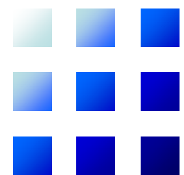
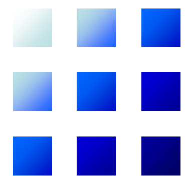
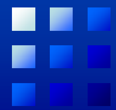


データベース
【9:リレーショナル データベース言語SQL】

石川 佳治

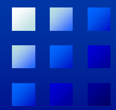


背景



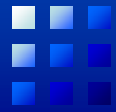
歴史的背景

- 1970年代よりリレーショナルDBMSの研究開発進む
  - System R：IBM
  - INGRES：UC Berkeley
- 1980年代からリレーショナルDBMSの実用化
- 実用的なデータベース言語が求められる
  - リレーショナル代数，リレーショナル論理では不十分な面がある
  - データの更新，スキーマの定義，アクセス権制御
  - 集計計算処理，ソーティング
  - 使い勝手



SQLの位置づけ

- データベース言語の国際的な標準
  - 「シークェル」と発音されることが多い
- System Rにおいて当初開発される
  - 当初はSEQUEL（Structured English Query Language）と呼ばれる
- ISOでの国際規格化
- 日本ではJIS規格化
  - 「エスキューエル」と呼ぶ



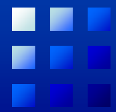
SQLの利点(1):ユーザの立場

① 一つのデータベース言語を学ぶことで各種DBMSが利用可能

② 異なるDBMS間のアプリケーションプログラムの移植・連携が容易

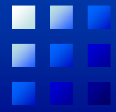
③ 汎用性のあるツールやユーティリティの開発が可能

④ DBMS利用技術のノウハウの共有



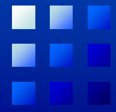
SQLの利点(2):ベンダーの立場

- 他社のDBMSとの連携が可能
- 今日のSQLの仕様は大規模
  - 各社ともSQLの仕様全体は実装していない
  - SQLは各ベンダーのDBMSの機能の「最大公約数」的な意義もある
  - 先進的な機能の実装
⇒SQLの拡張仕様に導入
⇒他社も追従



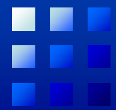
SQLの標準化：主な流れ

- 1986年：ANSIによる最初の標準規格
  - 1987年：ISOによるSQL規格第1版（SQL87）
  - 1989年：マイナーな更新（SQL89）
- 1992年：大幅な機能拡張
  - SQL92：通称SQL2
  - RDBの完成型，新たなニーズへの対応
  - 初級，中級，上級の3つの適合性レベル
- 1999年：さらに拡張
  - SQL99：通称SQL3
  - オブジェクトリレーショナルデータベース機能
- 2003年：Java/OLAP/XML機能，マルチメディア，外部データ連携



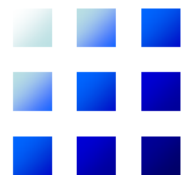
SQL標準と実際のシステム

- 市販のSQL製品はSQL92（SQL2)の初級レベルに準拠するものが多い
- 中級･上級レベル, SQL3の機能は選択的に対応

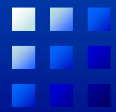


SQLの参考書籍

- 山平，小寺，土田：SQLスーパーテキスト，技術評論社，2004年，ISBN4-7741-1974-1
- 土田，小寺：－SQL最新標準規格―SQL2003ハンドブック，ソフト・リサーチ・センター，2004年，ISBN4-88373-207-X



基本概念

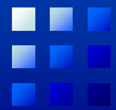


リレーショナルデータモデルとの相違点

A) 重複したタプルの存在
- リレーショナルデータベースでは重複は存在しない
- しかし，現実のデータ操作では重複が意味を持つことがある：集約処理（平均値，カウント）
- SQLでは重複したタプルの存在を許す

B) 属性やタプルの順序付け
- SQLでは属性および属性値は明示的に順序付けされたものとして扱う：リレーショナルデータベースでは並び順に意味はない
- 並び順を指定することもできる

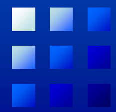
C) 属性にデータ型（例：整数，文字列）を対応付ける
- データ型に応じた検索条件が指定可能



