データベース

第14回　問い合わせ処理、障害回復

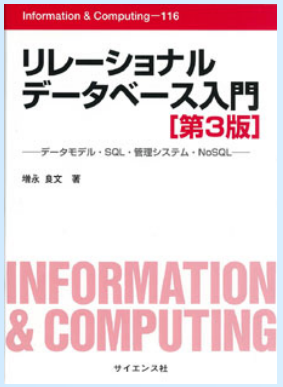
九州工業大学 情報工学部　尾下真樹

今日の内容

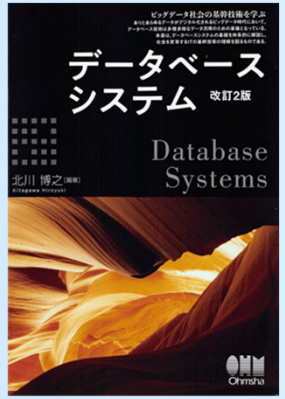
- 前回の復習
- 問い合わせ処理
  - 問い合わせ処理の最適化
  - 基本データ操作の実行方法
- 障害回復
- 授業のまとめ

教科書・参考書

- 「リレーショナルデータベース入門 第3版」
  増永良文 著、サイエンス社（3,200円）
  - 9章、10章（10.4～10.6）



- 「データベースシステム 改訂2版」
  北川 博之 著、オーム社（3,200円）
  - 9章、11章



前回の復習

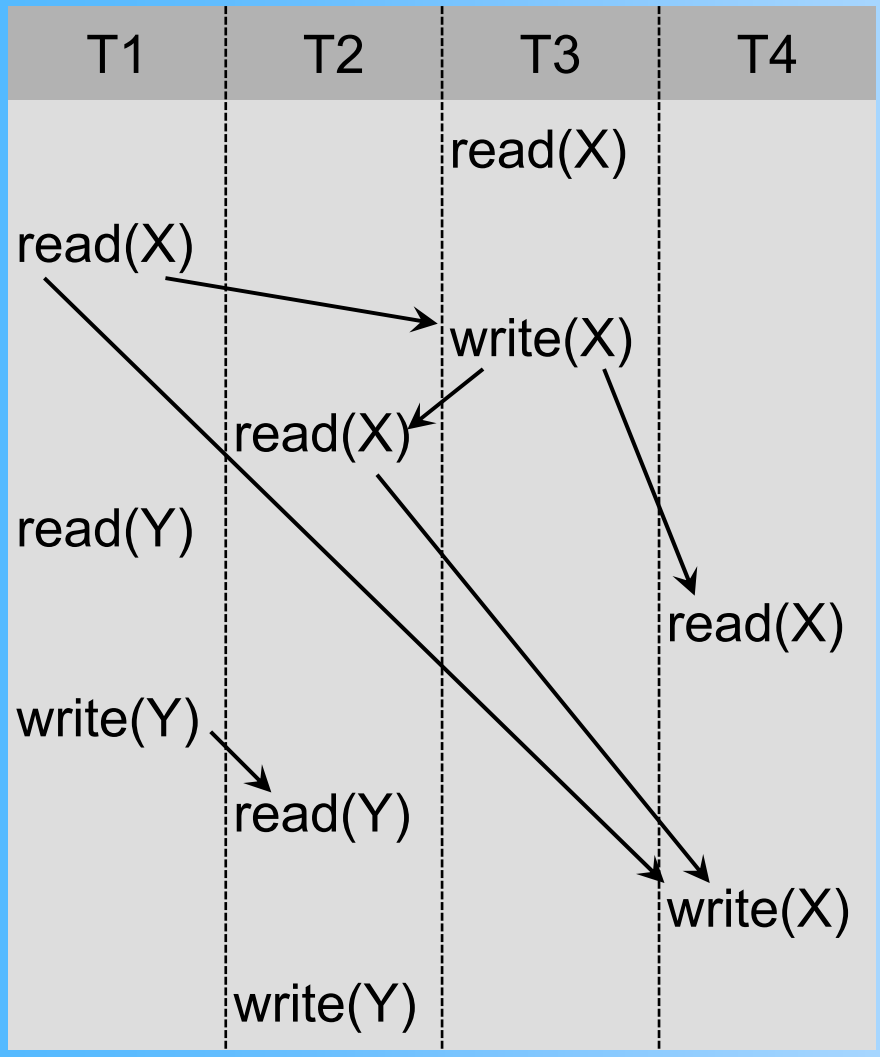
同時実行制御

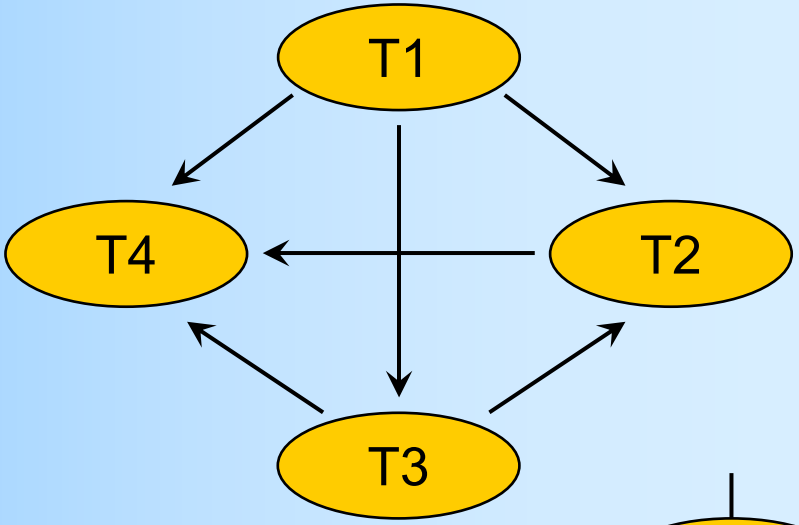
- トランザクション
- トランザクションの同時実行制御
- 計画的な並行処理
  - 直列可能性
  - ビュー等価と競合等価
  - 先行グラフによる競合等価の判定
- 逐次的な並行処理
  - ロックとデッドロック
  - デッドロックの回避方法

等価の定義と判定方法

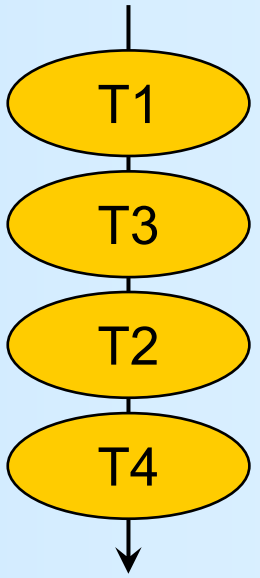
- ビュー等価
  - 各トランザクションが読み込む値が同じかどうかにより判定
  - 判定処理には時間がかかる
- 競合等価
  - 各トランザクションが競合するデータにアクセスする順番が同じかどうかにより判定
  - 等価なスケジュールも、等価でないと判定してしまうことがある（実用上は問題ない）
  - 比較的容易に判定できる

先行グラフの作成例





巡回はないので直列可能

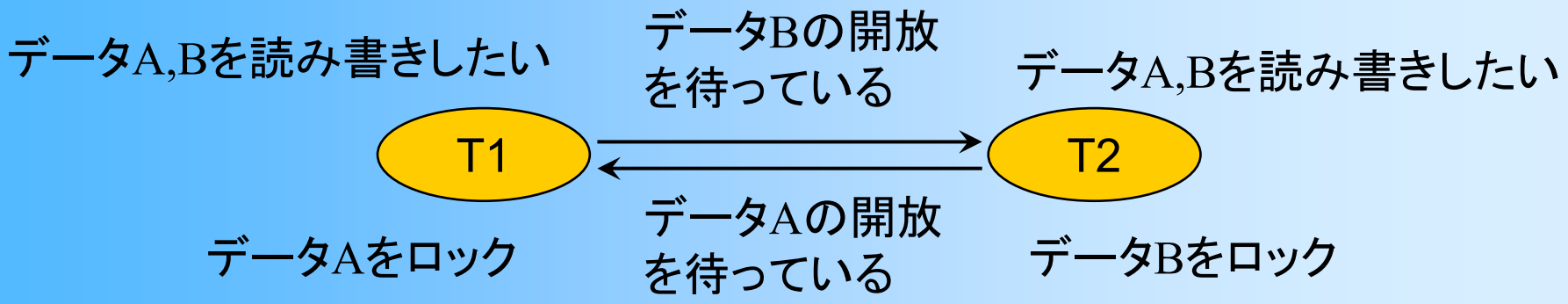


ロックを用いた同時実行制御

- ロック（Lock）
  - あるトランザクションだけが一部のデータを読み書きできるようにする仕組み
- ロックを使った処理の流れ
  - トランザクションがあるデータを読み書きする前に、そのデータに対して必ずロックを宣言
  - 他のトランザクションがそのデータを読み書きしようとした時には、後のトランザクションは、最初のトランザクションが完了するまで待たされる

