュールが回復処理を行うためにログデータを取得する．取得したログデータは，プラグインIDLで指定される回復処理契機で該当するプラグインに渡す．
	排他制御	プラグインモジュールで決められた規則に従い，資源名を用いて排他制御を行う．
ファイル操作	論理ファイル操作	ORDBMS管理のLOB用格納領域にページ単位で読み書き可能な論理的なファイルを作成する．トランザクションとの同期を保証し，バックアップの対象となる．
	DBファイル操作	DBファイルの生成，削除，読み書きを行う．ただし，ファイル内容への変更に関して，トランザクションとの同期は取らず，バックアップの対象にはならない．
	OSファイル操作	OSファイルの生成，削除，読み書きを行う．ただし，ファイル内容への変更に関して，トランザクションとの同期は取らず，バックアップの対象にはならない．
サービス	UDT列情報の取得[1)]	データ操作の対象列に関するディクショナリ情報(表名称，列名称，プラグイン句で指定する任意の文字列)を参照する．
	レジストリ参照サービス	プラグインモジュール固有の制御情報を取得する．
	レジストリ更新サービス	プラグインモジュール固有の制御情報を更新する．
	OSサービスインタフェース	OSシステムコールを直接発行せずに，このOSサービスインタフェースを介して間接的に呼び出す．
	エラー通知	プラグインモジュール内部で発生したエラーをODBMSに通知する．

注記
1) **データプラグインモジュールだけで呼び出される**
2) **インデクスプラグインモジュールだけで呼び出される**

このために，プラグインIDLで，プラグインモジュールの処理で使用するために必要な引数を取得できるようにインタフェースを定義する必要がある．

なお，プラグインの種別（データプラグインモジュール及びインデクスプラグインモジュール）により要求される機能の差があるため，プラグイン種別により使用可能なPPIを分けている．

3.3.5. データプラグインモジュールとインデクスプラグインモジュールの分離

データ処理機能を提供するデータプラグインモジュールと，インデクスに特化した機能を提供するインデクスプラグインモジュールを分離した．

インデクスプラグインモジュールにより，ORDBMSが標準に提供しているB-tree方式のインデクス機能以外に，新しいインデクス機能を追加することができる．

本論文では，SGMLパーサなどデータ操作を処理する部分と，全文検索を高速化するためのn-gram方式のインデクスを実装する部分を独立させ，それぞれプラグインで組込めるようにする．これにより，n-gram方式のインデクスプラグインモジュールをSGML以外のデータにも適用可能とする．

データプラグインモジュールから，PPIを経由してインデクスプラグインモジュールの検索機能を利用することができる．特定のデータプラグインモジュールからではなく，複数のデータプラグインモジュールから共通に利用することもできる．

インデクスプラグインモジュールには，CREATE INDEX文，DROP INDEX文などのインデクス実装に固有の呼び出し契機がある．この契機を利用して，インデクスを保守する

ことができる．例えば，大量の文書の登録・更新時には一時的にインデクスを削除しておくことにより，インデクス保守の効率化を図ることができる．

3.3.6. プラグインオプションとレジストリ表

表の列にSGML文書を格納する際，SQLによる列単位の操作では，その列に格納するSGML文書は同一の文書構造を持つことが期待される．この場合，列単位で文書構造情報などを指定することが必要になる．即ち，列単位でプラグインに必要な情報を保持する．

そこで，列単位にプラグインに関連する情報を指定可能にするプラグインオプションを提供する．また，この関連情報を保持するためのレジストリ表を提供する．

図3.8は，SGML文書の解析に必要なDTD(Document Type Definition)を列に対応させて定義した例である．

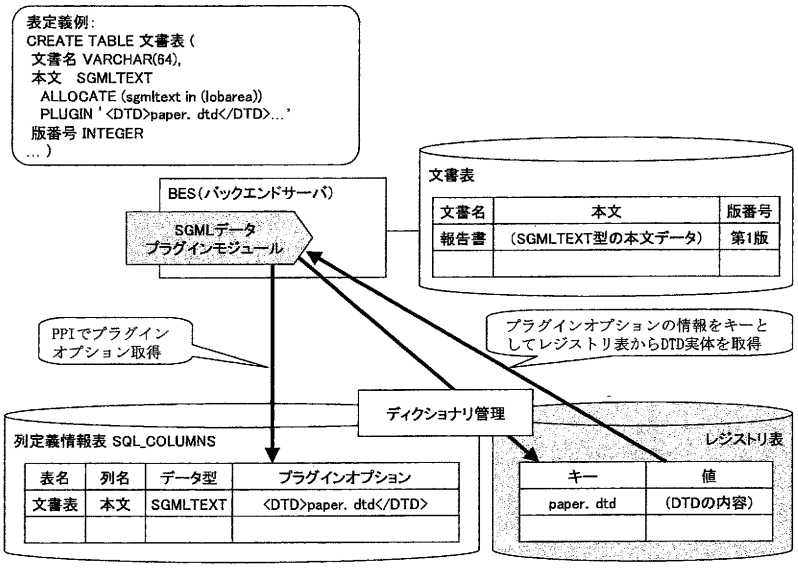


図3.8: 列定義でのプラグインオプション指定の例

表作成時に，列定義のプラグインオプション（PLUGINに続く' 'で囲まれた文字列）にDTD名称を指定する．レジストリ表には，DTD名称をレジストリキーとし，DTDの内容を値として登録する．プラグインは，PPIを用いて列に定義されたDTD名称を取得し，それをレジストリキーとして対応するDTD内容を読み出すことができる．

なお，SQL文で処理対象となるすべての列のプラグインオプションは，SQL文の処理に先立ってFESのSQL解析部がディクショナリから読み出し，中間コードに保持する．よって，プラグインからのプラグインオプションの参照は，メモリ参照であり高速に行える．レジストリ表の値もディクショナリサーバやFESでキャッシングしており，サーバ間の交信を削減している．

3.4. 適用評価

マルチメディアデータを扱う事例として，全文検索，文字列検索，XML 検索，画像検索，及び空間検索のプラグインを開発し，その適用性について評価する．表 3.3 は，それらプラグイン群で実装されたデータプラグインモジュール，インデクスプラグインモジュールの組合せ，及び提供される代表的な操作群である．

表 3.3: プラグイン実装例

種別	データプラグインモジュール	インデクスプラグインモジュール	ルーチン一覧		
			入力操作	出力操作	検索操作
全文検索[1]	SGMLTEXT	NGRAM[2]	-SGMLTEXT	-extracts -score	-contains -contains_with_score
文字列検索	FREEWORD	IXFREEWORD[2]	-FREEWORD	-extracts -score	-contains -contains_with_score
XML検索	XML	IXXML[2]	-XML	-extracts -score -XMLQUERY[3]	-contains -contains_with_score -XMLEXISTS[3]
画像検索	IMAGEFEATURE	FEATUREINDEX[4]	-IMAGEFEATURE	-GETDISTANCEIMAGEDATA -GETDISTANCEFEATUREDATA -GETFEATUREDATA	-SEARCHIMAGEDATA -SEARCHFEATUREDATA
空間検索	GEOMETRY	SPATIAL[5]	-GEOMETRY -GeomFromText -GeomFromWKB	-extracts -AsText -AsBinary	-Within -WithinRough -IntersectIn -IntersectInRough

注記
1) SGML文書及びXML文書の構造名を指定した検索も可能
2) n-gram方式のインデクスを実装
3) W3C XPath 1.0及びXQuery 1.0で規定
4) 多次元木方式のインデクスを実装
5) 四分木方式のインデクスを実装

3.4.1. UDT によるプラグイン拡張について

UDT によるプラグイン拡張によって，マルチメディアデータを扱うことの適用性を確認した．

1. マルチメディアデータ管理の一元管理

   従来のマルチメディア管理システムでは，メディア管理サーバと DB サーバとを独立に同時に管理することが一般的であった [67]．このような形態では，システム運用管理において，メディアデータと DB データとの間で整合性の矛盾が無いように管理する必要があり煩雑であった [61]．

   本論文では，マルチメディアデータを ORDBMS で一元管理することができるようになった．既存の RDB 資産を活かした上で，マルチメディアデータも管理することができる．一元管理することにより整合性を維持する運用が容易になる．さらに，マルチメディアデータを格納する単位は列であり，その列に回復処理を有するプラグイン

モジュールからなるUDTを対応付けるので，ORDBMS全体として回復処理が実現できる．

2. SQLインタフェースで統一したマルチメディア操作

   従来のメディア管理サーバとDBサーバを同時に利用するシステムでは，DBにアクセスする場合はSQLインタフェースで，メディアデータにアクセスする場合はサーバ独自のインタフェースを利用しなければならなかった．このような形態では，アプリケーション開発が煩雑であった．

   本論文のRDBモデル拡張では，マルチメディアデータのアクセスをSQLインタフェースに統一することにより，アプリケーション開発が容易になる．

   また，オブジェクト指向モデルを取り入れることにより，オブジェクト指向のアプリケーションとの親和性が高く，生産性の向上を図ることができる．本論文で示すORDBMSは，データベース言語SQLの標準仕様[51]に準拠し，利用者定義型に基づく型継承，多重定義を実現する．

   表3.3では，各プラグインで提供されるデータ型で提供されるルーチン群を示す．アプリケーションから同一の操作概念については，総称的に同じルーチン名を付けることでオブジェクト指向のアプリケーションとの親和性に配慮した．例えば，全文検索，文字列検索，XML検索については，いずれもテキストを対象にするので，検索条件contains及びcontains_with_score，ランク値算出scoreは，同じルーチン名としている．また，検索した対象を取り出す出力操作extractsについては，全文検索，文字列検索，XML検索に加えて，空間検索でも，同じルーチン名としている．

3.4.2. プラグイン機構について

これら全文検索，文字列検索，XML検索，画像検索，及び空間検索に対応するプラグインモジュールをDBシステムに組込んだ結果から，プラグイン機構の適用性を確認した．

- 本論文で提案するプラグインIDLを基に，プラグインの外部仕様を規定してプラグインモジュールを開発した．
- ルーチンをプラグインの実装関数としてUDTで定義し，ルーチン呼び出しを含むSQLを実行した．ORDBMSと密に連携したプラグインの実行制御を確認することができた．
- 複数のBESによる並列DB環境で，プラグインを並列に実行することができた．
- PPIが提供する回復機能により，プラグインからDBリソースをアクセスすることでデータベースの回復処理を実現した．
- データプラグインモジュール及びインデクスプラグインモジュールを分離して開発し，それぞれ独立に組込んで実行した．また，インデクスプラグインモジュールの共有を図ることができた．

具体的には，プラグインを，メディア操作に対応する機能をORDBMSに追加するデータプラグインモジュールと，メディアに対する高速検索機能を追加するインデクスプラグインモジュールに分類し，それぞれについて定義を規定した．データプラグインモジュールの外

部仕様は，UDT 定義によって規定され，その実装はプラグインモジュールと呼ばれるプログラムによって行われる．一方，インデクスプラグインモジュールは，利用者から直接的な呼び出しインタフェースを持たないが，インデクスの保守のために規定する実装関数を，プラグイン IDL を用いてインタフェースの記述を行う．

また，プラグイン IDL では，UDT 定義中でプラグインでの実装を指定されているルーチンが呼び出された場合，プラグインモジュールのどの実装関数が呼び出されるかを指定できるような仕様とした．さらに，それぞれの実装関数が呼び出される際に ORDBMS の実現に必要な引数をプラグイン IDL で指定する仕様とした．この場合，UDT 定義で指定されたルーチン呼び出しの他に，更新処理およびシステム制御系の呼び出し（トランザクションの開始終了時，セッションの開始終了時，回復処理の開始終了時，インデクス保守時）の定義を指定する仕様とした．