SQL特有の用語

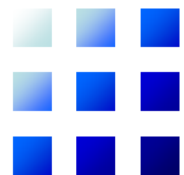
- リレーショナルデータモデルとの相違点により，SQLでは異なる用語を用いる

リレーショナルデータモデル	SQL
リレーション	表（table）
タプル	行（row）
属性	列（column）

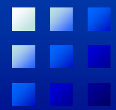


SQLの利用形態

- 直接起動
  - 直接ユーザが利用
- プログラムからの利用
  - プログラミング言語(C, C++, Java, …)と組み合わせる
  - さまざまなアプローチが存在
    - 埋め込みSQL
    - ODBC, JDBC
    - …

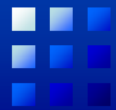


データ定義



SQLの表の種類

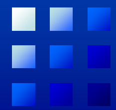
- 実表（base table）
  - データの実体を伴う表
- ビュー表（viewed table）
  - ビューを表現
  - データの実体は存在しない仮想的な表
  - 問合せ時に実表からデータを求める
- 導出表（derived table）
  - 問合せ結果として一時的にできる表



表の定義(1):定義例

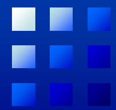
- 実表の定義例

```
CREATE TABLE 科目
(科目番号 CHAR(3) NOT NULL,
 科目名     CHAR(12) NOT NULL,
 単位数     INTEGER,
  PRIMARY KEY (科目番号),
  CHECK (単位数 BETWEEN 1 AND 12))
```



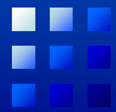
表の定義(2):データ型

- 文字列
  - CHAR, CHARACTER
- 数
  - 真数値:INTEGER, SMALLINT, DECIMAL, NUMERIC
  - 概数値:REAL, FLOAT, DOUBLE
- ビット列
- 日時
- 時間隔



表の定義(3):整合性制約

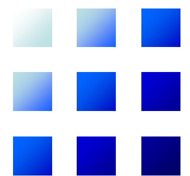
- 空値に関する制約
  - NOT NULLを指定すると，指定した列の値は空値（ナル値）をとることができない
- 主キーの記述
  - PRIMARY KEY（科目番号）などと指定
  - 候補キーも指定可能
- ドメイン制約
  - 値の範囲を指定:CHECK（単位数 BETWEEN 1 AND 12）
- 定義域（domain）が定義可能
  - CREATE DOMAIN 単位数 INTEGER CHECK (VALUE BETWEEN 1 AND 12)
  - データ型の代わりに利用可能
- 参照整合性制約
  - 外部キーを指定:FOREIGN KEYを用いる



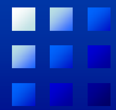
表の定義(4):定義例2

- 実表の定義例2

```
CREATE TABLE 履修
(科目番号 CHAR(3) NOT NULL,
 学籍番号 CHAR(5) NOT NULL,
 成績       INTEGER,
 PRIMARY KEY (科目番号, 学籍番号),
 FOREIGN KEY (科目番号)
     REFERENCES 科目(科目番号),
 FOREIGN KEY (学籍番号)
     REFERENCES 学生(学籍番号),
CHECK (成績 BETWEEN 0 AND 100))
```



問合せ



問合せの基本形

- SQLの典型的問合せ記述

```
SELECT T_i1.C_1, ..., T_im.C_m
FROM   T_1, ..., T_n
WHERE  ψ
```

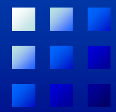
  - $T_1, \ldots, T_n$ は表名，$C_1, \ldots, C_n$は列名，ψ は条件式

- リレーショナル代数式での表現

$$\pi_{T_{i_1}.C_1,\ldots,T_{i_m}.C_m}(\sigma_{\psi'}(T_1 \times \cdots \times T_n))$$

- タプルリレーショナル論理式での表現

$$\{t^{(m)} \mid (\exists t_1)\cdots(\exists t_m)(T_1(t_1) \wedge \cdots \wedge T_n(t_n) \wedge \psi'' \\ \wedge t[C_1] = t_n[C_1] \wedge \cdots \wedge t[C_m] = t_{i_m}[C_m])\}$$



