具体的な講義の項目

- イントロダクション
- データベースのモデル
- 関係データベースの操作（関係代数、関係論理）
- 正規化、概念設計
- 問い合わせ言語
- トランザクション処理
- データベースの内部構造
- データベースの問い合わせ処理
- その他（オブジェクト指向DB、XML/RDF-DB等々）

データ定義（制約）

- 一貫性制約 (Integrity Constraint)
  - 更新でデータベースの内容に矛盾が生じないようにする
  - キー制約 (Key Constraint)
    - キー属性に格納されるデータはnull値になってはならない
  - 外部キー[参照]制約 (Foreign Key [Referential] Constraint)
    - 外部からキーとして参照される値は消えてはならない

データ定義例（1）

- 関係の作成

```
CREATE  TABLE   製造元
( 商品番号  CHAR(16) NOT NULL,
  商品名 CHAR(32) VAR,
  製造元 CHAR(16) VAR,
  希望小売価格  INTEGER,
  PRIMARY KEY(商品番号),
  FOREIGN KEY (商品番号) REFERENCES 秋葉原支店);
```


データ定義例（2）

- 関係の削除
  - DROP TABLE 秋葉原支店;
- 属性の追加／変更／削除
  - ALTER TABLE 製造元 ADD COLUMN 製造国 CHAR(8);
  - ALTER TABLE 製造元 ALTER COLUMN 製造元 SET 製造会社 CHAR(8);
  - ALTER TABLE 製造元 DROP COLUMN 希望小売価格;

埋め込み(組み込み) SQL

- 埋め込み SQL の文の先頭に EXEC SQL を付け、親言語の文と区別
- 親言語から変数で参照するため、: を変数の頭につける
- SQL の中では INTO 句を用いて親変数に代入
- 検索結果のタプルを1つずつアクセスするためにカーソルを使う

埋め込み(組み込み) SQL例（1）

```
EXEC SQL BEGIN DECLARE SECTION;
  char item[16];
  int  price, amount;
EXEC SQL END DECLARE SECTION;
  int x;
EXEC SQL CONNECT TO XXX;
EXEC SQL DECLARE c CURSOR FOR
  SELECT ITEM, PRICE, AMOUNT
  FROM Akihabara;
EXEC SQL OPEN c;
```


埋め込み(組み込み) SQL例（2）

```
EXEC SQL WHENEVER NOTFOUND GOTO printresult;
    x = 0;
    While(TRUE) {
        EXEC SQL FETCH c
          INTO :item, :price, :amount;
        if(price >= 100,000) x += amount;
    }
printresult:
    EXEC SQL CLOSE c;
    printf("Total Amount = %d¥nl", x);
```


ちょっと考えてみよう（6－1）

- DECLARE c CURSOR で c というカーソルを宣言して、FETCH c でカーソルのあるタプルのデータを取り込んでいる訳ですが、そのためにはどの時点でカーソルを宣言した SQL 文を実行しているか考えてみましょう

VIEW

- 仮想的な関係の定義
  - 外部スキーマと見ることもできる
- 目的
  - 必要な情報のみを提供 (セキュリティ)
  - 同じ問い合わせ文を何度も書かなくていよう
  - 複雑な問い合わせを書けないユーザのサポート
- 良く使われる
  - MS Access の問い合わせ定義は実は VIEW 定義と見なせる
- 非正規形の関係を定義しておくことも可能
- ただし、更新の対象とするのは難しい
- 一般に再帰的な定義はできない

VIEW の定義例（1）

- 商品分類がComputerであるような商品の商品番号と商品名、製造元を示す仮想的な関係を定義

```
CREATE VIEW Computer製品(商品番号, 商品名, 製造元)
   AS SELECT      商品番号, 商品名, 製造元
        FROM      製造元関係
        WHERE     商品名 IN (
                SELECT      商品名
                FROM        商品分類
                WHERE       商品分類 = 'Computer');
```


VIEW の定義例（2）

- VIEW を用いた VIEW

```
CREATE VIEW Computer在庫(商品番号, 在庫)
    AS SELECT    商品番号, 在庫
       FROM      秋葉原支店
       WHERE     商品番号 IN (
          SELECT      商品番号
          FROM        Computer製品);
```