デッドロック

- デッドロック
  - 複数のトランザクションが互いに必要なリソースをロックし合っており、互いに処理を遂行できない状態
  - 基本的にはどちらか一方のトランザクションを一度アボートする必要がある

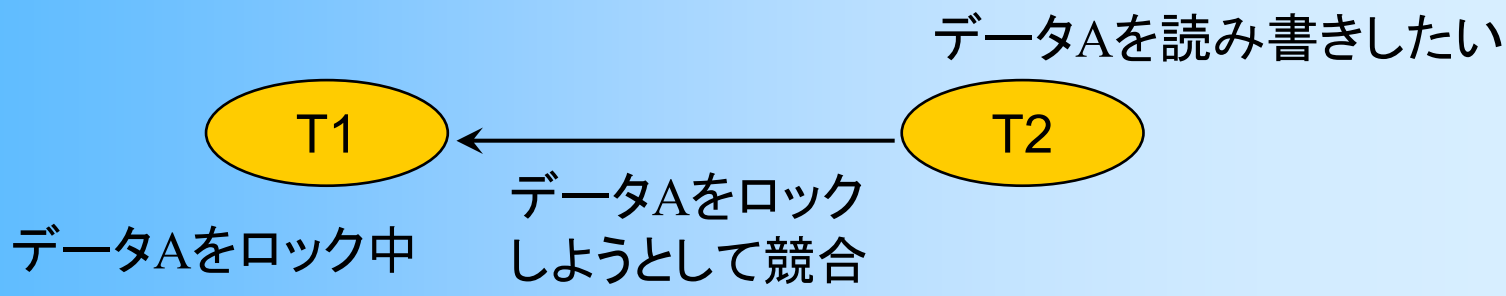


デッドロックへの対処方法

- デッドロックの検知
  - 何らかの方法で、デッドロックを起こしているトランザクションを特定し、終了させる
- デッドロックの防止
  - 何らかの方法で、デッドロックが発生しないよう防止する
- デッドロックの回避
  - 何らかの方法で、デッドロックが発生しても停止しないような処理を行う

デッドロックの回避方法

- ウェイト・ダイ（wait-die）方式
  - 最初にロックしていた方が古ければ、後からロックした方をアボート
- ワウンド・ウェイト（wound-wait）方式
  - 後からロックしようとした方が古ければ、最初にロックしていた方をアボート



問い合わせ処理

問い合わせ処理の効率化

- 問い合わせ処理の実行
  - データベースシステムは、入力された問い合わせ（SQL）を効率的に処理する必要がある
  - 問い合わせは、複数の基本データ操作（選択や結合）を組み合わせることで実行される
- 効率的に処理を実行するための技術
  - 問い合わせ処理の最適化
    - 複数の基本データ操作をどのように組み合わせるか？
  - 基本データ操作の実行方法
    - それぞれの基本データ操作の効率的な実行方法

問い合わせ処理の最適化

問い合わせ処理の最適化

- 問い合わせ処理
  - SQLによる問い合わせの例
    - 学生番号が00100の学生が履修している全科目の科目番号、科目名、成績を出力

```
SELECT   科目.科目番号, 科目名, 成績
FROM     科目, 履修
WHERE    科目.科目番号 = 履修.科目番号  AND
         学生番号 = '00100'
```

  - SQLでは、何を取り出すか（What）のみを指定し、どうやって取り出すか（How）までは指定しない
    - 実際の処理方法によって速度は大きく異なる

問い合わせの実現方法

- 同じ問い合わせを実現するときでも、複数の方法がある

①π科目.科目番号, 科目名, 成績(σ科目.科目番号＝履修.科目番号∧
学生番号＝'00100'(科目 × 履修))

②π科目番号, 科目名, 成績(σ学生番号＝'00100'
(科目$\bowtie$科目.科目番号＝履修.科目番号履修))

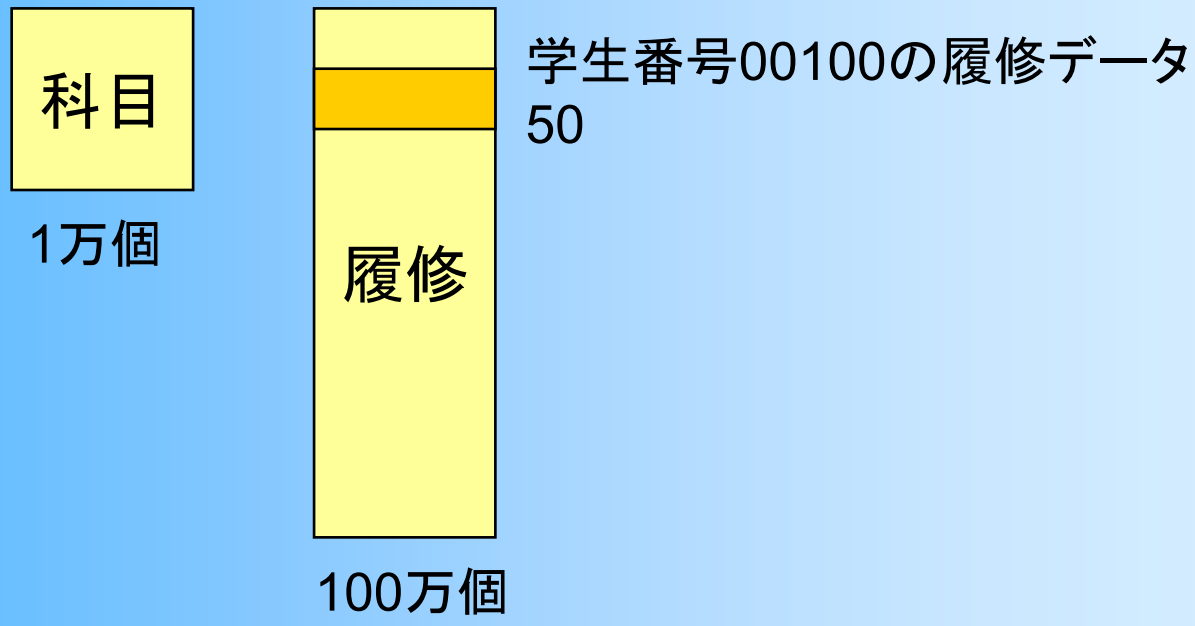
③π科目番号, 科目名, 成績(科目$\bowtie$科目.科目番号＝履修.科目番号
(σ学生番号＝'00100'履修))

科目

履修

問い合わせ方法による効率の違い

- 仮定
  - 科目は1万個、履修は100万個のデータとする
  - 学生番号00100の学生は、50科目を履修

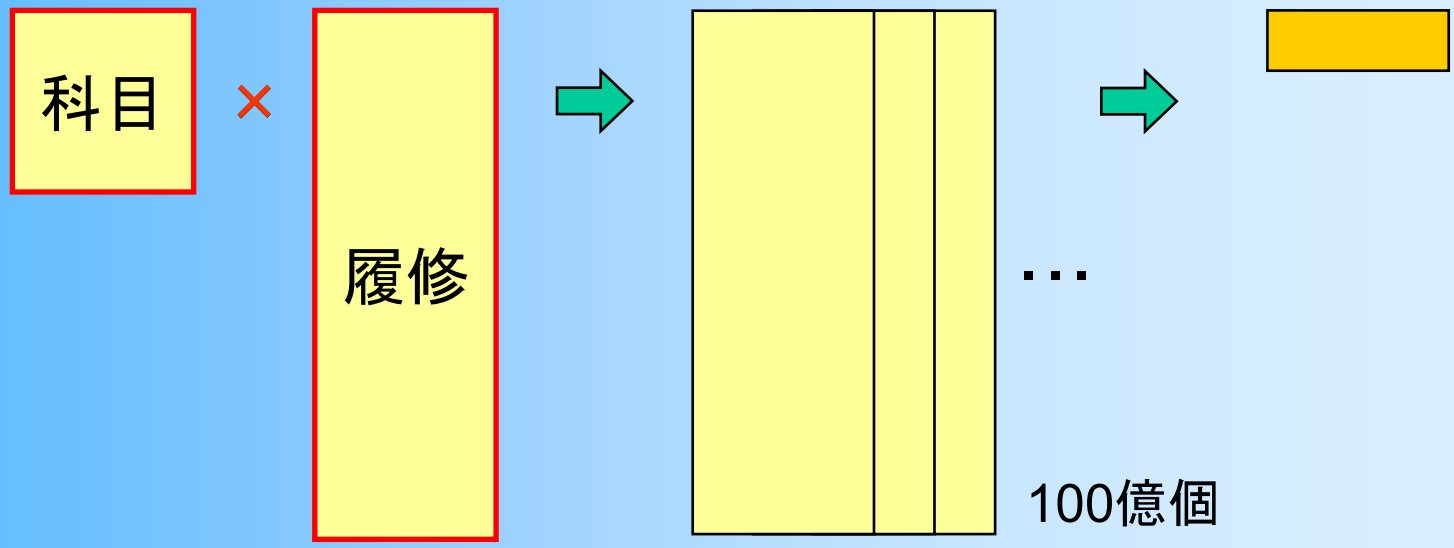


方法①での処理効率

- 方法①での処理効率

  ①π科目.科目番号, 科目名, 成績(σ科目.科目番号＝履修.科目番号∧学生番号＝’00100’(科目 × 履修))