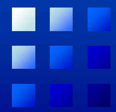
問合せ例(1)

- Q1:科目番号005の科目の履修者の学籍番号と成績の一覧

```
SELECT 履修.学籍番号, 履修.成績
FROM   履修
WHERE  履修.科目番号 = ’005’
```

  – 「履修.学籍番号」で「履修」表の「学籍番号」列を示す
  – 曖昧でない場合は表名を省略できる

```
SELECT 学籍番号, 成績
FROM   履修
WHERE  科目番号 = ’005’
```

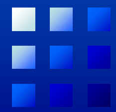


問合せ例(2)

- Q2:学籍番号00100の学生が履修した科目の科目番号，科目名，成績の一覧

```
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'
```

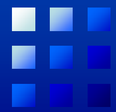
- 概念レベルの処理
  - 「科目」と「履修」の直積演算を実行
  - WHERE句の条件で選択演算を実行
  - SELECTで指定された列について射影演算を実行
- 「科目番号」は両方の表に存在:表名を省略不可



問合せ例(2’)

- Q2:学籍番号00100の学生が履修した科目の科目番号，科目名，成績の一覧
  - SQL92規格では，自然結合を表すNATURAL JOINが導入された
  - 先のSQL問合せとほぼ等価な問合せ

```
SELECT 科目番号, 科目名, 成績
FROM     科目 NATURAL JOIN 履修
WHERE  学籍番号 = ’00100’
```

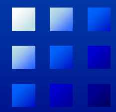


問合せ例(3)

- Q3: 情報工学専攻のいずれかの学生が履修した科目の科目番号と科目名の一覧

```
SELECT 科目.科目番号, 科目名
FROM    科目, 履修, 学生
WHERE  科目.科目番号 = 履修.科目番号
   AND 履修.学籍番号 = 学生.学籍番号
   AND 専攻 = '情報工学'
```

  - 教科書には「N'情報工学'」と書くように指示されているが,「'情報工学'」でよい

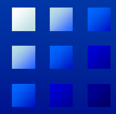


問合せ例(3’)

- Q3:情報工学専攻のいずれかの学生が履修した科目の科目番号と科目名の一覧

```
SELECT 科目番号, 科目名
FROM   科目 NATURAL JOIN 履修
   NATURAL JOIN 学生
WHERE  専攻 = ’情報工学’
ORDER BY 科目番号
```

  - NATURAL JOINを用いた例
  - ORDER BYにより，科目番号順にソートする
  - 降順にソートの場合は「ORDER BY 科目番号 DESC」と指定

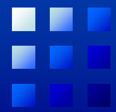


問合せ例(4)

- Q4：科目番号005の科目に関して学籍番号00100の学生よりも成績のよかった学生の学籍番号の一覧

```
SELECT y.学籍番号
FROM    履修 x, 履修 y
WHERE x.科目番号 = ’005’
  AND x.学籍番号 = ’00100’
  AND y.科目番号 = ’005’ AND y.成績 > x.成績
```

  - 「履修」を二重の意味で参照
  - x, y という**相関名**（correlation name）により区別
  - 教科書では「AS」が使われているが，なくてもよい



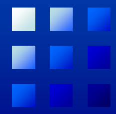
問合せ例(5)

- Q5:全科目の科目名と単位数の一覧
  - 問合せ条件がなければWHERE句を省略可

```
SELECT 科目名, 単位数
FROM    科目
```

  - 重複を除去したい場合にはDISTINCTを使用

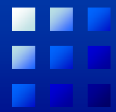
```
SELECT DISTINCT 科目名, 単位数
FROM    科目
```



問合せ例(6)

- Q6:単位数が3単位以上の科目番号，科目名，単位数の一覧
  - FROM句で指定した表のすべての列を問合せ結果に含めたい場合は,「*」を用いる

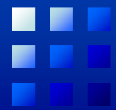
```
SELECT  *
FROM    科目
WHERE   単位数 >= 3
```



集合関数(1)

- 集合関数(set function)
- 集計演算を実行
  - COUNT(行数のカウント), SUM(合計), AVG(平均値), MAX(最大値), MIN(最小値)
- Q7:科目番号005の科目の平均点

```
SELECT AVG(成績)
FROM　履修
WHERE 科目 ='005'
```