表 3.3 では，各プラグインで提供されるデータ型とそのインデクス実装との対応付けを示す．例えば，全文検索，文字列検索，XML 検索については，いずれもテキストを対象にするが，それらのテキストを対象にする検索を高速化する手法はいずれも n-gram 方式で実装されている．もし，データプラグインモジュール・インデクスプラグインモジュールを分離せずに開発すると，同じ n-gram 方式を採用しても個別の実装が必要となる．一方で，データプラグインモジュール・インデクスプラグインモジュールを分離することで，実装の独立性が高まり，拡張性も容易となることが期待できる．

一方で，いくつかの考慮点が考えられる．

1. プラグインモジュールは，DB 処理と同一のスレッドで実行されるので，スタックサイズの制約，システムコールの発行制限に配慮した難易度の高いスレッドプログラミングが要求される．

2. データベースの回復処理に対応したプログラム設計及び実装を必要とされる．

3. ORDBMS の実行環境中でプラグインモジュールを DB 処理スレッドと同一のスレッドで実行するので，DB 処理のプロセスサイズが増大するため，スループットへの影響を測る必要がある．

3.4.3. プラグイン方式の評価

データへの操作機能を，3.2 節に示すルーチンの実装方式に従い，SQL 起動ルーチン方式，外部起動ルーチン方式，及びプラグイン方式で評価する．表 3.4 は，データへのアクセス方法，記述容易性，及び処理性能で評価を行う．

データへのアクセス方法について，SQL 起動ルーチン方式は，計算完備な SQL 手続き文を用いて DB 参照，DB 更新が可能ではあるが，スキーマオブジェクトだけ操作が可能である．一方，外部起動ルーチン方式及びプラグイン方式は，DB 参照がルーチン呼び出しの引数で渡される値でのみではあるが，C 言語などプログラミング言語で記述される種々のメディア対応の処理が組み込み可能である．

また，記述容易性について，SQL 起動ルーチン方式は，スキーマオブジェクトではないトランザクション情報，ログ情報，及び格納領域情報からなる DB リソースへのアクセス手段はない．しかし，外部起動ルーチン方式及びプラグイン方式は，これら DB リソースへのアクセス及び回復手段をもつ．ただし，外部起動ルーチン方式は，トランザクション制御及び

表 3.4: プラグイン方式の評価

	SQL起動ルーチン方式	外部起動ルーチン方式	プラグイン方式 (プラグイン機構による 外部起動ルーチン)
記述言語と 処理方法	計算完備なSQL手続き文 でスキーマオブジェクト だけ操作可能	プログラミング言語で 種々のメディア対応の 処理が組み込み可能	プログラミング言語で 種々のメディア対応の 処理が組み込み可能
DB参照手段	select文	ルーチン呼び出しの 引数で渡される値のみ	ルーチン呼び出しの 引数で渡される値のみ
DB更新手段	insert文, update文 delete文	—	—
DBリソース[1)]へのデータアクセス	×	○	○
DBリソースのデータ回復手段	×	○	○
DB操作の 記述容易性	—	×[2)]	○[3)]
処理形態	DB処理スレッド実行 (インタプリタ実行)	外部プロセス実行 (ネイティブ実行)	DB処理スレッド実行 (ネイティブ実行)
処理速度	× (インタプリタ実行)	△ (アドレス空間切替)	○ (同一アドレス空間)

凡例：1) トランザクション情報，ログ情報，及び格納領域情報
2) トランザクション制御及びDB回復処理に対応した順序，タイミングで処理を行わないと，DB破壊を招く可能性あり
3) DBリソースを利用するDB操作，トランザクション制御及びDB回復処理に対応した順序，タイミングの処理契機に応じる処理の記述が，プラグインIDL(Interface Definition Language)及びPPI(Plug-in Programming Interface)により可能

DB 回復処理に対応した順序，タイミングで処理を行わないと，DB 破壊を招く可能性がある．プラグイン方式は，DB リソースを利用する DB 操作，トランザクション制御及び DB 回復処理に対応した順序，タイミングの処理契機に応じる処理の記述が，プラグイン IDL 及び PPI により可能である．

さらに，処理性能について，SQL 起動ルーチン方式は，DBMS と同一アドレス空間の DB 処理スレッドで実行されるが，インタプリタ実行となる．外部起動ルーチン方式は，ネイティブ実行ではあるが，DBMS とは異なるアドレス空間での外部プロセス実行となる．プラグイン方式は，DBMS と同一アドレス空間の DB 処理スレッドで実行され，かつネイティブ実行となる．

以上から，プラグイン方式は，性能を維持するためにネイティブ実行による外部起動ルーチンを DBMS と同じアドレス空間で実行するが，一方で回復処理を記述する外部起動ルーチンを実行することにより引き起こされる不具合でデータベースを破壊する可能性をなくすために，プラグイン IDL 及び PPI を活用することで，DB リソースを利用する DB 操作，トランザクション制御及び DB 回復処理に対応した順序，タイミングの処理契機に応じる処理の記述を可能としている．これらによって，信頼性の確保及び性能向上の達成，及びそれらの両立を達成しており，SQL 起動ルーチン方式及び外部起動ルーチン方式より優位である．

3.5. 関連研究

1. ルーチン呼び出し方式の高速化について

   ORDB は，従来の RDB をベースとし，オブジェクト指向モデルを取り入れた DB である．主要な RDBMS ベンダは ORDBMS を開発し，製品化している [50][62][66].

   この ORDBMS は，データベース言語 SQL の標準仕様に準拠する形で実装が行われている [63]. SQL データベースの上にオブジェクト指向機能を実現するために利用者定

義型が導入されている．利用者定義型では，値の振る舞いを規定する操作を関数として定義ができ，これらの操作はルーチンとして指定できる．ルーチンのインタフェースは利用者定義型の中で宣言され，そのルーチンの実装は利用者定義型とは別にルーチン定義によって行うことができる．具体的には，ルーチンの実装はSQL起動ルーチンあるいは外部起動ルーチンである．

DB2[62]は，外部起動ルーチンが実行するアドレス空間をDBMSと同じアドレス空間にするのか，あるいは異なるアドレス空間にするのかを選択できる．利用者定義関数の定義時に，FENCEDあるいはNOT FENCEDを指定する．NOT FENCEDを指定すると，当該関数はDBMSと同じアドレス空間で実行される．しかし，関数の呼び出しが高速化される利点を有するものの，関数の動作で引き起こされる突発的な不具合からデータベースが破壊される可能性がある．そのため，NOT FENCEDを指定する利用者定義関数は，DBA権限を有する利用者が定義可能とし，権限管理の配下に置く．また，利用者定義関数のテスト段階では，DBMSと異なるアドレス空間で実行するFENCEDを指定し，正しく動作することが信頼できれば，NOT FENCEDを指定することを推奨している．

提案方式では，性能を維持するために外部起動ルーチンをDBMSと同じアドレス空間で実行するが，一方で回復処理を記述する外部起動ルーチンを実行することにより引き起こされる不具合でデータベースを破壊する可能性がある．信頼性の確保及び性能向上は，トレードオフの関係にある．

2. 機能の組み込みの容易化について

Oracle[4]データカートリッジ[66]の実装において，インデクスと表データとの一貫性の維持，ファイル及びデータベースとページなど永続化記憶との間の管理（トランザクション，バックアップ，データベース回復，記憶領域割り当て）に留意することが示されている．また，静的変数を使用せずにスレッドセーフに関数を記述し，OSシステムコールは発行しないなどのコーディング規則が明記されている．

Informix[5]は，DataBlade[6]と呼ぶモジュールを追加する機構を有する[50]．種々のマルチメディアに対応するDataBladeを有する．このDataBladeではアクセスメソッド実装で，インデクスに対するロック確保，解放によって，DBMSのロック管理と連携が可能である．また，データを回復可能とする，ログ管理との連携も可能である．さらに，アクセスメソッドでは，ディスク上のページを扱うために，DBMSのバッファ管理と連携が行われ，当該バッファへのページ書き込みを制御することも可能である．また，インデクスによるアクセスメソッド実装には，インデクススキャンの開始，スキャン中のレコード取り出し，レコード操作（挿入，削除，更新），インデクススキャンの終了など，12個の関数が用意される[50]．

MySQL[7]は，Pluggable Storage Engine Architectureにてストレージエンジン APIを公開しており，InnoDB[8]など組込まれたエンジンでの稼動実績がある[64][65]．このス

[4]Oracle及びOracle Database 10gは，Oracle Corporation及びその子会社，関連会社の米国及びその他の国における登録商標または商標です．

[5]Informixは，米国およびその他の国における International Business Machines Corporationの商標です．

[6]DataBladeは，米国およびその他の国における International Business Machines Corporationの商標です．

[7]MySQLは，米国Sun Microsystems, Inc.の米国およびその他の国における商標または登録商標です．

[8]InnoDBはInnobase Oy社の商標です．

トレージエンジン API は，RDB で提供する既定義型に対して，最適化で生成する実行プランを実行解釈する処理で各行レベルのデータ操作及びインデクス操作を規定している．ただし，DB エンジン全体を組込む作業を前提にした API 仕様であり，マルチメディアに対応したデータ型拡張，DB 回復の処理規定，及び柔軟にモジュールを追加できる機構とはなっていない．

いずれも，API 仕様，及びマニュアルに記載されるコーディング規則に従って実装するしかないが，提案方式では，DB リソースを利用してデータベースへの操作，トランザクション制御及びデータベース回復処理に対応した順序，タイミングの処理契機に応じる処理の記述を可能とすることで難易度が高いと考えられるプラグインモジュールの組込みを簡単にしている．

3.6. おわりに

本論文では，様々なメディアデータの多様化・拡張に対応可能なオブジェクト・リレーショナルデータベース管理システムを開発し，オブジェクト指向アプリケーションと親和性の高い SQL インタフェースを提供することで，先進的なマルチメディア管理技術を取り込んで容易に拡張可能な DB システム基盤を提供することができた．具体的には，下記の特長を有するプラグイン機構を実装した．

- プラグイン実装のルーチン
- プラグイン・インタフェース定義
- プラグイン・プログラミング・インタフェース
- DB と同一スレッドでのプラグイン実行
- データプラグインモジュールとインデクスプラグインモジュールの分離
- プラグインオプションとレジストリ表

また，全文検索，文字列検索，XML 検索，画像検索，及び空間検索のプラグインを開発し，これらの技術の適用性を確認した．

1. マルチメディアデータ管理の一元管理マルチメディアデータを DB サーバで一元管理することができた．既存の RDB 資産を活かした上で，マルチメディアデータも管理することができた．一元管理することにより整合性を維持する運用が容易になった．さらに，ORDBMS の回復機能により高信頼なデータ管理が可能になった．

2. SQL インタフェースで統一したマルチメディア操作マルチメディアデータのアクセスを SQL インタフェースに統一することにより，アプリケーション開発が容易になった．また，オブジェクト指向モデルを取り入れることにより，オブジェクト指向のアプリケーションとの親和性が高く，生産性の向上を図ることができた．今後，マルチメディアデータを順次データベースシステムに取り込むことが益々要望され，このようなマルチメディアデータの追加が容易になることが期待できる．

第4章　Webフォーム基盤アーキテクチャの開発及びその応用事例

電子申請システム，ワークフローシステムなどでWeb化が進展している．本論文では，Webフォーム基盤のアーキテクチャとして機能配置方法，記述方法，及びデータ連携・交換方法を提案し，それを実装し電子申請システム，ワークフローシステムに適用した事例を示す．

