データベースS

第14回　問い合わせ処理、障害回復

システム創成情報工学科 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
- 講義のまとめ
- 授業アンケート

教科書

- 「リレーショナルデータベース入門 」
  増永良文 著、サイエンス社 (2,600円)
  - 8章 219～241ページ(最適化)
  - 9章 252～271ページ(障害回復)
- 「データベースシステム」
  北川 博之 著、昭晃堂 出版 (3,200円)
  - 7章 125～146ページ(最適化)
  - 9章 174～188ページ(障害回復)

前回の復習

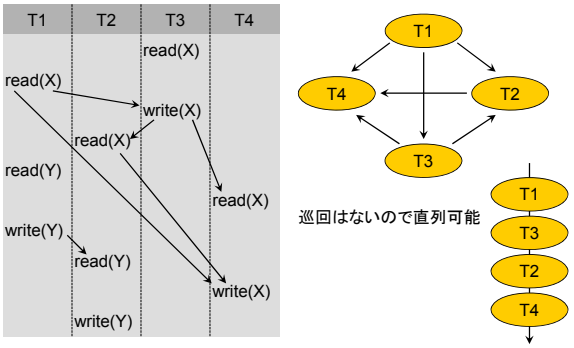
同時実行制御

- 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

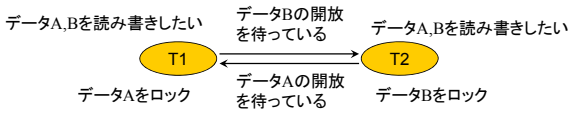


ロックを用いた同時実行制御

- 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

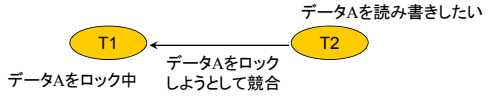


デッドロックへの対処方法

- 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



例題

時刻	T1	T2	T3
t0			read(Y)
t1	read(X)		
t2			write(Y)
t3	read(Y)		
t4		read(Y)	
t5			read(X)
t6			write(X)
t7			commit
t8	write(Y)		
t9	commit		
t10		write(Y)	
t11		commit	

- ワウンド・ウェイト方式での結果
- ウェイト・ダイ方式での結果

※トランザクションがアボートされたら次の時刻に再開するものとする

ワウンド・ウェイト方式

時刻	T1	T2	T3	ワウンド・ウェイト方式
t0			read(Y)	T3がYをロック
t1	read(X)			T1がXをロック
t2			write(Y)	
t3	read(Y)			T1がYをロックしようとして停止（T1＞T3）
t4		read(Y)		T2がYをロックしようとして停止（T2＞T3）
t5			read(X)	T3がXをロックしようとしてT1をアボート
t6			write(X)	（T3＜T1）
t7			commit	T3コミット
t8	write(Y)			T1がYをロックしようとしてT2をアボート
t9	commit			（T1＜T2）
t10		write(Y)		
t11		commit		T1コミット → T2コミット

ウェイト・ダイ方式

時刻	T1	T2	T3	ウェイト・ダイ方式
t0			read(Y)	T3がYをロック
t1	read(X)			T1がXをロック
t2			write(Y)	
t3	read(Y)			T1がYをロック、アボート（T1＞T3）
t4		read(Y)		T2がYをロック、アボート（T2＞T3）
t5			read(X)	T3がXをロックしようとして停止（T3＜T1）
t6			write(X)	再開したT1が先にXをロックするため
t7			commit	T2は、Yをロック、アボートし続ける
t8	write(Y)			T1がYをロック、アボート（T1＞T3）
t9	commit			T1がアボートしたらT3が実行できコミット
t10		write(Y)		T1がYをロックしようとして停止（T1＜T2）
t11		commit		T2コミット → T1コミット

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
    - 学籍番号が00100の学生の履修している各科目の科目番号、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
         学籍番号 = '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

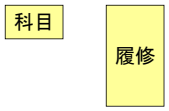
問い合わせの実現方法

- 同じ問い合わせを実現するときでも、複数の方法がある

①$\pi_{\text{科目.科目番号, 科目名, 成績}}(\sigma_{\text{科目.科目番号＝履修.科目番号}\wedge\text{学籍番号＝'00100'}}(\text{科目} \times \text{履修}))$

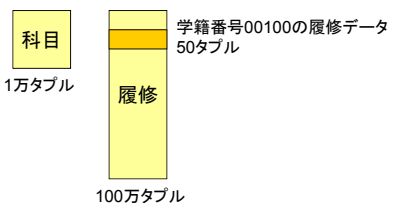
②$\pi_{\text{科目番号, 科目名, 成績}}(\sigma_{\text{学籍番号＝'00100'}}(\text{科目} \bowtie_{\text{科目.科目番号＝履修.科目番号}} \text{履修}))$