集合関数(2)

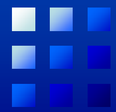
- 例:科目番号001の科目の履修者数

```
SELECT COUNT(*)
FROM   履修
WHERE 科目 ='001'
```

  - COUNTのみ引数に * を指定可能:行数のカウントを意味する
  - 図の場合，結果は2

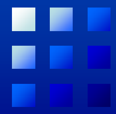
- 例:科目番号001の科目で成績が出ている履修者の数
  - COUNT(*) → COUNT(成績)に修正
  - 結果は1:空値は対象外

科目番号	学籍番号	成績
001	00001	90
002	00001	70
002	00001	90
001	00002	NULL



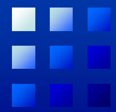
探索条件(1)

- WHERE句で指定
- 比較述語
  - =, >=, <=, >, <, <>
  - AND, OR, NOTで結合
- BETWEEN述語:範囲指定
  - WHERE 年齢 BETWEEN 20 AND 30
- IN述語:いずれかと等しい
  - WHERE 学部 IN ('工学部', '文学部', '理学部')
- NULL述語:空値かどうかを検査
  - WHERE 成績 IS NULL



探索条件(2)

- LIKE述語
  - 文字列に適用:パターンを指定可能
  - 例:WHERE 名前 LIKE '石川%'
- パターンの例
  - 'abc%':abcで始まる文字列
    - %は0以上の任意の文字列にマッチ
  - 'abc_ _ _ghi':1～3文字目がabc, 7～9文字目がghiの文字列
    - _ は任意の1文字にマッチ

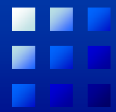


グループ表（1）

- Q8：全科目について科目番号と平均点の一覧

```
SELECT 科目番号, AVG(成績)
FROM    履修
GROUP BY 科目番号
```

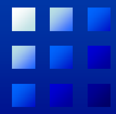
  - 「GROUP BY 科目番号」によりグループ化を指定
  - 同じ科目番号の値を持つ行が一つのグループにまとめられたグループ表（grouped table）を作成
  - グループ表では，SELECTの後ろにリストできるのは，各グループについて一意に値が決まる項目のみ



グループ表(2)

- グループ表の例

科目番号	学籍番号	成績
001	00001	90
001	00002	80
002	00001	90
002	00003	70
…	…	…

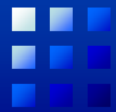


グループ表(3)

- Q9:情報工学専攻の学生が履修した科目の科目番号と情報工学専攻の学生に関する平均点の一覧

```
SELECT 履修.科目番号, AVG(成績)
FROM　 履修, 学生
WHERE 履修.学籍番号 = 学生.学籍番号
　AND　 専攻 = ’情報工学’
GROUP BY 履修.科目番号
```

  – GROUP BY句とWHERE句の組合せの例

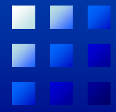


グループ表(4)

- Q10：履修者が30名以上の科目の科目番号，履修者数，平均点の一覧

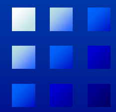
```
SELECT 科目番号, COUNT(*), AVG(成績)
FROM    履修
GROUP BY 科目番号
HAVING COUNT(*) >= 30
```

  – グループ表の中で選択演算を行うためにHAVING句を用いる



集合演算(1)

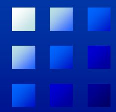
- UNION(和), EXCEPT(差), INTERSECT(共通部分):これらについては重複除去を実施
- UNION ALL, EXCEPT ALL, INTERSECT ALL と指定すれば重複を除去しない



集合演算(2)

- Q11：実習課題があるか，あるいは単位数が5単位以上の科目の科目番号，科目名，単位数の一覧

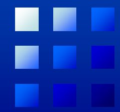
```
SELECT 科目.*
FROM　科目, 実習課題
WHERE 科目.科目番号 = 実習課題.科目番号
UNION
SELECT *
FROM　科目
WHERE 単位数 >= 5
```



集合演算(3)