  – 科目 × 履修の直積により100億個の中間データ領域が必要

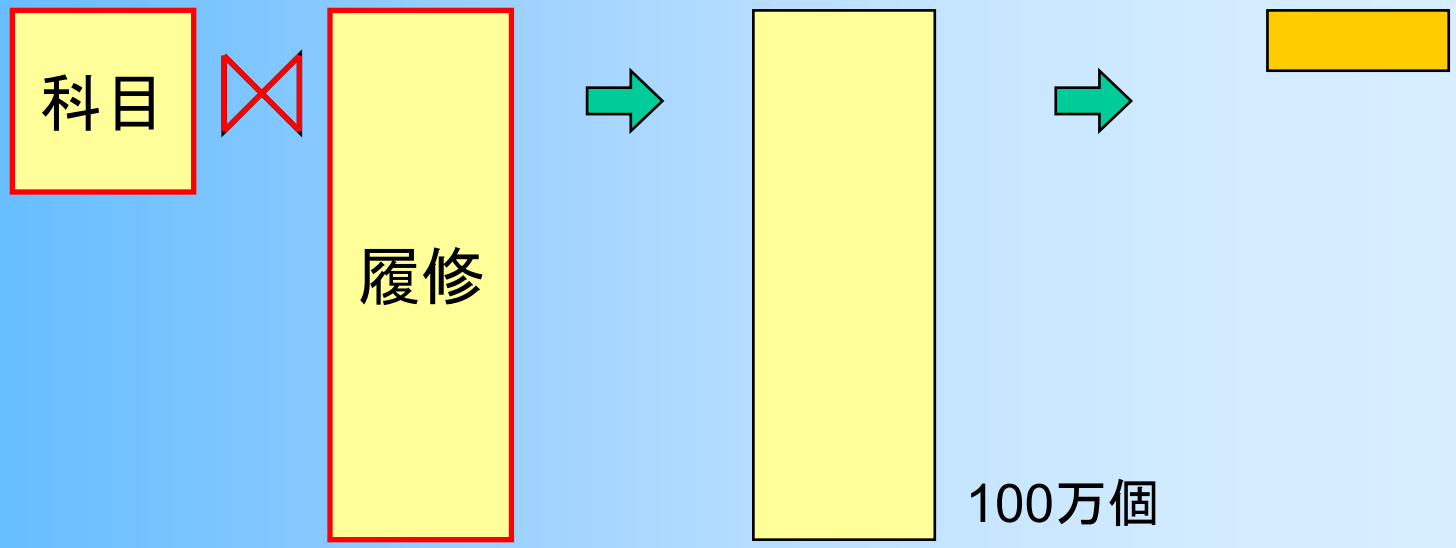


方法②での処理効率

- 方法②での処理効率

②π科目番号, 科目名, 成績（σ学生番号＝’00100’
（科目⋈科目.科目番号＝履修.科目番号履修））

– 科目と履修の等結合により、100万個の中間データ領域が必要

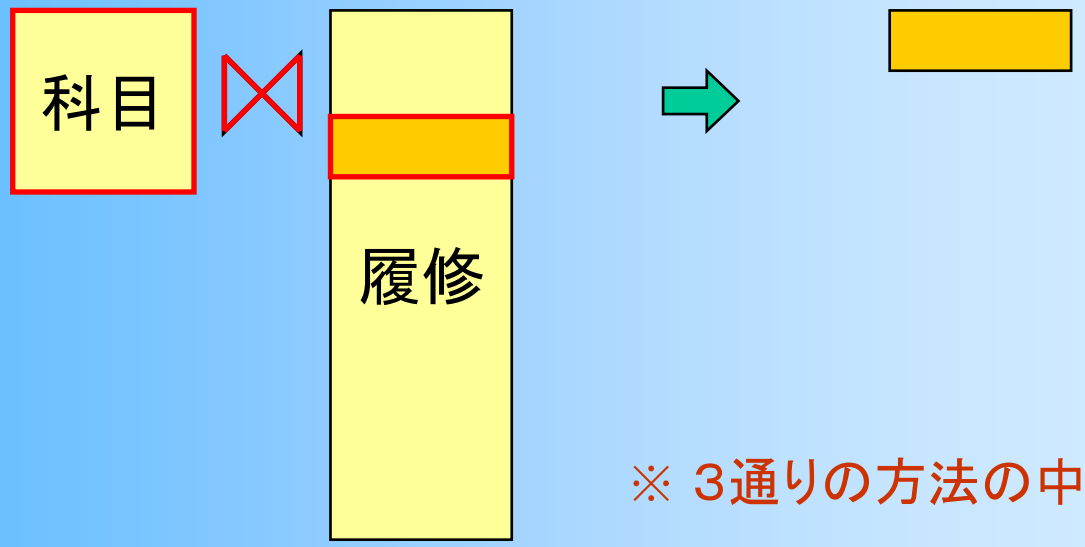


方法③での処理効率

- 方法③での処理効率

  ③π科目番号, 科目名, 成績（科目$\bowtie$科目.科目番号＝履修.科目番号（σ学生番号＝’00100’履修））

  – 先に50個の履修のデータを選択してから結合するため、中間データ領域は50個分で良い



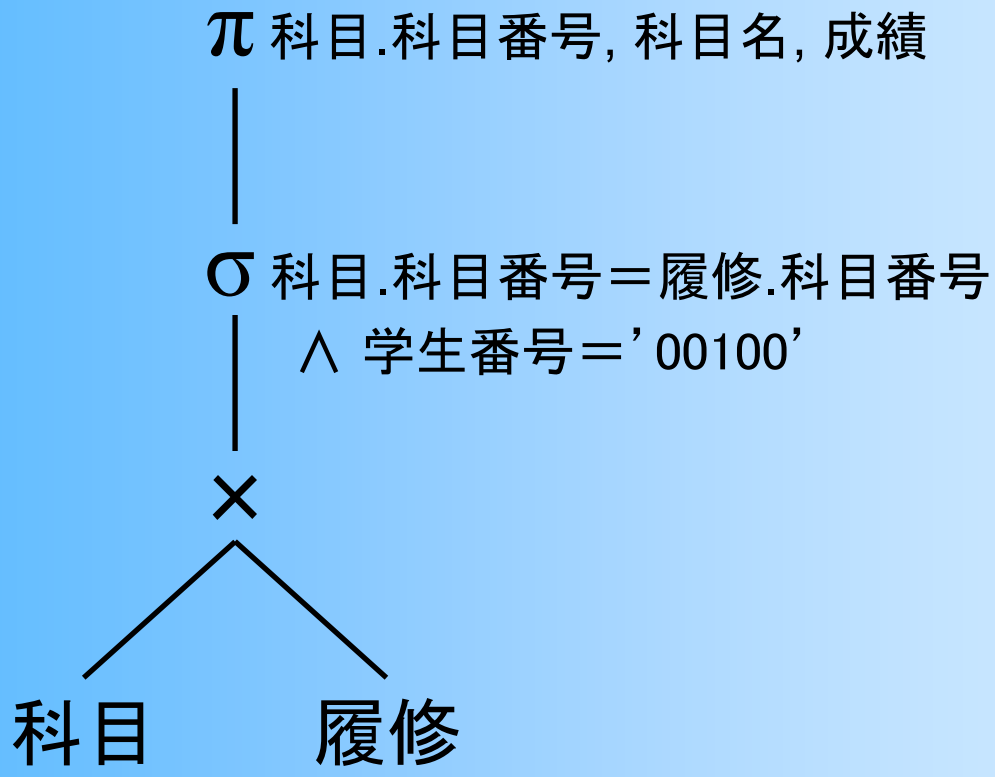
※ 3通りの方法の中では最も効率的

最適化の原則

- なるべく処理すべき中間データを減らす
  - 問い合わせ結果に関与しないデータを除去するため、選択をできるだけ早い段階で適用する
  - 中間結果のデータ量を削減するため、射影による不要な属性の削除をできるだけ早い段階で行う
  - 直積とその直後の選択が結合にまとめられる場合は、そのようにする

処理木を使った最適化の方法(1)