③$\pi_{\text{科目番号, 科目名, 成績}}(\text{科目} \bowtie_{\text{科目.科目番号＝履修.科目番号}} (\sigma_{\text{学籍番号＝'00100'}}\text{履修}))$



問い合わせ方法による効率の変化

- 仮定
  - 科目は1万タプル、履修は100万タプルとする
  - 学籍番号00100の学生は、50科目を履修

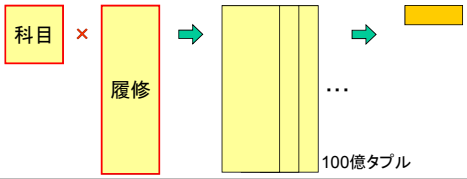


方法①での処理効率

- 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∧学籍番号＝'00100'(科目 × 履修))

– 科目 × 履修の直積により100億タプルの中間データ領域が必要

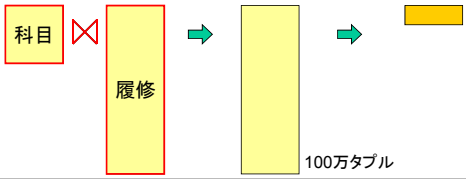


方法②での処理効率

- 方法②での処理効率

②π科目番号, 科目名, 成績(σ学籍番号＝'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、100万タプルの中間データ領域が必要

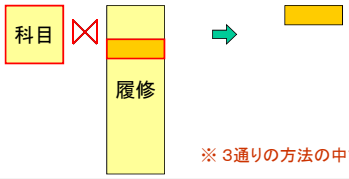


方法③での処理効率

- 方法③での処理効率

③π科目番号, 科目名, 成績(科目⋈科目.科目番号＝履修.科目番号(σ学籍番号＝'00100'履修))

– 先に50個の履修のタプル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



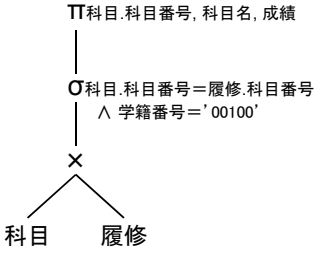
※ 3通りの方法の中では最も効率的

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タプルを除去するため、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