- Q12:実習課題のない科目の科目番号と科目名の一覧

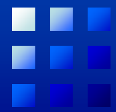
```
SELECT 科目番号, 科目名
FROM 科目
EXCEPT
SELECT 科目.科目番号, 科目名
FROM 科目, 実習課題
WHERE 科目.科目番号 = 実習課題.科目番号
```



副問合せ（1）

- 副問合せ（subquery）: 入れ子問合せの機能
- Q3: 情報工学専攻のいずれかの学生が履修した科目の科目番号と科目名の一覧

```
SELECT 科目番号, 科目名
FROM 科目
WHERE 科目番号 IN
    (SELECT 科目番号
     FROM    履修, 学生
     WHERE  履修.学籍番号 = 学生.学籍番号
        AND 専攻 = '情報工学' )
```



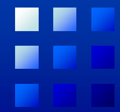
副問合せ（2）

- Q12：実習課題のない科目の科目番号と科目名の一覧

外への参照：内側の問合せから外側の問合せの表を参照可能

```
SELECT 科目番号, 科目名
FROM　 科目
WHERE NOT EXISTS
   (SELECT *
    FROM　 実習課題
    WHERE 実習課題.科目番号 = 科目.科目番号)
```

  - EXISTS述語：副問合せの結果が空でないとき真
  - NOT EXISTSは，副問合せに何か結果があれば偽

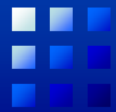


副問合せ(3)

- 例: 平均年齢より年齢が大きい学生を求めよ

```
SELECT 学生名
FROM   学生
WHERE 年齢 >
   (SELECT AVG(年齢) FROM 学生)
```

  – 副問合せの結果得られる行数が1行となる場合，比較述語の右辺に副問合せを指定可能

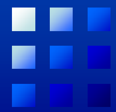


限定比較述語(1)

- 例:少なくとも一人が100点をとった科目の情報

```
SELECT *
FROM    科目
WHERE 科目番号 = ANY (SELECT  科目番号
                      FROM     履修
                      WHERE   成績 = 100)
```

  - 限定子ANY:副問合せの結果のどれか一つが比較条件を満たすと真になる
  - 限定子SOME:ANYと同じ意味

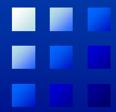


限定比較述語(2)

- 例:各科目の最高点の情報を列挙

```
SELECT *
FROM    履修 x
WHERE x.成績 >= ALL
   (SELECT 成績
    FROM    履修 y
    WHERE x.科目番号 = y.科目番号)
```

  – 限定子ALL:副問合せの結果のすべてが比較条件を満たすと真になる



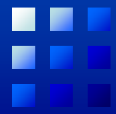
SELECT句に関する補足

- 問合せ結果の列名を指定することが可能
  - 例:科目ごとの平均点

```
SELECT 科目番号, AVG(成績) AS 平均点
FROM　履修
GROUP BY 科目番号
```

- 算術演算を指定可能
  - 例:学生の10年後の年齢

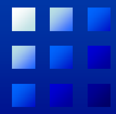
```
SELECT 氏名, 年齢 + 10
FROM　学生
```



ビュー(1)

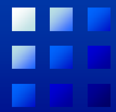
- ビュー表(view table)
- 実際のデータは存在しない仮想的な表
- 問合せが発生した時点で実表から計算
- 通常の表のように問合せ可能
- 例:実習を伴う科目だけに関するビュー表

```
CREATE VIEW
   実習科目(科目番号, 科目名, 単位数) AS
  SELECT 科目.*
  FROM     科目, 実習課題
  WHERE  科目.科目番号 = 実習課題.科目番号
```



ビュー(2)

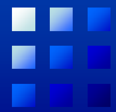
- ビュー表の利点
  - アクセス権を適切に設定することで，ある人にのみ見せたいデータ／見せたくないデータのアクセス管理が可能
  - ユーザが興味あるデータのみに絞って情報を表現
- 注意点
  - ビュー表の更新では問題が発生する場合がある: ビュー更新問題
  - いくつかの制約条件を満たす場合のみ，ビューの更新が可能



データ更新(1)

