

コンピュータ概論B

－ ソフトウェアを中心に －

#08 データベース (前編)

京都産業大学
安田豊

データベースとは

- 教科書 pp.103-
- 外見
 - － データを決まった形式（フォーマット）で整理し蓄積したもの
 - － レコード (Record) の存在
 - － オブジェクト指向データベースのように決まったデータ型を用意しないタイプもある（例外は常にある）

データベースの目的

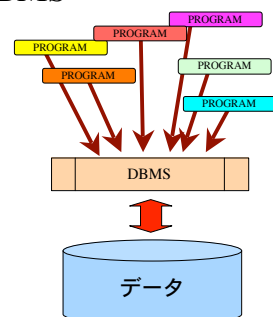
- 目的
 - － 入力・更新
 - － 高速な検索、再利用
- 一元管理
 - － あちこちにあるデータを一元管理したい
 - － 多数ユーザに最新で正しいデータを提供する
- 共有
 - － 多くのユーザで参照、更新したい
 - － 一元管理と常にセットで現れる問題

一元管理と共有の間で

- データと処理プログラムの独立性の確保
 - － 項目が一つ増えたらプログラム全部修正？
- 整合性の確保
 - － 複数箇所に同じ値がある
 - － 学費データと履修データの両方に在籍情報がある
 - － 多数利用者の同時更新から生まれる矛盾抑制
- データの保全
 - － プログラムの中断やシステムダウンからの保護
- アクセス制限
 - － 複数利用者が前提
- 簡易な問い合わせ言語機能の利用

DBMS

- 前出の機能をどうやって実現するか？
 - － データをプログラムが直接扱えないようにする
 - － DBMS の登場
 - － Database Management System
 - － 全ての作業はDBMSというプログラムを経由する
 - － 独立性、整合性、保全、アクセス制限



独立性・整合性

- 独立性
 - － データのフォーマットはDBMSに定義・管理
 - － 処理プログラムはその定義を引用して動作する
- 整合性
 - － 処理プログラムの手続きはすべてDBMSに対する指示として実行される
 - － DBMS は実行時に整合性管理、アクセス制限管理
 - － 順序処理、排他制御も行う

排他制御

- あるデータをカウントアップする
 - 「読んで」から「足し」で「書く」
 - 複数の処理リクエストが来た場合、正しくカウントアップできない
 - 「読・足・書・読・足・書」なら良いが
 - 「読・読・足・足・書・書」なら？
- 排他制御
 - 「この処理が終わるまで、この資源はロック」
 - デッドロックに注意
 - DBMSではロールバックの必要性につながる
 - DBMSに限らず多用されている
 - 計算機科学・技術の重要な概念の一つ

データ保全

- 処理プログラムの中断
 - バグ、オペレーションミス、システムダウン
- 一貫性の保持
 - 更新処理途中での停止
 - 会員資格更新時に会員番号 100 までで止まった
 - 会員マスターは更新したが支払いデータは未更新
 - 作業しなかったか、完了したかのどちらかに確定しないとけない
- トランザクションとロールバック

トランザクション

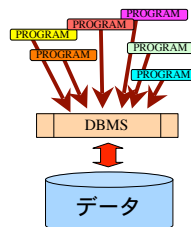
- データの整合性を保つために必要な最小の一連処理
 - その途中で終了した場合、データに矛盾が生じる
 - 大量データの削除処理などもそう
 - プログラマにしかトランザクションの存在が分からないケースもある
 - 明示的なトランザクションもある
- ロールバック
 - トランザクションを完了できなかった場合、トランザクション前の状態に巻き戻す

バックアップ・レストア

- DBMS自体の不意の中断
 - バグ、オペレーションミス、システムダウン
 - それでも一貫性を保持しなければならない
 - あるポイントでバックアップを取る
 - そこからは記録された更新情報を元に再現
- ログ管理
 - 更新記録(Log)をトランザクション単位で記録
 - レストアでは最終バックアップからログを頼りに更新作業を再現
 - ログがいっぱいになったら通常はDBMS自体が自動停止してデータを保護する。

DBMS のまとめ

- データベースが守るべき要件
 - データ独立、整合性管理、データ保全、アクセス管理
 - 多くはマルチプログラミングからの保護
 - 排他制御
 - バックアップ・レストア、ログ管理
- いずれも一貫性保持のため
 - そのためにプログラムとデータの間にDBMSという「仲介人」を入れる