- 処理の手順を木表現で表す

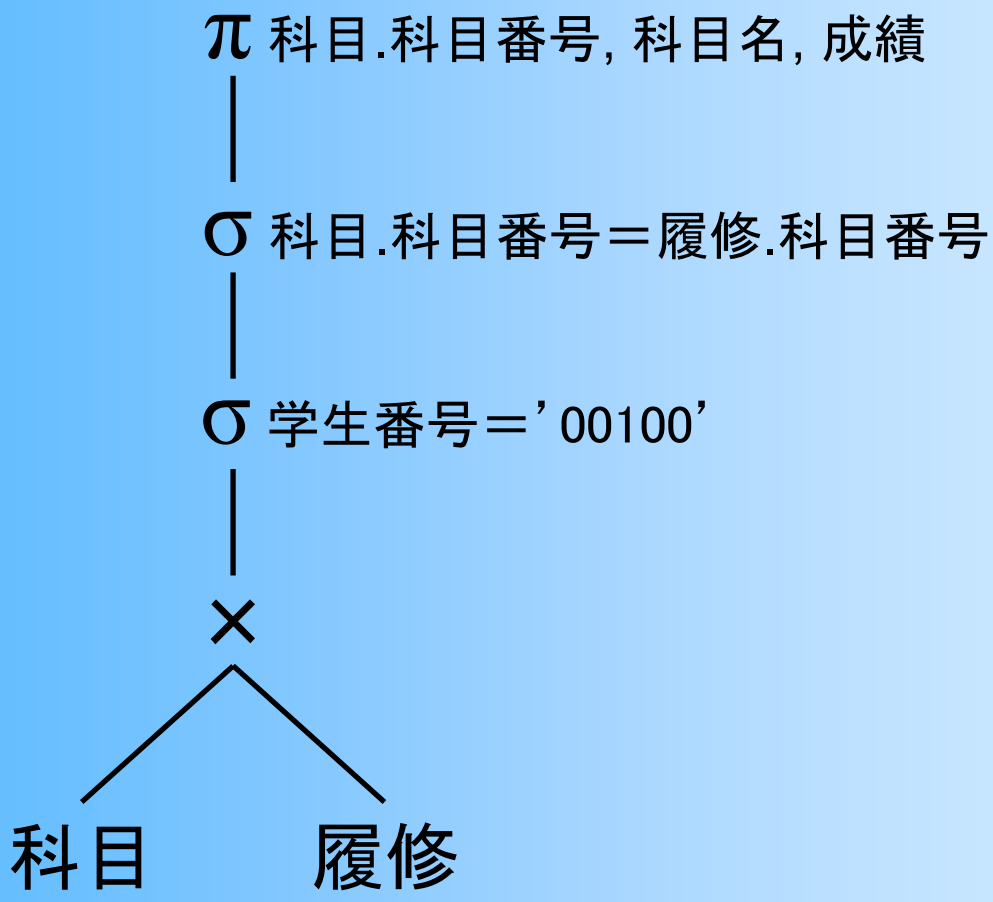


処理木を使った最適化の方法(2)

- 処理木に、最適化のためのルールを順番に適用していくことで、最適化を実現できる
  1. 選択条件を分解
  2. 選択を可能な限り下に移す
  3. 連続する直積と選択を結合にまとめる
  4. 射影を可能な限り下に移す
  5. 連続した射影・選択をまとめて一つにする

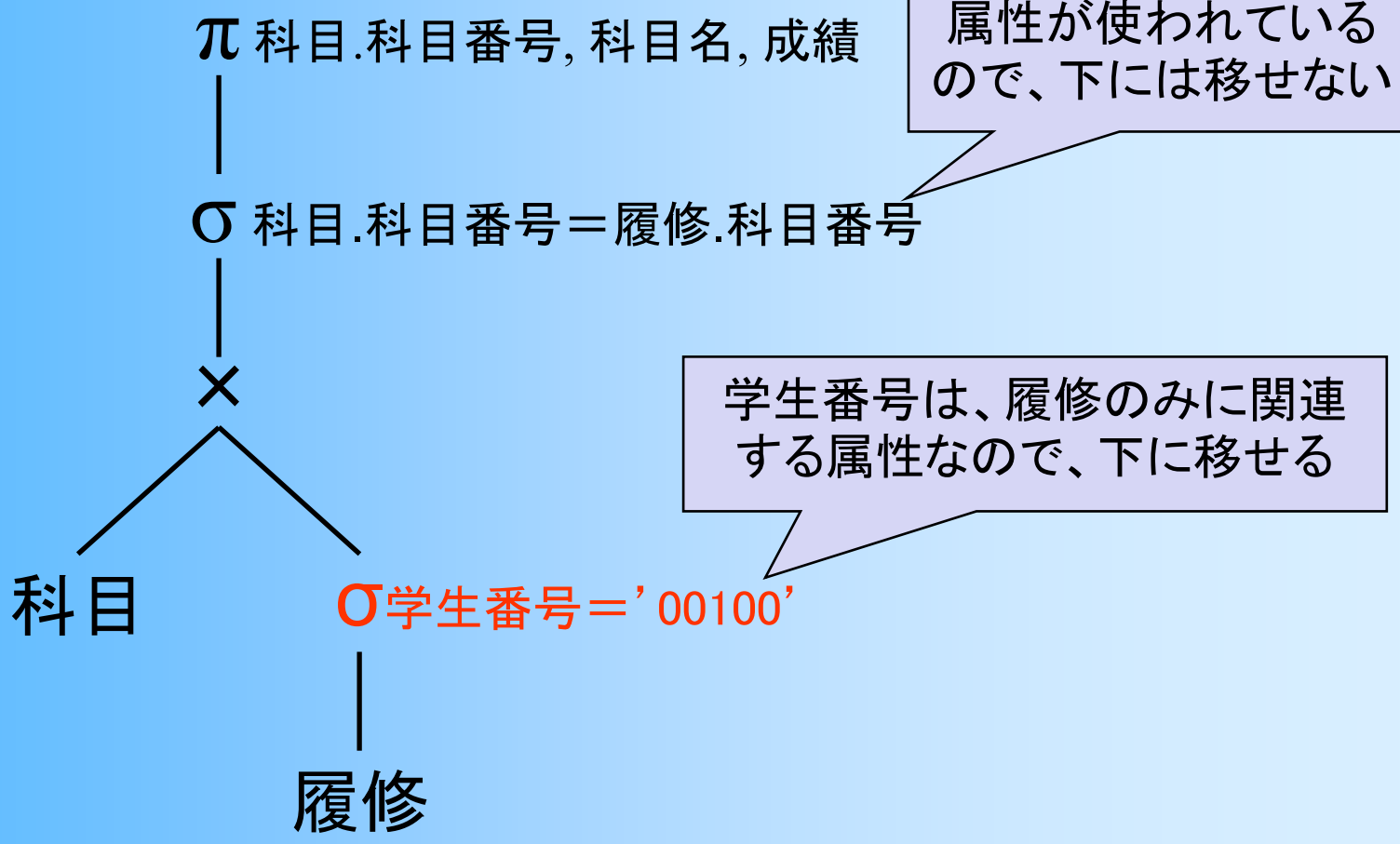
処理木を使った最適化の例(1)

- 1. 選択条件を分解



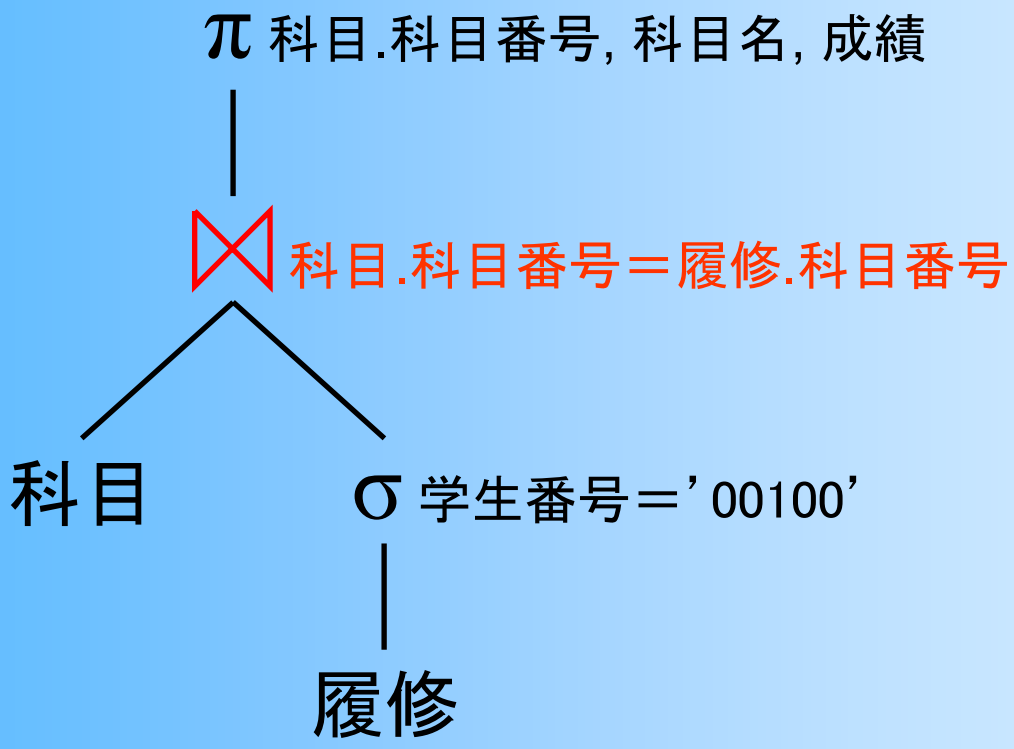
処理木を使った最適化の例(2)