4.1. はじめに

Webアプリケーションで構築される業務システムは，顧客からの要望に合わせて，短い周期で継続的に拡張や改善が行われている．表示層にはWebブラウザを利用し，HTTPやHTMLなどを解釈できればよいため，PCだけではなくPDAや携帯電話などでもオンライン情報の閲覧，オンラインショッピング，銀行の振込みなどのサービスが利用できるようになってきている．また，アプリケーション層にはJSP（JavaServer Pages），Servlet，EJB（Enterprise JavaBeans）などのJavaTM関連技術を利用し，Webフレームワーク（MVCモデル[68][69]）及びコンポーネント部品の機能配置及び記述方法に基き，各々の役割分担に従うことでシステム構築が可能である． したがって，表示層及びアプリケーション層での再利用性が高く，顧客の要望に素早く対応できるWebアプリケーション開発への期待が益々高まってきている．

一方，インターネット及びイントラネットを利用したWebシステムでは入力画面及び出力帳票開発の容易化，入出力データのXML基盤への対応などが要望されている．政府主導によるIT化推進，自治体や民間企業の電子申請システム，ワークフローシステムなどでWeb化が急速に進められている．このように，電子帳票，電子フォーム市場でのWeb化の要望が高まってきている．

これらの背景を踏まえて，Webアプリケーションでの電子フォームシステム・アーキテクチャのコンポーネント，機能構成，及びコンポーネント間インタフェースを開発し，また上記のアーキテクチャをWebフォーム基盤ミドルウェアとして実現した．具体的には，Webフォーム基盤アーキテクチャとして，Webアプリケーションフレームワークを支える機能配置方法，記述方法，及びデータ連携・交換方法を開発した．

本論文では，携帯電話などのモバイル機器を始めとして，RFIDなどユビキタス基盤を支える情報機器への対応も視野に幅広く適用が期待されているWebフォーム基盤のアーキテクチャを提案し，それらを実装し様々な応用事例に適用した結果を示す．

4.2. 現状と課題

Webアプリケーションは，アプリケーションの保守性を向上させるために，表示層及びアプリケーション層の各階層で機能分担する．図4.1は，Webアプリケーションの機能配置を示す．このような機能分担をMVCモデルへ適用すると，いくつか考慮するべき点がある．まず，表示層で行う単純データチェックや単純データ型(形式)変換と，アプリケーション層で行う複雑なデータチェックや複雑なデータ型(形式)変換の配置及び役割分担を，チェック処理間での重複を避け，お互いに矛盾がないように整合性を保ちつつ決めることが重要である．また，データの入出力に伴う画面と画面遷移が一体化してしまうと保守が困難になる．

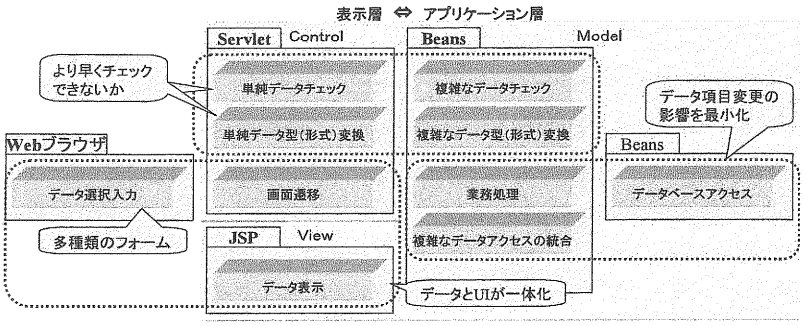


図 4.1: Webアプリケーションの機能配置

4.2.1. 表示と業務ロジック分離の難しさ

View部分のJSPは，JavaBeansあるいはサーブレットによって影響をうけない部分はHTMLで記述し，それらの結果によって動的に変更する部分は%～%で囲みスクリプトレットで$HTML$出力している．つまり，この部分は，業務ロジック（$JavaBeans$）あるいは制御フロー（$Servlet$）に影響を受ける．JSPで表示する定型コンテンツの量（この場合はデータ入力項目数など）が多くなってしまうと,%～%で囲まれたスクリプトレットが数多くなって，記述されたコードの可読性が悪くなり，再利用性が低くなる傾向になる．JSPは基本的に$HTML$の任意の場所に%～%で囲まれたスクリプトレットを書ける能力を持つ一方，$HTML$とコードの部分が混ざっているので可読性を犠牲にしている．この傾向は，記述量が増えると顕著になる．コードの可読性が低いということは，再利用できるかの判断が困難になることを意味している．また，JSPは本質的にコードであるがゆえに，表示のイメージを直感的に得ることは不可能である．$WYSIWYG$で見たとおりに印刷するには，さらに別途整形用のプログラミングを必要とする．

業務ロジックに専念して開発を行うべきであるが，画面，制御フロー及び業務ロジックとの連携部分に多大な工数を割かなくてはならない．View 部分で入力されたデータは，Model 部分で処理されるため，現実的には，画面の設計と業務ロジックの設計の分離が難しい．本来は早期の時点で画面と業務ロジックの整合性を取るのが望ましいが，実際は，利用者のイメージに合わず仕様変更になったりする．表示仕様は，業務要件の変化が激しいために，固定的に作成して終わりということではなく，変更を前提にして表示仕様を考えて行くことが重要になりつつある．一端，作成した仕様を，変更，保守まで考慮した開発の考え方が望まれていると言える．このことから，画面（View）と業務ロジック（Model）をもっと分離させて開発する技法が望まれる．

4.2.2. HTML フォームの限界

HTML を利用したフォームによって View 部分が実装されてきているが，HTML によるフォームの設計に様々な課題があり，それらの課題を解消する必要性に迫られている．HTML をフォームの基盤として利用する場合には，次の課題がある．

1. データとユーザインタフェース (UI) が一体化

   HTML では，フォームで扱うデータと UI が密接に連携している．フォームを定義する場合，データの扱いと UI は一体化してコーディングされ，当然ながら適用する UI は決まってしまう．例えば，予め用意する選択肢の中から排他的に項目を選ぶ場合はラジオボタンあるいはドロップダウンリストの UI を利用し，また同時に複数の項目を入力させる場合はチェックボックスの UI を利用する．HTML では一般的な UI だけが提供され，いずれの場合もデータと UI とは一体化して定義されている．必然的に UI が変更になれば，データ部分も影響を受けることになる．

2. 単純なデータチェックにもプログラミングが必須

   フォームの中で閉じて動的にデータ間でチェックが必要な場合があり，フォームへのデータ入力時点でできるだけチェックを済ませたい要望がある．例えば，旅費申請で宿泊費の上限があり，その上限額を超えて申請が行われていないか，宿泊期間の開始日と終了日が逆転して矛盾していないかなど，フォームに入力されたデータを申請時点でチェックできることが望ましい．また，入力データの範囲チェック，データ型のチェック，入力候補値の表示などの豊富な UI コントロールなどは，JavaScript などでさらにプログラミングを必要とする．さらに，このようなチェックを HTML で実装する場合，データや UI を動的に変更できないために，別の HTML フォームを予め作成しておくか，JavaScript によるプログラミングを駆使しなくてはならない．

4.2.3. 多種類の定型フォーム

Web アプリケーションの画面を目的別に分類すると，ナビゲート画面と業務画面に分けることができる．表 4.1 は，Web フォームの特徴を示す．

ナビゲート画面は，利用者の入力を促すための表示部分と利用者からの入力内容とからなり，それらの入力内容に応じて表現構造が可変となる特徴がある．用途としては，ログイン画面，業務選択画面，エラー画面などがある．一方，業務画面は，紙ベースの帳票と同様に

表 4.1: Web フォームの特徴

	フォーム （データ入力が中心）	レポート （データの加工・出力が中心）
静的コンテンツ （表現構造が不変）	定型フォーム （業務画面）	（目的別集約結果画面） レポート （ドリルアップ・ダウン画面）
動的コンテンツ （表現構造が可変）	非定型フォーム （ナビゲート画面）	

凡例）
業務画面：オーダエントリなど入力を中心
ナビゲート画面：ログイン画面，業務選択画面，エラー画面など

表現構造が不変となる特徴がある．帳票を用途で分類すると，オーダエントリなど入力を中心に行うフォームとデータの活用及び加工を中心に行うレポートに分けられる．

また，フォームは，定型フォームと非定型フォームに区別できる．定型フォームは表示される画面が固定化されている．一方，非定型フォームは表示される画面のコンテンツが，プログラム処理で自由自在に加工される．定型フォームには，単純な画面（入力確認，項目承認など）から複雑な画面（項目間での整合性を確認など）まである．業務の中で画面の遷移に対応して各フォームを用意する必要があるので，定型フォームは多くなる．そのため，画面間でのデータの受け渡しが簡単に記述できることが重要である．

図 4.2 は，典型的な申請業務の画面制御のための処理制御の流れを示しており，典型的な処理内容とデータ種別毎に，画面構成と画面遷移を要件に従って実装するナビゲート画面（非定型フォーム）が JSP，申請画面，承認画面など業務システムで代表的な業務画面（定型フォーム）が HTML から構成されていることを示している．ナビゲート画面は受注や出荷など時間に関するの業務上の時間制約，順序性を考慮し業務をナビゲートする用途で作成されるので，概して複雑な画面遷移を含むが種別数は数多くならない．一方，業務画面は業務内容毎に作成されるので，画面数が多くなる傾向にある．

4.2.4. 定型フォームと業務ロジックの依存性

画面（View）と業務ロジック（Model）の依存性を極力排除して，分離させることが重要であることを述べてきたが，さらに利用者からフォームに入力されたデータとサーバで処理されるデータを分離させる必要もある．この課題を解決するために，なぜ XML が基盤として必要であるか，考えてみたい．定型フォームを利用する Web 環境での業務システムを開発する場合，開発者は何に留意するべきであろうか．

利用者からは，入力のチェックを柔軟に行いたいとか設計時点とテスト時点で画面を変更，追加して欲しいなどと様々な要望がある．また，実際に HTML フォームでは，HTTP 基盤

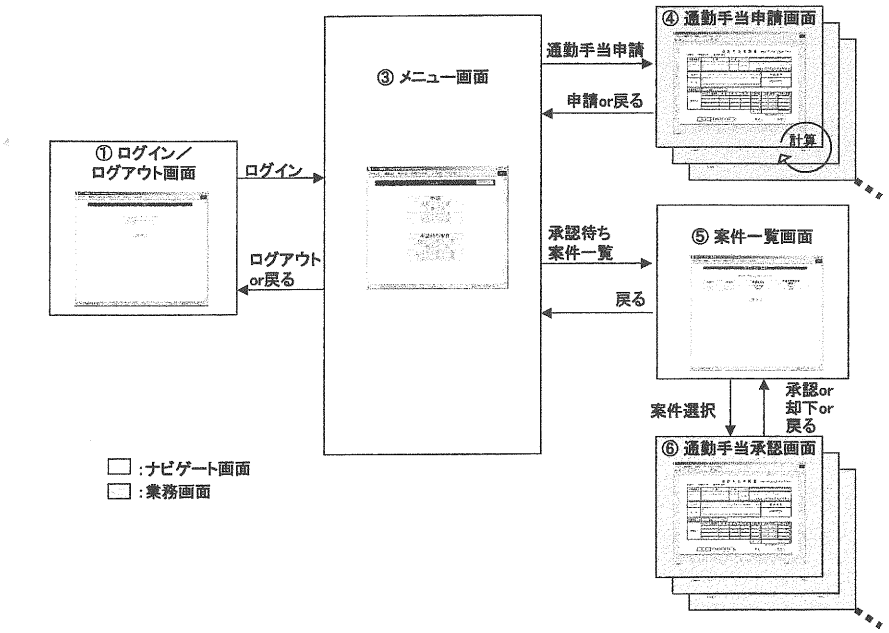


図 4.2: アプリケーション例の画面遷移

を利用して Web ブラウザから入力されたデータがサーバへ転送され，定型フォームで入力されたデータを業務に受け渡すだけではなく，データベースに格納することまでも必要としている．そのため，そのデータを処理するサーバの業務システムでは，データの項目名及びデータ型が変更されるために，データ構造がフォームに依存する問題がある．当然ながら，変更されたフォームに対応する業務システムのプログラムも影響を受けることになる．