- VIEW の削除

```
DROP VIEW Computer製品;
```


ちょっと考えてみよう（6－2）

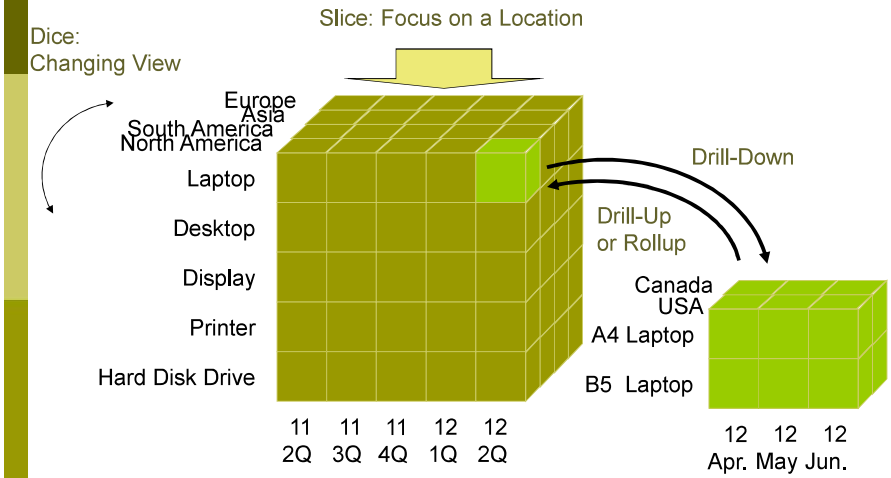
1. 以下の問いを示す商品分類毎最低価格（商品分類、最低販売価格）という VIEWを定義してみよう。「秋葉原支店において在庫数が 5 以下である商品の商品分類ごとの最低販売価格を知りたい。」

2. 上で定義した商品分類毎最低価格という VIEW を用いて以下の問いに対応するSQL文を書いてみよう。「秋葉原支店において在庫数が 5 以下である商品で最低販売価格が50,000 円以上の商品分類を知りたい。」

VIEW Materialization

- 一般に、VIEW は問い合わせの形で記憶
  - VIEW を用いた問い合わせが発生すると、その場で記憶しておいた問い合わせに置き換える
- 頻繁に同じ VIEW が用いられる場合
  - 検索結果の関係を記憶しておいた方が効率がよい
    - 例えば、何度も結合演算をしなくてすむ
  - これを Materialization（実体化）と呼ぶ
- メリット
  - VIEW の検索性能が向上
- デメリット
  - 元の関係が更新された場合に、その更新が反映されない
  - 更新を反映するための機構が必要

データキューブ (Data Cube)



ちょっと考えてみよう（6－3）

1. **データキューブの面を何回も切り替えて表示することを考えたとき、実体化した場合としなかった場合の計算コストの大小を考えてみましょう。**

2. **店舗の売り上げの履歴等をデータキューブに格納する場合、過去のデータに対する更新の頻度が実体化とどう関係するか考えてみましょう。**

関係データベースと Prolog

- 強い関連
- 関係データベースの各タプル
  - Prolog の fact と見なす
    - 一階述語論理（First Order Logic）の単位節（Unit Close）
  - 関係名を述語名
  - 属性名は引数番号との対応
- 負節は問い合わせ
  - 負節の述語名で関係を示す
  - 検索条件は変数束縛
  - 結果も変数束縛

対応例

```
製造元(GDC10, Desktop, GELL).
製造元(GLC20, Laptop, GELL).
製造元(FLC33, Laptop, Fepson).
    :

秋葉原支店(GDC10, 150,000, 5).
秋葉原支店(GLC20, 90,000, 12).
秋葉原支店(FLC33, 70,000, 3).
        :

商品分類(Computer, Desktop).
        :
```


問い合わせ例

- 製造元がGELLであるような商品の商品番号は
  - ?- 製造元(A, __, GELL).
    - A = GDC10;
    - A = GLC20;
    - no.
- 秋葉原支店で在庫数が10以上の商品の商品番号、価格、在庫数は
  - ?- 秋葉原支店(B, C, D), D>10.
- 結合演算も2述語間の変数束縛
  - 製造元がFepsonである商品の秋葉原支店での販売価格
    - ?- 製造元(X, __, Fepson), 秋葉原支店(X, Y, __).
- OR の表現
  - ?- 秋葉原支店(X, Y, _); 日本橋支店(X, Y, _).

ちょっと考えてみよう(6－4)

- 商品分類がComputerに属するGELLの商品で、秋葉原支店か日本橋支店で販売価格が100,000円以下で売られている商品の商品名と商品番号を知りたいとき、Prolog で書くとどうなるでしょう。

VIEWとProlog

- Prolog の rule（一階述語論理の確定節）は VIEW と見なせる
  - Computer製品(A, B, C) :-
    製造元(A, B, C),
    商品分類(Computer, B).
- Prolog は再帰呼び出しも扱うことができる
  - 祖先(A, B) :- 親(A, B).
  - 祖先(A, B) :- 親(A, C),祖先(C, B) .
  - 一般の VIEW では再帰呼び出しはできない

Prologと関係データベースの差

- 集合全体ではなく1タプル毎に扱う
  - Tuple at a time
- データの更新と不揮発性
  - 更新は一階述語論理の外
    - assert(), retract() 述語で更新しても、プログラムを終了すれば更新内容は消える。
- データベースの共有
  - 同じデータベースを複数のユーザが同時に使うことはできない。

演繹データベース(Deductive Database)

- 一階述語論理と関係データベースの融合
  - 更新やデータ共有の可能な一階述語論理
  - データベースへの高度な機能の導入
    - 再帰的なVIEW
- これまでの研究成果
  - 停止性の問題
  - 否定の取り扱い
- 問題点
  - 現実のアプリケーションが少ない