- INSERT, DELETE:行の挿入，削除
- UPDATE:列の値の更新
- U1:科目番号002の科目の実習課題03として「シェル作成」を追加

```
INSERT INTO 実習課題
  VALUES ('002', '03', 'シェル作成')
```

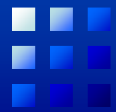


データ更新(2)

- U2：科目番号010の科目の履修者として，学籍番号が’00099’以下の学生を全員登録

```
INSERT INTO 履修(科目番号, 学籍番号)
  SELECT '010', 学籍番号
  FROM     学生
  WHERE  学籍番号 <= '00099'
```

  – 追加された行の「成績」列の値は空値となる



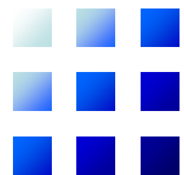
データ更新(3)

- U3：科目番号005の科目の実習課題をすべて削除

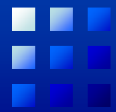
```
DELETE FROM 実習課題
WHERE  科目番号 = ’005’
```

- U4：科目番号010の科目の単位数を3単位に変更

```
UPDATE 科目
SET         単位数 = 3
WHERE   科目番号 = ’010’
```



余談：最近のSQL

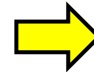


CASE式

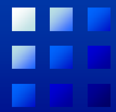
- 場合分け:学生の年齢をもとに世代情報を求める

```
SELECT
  氏名,
  CASE WHEN 年齢 < 20 THEN ’10代’
  CASE WHEN 年齢 >= 20 AND 年齢 < 30 THEN ’20代’
  ELSE ’30代以上’ END as 世代
FROM 学生
```

学籍番号	氏名	年齢
001	山田一郎	21
002	鈴木次郎	32
003	渡辺花子	19



氏名	世代
山田一郎	20代
鈴木次郎	30代以上
渡辺花子	10代



窓関数

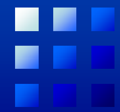
- リレーションをタプルのシーケンスとみなし，その上に区間（窓：ウィンドウ）を設定して処理を適用
- PARTITION BY句が代表的

```
SELECT 品名, 種別, 売上高
       RANK() OVER (PARTITION BY 種別
                    ORDER BY 売上高 DESC) ランク
FROM 売上
```

品名	種別	売上高
バナナ	果物	120
キャベツ	野菜	320
リンゴ	果物	250

⇒

種別	品名	売上高	ランク
果物	リンゴ	250	1
果物	バナナ	120	2
野菜	キャベツ	320	1

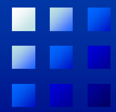


WITH句（1）

- 副問合せに名前をつけ参照可能とする
- WITH RECURSIVE句：再帰的問合せに使用

```
WITH RECURSIVE 信長子孫(名前) AS
  (SELECT 子
   FROM 親子関係
   WHERE 親 = ‘織田信長’
   UNION ALL
   SELECT 親子関係.子
   FROM 親子関係, 信長子孫
   WHERE 親子関係.親 = 信長子孫.名前)
SELECT COUNT(*) FROM 信長子孫
```

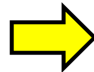
例：信長の子孫は何名いるか？



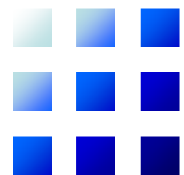
WITH句(2)

- タプルが増えなくなるまで再帰的に処理

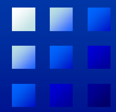
親	子
織田信秀	織田信広
織田信秀	織田信長
織田信秀	織田信行
織田信長	織田信忠
織田信長	織田信雄
織田信長	織田信孝
織田信忠	織田秀信
豊臣秀吉	豊臣秀頼
徳川家康	徳川秀忠
・・・	・・・



名前
織田信忠
織田秀信
・・・



余談：プログラムとの連携



プログラムからのSQL利用(1)

インターネット

・ウェブを介して 要求受付

・SQL問合せ を作成
・DBMSに発行

ウェブ サーバ

アプリ ケーション サーバ

SQL 問合せ

問合せ 結果

DBMS

DB

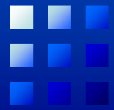
クライアント
(ウェブブラウザ)