- 2. 選択を可能な限り下に移す



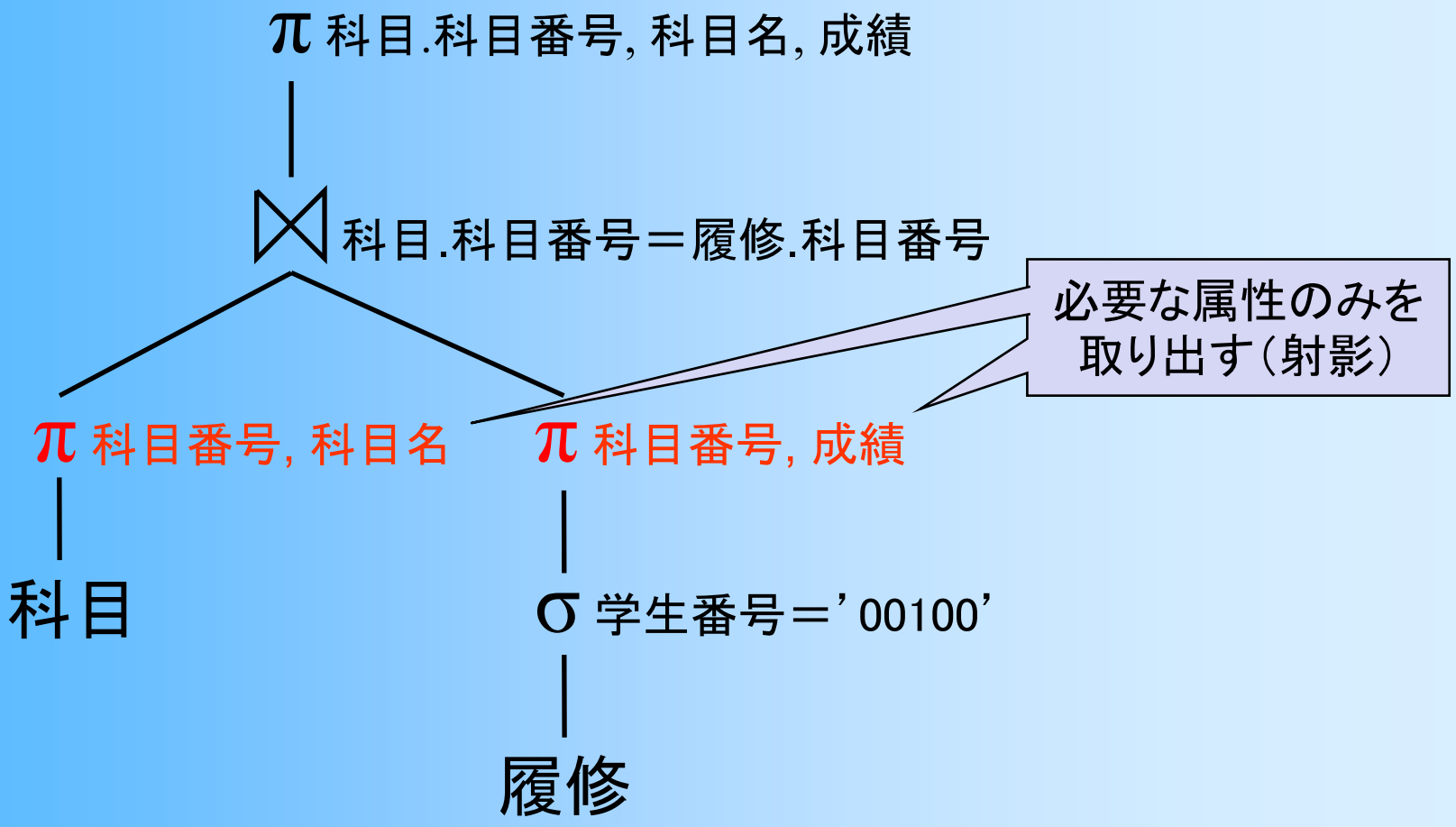
処理木を使った最適化の例(3)

- 3. 連続する直積と選択を結合にまとめる



処理木を使った最適化の例(4)

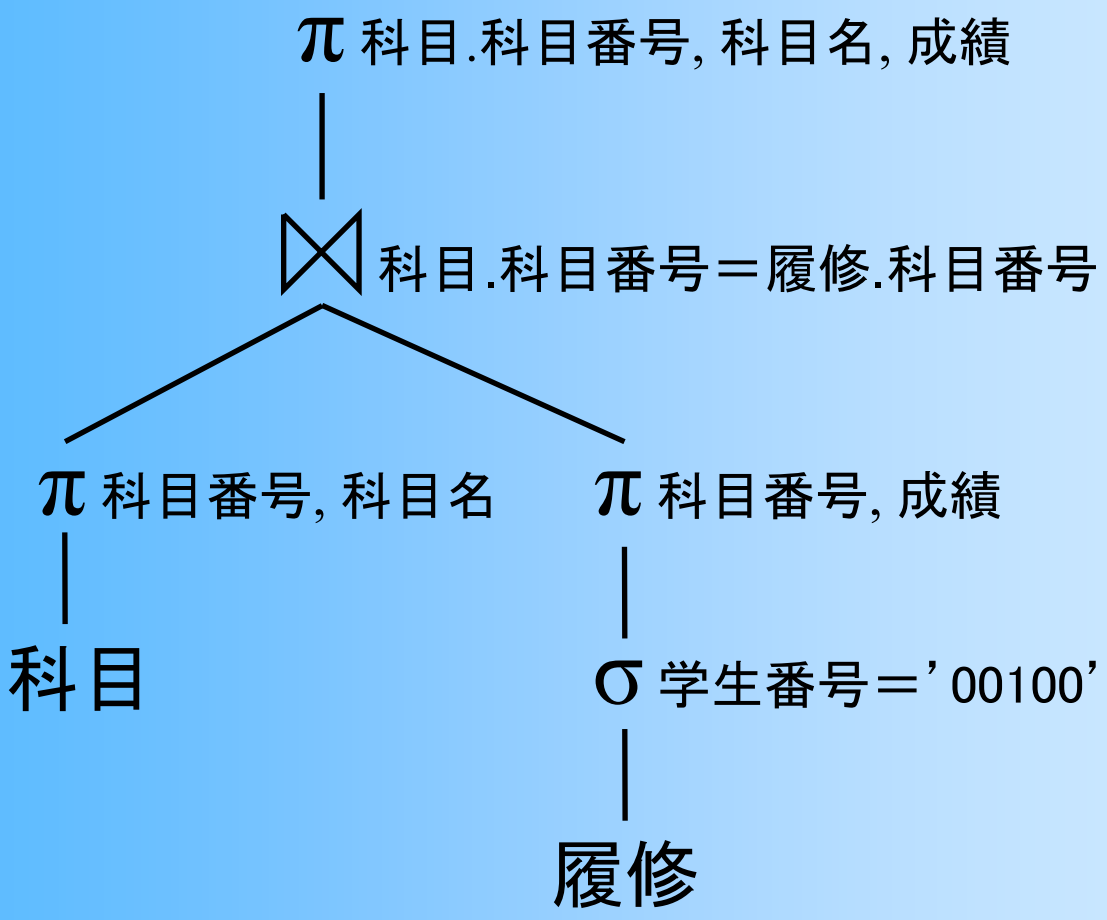
- 4. 射影を可能な限り下に移す



処理木を使った最適化の例(5)

- 5. 連続した射影・選択をまとめて一つにする

※ この例では不要



処理木を使った最適化の例(6)

- 最適化前の最大中間データ数
  - 全科目×全履修の直積のデータ
  - 1万個×100万個＝100億個
- 最適化後の最大中間データ数
  - 1学生分の履修のデータ
  - 50個
- 処理に必要な中間データを最小限にできる処理手順が得られた