このように，画面に着目する利用者と業務仕様に着目する開発者との間で，相互に意見を調整して対処することになる．開発者からは，業務システムのキーとなる開発項目をあらゆる場面で支援することが要請されていると言える．このようなデータ交換・格納用途に合致する基盤として，利用者及び開発者から見て XML を基盤として採用することは必然である．

4.3. アプローチ

Web アプリケーションの開発基盤に Web フォーム基盤が加わると，開発する手順及び構成が図 4.3 に示すように変わる．

MVC モデルを Web フォーム基盤に適用した Web アプリケーションの開発では，MVC モデルの Control 及び View に適用されることになり，Model である業務ロジックとは無関係に影響を与えず開発効率や操作性を向上させることができる．即ち，Web アプリケーションの開発基盤として，Java あるいは J2EE(Java2 platform Enterprise Edition) のフレームワークを利用して Web フォーム基盤で実現する [69]．HTML の画面では実現できなかったきめ細かなユーザインタフェースを提供することで，Web アプリケーションの開発課題を解決する．

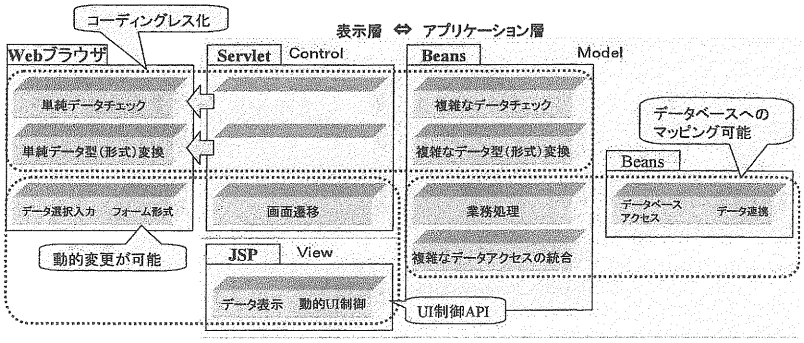


図 4.3: Web フォーム基盤を利用した Web アプリケーションの機能配置

Web フォーム基盤を利用する Web アプリケーションの特徴は以下の通りである.

- JSP で HTML によって記述されるビュー部分のロジックが，Web フォーム基盤の定型フォーム表示機能で置き換えが可能である．それによって，HTML で画面の配置，背景色など開発するコード量を大幅に減らすことができる.
- UI コントロールが業務仕様に沿っており，画面設計時点で設定される UI コントロールが Web アプリケーション実行時に，柔軟にかつ動的に制御選択可能であり，利用者の木目細かな要望に対応することができる．HTML フォームでは，Web ブラウザで単純なデータチェックやデータ型（形式）変換を行っていたが，Web フォーム基盤では GUI ベースでそれらの処理をコーディングレスで設定が可能である．即ち，新たに UI コントロールの実装のために別途 JavaScript によるプログラミングが不要である.
- Web アプリケーションの中で Servlet（画面遷移），JSP（データ表示），及び Web フォーム基盤（データ選択入力，単純データチェック及び単純データ型（形式）変換）と役割分担が明確に分離されるので，各処理部分の再利用が行い易い.

以下では，Web フォーム基盤の開発内容について詳解する.

4.3.1. データとユーザインタフェース (UI) を分離

Web フォーム基盤はデータと UI を分離する構造となっている．フォームを定義する際に，フォームに関連するデータ型及びデータ構造を指定し，個別のプロパティを除いて UI 種別を設定する．フォームを Web ブラウザで実行する際に，UI の個別プロパティを設定して各利用者に応じた UI のビューが決められる柔軟性を持っている．具体的なプロパティとして，入出力許可属性の設定，選択リスト及びバルーンヘルプの設定，配色属性などがある（表

2参照）．基本的に，Webフォーム基盤ではWebブラウザの実行時にUIの詳細を決めてフォームを設定しておけば，表示方法の拡張などにも合わせることも可能である．

また，一般的に画面の設計では，データ入力部分の表示項目名，桁数や入力形式などのデータ型，データ構造がフォーム毎に異なっており，画面部品の再利用を困難にしている．また，これらの木目細かな要件に対応して，各々JSPを駆使してコーディングすることが必要である．一方，Webフォーム基盤では，表示項目名やデータ型，データ構造を定型化したUIコントロールを導入することによって，表示される部品を簡単に貼り付ける操作だけで細かな位置決めを行ったり，テキストボックスの入力値をチェックしたりボタンが押されたらデータベースにデータを送るなどが出来るようになる．基本的にGUIベースの考え方で，入力項目の属性と処理を細かな部分まで設定することができる．従来まではJSPでコーディングされていた処理部分を，Webフォーム基盤設計時点にUIコントロールを貼り込む操作によって代替されていると言える．即ち，人手のコーディング部分は，フォームのソースファイルとして実装されている．このために，画面はコーディングレスで実現することが可能である．

図4.4はGUIベースでWebフォーム基盤設計時点にUIコントロールの貼り込む操作によってどのようなWebフォーム基盤のソースファイルが作成されるかを図示したものである．画面イメージで表示される情報は，二つに区分されている．まず，画面イメージの裏紙というべき背景レイアウトからなる「プレゼンテーション」である．この「プレゼンテーション」は，帳票ツール，オフィスツールやスキャナなどから読み取られて正確な位置情報とともに記述されている．一方，Webブラウザからデータ入力され，予め初期値として設定された「データ」とフォームを表現するデータ項目名，データ型，データ構造である「スキーマ」とからなる「モデル」である．Webフォーム基盤のソースファイルでは，「データ」とUIである「プレゼンテーション」が分離されて，個別のXMLタグが割り当てられて記述されている．例えば，通勤手当申請書の氏名の表示部分と「日立太郎」というデータは，Webフォーム基盤のソースファイルではマッピングによって関連付けが行われている．サーバに転送されるXMLは「データ」部分だけであるが，必要に応じてXSLTによるデータ変換処理が容易に実装できる．

4.3.2. 動的に初期値や入力制限を柔軟に変更が可能

Webフォーム基盤では，動的に初期値や入力制限を柔軟に変更することが可能となっている．即ち，初期値設定，入力属性の変更，選択リストの設定，バルーンヘルプの変更，表示色設定，印刷属性の変更などは，フォーム形式は一つであるが，場面によって初期値や入力制限したい範囲や強調したい部分が異なる場合への対応が可能である．例えば，ワークフローで申請者ごとに初期値（名前など）を変更し，また申請者と承認者で強調したい部分や入力制限する部分が異なるなど柔軟に変更したい場合に対応できる．

Webフォーム基盤で提供する様々なUIコントロールによる入力項目の設定のためのAPIは，表4.2に示す通りである．

4.3.3. XMLによるデータ管理の柔軟性

Webフォーム基盤では全面的にXML処理基盤を採用している．XMLによって，各種データの仕様が変更された際に大幅な時間とコストを掛けずに異なるアプリケーション間での

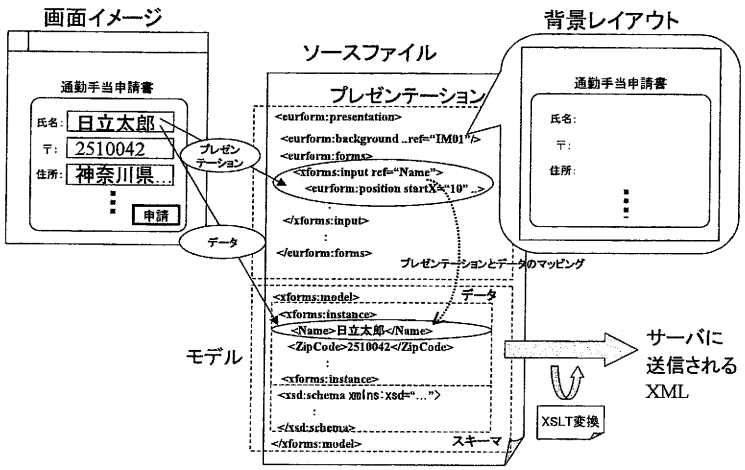


図 4.4: Web フォーム基盤の基本構造

データ連携，交換に対応することが期待されている。開発効率を向上させることができ，また項目の追加・変更が簡単に行えることで開発のスピードが早くなり，データの再利用性が高くなると言われている．業務処理，複雑なデータアクセスの統合（業務間での XML データ変換（例えば，申請業務と審査業務でデータを交換する例）など），及びデータアクセス（データベースへの格納（例えば，申請データが RDB にマッピングされて格納する例）など）への対応が重要である

具体的に，Web フォーム基盤は XML を採用したことによって次のメリットがある．

1. データ転送での XML 利用

   Web フォーム基盤では，Web ブラウザから入力されたデータは XML としてサーバの業務システムへ転送される．即ち，データに XML のタグ付けを行い，業務システムでのデータの扱いに基づいたデータの意味付けが可能であることを意味する．XML でデータを表現することで，データ構造の変更に左右されずに，XML タグを指定してプログラムで必要となるデータ項目を取り出せるようになる．フォーム形式を変更しても，送信するデータを XML で表現するので，プログラムへ影響を与えることはない．しかも，Web フォーム基盤はサーバの業務システムでのデータの扱いに合わせて，データ項目名，データ型，データ構造を XSLT 変換へ組み込むことも可能となっており，高い柔軟性を保持している．

   また，Web フォーム基盤では，HTTP 基盤に加えて SOAP 基盤もデータ転送のプロトコルとして利用することができる．Web フォーム基盤の作成時にいずれのプロトコルを利用するかを選択できる．データを取得する業務システムのプログラムは，HTTP の場合は Servlet あるいは JSP で作成する．また，SOAP の場合は，SOAP による呼出しを受け付けるサービスプログラムで作成する．今後増えるであろう Web サービス基盤へいち早く対応している

表 4.2: UI コントロールの機能一覧

機能	処理内容	UIコントロール名
UI初期値設定	各UIコントロールの初期値設定	EF, MLEF, DL, RB, CB
入力属性	データ入力可否の設定	EF, MLEF, DL
選択リスト・バルーンヘルプ設定	選択リスト及びバルーンヘルプの表示項目を設定	DL, RB, CB
表示色設定	文字色及び背景色の設定	EF, MLEF, DL, RB, CB
印刷属性	印刷対象可否の設定	EF, MLEF, DL
スクリプト機能	データ更新・送信実行時に実行させるスクリプト（項目間チェック，項目間演算結果の代入など）の指定	EF, MLEF, DL
データ属性	データの属性（初期値，データ種別（半角，全角，半角カナ，数字，半角英数字），桁区切り表示，制限指定（最大桁数あるいは最小値～最大値）の設定	EF, MLEF
入力制御機能	入力制御（必ず入力する）の設定	EF, MLEF
選択制御機能	選択制御（必ず選択する）の設定	DL

凡例　EF(EditField)　　：1行分のテキスト入力域
MLEF(MultiLineEditField)：複数行のテキスト入力域
DL(DropdownList)　　：リスト表示された文字列からの選択
RB(RadioButton)　　：ボタンによる単一選択
CB(CheckBox)　　：ボタンによる複数選択

2. データベースとの連携

データベースには，業務処理で利用する用途及びデータベース構造の種別に基づき三つにパターン分けして格納する．用途として，業務用データベース以外に，申請されるデータ，あるいは，Web フォーム基盤に表示する初期値データを管理するフォーム用データベースも考えられる．データベース構造には，XML データとしてそのまま格納することや RDB の正規化リレーションに格納することを想定している．