・ユーザは 検索条件 (書籍名など) を入力

・HTMLを 配信

・問合せ結果 をもとに 各種の処理
・HTMLに加工

- 求められる機能
  - プログラム中からのSQLの発行
  - 問合せ結果の受け取り
  - 動的なSQLの発行:ユーザの条件指定は そのつど異なる



プログラムからのSQL利用(2)

和書 詳細サーチ

最低1つのボックスに検索条件を指定してください。指定する条件を増やすと、

洋書詳細サーチはこちらへどうぞ。

著者:

◯必ず一致 ◉姓名に含む

タイトル: データベース

◯で始まる ◉タイトルに含む

トピック:

◯で始まる ◉トピックに含む

(ジャンルでの検索に有効です)

ISBN:

(ハイフンなしで入力してください)

出版社:

その他の検索条件:

判型: すべて

対象年齢: 全年齢

出版年月: 2000

月

以降

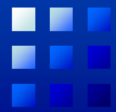
並べ替え: 売れている順番

サーチ

- 以下のような問合せをプログラム内で構築

```
SELECT *
FROM    書籍 x, 販売 y
WHERE  x.出版年 >= 2000
  AND x.タイトル LIKE ‘’%データベース%’
  AND x.書籍番号 = y.書籍番号
ORDER BY y.販売数 DESC
```

- DBMSに発行
- 結果として得られた行(タプル)の集合からHTMLを作成



PHP言語の例（1）

- HTML文書のひな型にデータベース問合せ結果を埋め込む（伝統的なPHPプログラムの例）

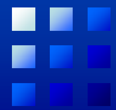
```
<html>
<h1>問合せ結果</h1>

$gakubu = argv[1];

<ul>
$result = exec(“SELECT 学籍番号, 氏名 FROM 学生 WHERE 学部 = $gakubu");
if ($result) {
  do {
    @$row = fetch_row($result, $i);
    if ($row) {
     print('<li>'. $row[0] . ', ' . $row[1] . '</li>' . "¥n")
    }
  } while ($row);
}
</ul>
</html>
```

プログラムの引数で変数$gakubuを初期化

exec( ) は，引数で与えられたSQL文を実行
$result変数には問合せ結果が入る

fetch_row( )は，問合せ結果から次の1行を読み込む

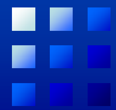
print( ) で出力する
‘.’ は文字列を結合することを表す
“¥n” は改行文字



PHP言語の例(2)

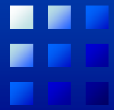
- 先のプログラムはHTMLのテンプレートで，$で始まる部分はHTML言語ではない
- ウェブ経由で問合せ条件が来ると，先のPHPプログラムをPHP言語処理系で解釈
  - $～の部分をPHP言語のルールで処理
  - 問合せ条件($gakubu)に変数を代入し，得られたSQLをDBMSに発行
  - 結果を受け取り加工してHTMLテンプレートに埋め込む
- 結果は右図のようになる

```
<html>
<h1>問合せ結果</h1>

<ul>
<li>001, 山田一郎</li>
<li>002, 鈴木花子</li>
…
</ul>

</html>
```



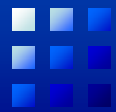
埋込みSQL（Embedded SQL）

- 教科書5.6節
- プログラム中にデータベース操作処理を埋め込む（図5.2）
- 埋込みSQLを含むプログラムをDBMS提供のプリプロセッサで処理
  - DBMSを呼び出すホスト言語の処理に置き換え
  - DBMS提供のライブラリなどとリンクし実行
- C言語/C++言語では最近では用いられない



ODBC, JDBC, SQL/CLI

- ODBC（Open Database Connectivity）
  - RDBMSにアクセスする共通インタフェースとしてMicrosoftが開発
  - DBMSに対する一連の処理をC言語の関数として提供
  - ソースレベルでのアプリケーションの可搬性を実現
- JDBC
  - ODBCのJava版
  - Java SEに含まれる
- SQL/CLI
  - CLIはCall Level Interface（呼出しレベルインタフェース）の略
  - ODBC, JDBCの考え方をもとに，規格にまとめたもの