基本データ操作の実行方法

基本データ操作

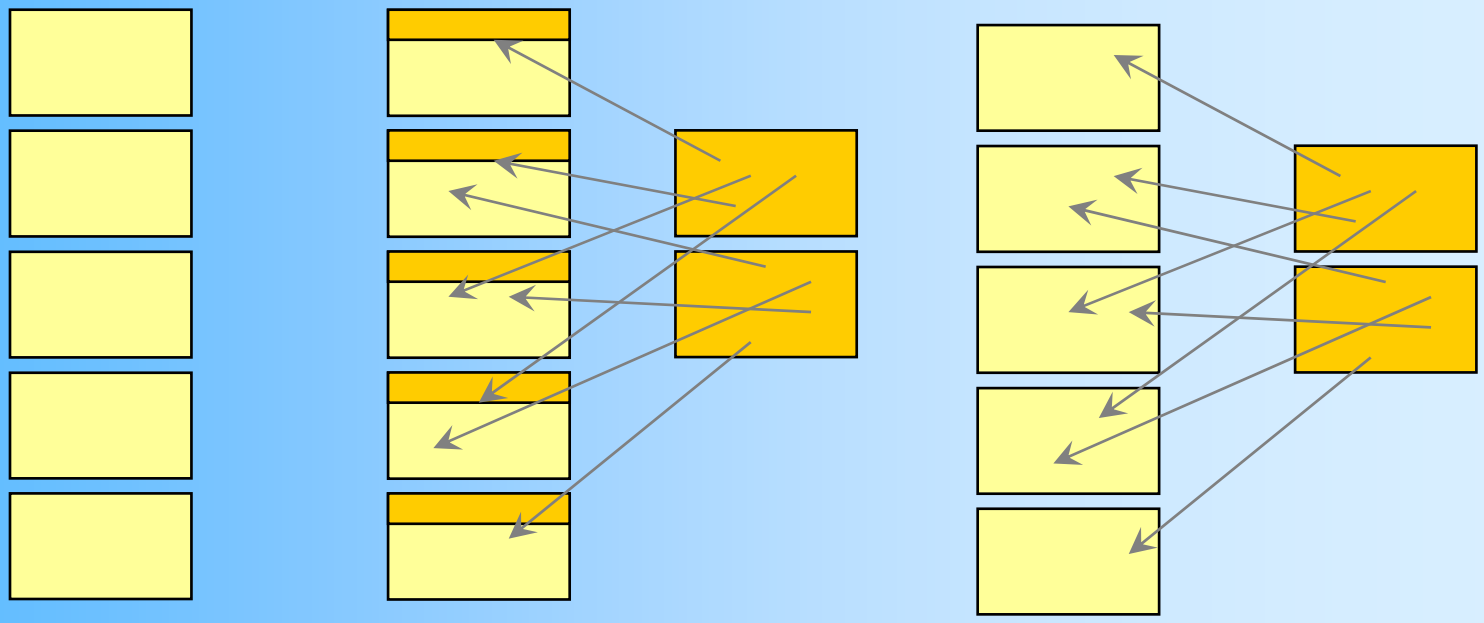
- 基本データ操作の方法
  - 複雑な問い合わせも、選択や結合などの基本的な操作の組み合わせによって実現される
    - 代数演算子を使った代数式で表される
  - それぞれの基本操作をどのように行うか？
    - インデックスの有無や、条件の種類によって、最適な処理方法は異なる
- 基本データ操作
  - 選択
  - 結合

データ操作の最適化

- 最適化において注意すべき点
  - データは、一般的には、ディスクに格納されていると考えられる
  - データは、ページ単位でディスクから読み込んでくる必要がある
  - ディスクからの読み込みの速度は、メモリからの読み込みの速度に比べると著しく遅い
  - ディスクから読み込み開始時にも時間がかかる
  - なるべくディクスアクセスの回数を減らすような最適化が必要