- 処理の手順を木表現で表す



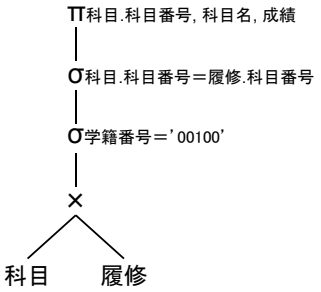
処理木を使った最適化

- 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

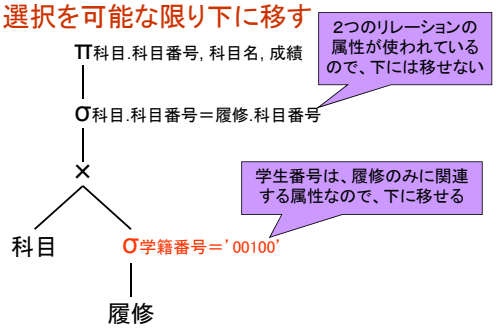
処理木を使った最適化

- 1. 選択条件を分解



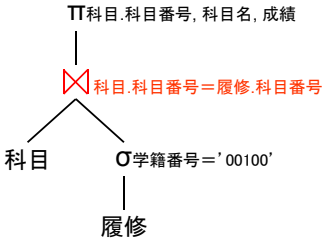
処理木を使った最適化

- 2. 選択を可能な限り下に移す



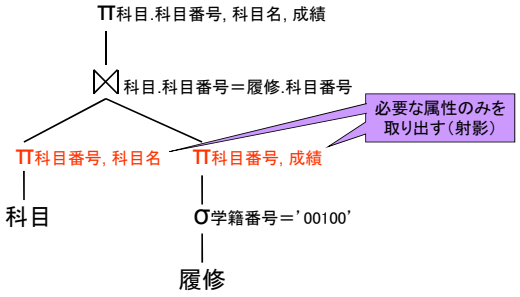
処理木を使った最適化

- 3. 連続する直積と選択を結合にまとめる



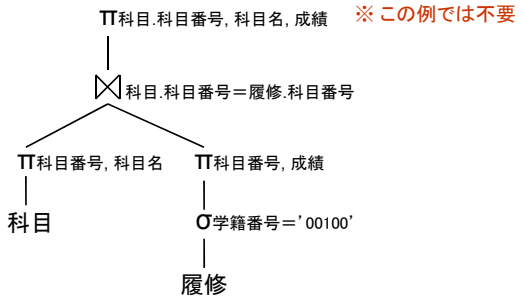
処理木を使った最適化

- 4. 射影を可能な限り下に移す



処理木を使った最適化

- 5. 連続した射影・選択をまとめて一つにする



基本データ操作の実行方法

基本データ操作

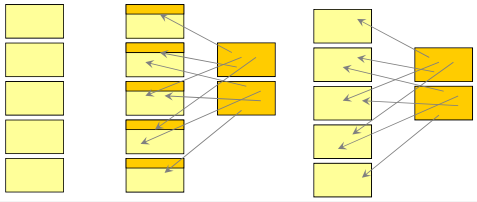
- 基本データ操作の方法
  - 複雑な問い合わせも、選択や結合などの基本的な操作の組み合わせによって実現される
    - 代数演算子を使った代数式で表される
  - それぞれの基本操作をどのように行うか？
    - インデックスの有無や、条件の種類によって、最適な処理方法は異なる
- 基本データ操作
  - 選択
  - 結合

データ操作の最適化

- 最適化において注意すべき点
  - データ量は、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
  - ディスクアクセスの速度は、メモリアクセスに比べると著しく遅い
  - なるべくディスクアクセスの回数を減らすような最適化が必要

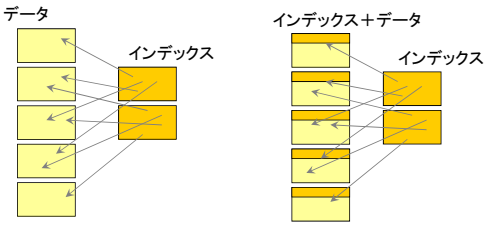
ページ単位での読み書き

- 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

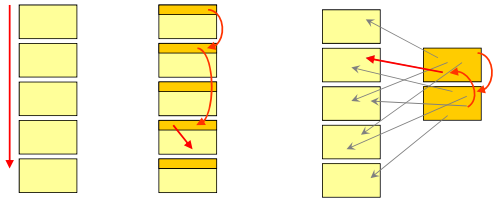


基本データ操作の実行方法

- 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

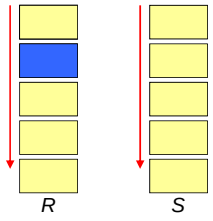
- 選択
- 結合

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

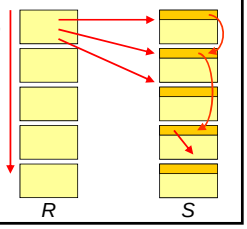
入れ子ループ結合

- 2つのリレーションの各タプル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



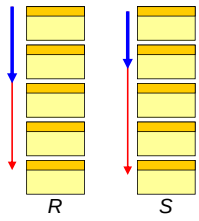
インデックスを用いた結合

- 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*の各タプルごとに処理
    - 結合対象の*S*のタプルを*S*のインデックスを用いて探索



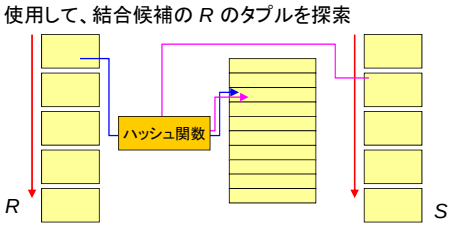
マージ結合

- 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の同じタプル同士を結合
  - 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* の各タプルを、結合する属性でハッシュを使用して分ける
  - リレーション*S* の各タプルごとに、同じハッシュ関数を使用して、結合候補の *R* のタプルを探索



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、データベースに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
  - データベースでは、何が起こってもデータの整合性が保たれることが重要

障害の種類

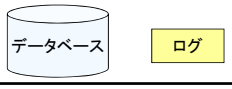
- トランザクション障害
  - トランザクションが、アボートされるなどして、完了できない状態
  - 並列システムではよくある事態
- システム障害
  - OSのハングアップやハードの障害などで、システムが強制終了されるような状態
  - これもたまに発生する
- メディア障害
  - ハードディスクなどのメディアに障害が生じてデータが読み出せなくなるような状態

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

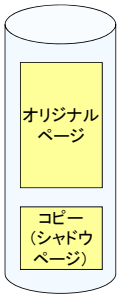
トランザクション障害への対応法（1）

- 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しないといけないので、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)

- シャドゥページング法
  - 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ら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
- 講義のまとめ
- 授業アンケート

講義のまとめ

講義の達成目標（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。

講義の位置づけ

- 情報の必修科目
  - 本科目の単位を取得しないと、卒業できない
- 学科の学習・教育目標(B) に対応
  - コンピュータ応用とシステム理論を学び、時代の要請に呼応した新たな情報システムを創造し、開発を行うための基礎能力を身に付ける