- フォーム用データベースにフラット形式で格納
- フォーム用データベースに XML データ形式で格納
- 業務用データベースに直接格納

業務処理では，例えば送信データの審査処理が複雑で，データの受付処理と審査処理を個別に非同期でおこなう場合，受付処理で，必須項目の有無など簡単な検証処理を行い，受付番号を発行し，フォーム用データベースに格納する．その後，夜間バッチ処理などにより，申請データを業務用データベースに登録し，審査処理では，データを部分的に取り出し，演算やマスタとの検証処理などを行うこともある．

4.4. 適用事例

マルチメディアデータを扱う事例として，全文検索，文字列検索，XML 検索，画像検索，及び空間検索のプラグインを開発し，その適用性について評価する．表 3.3 は，それらプラ

グイン群で実装されたデータプラグイン，インデクスプラグインの組合せ，及び提供される代表的な操作群である．

4.4.1. 電子申請システム

2001 年 3 月に e-Japan 計画が発表され，「電子商取引等の促進」などが重点政策として挙げられている [74]．この流れを受けて，電子政府の実現を目指し，財務規則等で定められている帳票及び文書を電子化することが要請されている．図 4.5 は，電子申請システムへの適用例である．地方自治体でも電子化すべき帳票様式が，4,000 を超えると言われており，総務省ガイドラインに準拠した受付処理・審査機能を実現する必要がある [75][76]．申請様式を GUI で容易に作成・拡張することで，申請業務の拡大に柔軟に対応でき，自治体毎の様々な申請様式にも対応できる電子申請システムが構築されている．

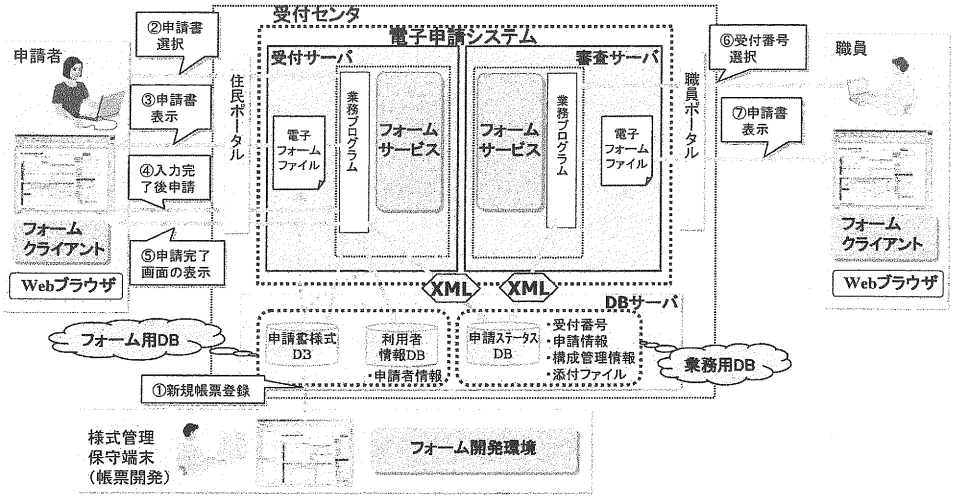


図 4.5: 電子申請システムへの適用例

4.4.2. ワークフローシステム

企業内で利用される各種申請書類，企業間で取り交わされる契約書や注文書，請求書などの帳票類の電子化が序々に進展し，企業内外での業務プロセスをインターネット上で電子的に処理するワークフローシステムが重要になりつつある．図 4.6 は，ワークフローシステムへの適用例である．紙帳票イメージのままで，しかも帳票スタイルの Web 画面でデータを入力し，直接ワークフローへの投入が可能である．帳票作成からワークフローまでプログラ

ムレスで提供されるので，基本的に業務のワークフローを Web ブラウザで電子帳票を利用した申請業務が違和感なく行える．

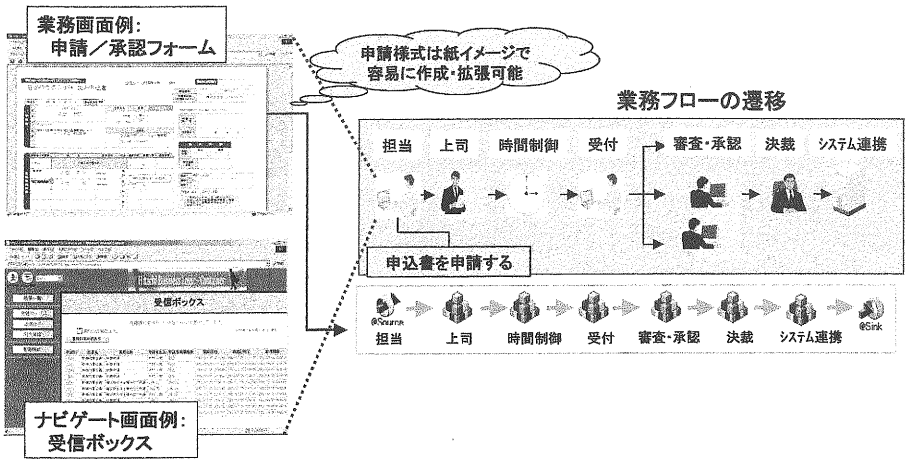


図 4.6: ワークフローシステムへの適用例

4.5. 関連研究

本論文と関連する主に Web 基盤でのフォームの研究内容及び標準化動向を示す．

中西は，帳票の表示パターンがデータモデル構造から演繹できることを利用して構築された「帳票の概念モデル」でフォーム生成機能を評価して，ユーザの満足度を高めるための改善点とフォーム生成機能のあり方を議論している [70]．今村らは，Web ブラウザを用いた製品仕様交換を容易にすることを目的として文書内容制約の入力時チェック機能をもつ XML 文書入力方式を提案している [71]．文書内容制約のチェックロジックが入力フォームを生成するプログラムから分離できるので，Perl や Java により入力フォームを直接プログラムする方式と比べて，文書内容チェック機能の保守を容易にすることができるとしている．野中らは，対話型ソフトウェアの画面仕様を，Web フォームの意図と表現方式を分離し記述する XForms 形式 [73] を利用して，対話のセマンティクス記述及びその計算機処理を記述する妥当性を述べている [72]．また，XForms 形式の画面仕様を用いて，対話型ソフトウェアの標準機能規模を自動計測する技法も提案している．

また，次世代 Web フォーム基盤（ポスト HTML，XHTML の代替）として W3C で XForms が規定されている．W3C は，XForms を次世代 Web フォーム規格として，広く一般の利用者及び開発者に対して，XForms 規格の実装と相互運用性の検証を呼び掛けている．XForms

が単なるフォーム記述仕様に留まらず，高度なWebアプリケーションを構築するための基盤技術との期待から，この規格の発展方向が注目される．XFormsは，表示される情報機器に合わせて画面を動的に対応させることを可能とする特徴があり，PDA，携帯電話などのモバイル機器を始めとして，将来的にはバーコード，ICタグ，RFIDなどユビキタス基盤を支える情報機器への対応も視野に幅広く適用が期待できる．

4.6. おわりに

インターネット及びイントラネットを利用したWebシステムで，業務の継続的な拡張，柔軟性の確保によって，さらなる業務効率の向上，スピード経営を目指す要望が益々高くなってきている．これらの要望に応えるため，Java及びXML基盤を基にしたWebフォーム基盤アーキテクチャを提案し，また帳票開発ツールで培った技術と融合させることで，利用者との直接的なインタフェースを支える表示基盤としてWebフォーム基盤ミドルウェアを開発した．既に，Webフォーム基盤ではWebブラウザから入力されたデータを，サーバへSOAP基盤を利用してXMLによるデータ転送に対応している．多様な業務システムの要件に合致させるために，Webサービスへの対応が要望されており，Webフォーム基盤でもWebサービスの進展に合わせて適用して行く．

第5章　結言

5.1.　成果のまとめ

5.1.1.　並列問合せ処理機構の開発

並列データベースシステムにおいて，問合せ処理の並列化方式を提案した．具体的には，SQL の解析処理によって並列性を導き，具体的に並列に実行するためのパイプライン並列及びデータ並列の処理手順の表現方法，問合せ変換方法，並列実行環境など，実適用を伴う問合せ処理機構の実現方式及びその評価を示した．

負荷平準化を目的とするフロータブルサーバ方式は，プロセサ間での処理負荷の平準化のために，DB を格納していないプロセサに処理の一部を分担させる方法である．これによって，ジョイン処理にデータ並列及びパイプライン並列を適用することで，応答時間が改善される．また，サーバ数を増やすことにより各サーバの処理手順を起動するための処理時間及びスケーラビリティとの関係を評価した．具体的には，負荷平準化を目的とするフロータブルサーバ方式を適用し，パイプライン並列の効果により従来と比較して処理時間比で最大約24 %削減されることを確認することで実システムへの適用が可能であることを示した．さらに，フロータブルサーバ方式を好適に活用するために 2 段階からなる SQL の最適化方式を提案した．SQL 文を SQL 受付サーバが受け取って解析処理する段階と，実際にその SQL の処理手順を実行する問合せの実行処理時に最適化する段階からなる．大規模な表に対するジョイン処理及びソート処理は，処理手順の選択を間違うと大幅に性能が低下する懸念があるが，2 段階の最適化方式は，これらの課題を解決するものである．この 2 段階最適化方式の負荷評価を行い，処理時間比で問合せ処理実行時間の高々約 3 %以下であることを確認することで実システムへの適用が可能であることを示した．

今後の課題について述べる．

- 並列ジョイン処理として，PNL 処理と PSM 処理を代替の処理手順として評価しているが，並列ハッシュジョインについても同様の評価を実施する．
- 千台を超えるサーバ環境でのシステム評価によって，提案方式の妥当性を検証する．
- ジョイン処理などの高負荷な問合せ処理とトランザクション処理を並行して実行する場合のトランザクション処理の応答性能への影響及びスループット評価

5.1.2.　プラグイン機構の開発

本論文では，様々なメディアデータの多様化・拡張に対応可能なオブジェクト・リレーショナルデータベース管理システムを開発し，オブジェクト指向アプリケーションと親和性の高い SQL インタフェースを提供することで，先進的なマルチメディア管理技術を取り込んで

容易に拡張可能なDBシステム基盤を提供することができた．具体的には，下記の特長を有するプラグイン機構を実装した．

- プラグイン実装のルーチン
- プラグイン・インタフェース定義
- プラグイン・プログラミング・インタフェース
- DBと同一スレッドでのプラグイン実行
- データプラグインモジュールとインデクスプラグインモジュールの分離
- プラグインオプションとレジストリ表

また，全文検索，文字列検索，XML検索，画像検索，及び空間検索のプラグインを開発し，これらの技術の適用性を確認した．

1. マルチメディアデータ管理の一元管理マルチメディアデータをDBサーバで一元管理することができた．既存のRDB資産を活かした上で，マルチメディアデータも管理することができた．一元管理することにより整合性を維持する運用が容易になった．さらに，ORDBMSの回復機能により高信頼なデータ管理が可能になった．
2. SQLインタフェースで統一したマルチメディア操作マルチメディアデータのアクセスをSQLインタフェースに統一することにより，アプリケーション開発が容易になった．また，オブジェクト指向モデルを取り入れることにより，オブジェクト指向のアプリケーションとの親和性が高く，生産性の向上を図ることができた．今後，マルチメディアデータを順次データベースシステムに取り込むことが益々要望され，このようなマルチメディアデータの追加が容易になることが期待できる．