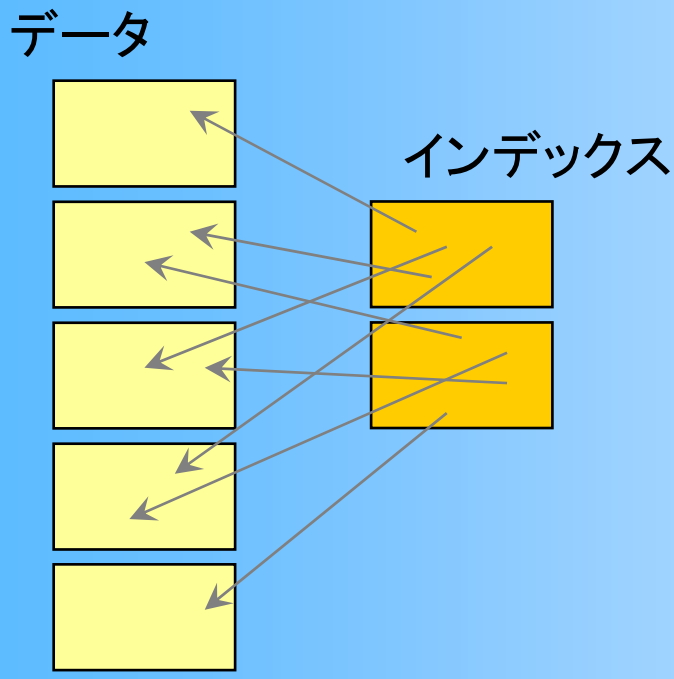
ページ単位での読み書き

- ページ配置の例
  - データ（ソート有り or 無し）
  - １次インデックス付きデータ
  - ２次インデックスイ付きデータ

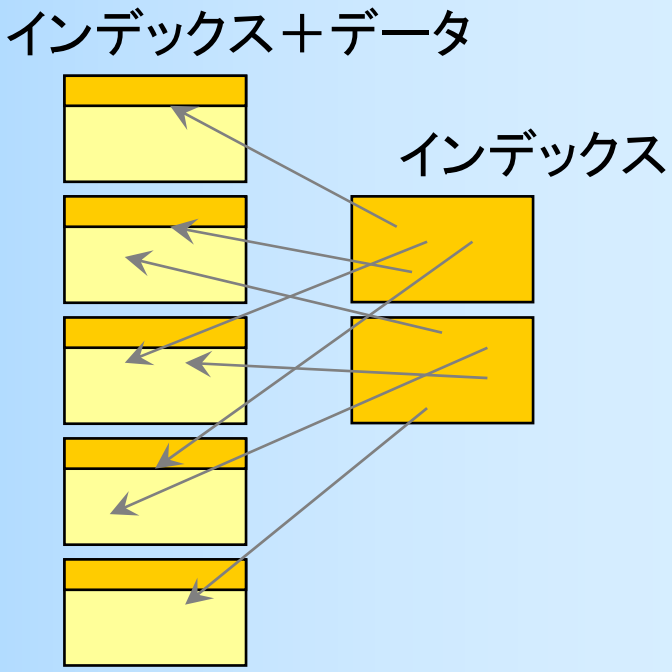


インデックスの種類

- ２次インデックス
  - インデックスとデータが別ページに存在



- １次インデックス付きデータ
  - インデックスとデータが同一ページに混在

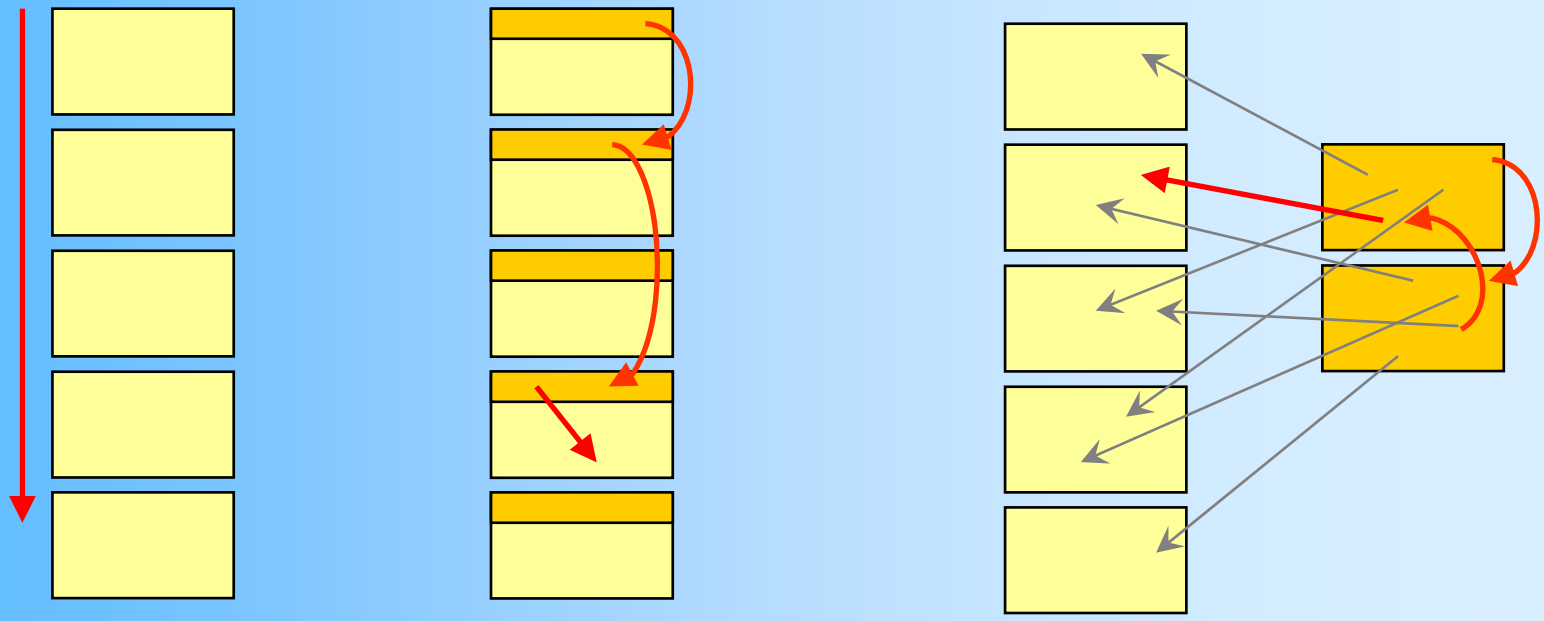


基本データ操作の実行方法

- 選択
- 結合

選択操作の方法

- 線形探索
- １次インデックスを用いた探索
- ２次インデックスを用いた探索



基本データ操作の実行方法

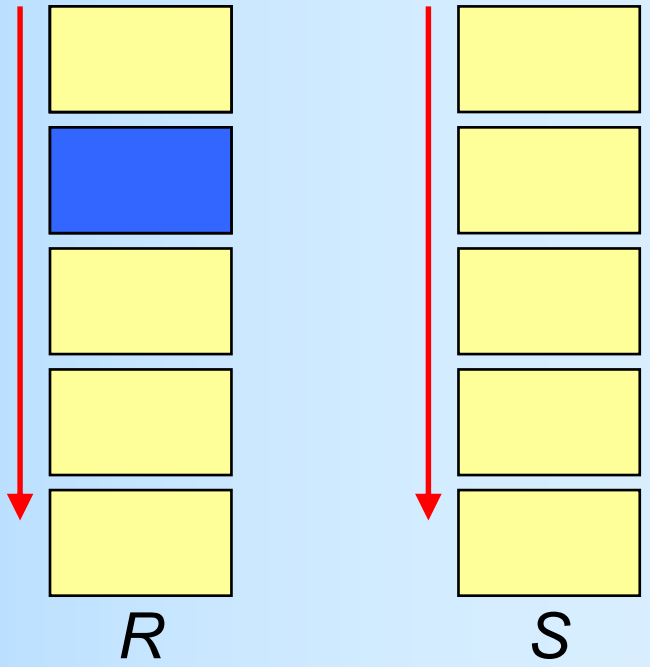
- 選択
- 結合

結合操作の方法

- 結合操作
  - 問い合わせの中でも非常に手間がかかる処理
  - 結合操作をいかに高速化するかが重要
- 結合操作の方法
  - 入れ子ループ結合
  - インデックスを用いた結合
  - マージ結合
  - ハッシュ結合

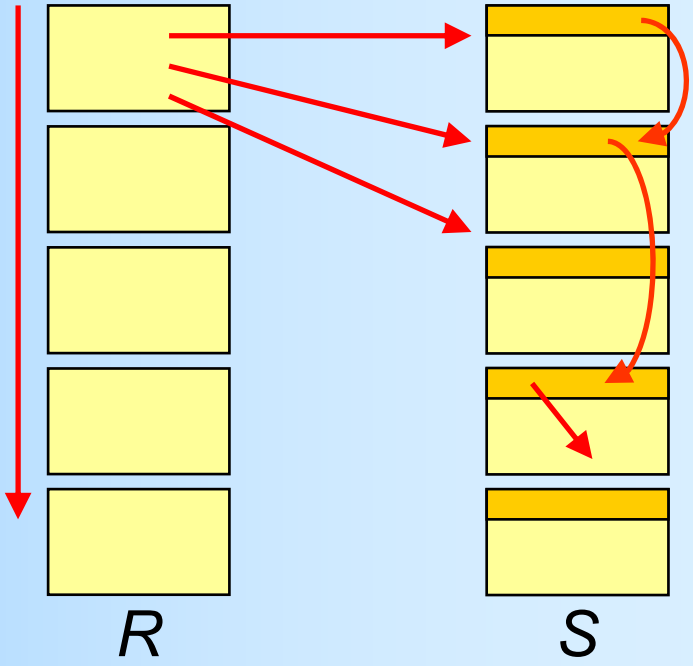
入れ子ループ結合

- 2つのリレーションの各インスタンス同士の組み合わせを全てチェック
  - 実際には、各ページごとに処理
    - ループが2重になるので、入れ子ループと呼ぶ
    - 内側のリレーションの呼び出しを繰り返し行う必要がある
  - 非常に効率が悪い



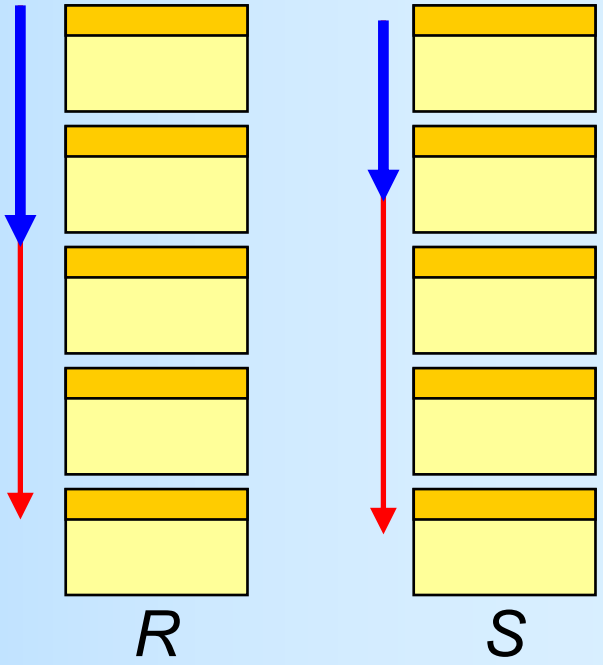
インデックスを用いた結合

- インデックスを用いた結合
  - 少なくとも一方のリレーションが、結合条件の属性のインデックスを持っているときに、適用可能
  - *R*の各インスタンスごとに処理
    - 結合対象の*S*のインスタンスを*S*のインデックスを用いて探索



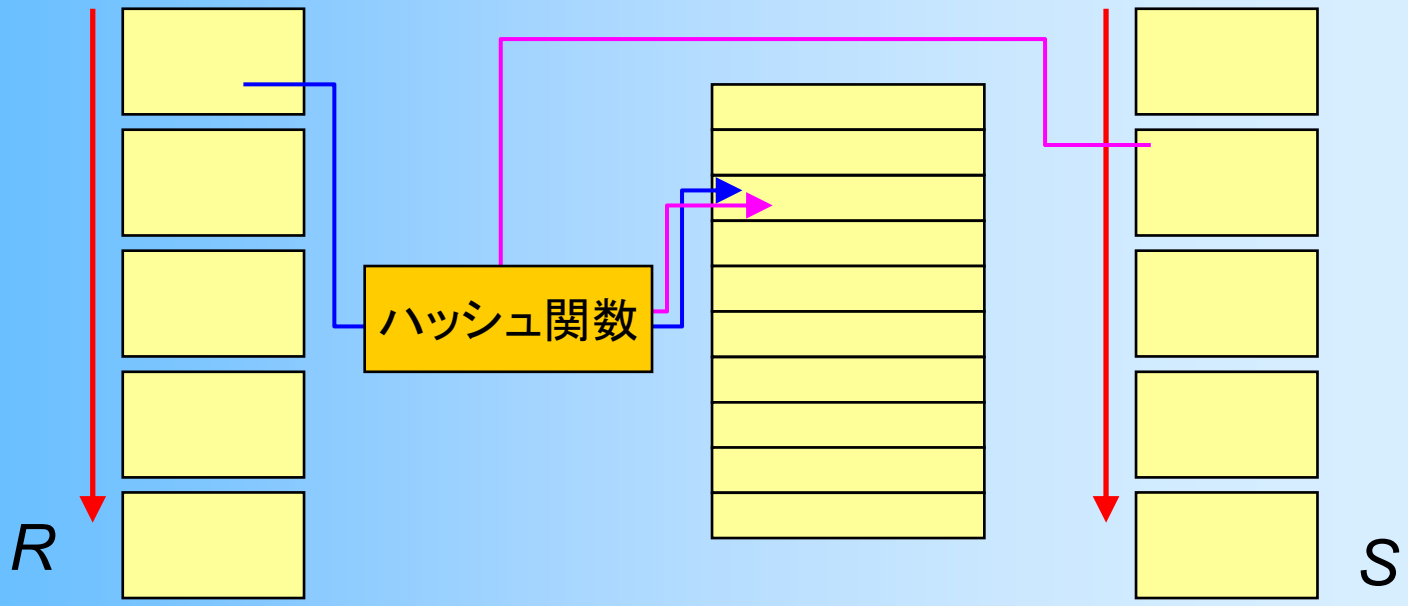
マージ結合

- マージ結合（ソートマージ結合）
  - 2つのリレーションが、結合条件の属性でソートされているとき（1次インデックスを持っているとき）に、適用可能
  - 2つのリレーションをそれぞれ順番にたどりながら属性値が同じインスタンス同士を結合
  - 同じ値のデータが複数存在する場合は、後戻りが必要