単位の認定方法（確認）

- 期末テスト（40点）
  - 教科書・ノート持込不可
    - 基本的な原理を理解しておけば解ける問題を出題
- 期末レポート（40点）
  - 与えられた課題を作成して提出
- 毎回の講義の演習問題・演習課題（20点）
  - 講義中の演習問題、データベース演習課題
- 出席（ICカード＋クリッカー＋演習問題提出）
  - 成績には考慮しない、一定回数の欠席で不合格
    - 自分の出席回数は、きちんと自分で把握すること

期末試験・レポート

- 期末試験
  - 7月29日（水）3限目 2102講義室
    - 正確な時間・会場は、掲示を確認すること
  - 試験範囲は、講義で扱った内容全て（演習も）
- 期末レポート
  - 8月10日（月）17:00（厳守）
    - 締め切り以降の提出は一切認めない
  - Moodleから提出
  - 端末室の利用可能期間に注意すること

レポート課題

- 課題内容
  - 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. スキーマの設計
     - データベースに格納するデータを決めて、思いつく属性を挙げる → 正規形を満たすように正規化
  2. テーブルの作成、データの追加
  3. Webインターフェースの作成
     - 一覧表示・追加・削除・修正
     - なるべく実用的に使えるような検索機能などを追加

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、試験に自信のない人は、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

データベースの概要

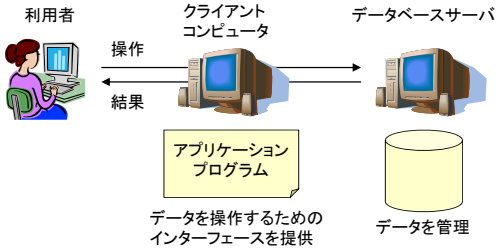
- データベースって何だろう？
  - 大量のデータを効率良く管理するためのシステム
  - データのモデリング、検索インターフェース、大量データ処理などの機能を提供
  - 他のプログラムからデータを利用できる
- データベースの応用
  - 企業の顧客・売上データなど
  - 最近では、地理データベース、マルチメディア、DNAデータ、科学技術データ なども扱われる
  - SEなど情報系に就職する人の多くは、何らかの形でデータベースシステムに関わる可能性が高い

データベースの社会での応用

- 店頭で販売されるソフトウェアはごく一部
- 実際には、企業からの注文で、企業内部で使われるソフトウェアの開発が多い
  - 企業のデータ（顧客・売り上げ・業務情報など）を管理するようなソフトウェア
  - データの管理部分にはデータベースシステムを利用し、ユーザーインターフェースのみを開発
  - データベースの設計・構築・最適化などの作業
  - データベースと密接に関わる仕事が多くなる

データベースの応用のイメージ

- データベース作成と Webインターフェース



情報系の職種（おおまかな分類）

- プログラマ（コーダー）
  - 指示された通りにプログラムを書く人
- 開発者・設計者、システムインテグレータ
  - どのようにシステムを作るか設計をし、プログラムも書く
- システムエンジニア
  - データベースの構築や管理
  - コンピュータやネットワークの管理
  - 利用者の要求を聞き、ソフトウェアの仕様をまとめる
  - プログラマや設計者・開発者の仕事をすることもある
- プロジェクトマネージャ
  - システムの開発を管理

この講義はどのような役に立つか

- この講義を受講したからと言って、何らかの企業に簡単に就職できるようなことはない
- 本講義の範囲内で扱うのは、ごく基礎的な知識・演習のみ
  - 単にデータベースの知識だけでなく、情報工学の理解を深める、という意味でも重要
- 情報工学の常識として、最低限知っておくべき内容
- 自分でより勉強したい人のための出発点

企業や就職との関係

- 講義では基本的な演習しか行わないので、情報系の仕事で働きたい人は、自分でいろいろな技術を勉強することをすすめる
  - 就職時は、講義以外でどのような努力・勉強をしてきたのかが問われる
  - 演習やプロジェクトでどのような工夫をしたのか、などもアピールになる（理解度・発想力・説明力）
- 情報系の仕事を目指さない人も、本講義・演習の内容くらいは押さえるべき
  - どういう仕事を目指すのかを決めることが必要

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

  - 期末試験
  - レポート提出

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 などのフリーなシステム
  - Apach 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)などのライブラリによるUI
    - enchant.js ゲーム開発ライブラリ
- 資格
  - Oracle認定試験 など

演習問題・授業アンケート

- 演習問題・授業アンケート(選択式)
  → いつものマークシートに記入、回収
- 授業アンケート(選択式・記述式)(学部共通)
  → 他の科目と一緒に、Webから入力

今日の内容のまとめ

- 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
- 講義のまとめ
- 授業アンケート

今後の授業日程

- 期末試験
- 試験後の授業(第15回目)はなし
  - レポート課題のための自習演習時間とする
  - 質問等があれば対応する
- レポート課題締切