このプラグイン方式により，耐久性(Durability)の実現と高速性の確保，かつそれらの両立，及びORDBMS(オブジェクト指向リレーショナルデータベース管理システム)製品でのSGML文書，文字列，XMLへの構造検索を始めとして，類似画像検索，空間検索として実用化することにより組込みが容易であることを確認した．

5.1.3. Webフォーム基盤アーキテクチャの開発

インターネット及びイントラネットを利用したWebシステムで，業務の継続的な拡張，柔軟性の確保によって，さらなる業務効率の向上，スピード経営を目指す要望が益々高くなってきている．これらの要望に応えるため，Java及びXML基盤を基にしたWebフォーム基盤アーキテクチャを提案し，また帳票開発ツールで培った技術と融合させることで，利用者との直接的なインタフェースを支える表示基盤としてWebフォーム基盤ミドルウェアを開発した．既に，Webフォーム基盤ではWebブラウザから入力されたデータを，サーバへSOAP基盤を利用してXMLによるデータ転送に対応している．多様な業務システムの要件に合致させるために，Webサービスへの対応が要望されており，Webフォーム基盤でもWebサービスの進展に合わせて適用して行く．

XML データベースの応用事例として，電子申請システム，ワークフローシステムなどで Web 化が進展している．本論文では，Web フォーム基盤のアーキテクチャとして機能配置方法，記述方法，及びデータ連携・交換方法を開発し，それらを実装することで，電子申請システム，ワークフローシステムでの有効性を確認した．

今後，Web システムにおいて「帳票」の電子化から，「帳票」と「ナビゲーション」を統合した Web フォームへと進化させ，また様々な情報機器への対応，法制化に伴う帳票長期保存用途への対応など今後の検討課題である．

5.2. 今後の展望

5.2.1. IT 基盤を取り巻く環境

市場のニーズが多様化し，また市場を取り巻くビジネス環境も劇的に変化を続ける中で，既存サービスの競争優位性を図りつつ，今までにない業務サービスを試して新たな価値を創生することが重要になってきている．図 5.1 は，IT 基盤の変革を示す．電子マネー，IC カード，電子商取引，及び RFID 利用により，活用した現場で関与するヒト，モノ，位置情報などからなるイベント及びデータを管理する必要性から，通信料及びデータ量とも爆発的に増大する．この傾向は，利便性が広まるにつれて益々増大する．これによって，これらの膨大なデータを処理するための IT 基盤への要件が変革しつつあることが認識されている．

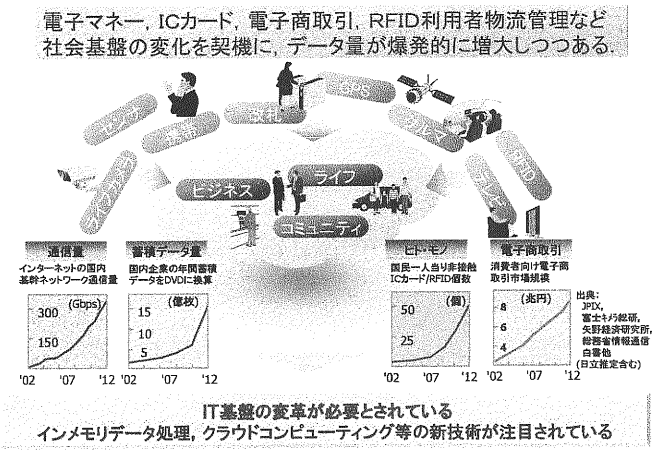


図 5.1: IT 基盤の変革

5.2.2. システム要件の変化

このような情報爆発時代を向えて，電子マネー，IC カード，電子商取引，及び RFID 利用から発生するデータを産み出す実世界とそれらのデータを処理する IT 基盤の間のビジネス速度が飛躍的に向上することも期待されている．図 5.2 は，IT 基盤の性能トレンドを示す．典型的な OLTP 業務を想定した TPC-C ベンチマーク [49] の性能トレンドを年代毎にプロットしている．これによれば，IT 基盤は年率 1.3 倍で性能向上を達成していることが分かる．

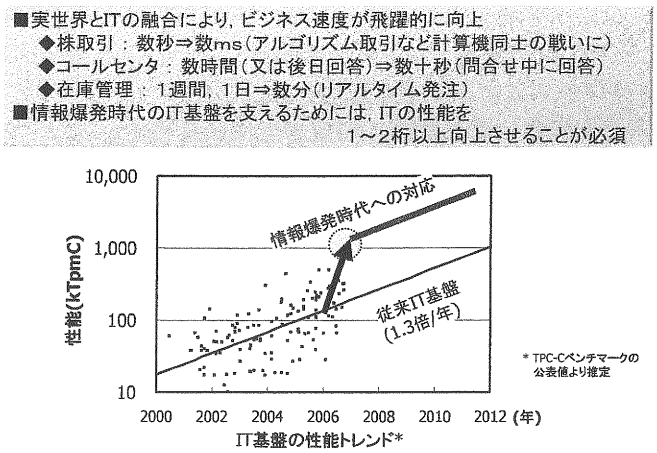


図 5.2: IT 基盤の性能動向

一方，ビジネス要件に関して，実世界から産み出される膨大なデータを処理する IT 基盤は現状の 1～2 桁以上の性能向上を図ることが必須であると認識されている．例えば，株取引では，アルゴリズム取引によって売買処理が数秒オーダから数ミリ秒オーダへと短縮されている．また，コールセンタ業務は，後日の回答から即座の回答へと要件が変わり，在庫発注業務でも即座に発注するなど，リアルタイム化が進展している．

従って，IT 基盤の処理要件は，情報爆発時代に向けた対応を図る必要に迫られている．

また，データ量の増加に対して，DBMS 技術の対応状況を示す．図 5.3 は，基幹系システム及び情報系システムでのストレージ需要の伸びを示す．基幹系システムのストレージ需要の伸びは，メモリ容量の増大でカバーされ，ほぼオンラインで利用する DB 容量を格納することが可能な状況となりつつある．

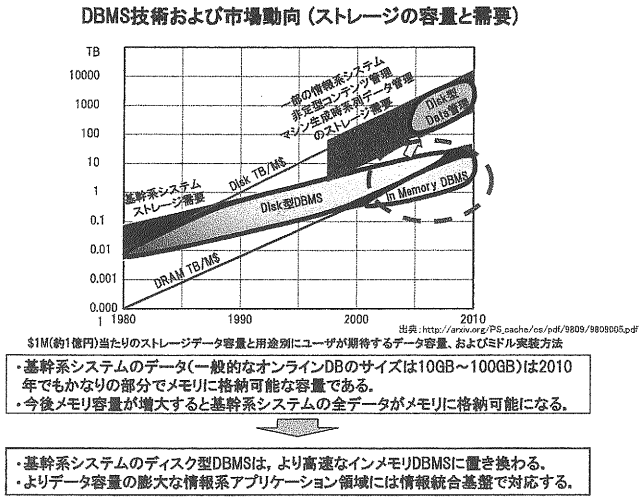


図 5.3: DBMS 技術の対応

従って，基幹系システムは，ディスク型DBMSからインメモリ型DBMSへの置き換えが進むものと考えられる．

他方，情報系システム，特に文書，画像，映像のようなマルチメディア情報からなる非定型コンテンツ，及び電子マネー，ICカード，電子商取引，及びRFID利用からマシンが時系列で自動的に生成するデータを格納する場合，基幹系システムのストレージ容量の2〜3桁上回る．このストレージ容量の伸びの傾向は，2010年以降も引き続くことが期待されている．

これらの動向から，DBMSは新たな要件に対処する必要があると考えられる．

5.2.3. 情報統合基盤での対応

仮想化及びクラウドコンピューティングなど新技術を先進的に取り入れ，活用される時代が到来している．このようなビジネス動向，ITシステム動向を踏まえて，この情報爆発時代を支える高性能・高信頼データベースに資する情報統合基盤が求められている．図5.4は，情報統合基盤のアーキテクチャである．

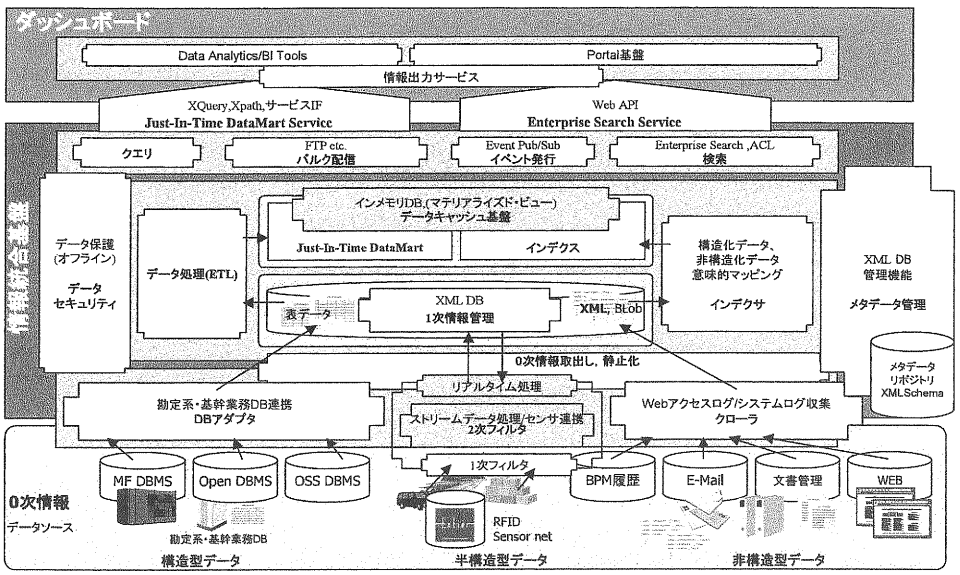


図 5.4: 情報統合基盤

大幅な性能向上によって今までにないサービスを追加することで，新たなビジネス価値を創出する，ブレイクスルーへの挑戦に資する情報統合基盤を提供する．また，課題発見のリアルタイム化と適切な対処によって，既存サービスの競争優位性を維持し，継続的な改善を行いつつビジネス価値を向上する，オペレーショナル・エクセレンスの追求に応える．

データソースは，基幹系システムから入力される日々発生するトランザクションデータ，また業務プロセスの履歴情報，メール・文書，システムの稼働ログ情報に加えて，電子マ

ネー，ICカード，電子商取引，及びRFID利用から産み出されるストリームデータからなる．これらのデータは，構造型データであればETL(Extract Transform Load)ツールを活用して，データ活用がなされる．非構造型データ及び半構造型データは，意味構造をマッピングするインデクサを介して，サーチが可能な形式へと変換される．

それぞれ活用可能なデータ形式に変換されると，データ分析/BIツール及びポータル基盤を介して，データサービスに供される．

謝辞

本論文は，株式会社日立製作所システム開発研究所，同情報・通信開発本部，同ビジネスソリューション事業部，及び同ソフトウェア事業部において，1987 年から 2006 年の 20 年間にわたって行われた研究開発をまとめたものである．論文の執筆に当たっては，静岡大学情報学部教授 石川博博士の懇切なるご指導を頂きました．また，論文の内容に関して有益なご助言を頂いた静岡大学情報学部教授 渡辺尚博士，中谷広正博士，静岡大学工学部教授 廣本宣久博士に厚く御礼申し上げます．

また，研究の遂行にあたり，常にご指導，ご協力頂いた，元日立製作所システム開発研究所第 6 部部長 米田茂氏に厚く御礼申し上げます．さらに，様々な観点から御助言を頂いた元日立製作所システム開発研究所主任研究員 鳥居俊一博士，元日立製作所ソフトウェア事業部 DB 設計部部長 正井一夫氏，元日立製作所ソフトウェア事業部 DB 設計部主管技師 宮崎光夫氏に心より御礼申し上げます

本研究を遂行する環境を整えてくださった，元日立製作所システム開発研究所副所長 高橋栄博士，元日立製作所システム開発研究所企画室室長 久保隆重氏に心より感謝申し上げます．また，情報科学領域での研究開発を行うに当たり，研究に向かう姿勢や企業内研究者のあり方のご示唆，ご助言頂いた元筑波大学情報学類教授 甘田早苗博士，益田隆司博士に心より感謝申し上げます．

論文をまとめるに当たり，業務と執筆を両立させるためのご理解，ご支援して頂いた日立製作所ソフトウェア事業部先端情報システム研究開発本部の上司及び同僚の方々に深く感謝いたします．

最後に，これまで私をあたたかく応援してくれた両親と，私を明るく励まし続けてくれた妻 美紀，息子 智大，娘 茜に心から感謝します．

参考文献

[1] Inmon, W.H.: Building The Data Warehouse, John Wiley & Sons (1992).