ハッシュ結合

- ハッシュを使った結合
  - リレーションR の各インスタンスを、結合する属性をもとに、何らかのハッシュ関数を使用して分ける
  - リレーションS の各インスタンスごとに、同じハッシュ関数を使用して、候補となる R のインスタンスを探索



問い合わせ処理のまとめ

- 問い合わせ処理の最適化
- 基本データ操作の実行方法

障害回復

障害回復

- 障害回復
  - 何らかの理由で、データベースシステムに障害（そのままでは処理を進めることができないような事態）が発生したときに必要となる処理
  - データベースシステムでは、障害が発生した場合でも、データの整合性を保つことが重要

障害の種類

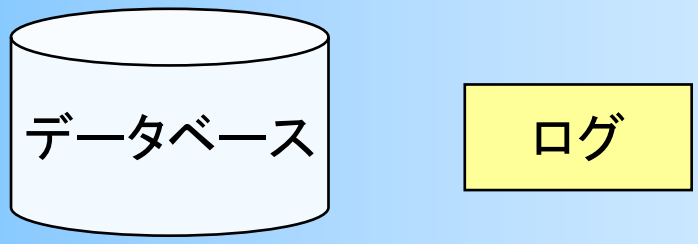
- トランザクション障害
  - トランザクションが、アボートされるなどして、完了できない状態
  - 並列システムではよくある事態
- システム障害
  - OSのハングアップやハードの障害などで、システムが強制終了されるような状態
  - これもたまに発生する
- メディア障害
  - ハードディスクなどのメディアに障害が生じてデータが読み出せなくなるような状態

障害への対応方法

- 一般に、大きく2通りの方法がある
  - ログ法
  - シャドウページング法

トランザクション障害への対応法(1)

- ログ法
  - トランザクションがデータベースへの処理を行う度に、変更内容をデータベースとは別のログファイルに出力
  - トランザクション障害が発生したら、ログをもとに開始前の状態に戻す
  - 更新を行うたびに同時にログを出力する必要があるため、処理効率がやや落ちる



トランザクション障害への対応法(2)

- シャドウページング法
  - 更新前のページと更新後のページを用意
  - 更新時は、更新前のページをコピーして更新後のページに書き込み
  - トランザクションがアボートしたら、更新前のページを使用するように戻す
  - トランザクションがコミットしたら、更新後のページを引き続き使用する
    - 次のトランザクションでは、前回更新されたページが更新前のページとなる
  - 障害時の処理は楽だが、メモリ管理が大変

システム障害への対処方法

- ログ法
  - システムが再起動した後に、残されたログを適用
  - REDO/UNDO法
    - トランザクションはコミットしていたが、遅延書き込みなどの理由でまだデータベースには反映されていなかった場合、ログを頼りに結果を再適用（REDO）
    - トランザクションがコミットしていなかったら、ログを頼りに開始前の状態に戻す（UNDO）
- シャドウページング法
  - 全て更新前のページを使用

メディア障害への対処方法

- メディア障害が起ると、シャドウページにもアクセス不可能
- ログ法は差分を記録するだけなので、データベース自体が読めなくなると復元できない
- 完全コピーとログ法の組み合わせ
  - 定期的にデータベースを完全バックアップ
  - 前回のバックアップ以降の全ログを記録しておく
  - メディア障害が発生したら、最後にとったバックアップに、それ以降の全てのログを適用
- 完全バックアップを取るためには、全てのトランザクションを停止させる必要がある

障害回復のまとめ

- 障害の種類
  - トランザクション障害
  - システム障害
  - メディア障害
- 対処方法
  - ログ法
  - シャドウページング法

今日の内容

- 前回の復習
- 問い合わせ処理
  - 問い合わせ処理の最適化
  - 基本データ操作の実行方法
- 障害回復
- 授業のまとめ

授業のまとめ

本科目の達成目標(シラバスより)

- リレーショナルデータベースを扱う上で必要な、スキーマの設計方法やSQLの使い方などの基礎的な知識を理解させる。
- リレーショナルデータベースの内部で用いられる、データ格納方式や高速化のための基礎的な技術を理解させる。
- データベース設計・操作を体験させ、データベースを利用するための基礎的な技術を習得させる。

本科目の位置づけ

- 科目区分（必修・選択必修・選択）
  - 各自の所属学科・コースによって異なるので、履修課程表で確認する
  - 必修の場合は、本科目の単位を修得しなければ、卒業できない
    - 不合格になった場合は、翌年度以降に再履修する
- 学習・教育目標との対応
  - 各学科の学習・教育目標の、情報工学・計算機工学の基礎・応用技術を学ぶ項目に対応する
    - 知能（B）、情報・通信（C）、物理（C）、生命（C）

成績評価

- 期末試験（40点）
- 期末レポート（40点）
- 毎回の授業の演習問題・演習課題（20点）
  - 授業中の演習問題、データベース演習課題
- 出席
  - 成績には考慮しない、一定回数の欠席で不合格
  - 成績評価や出席確認の方法の詳細は、第1回の講義の説明を参照

期末試験・レポート

- 期末試験
  - 実施方法や日時は、試験時間割や Moodle を参照
  - 試験範囲は、講義で扱った内容全て（演習も）
    - 記述式の問題にもきちんと回答できるように勉強すること（例：代数演算式、SQL、正規化の説明、等）
- 期末レポート
  - 作成方法や提出締め切りは、Moodleを参照
    - 締め切り以降の提出は一切認めない

レポート課題

- 自分で決めた何らかのテーマを題材にして、データベースとWebインターフェースを作成
  1. データベースのスキーマの設計
     - 全ての正規形を満たすように正規化を行う
  2. データベースの作成
     - テーブルの作成、データの追加
  3. データベースへの問い合わせ
     - 問い合わせの具体例を考えてSQLを作成・実行
  4. Webインターフェースの作成
     - 一覧表示、追加・削除・更新、実用的な検索

再試験・再レポート

- 再試験・再レポート提出は一切行わない
- 最初から試験・レポートに全力を尽くすこと
- 特に、試験に自信がない人は、レポートを頑張ること
- 万一、再試験・再レポート等を実施する場合は、Moodleで連絡する

講義のまとめ(1)

- 講義・演習
  - 第1回 ガイダンス、データベースシステム
  - 第2回 データモデル
  - 第3回 リレーショナル代数
  - 第4回 データベース言語 SQL (1)
  - 第5回 演習：PostgreSQLによるデータベース構築
  - 第6回 データベース言語 SQL (2)
  - 第7回 リレーショナルデータベースの設計 (1)
  - 第8回 リレーショナルデータベースの設計 (2)
  - 第9回 リレーショナルデータベースの設計 (3)

講義のまとめ(2)

- 講義(続き)
  - 第10回 演習: PHPによるWebインターフェース(1)
  - 第11回 演習: PHPによるWebインターフェース(2)
  - 第12回 物理的データ格納方式
  - 第13回 同時実行制御
  - 第14回 問い合わせ処理、障害回復
  - 第15回 総合演習
  - 期末試験
  - レポート

仕事との関連

- 情報システムの開発に関わる仕事をする人は、データベースシステムと連携したシステムを開発することになる可能性が高い
- 情報システムを利用するだけの人も、データベースシステムに関する知識が求められる可能性が高い
- 実際のシステム開発には、より実用的な知識が求められる

研究との関連

- 本科目の内容は、「データ工学」と呼ばれる研究分野の基礎となる
  - 効率的なデータ構造やアルゴリズム
  - データベースシステムや応用システム
- 本学科（本学部）では、データ工学を専門的に研究している研究室はない？
  - 情報工学に関する研究を行っている多くの研究室では、データ工学自体を直接研究はしなくとも、基礎技術として利用することになる可能性は高い

より詳しく勉強したい人へ

- データベースシステムの利用
  - PostgreSQL、MySQL などのフリーソフトウェア
  - Apache などのウェブサーバ
- ウェブプログラミング
  - HTML5 ＋ JavaScript を使ったプログラミング
    - jQuery（AJAX）などのライブラリを使ったUI
    - enchant.js などのゲーム開発ライブラリ
- 資格
  - Oracle認定試験 など

授業アンケート

- 本科目独自の授業アンケート
  → Moodleの本科目のコースから回答
- 学部共通の授業評価アンケート
  → 他の科目と一緒に学修自己評価システムから回答
  - 学修自己評価システムには、LiveCampus からアクセスできる
    - メニュー → 学習教育支援 → 学修自己評価システム
  - 詳細は、Moodle の学修自己評価システムと授業評価アンケートのコースを参照

まとめ

- 前回の復習
- 問い合わせ処理
  - 問い合わせ処理の最適化
  - 基本データ操作の実行方法
- 障害回復
- 授業のまとめ