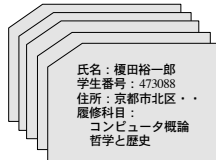


データベースの種類

- データモデルに適したタイプ
- カード型
 - 図書館蔵書カードのような一件一枚のもの
- ネットワーク(型)データベース
 - データの親子関係に注目
- リレーショナル(型)データベース
 - Relational Database
 - データの関係 (relation) に注目
 - 現在もっとも良く使われている
- 学生情報データベースを考える

カード型による学生情報データベース

- 一人一件
- 利点
 - 全ての情報がカードの中にあるのでカードを見つければあとの処理が簡単
- 欠点
 - 柔軟な検索が出来ない
 - キー以外の検索は一枚一枚繰ること
 - 通常はキーでソートして検索を容易にする
 - インデックス(複数)の利用も可能



探索法

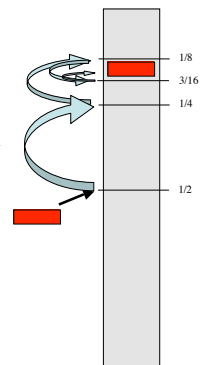
- より高速な検索のために
 - 高速とは?
 - CPU処理量(計算量)が少ない
 - ディスクアクセス量が少ない
- 多様な探索手法の存在
 - シーケンシャルアクセスとソート、二分探索
 - ランダムアクセスとハッシュ、インデクシング

シーケンシャルな探索

- 順次当たる方法 sequential
 - 単純総当たり
 - 図書カードをタイトルキーワードで繰る
 - ソート sort
 - 図書カードをタイトル順で並べておく
 - 妥当なところまでスキップ(調べるより送るだけの方がCPU処理量が少ない場合に有効)
- カード型データベースでは
 - 以下の記述は原理的な話として、ユーザインタフェースとしてカード型に見せているだけで、内実は違う、という場合もありうる
 - 何か一通りの方法でのみソート可能
 - それ以外の方法でタイトル順のカードを作者で当たるときは総当たり

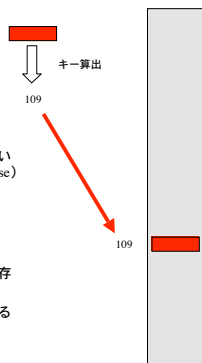
ランダムアクセスを利用した探索

- 二分探索 binary search
- 手法
 - sortされているカードの真ん中位置をまずアクセス
 - キーの大小から判断して、上下いずれのブロックに含まれるかを判定
 - 該当ブロックの真ん中を次にアクセス
- 利点: 高速な検索 (log N)
- 欠点: 順列のある場合だけ検索可能
 - 文字列部分マッチなどには使えない
 - ランダムアクセス可能なデバイス必須



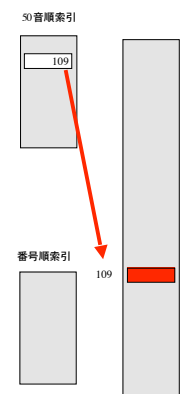
ランダムアクセスを利用した検索

- ハッシュ法 (hash)
- 手法
 - キーワードなどから何らかの数値を計算
 - 十分に拡散し、衝突が少なくなるように良い計算式を選ぶ(定式は逆に無く、case by case)
- 利点
 - うまくするとワンクッションだけでヒット
- 欠点
 - 重なったときの処理が面倒
 - ヒット率が入力データと計算式の相性に依存
 - データ領域の充填率が低い
 - 空きがあることを利用した高速化手法である



ランダムアクセスを利用した検索

- インデクシング (indexing)
- 索引(index)を利用する
 - 直接データを検索せずに
 - 検索に必要なデータだけをまとめた index を検索
 - そこにデータ位置が書かれている
 - 現実世界でも常用されている手段
- 利点
 - 複数のインデックスを持てる(名前順、学生番号順)
 - データの特性に依らず一般的に有効
- 欠点
 - インデックス作成に時間が掛かる(場合がある)
 - 追加より検索が圧倒的に多い場合に事前努力をする方式



ここまでのまとめ

- データベースの目的
 - 仲介人としてのDBMSの果たす役割
 - データ保護、一貫性の維持
- データベースの種類
 - カード型、etc. etc.
- 探索手法
 - 高速な探索のための手法
 - 二分探索、インデクシング、ハッシュ
 - データベース=一群の目的のための工夫の集積体