[2] Codd, E.F., Codd, S.B. and Sally, C.T.: Providing OLAP - On-Line Analytical Processing - to User-Analysts, An IT Mandate (1993).

[3] Date, C.J.: An Introduction to Database Systems, Volume & , 3rd Edition, Addison-Wesley (1983).

[4] Melton, J. and Buxton, S.: Querying XML: XQuery, XPath, and SQL/XML in context, Morgan Kaufmann Publisher (2006).

[5] X/Open, Distributed Transaction Processing: The TX (Transaction Demarcation) Interface Specification, X/Open company Ltd.

[6] X/Open, Distributed Transaction Processing: The XA Interface Specification, X/Open company Ltd.

[7] Bayer, R. and McCreight, E.: Organization and Maintenance of Large Ordered Indexes, Acta Informatica, Vol.1, No.3, pp.173-189 (1972).

[8] Bentley, J.L.: Multidimensional binary search trees used for associative searching, Comm. ACM, Vol.18, No.9, pp.509-517 (1975).

[9] Guttman, A.: R-Trees: A Dynamic Index Structure for Spatial Searching, Proc. of ACM SIGMOD International Conference on Management of Data, pp.47-57 (1984).

[10] Finkel, R. and Bentley, J.L.: Quad Trees: A Data Structure for Retrieval on Composite Keys, Acta Informatica, Vol.4, No.1, pp.1-9 (1974).

[11] Fagin, R., Nievergelt, J., Pippenger, N. , et.al.: Extendible Hashing - A Fast Access Method for Dynamic Files, ACM Transactions on Database Systems, Vol.4, No.3, pp.315-344 (1979).

[12] Litwin, W.: Linear hashing: A new tool for file and table addressing, Proc. 6th Conference on Very Large Databases, pp.212-223 (1980).

[13] Larson, P.A.: Dynamic Hash Tables, Comm. ACM, Vol.31, pp.446-457 (1988).

[14] Tsuchida, M. and Oomachi, K.: US5,091,852: System for optimizing query processing in a relational database (1992).

[15] Pike, R.C., Quinlan, S., Dorward, S.M., et.al.: US7,590,620: System and method for analyzing data records (2009).

[16] Tsuchida, M., Nakano, Y., Kawamura, N., et.al.: US5,806,059: Database management system and method for query process for the same (1998).

[17] Tsuchida, M., Nakano, Y., Kawamura, N., et.al.: US6,026,394: System and method for implementing parallel operations in a database management system (2000).

[18] Tsuchida, M., Nakano, Y., Kawamura, N., et.al.: US6,256,621: Database management system and query operation therefor, including processing plural database operation requests based on key range of hash code (2001).

[19] Tsuchida, M., Nakano, Y., Kawamura, N., et.al.: US6,556,988: Database management apparatus and query operation therefor, including processing plural database operation requests based on key range of hash code (2003).

[20] Tsuchida, M., Nakano, Y., Kawamura, N., et.al.: US6,567,806: System and method for implementing hash-based load-balancing query processing in a multiprocessor database system (2003).

[21] Dean, J. and Ghemawat, S.: US7,650,331: System and method for efficient large-scale data processing (2010).

[22] Tsuchida, M., Masai, K., Torii, S.: US6,192,359: Method and system of database divisional management for parallel database system (2001).

[23] 淵一博監修，古川康一，溝口文雄共編：プログラム変換，二村良彦著 第4章 部分計算，pp.63-79，共立出版 (1987).

[24] 二村良彦，野木兼六:一般部分計算法，コンピュータソフトウェア，Vol.5，No.2，pp.27-40 (1988).

[25] Loosely，C.，Douglas, F. 著，間宮あきら訳：データベースチューニング 256 の法則，第 13 章 並列化の原則，pp.355-396，日経 BP (1999).

[26] Satoh, K., Tsuchida, M., Nakamura, F., Oomachi, K.: Local and Global Query Optimization Mechanisms for Relational Databases, Proc. of VLDB, pp.405-417 (1985).

[27] 根岸和義，土田正士，正井一夫：並列リレーショナルデータベースシステム HiRDB の概要と基本技術，電子情報通信学会データ工学研究専門委員会, Vol.95, pp.407-410, DE95-79 (1995).

[28] Shimokawa, T., Takayama, H. and Masai, K.azuo: Highly Scalable Parallel Relational DBMS, Hitachi Review, Vol.45, No.2, pp67-70 (1996).

[29] HiRDB Version 8 UAP 開発ガイド (第 5 版)，3020-6-356-40 (2009 年 1 月).

[30] 土田正士，武藤英男：RDB における埋込み型問合せの最適化方式の提案，情報処理学会第 36 回全国大会 (1988).

[31] 岩田守弘，土田正士，根岸和義，正井一夫，宮崎光夫：並列 DB サーバシステムにおける問合せ処理方式の提案，情報処理学会第 48 回全国大会 (1994).

[32] 正井一夫，根岸和義，土田正士，松沢茂：更新処理を並列実行するUNIX向けDBMSを開発ー更新処理を並列実行するUNIX[1]向けDBMSを開発ー，日経エレクトロニクス, No.630 pp.101-114 (1995).

[33] Piatetsky-Shapiro, G. and Connell, C.: Accurate Estimation of the Number of Tuples Satisfying a Condition, Proc. of ACM SIGMOD (1984).

[34] Muralikrishna, M. and Dewitt, D.J.: Equi-Depth Histograms for Estimating Selectivity Factors for Multi-Dimensional Queries, Proc of ACM SIGMOD (1988).

[35] Haas, P.J., Naughton, J.F., Seshadri, S., Stokes, L.: Sampling-Based Estimation of the Number of Distinct Values of an Attribute, Proc. of VLDB (1995).

[36] Graefe, G. and Ward, K.: Dynamic Query Evaluation Plans, Proc. of ACM SIGMOD (1989).

[37] Cole, R., et.al.: Optimization of Dynamic Query Evaluation Plans, Proc of ACM SIGMOD, pp.150-160 (1994).

[38] Lee, A.W., et.al.: Closing The Query Processing Loop in Oracle 11g[2], Proc of VLDB, pp.1368-1378 (2008).

[39] DeWitt, D.J., et.al.: Parallel Database Systems: The Future of High Performance Database Systems, Comm. of the ACM, Vol.35, No.6, pp.85-98 (1992).

[40] DeWitt, D.J. and Gerber, R.H.: Multiprocessor Hash-based Join Algorithms, Proc. of VLDB (1985).

[41] DeWitt, D.J., Gerber, R.H., Graefe, G., et.al.: GAMMA - A High Performance Dataflow Database Machine, Proc. of VLDB (1986).

[42] Fushimi, S., Kitsuregawa, M. and Tanaka, H.: An Overview of The System Software of A Parallel Relational Database Machine, Proc. of VLDB (1986).

[43] Hong, W. and Stonebraker, M.: Optimization of Parallel Query Execution Plans in XPRS, Proc. of Conference on Parallel and Distributed Information Systems (1991).

[44] Selinger, P.G., et.al.: Access Path Selection in a Relational Database Management System, Proc. of ACM SIGMOD, pp.405-417 (1979).

[45] Pavlo, A., et.al.: A Comarison of Approaches to Large-Scale Data Analysis, Proc. of ACM SIGMOD, pp.165-178 (2009).

[46] Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, Proc. of OSDI (2004).

[47] Hellerstein, J.M., et.al.: Adaptive Query Processing: Technology in Evolution, IEEE Data Engineering Bulletin, Vol.23, No.2, pp.7-18 (2000).

[1]UNIXは，The Open Groupの米国ならびに他の国における登録商標です.

[2]Oracle及びOracle Database 11gは，Oracle Corporation及びその子会社，関連会社の米国及びその他の国における登録商標または商標です.

[48] Liu, B. and Rundensteiner, E.A.: Revisiting Pipelined Parallelism in Multi-Join Query Processing, Proc. of VLDB, pp.829-840 (2005).

[49] TPC-C Benchmark, available from (http://www.tpc.org/tpcc/spec/).

[50] Stonebraker, M.: Object-Relational DBMSs The Next Wave, Morgan Kaufmann Plublishers (1996).

[51] ISO/IEC 9075-2 Database Language - SQL Part 2 : Foundation (SQL/Foundation) (2008).

[52] 鳥居俊一ほか：高拡張性を目指した並列RDBサーバ，電子情報通信学会技術研究報告，DE94-49，pp.41-48 (1994).

[53] 日立製作所，共通マニュアル スケーラブルデータベースサーバ HiRDB Version 5.0 (Object Option付) (1998).

[54] 鳥居俊一ほか：マルチメディアデータベース基盤技術の開発，平成9年度次世代電子図書館システム研究開発事業 報告会 (1998).

[55] 鳥居俊一ほか：マルチメディアデータの一元管理を実現するHiRDB Universal Server，日立評論 1998年5月号 (1998).

[56] Shun'ichi Torii, et.al.: Integrated Multimedia Database, Hitachi Review, No.47, No.6, pp.296-299 (1998).

[57] 岩田守弘，土田正士，ほか：ORDBにおける多重定義関数の呼び出し処理の高速化，情報処理学会第58回全国大会 (1998).

[58] 小林挙，土田正士，ほか：ORDBにおけるプラグイン組込み仕様と実行制御，情報処理学会第58回全国大会 (1998).

[59] 原憲宏，土田正士，ほか：ORDBにおけるメディア対応ライブラリ組込みのためのアーキテクチャ提案，情報処理学会第58回全国大会 (1998).

[60] Gray, J., et.al.: Transaction Processing: Concepts and Techniques, Morgan Kaufmann Publiskers (1993).

[61] 土田正士，小寺孝：SQL2003ハンドブック，ソフトリサーチセンター (2004).

[62] Chamberlin, D.: Using the New DB2: IBM's Object-Relational Database System, Morgan Kaufmann Publiskers (1996).

[63] 土田正士，小寺孝，芝野耕司：SQLの20年と現状および今後の展開（後編），情報処理，Vol.45，No.6，pp.624-630 (2004).

[64] MySQL 5.1 Reference Manual Chapter 14. Pluggable Storage Engine Architecture.

[65] MySQL 5.0 ' s Pluggable Storage Engine Architecture（White Paper）.

[66] Oracle Database データカートリッジ開発者ガイド 10gリリース2(10.2) B19251-02.

[67] Melton, J., et.al.: SQL and Management of External Data, SIGMOD Record, Vol.30, No.1, pp.70-77 (March 2001).

[68] Crupi, J., et.al.：J2EEパターン—明暗を分ける設計の戦略，ピアソン・エデュケーション (2001).

[69] Kassem, N., et.al.：Java2 Platform Enterprise Edition アプリケーション設計ガイド — J2EE Blueprint —，Sun Microsystems (2000).