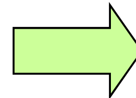


ODBCのイメージ

アプリケーションプログラム
- C言語で記述
- 各種ODBC関数の呼出しを含む

ODBCドライバ
- C言語のライブラリ
- DBMSのベンダーが提供

コンパイル



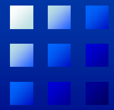
実行プログラム

ODBC関数群

SQL発行

問合せ結果取得

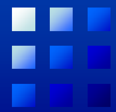
DBMS



JDBCの記述例

- 従業員テーブルから1行ずつ読み出す
- 各レコードから名前属性の値を取り出す
- ResultSetはJDBCが提供するクラス

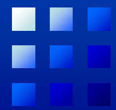
```
…

Class JdbcSample {
  …
  ResultSet resultSet;

  String sql = "SELECT * FROM 従業員";

  try {
    resultSet = statement.executeQuery(sql);
    while (resultSet.next()) {
      String name = resultSet.getString("名前");
      System.out.println(…);
    }
    …
  }
  …
}
```



DBMSが提供するプログラミング言語

- OracleのPL/SQLが代表的
  - PostgreSQLならPL/pgSQL
- 変数が使え，if, for, loopなどの制御構造を持つスクリプト言語
- SQLを埋め込みできる
- SQLのみでは書けない簡単な処理を記述

```
DECLARE
  CURSOR cur IS SELECT name FROM emp;
  str VARCHAR2(10);
BEGIN
  OPEN cur;
  LOOP
    FETCH cur INTO str;
    EXIT WHEN cur%NOTFOUND;
    DBMS_OUTPUT.PUT_LINE(str);
  END_LOOP;
  CLOSE cur;
END;
```

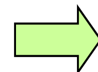


O/Rマッピング（1）

- オブジェクト-リレーショナルマッピング
  - ORMとも書く
- Javaなどのオブジェクト指向言語のクラスをRDBMSのテーブルに対応づける
  - プログラマはSQLを直接扱わなくてよい
- 例：Ruby on RailsのActive Record
  - students(id, name, age, …) というテーブルがすでにRDB上に作られていると想定

```
class Student < ActiveRecord::Base
end
```

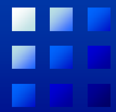
と宣言するだけで，自動的に



```
s = Student.new
s.id = “001”
s.name = “山田一郎”
…
```

などと書ける

studentsテーブルにレコードが挿入される

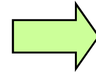


Ruby on RailsのActive Record（1）

- 問合せの例

```
s = Student.find(1)
```

idが1の学生のオブジェクトを
見つけて変数sに代入



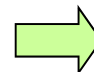
```
SELECT *
FROM students
WHERE id = 1
LIMIT 1
```

実は裏で
SQLが
発行されて
いる

「LIMIT 1」は，最初に見つかった
レコードを返す指示

```
s = Student.where(name: '山田一郎').take
```

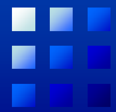
名前が山田一郎である学生のオブジェクトを
見つけて変数sに代入



```
SELECT *
FROM students
WHERE name = '山田一郎'
LIMIT 1
```

O/Rマッピングの注意点

- 複雑な問合せの記述は困難
- 必ずしもよいSQL問合せが発行される
とは限らない



Ruby on RailsのActive Record（2）

- 関連付け：例

```
class Customer < ActiveRecord::Base
  has_many :orders
end

class Order < ActiveRecord::Base
  belongs_to :customer
end
```

CustomerとOrderが
1対多で関連する
ことを示す

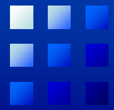
以下のようなコードが書ける

```
Customer.find(1).orders each do |o|
  print(“order date: “, o.date)
end
```

idが1の顧客による
発注のそれぞれの
日付を出力

裏で自動的に結合処理が行われる

LINQ



- マイクロソフトの.NET環境で利用可能
  - C#, Visual Basicなど
- LINQ to SQLの例
  - 若い従業員の名を求めるC#プログラム

```
…

var query =
  from e in db.Employees
  where e.Age < 30
  select n;

foreach (var x in query) {
  Console.WriteLine(“id = {0}”, x.Name);
}
```

埋め込みSQLと異なり,
プログラミング言語処理系が
LINQの構文を理解している
→ 型のチェックなどが可能