[70] 中西昌武：概念モデルに利用によるフォーム生成ツールの評価 - Microsoft Access の場合 - ，情報処理学会研究会報告 DBS 131-12 (2003).

[71] 今村誠：WWW ブラウザによる XML 文書入力方式について，情報処理学会研究会報告 DD 17-1 (1999).

[72] 野中誠：XForms 形式の画面仕様書を用いた対話型ソフトウェアの機能規模計測，情報処理学会研究会報告 SE 138-12 (2002).

[73] Dubinko, M., et.al：XForms 1.0-W3C Recommendation (2003).

[74] 電子申請推進コンソーシアム編：インターネット電子申請，オーム社 (2002).

[75] 総務省編：地方公共団体における申請・届出等手続に関する汎用受付システムの基本仕様 (2003).

[76] 総務省編：汎用受付システム構築の参考資料 (調達編・共同方式の場合)(第 1.1 版) (2003).

論文リスト

学位論文申請資格に関わる論文

1. 土田正士，河村信男，中野幸生，原憲宏，石川博：オブジェクト・リレーショナルデータベース管理システムにおけるプラグイン機構の開発，情報処理学会論文誌 データベース (TOD)，Vol.4，No.1，pp.1-14（2011）.

学位論文内容に関わる論文（未発表論文も含む）

1. 土田正士，河村信男，中野幸生，原憲宏，石川博：リレーショナルデータベース管理システムにおける並列問合せ処理機構の開発，情報処理学会論文誌 データベース (TOD)，Vol.4，No.2（2011）. (投稿中)

その他の論文

1. Shin'ichi Konomi, Souzou Inoue, Takashi Kobayashi, Masashi Tsuchida, Suguru Kitsuregawa：Supporting Colocated Interactions Using RFID and Social Network Displays，IEEE Pervasive Computing, Vol. 5, No. 3, (2006).

2. 井上創造，木實新一，小林隆志，土田正士，喜連川優：RFID を用いた学会参加者ネットワーク表示システムとその応用，日本データベース学会 DBSJ Letters Vol.5 No.1 pp.81-84（2006）

3. 土田正士，小寺孝，芝野耕司：SQL の 20 年と現状および今後の展開（後編），(社) 情報処理学会情報処理学会誌 Vol.45 No.6 2004-6（2004）.

4. 土田 正士，鳥居 俊一：データベースプロセッサ 内蔵データベースプロセッサ IDP とフィルタリングプロセッサ RDSP の概要，(社) 情報処理学会誌 Vol.33 No.12 pp.1409-1415（1992）.

5. Kazuhiro Satoh, Masashi Tsuchida, Fumio Nakamura, Kazuhiko Oomachi：Local and Global Query Optimization Mechanisms for Relational Databases，Proc. of VLDB 1985, pp. 405-417 (1985).

書籍

1. 土田正士，小寺孝共著：SQL2003 ハンドブック（2004 年 11 月　SRC 社）

2. 山平耕作，小寺孝，土田正士共著：SQL スーパーテキスト（2004 年 3 月　技術評論社）

3. 芝野耕司監訳　小寺孝，白鳥孝明，田中章司郎，土田正士，山平耕作共訳：SQL:1999 リレーショナル言語詳解（2003 年 11 月　ピアソン・エデュケーション社）

4. 芝野耕司監訳　土田正士，小寺孝，白鳥孝明，山平耕作共訳：標準口座 XQuery - XQuery，XPath，SQL/XML の文脈で問い合わせる（2008 年 3 月　翔泳社）

口頭発表など

1. 木實新一，井上創造，小林隆志，土田正士，喜連川優：学術会議における参加者関係発見のためのネットワーク表示システムの利用，(社) 電子情報通信学会データ工学ワークショップ DEWS2006（2006）.
2. 土田正士：SQL の最新動向，(社) 情報処理学会第 67 回全国大会標準化特別セッション（2005）.
3. 土田正士，芦田仁史，谷口尚子，高橋信雄：Web フォーム基盤アーキテクチャの提案及びその応用事例，（社）情報処理学会データベースシステム研究会 DBS-133/FI-75（2004）.
4. 土田正士，芝野耕司：SQL の最新動向－ SQL/OLB と SQL/MED －，(社) 電子情報通信学会データ工学研究会 DE99-3 Vol.99 No.117（1999）.
5. 岩田守弘，土田正士，他：ORDB における多重定義関数の呼び出し処理の高速化，情報処理学会第 58 回全国大会（1998）.
6. 小林挙，土田正士，他：ORDB におけるプラグイン組込み仕様と実行制御，情報処理学会第 58 回全国大会（1998）.
7. 原憲宏，土田正士，他：ORDB におけるメディア対応ライブラリ組込みのためのアーキテクチャ提案，情報処理学会第 58 回全国大会（1998）.
8. 根岸和義，土田正士，正井一夫：並列リレーショナルデータベースシステム HiRDB の概要と基本技術，(社) 電子情報通信学会データ工学研究会　Vol.95 No.407-410 DE95-79（1995）.
9. 土田正士，河村信男，中野幸生，武藤英男，北嶋弘行，米田茂：高速フィルタリングプロセッサの方式と性能評価，(社) 電子情報通信学会計算機アーキテクチャ研究会 90-4（1991）.
10. 土田正士，武藤英男：RDB における埋込み型問合せの最適化方式の提案，情報処理学会第 36 回全国大会（1988）.
11. 土田正士，酒井直文，大町一彦：DB マシンを含めた DBMS の最適化方式の提案，情報処理学会第 34 回全国大会（1987）.
12. 土田正士，佐藤和洋：リレーショナル DBMS における問合せ処理最適化方式の実験評価（その２），情報処理学会第 31 回全国大会（1985）.
13. 土田正士：新国際規格 SQL3 の紹介－その応用と今後の動向－，第６回ＯＳＰＧ開発技術研究部会.
14. 正井一夫，根岸和義，土田正士，松沢茂：更新処理を並列実行する UNIX 向け DBMS を開発－更新処理を並列実行する UNIX 向け DBMS を開発－，日経エレクトロニクス No.630 pp.101-114（1995）.

特許リスト

日本特許公告 37 件（内 代表発明特許公告 22 件，共同発明特許公告 15 件）

代表発明特許 22 件

1. 特許 2753228 号：データ処理装置
2. 特許 2755390 号：データベース処理装置及びデータベース処理方法
3. 特許 2760794 号：データベース処理方法および装置
4. 特許 3266351 号：データベース管理システムおよび問合せの処理方法
5. 特許 3538322 号：データベース管理システムおよび問合せの処理方法
6. 特許 3667997 号：データベース管理装置
7. 特許 3732655 号：データベース管理システム，データベース管理装置および問い合わせ処理方法
8. 特許 3819694 号：データベース管理システムおよび問合せの処理方法
9. 特許 3819695 号：データベース管理システムおよび問合せの処理方法
10. 特許 3668243 号：データベース管理システム
11. 特許 3023441 号：データベース分割管理方法および並列データベースシステム
12. 特許 3060222 号：データベース管理方法およびシステム
13. 特許 3060222 号：データベース管理方法およびシステム
14. 特許 3060223 号：データベース管理方法およびシステム
15. 特許 3060224 号：データベース管理方法およびシステム
16. 特許 3060225 号：データベース管理方法およびシステム
17. 特許 3156199 号：データベース管理方法
18. 特許 3172793 号：データベース管理方法
19. 特許 3236999 号：データベース管理方法およびシステム
20. 特許 3806609 号：並列データベースシステムおよび分散フアイルシステム
21. 特許 3599055 号：記憶装置管理方法およびシステム
22. 特許 3838248 号：データ格納方法およびデータ管理システム

共同発明特許 15 件

1. 特許 1838737 号：データベース処理方法および装置
2. 特許 2063628 号：サーチ方法および装置

3. 特許 2018661 号：データベース管理方法
4. 特許 2023134 号：検索方法及び装置
5. 特許 2587434 号：データの入出力処理方法
6. 特許 3453757 号：バツファア管理方法
7. 特許 3747477 号：排他制御方法及びこれを実現するシステム
8. 特許 3800228 号：排他制御方法
9. 特許 3800243 号：ロツク制御方法
10. 特許 3808941 号：並列データベースシステム通信回数削減方法
11. 特許 3747525 号：並列データベースシステム検索方法
12. 特許 3777666 号：データベース処理方法およびシステム
13. 特許 3882835 号：データベース管理方法および並列データベース管理システム
14. 特許 3742177 号：並列データベースシステムルーチン実行方法
15. 特許 3763982 号：データベース処理方法及びその実施装置並びにその処理プログラムを記録した媒体法

米国特許公告 40 件（内 代表発明特許公告 13 件，共同発明特許公告 27 件）
代表発明特許 13 件

1. US6,829,623：Method and system for managing multiple database storage units
2. US6,801,921：Method and system for managing multiple database storage units
3. US6,567,806：System and method for implementing hash-based load-balancing query processing in a multiprocessor database system
4. US6,556,988：Database management apparatus and query operation therefor, including processing plural database operation requests based on key range of hash code
5. US6,510,428：Method and system of database divisional management for parallel database system
6. US6,256,621：Database management system and query operation therefor, including processing plural database operation requests based on key range of hash code
7. US6,192,359：Method and system of database divisional management for parallel database system
8. US6,101,495：Method of executing partition operations in a parallel database system

9. US6,026,394 : System and method for implementing parallel operations in a database management system

10. US5,813,005 : Method and system of database divisional management for parallel database system

11. US5,806,059 : Database management system and method for query process for the same

12. US5,317,727 : Method apparatus for determining prefetch operating for a data base

13. US5,091,852 : System for optimizing query processing in a relational database

共同発明特許 27 件

1. US7,113,951 : Method and system for detecting tables to be modified

2. US7,100,126 : Electrical form design and management method, and recording medium

3. US6,959,302 : Querying database system to execute stored procedures using abstract data type attributes, retrieving location information of data, sub-data between first and second servers

4. US6,938,043 : Database processing method, apparatus for implementing same, and medium containing processing program therefore

5. US6,789,074 : Database processing method and apparatus, and medium for recording processing program thereof

6. US6,728,710 : Database processing method, apparatus for implementing same, and medium containing processing program therefore

7. US6,711,566 : Database processing method and apparatus using handle

8. US6,694,312 : Database processing method and apparatus for processing data by analyzing query input

9. US6,564,205 : Querying parallel database system that execute stored procedures using abstract data type attributes, retrieving location information of data, sub-data between first and second server

10. US6,505,199 : Database processing method, apparatus for implementing same, and medium containing processing program therefore

11. US6,480,833 : Method of resolving overloaded routines, system for implementing the same and medium for storing processing program therefore

12. US6,424,964 : Database processing method and apparatus using handle

13. US6,405,193 : Database processing method and apparatus using handle

14. US6,374,238 : Routine executing method in database system
15. US6,327,585 : Database processing method and apparatus using handle
16. US6,212,516 : Parallel database management method and parallel database management system
17. US6,076,085 : Routine executing method in database system
18. US5,983,228 : Parallel database management method and parallel database management system
19. US5,983,213 : Database processing method and apparatus using handle
20. US5,940,289: Parallel database system retrieval method of a relational database management system using initial data retrieval query and subsequent sub-data utilization query processing for minimizing query time
21. US5,930,800 : Execution of user defined ADT function implemented by embedded module in a database management method
22. US5,261,065 : Input/output processing method in a database management system
23. US5,237,661 : Buffer management method and system therefor using an I/O buffer on main memory and utilizing virtual memory and page fixing
24. US5,185,888 : Method and apparatus for data merging/sorting and searching using a plurality of bit-sliced processing units
25. US5,023,774 : Data I/O transaction method and system
26. US4,967,341 : Method and apparatus for processing data base
27. US4,916,655 : Method and apparatus for retrieval of a search string