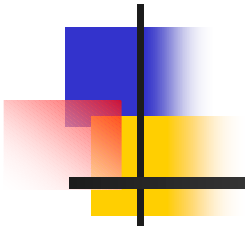
Club Db2 第103回

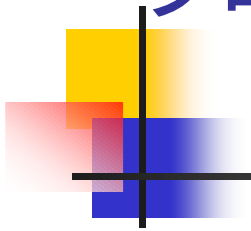
リレーショナルとはどんなことか

～ RDBとSQLの過去・現在・未来 ～



講演者：ミック

プロフィール



経歴：SIer勤務。主にDWH/BIシステムを得意としています。
　　現在は、パフォーマンス・チューニング専門チームに所属
　　サイジング、ベンチマーク、プロト性能検証（および火消し）

著書：『達人に学ぶ SQL徹底指南書』(翔泳社 2008)
　　『プログラミング学習シリーズ SQL』(翔泳社 2010/06)
訳書：J.セルコ『SQLパズル 第2版』(翔泳社 2007)

『CodeZine』、『Web+DB Press』などでSQLとRDBについての記事を連載。

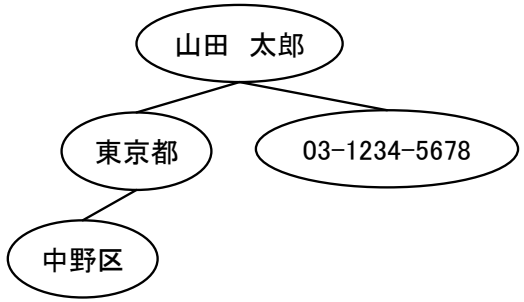
Twitterアカウント：copinemickmack

RDBと他のデータベースの違い

なぜリレーショナル・データベース？

データベースのモデルは他にもあるのに、なぜRDBが主流なのか。

・階層型データベース



・XMLデータベース

```
<address>
  <name>山田 太郎</name>
  <prefecture>東京都</prefecture>
     ・・・・・・
</address>
```

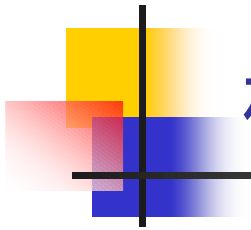
・リレーショナルデータベース

山田 太郎	東京都	03-1234-5678
加藤 一郎	千葉県	042-34-9999
鈴木 正一	福岡県	072-55-6789
・・・・・・		

・オブジェクト指向データベース

・キー・バリュー型データベース

（山田 太郎, 東京都）
（加藤 一郎, 千葉県）
（鈴木 正一, 福岡県）
・・・・・・



破壊的技術としてのRDB

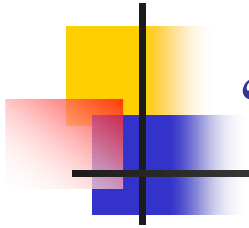
破壊的技術：既存の価値基準に照らすと性能の悪いオモチャに見えるが、新しい価値基準に照らすと既存技術を凌駕する技術のこと。

例：デジカメ ⇔ 銀塩カメラ

ワークステーション ⇔ メインフレーム

電気自動車 ⇔ ガソリン車

RDB ⇔ 階層型DB



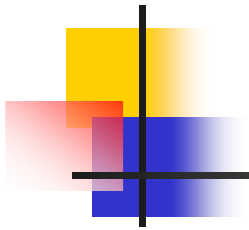
“ユーザフレンドリー”という新しい基準

- RDB VS 階層型DB
- 二次元表というユーザにとって馴染みやすいデータ構造を採用することで、「使いやすさ」という基準で有利だった。

　→ 逆に言うと、パフォーマンスや信頼性などの基準では、決して従来のモデルに勝てるものではなかった。

Ref.

「以前は情報システム部門の若手が時代に先がけてリレーショナルデータベース管理システムの導入を提案しても、上司がそれを斥けてきた。システムのパフォーマンスが良くないことが大きな理由の一つだったと聞いている。」（増永良文『リレーショナルデータベース入門』（1991, サイエンス社）



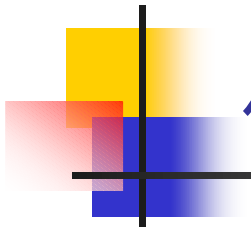
トレードオフのパズル

- 二次元表がデータ管理に便利であることは明らか。でも、それを管理するための言語が難しければ、総体としての利用コストは高くなってしまう。

→ 簡単にデータ管理出来る方法を探そう。

①アドレスなんて意識しない方が便利に決まっている。

②ループも無い方がいい。



住所なんて分からなくていい

このように、現在ではプログラミングとは基本的には、フォン・ノイマン型隘路を通る語の恐るべき交通量を計画し、細かく面倒を見ることである。そしてその交通量の多くは、意味のあるデータではなくて、**それがどこにあるかを見つけるためのもの**なのである。

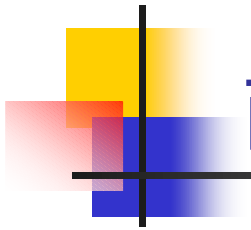
——ジョン・バッカス「プログラミングはフォン・ノイマン・スタイルから解放されうるか？」

関係型データベースにポインタはないというとき、物理レベルにおいてポインタが全くありえないということをいおうとしているのではない。逆に、そのようなレベルにおいてはポインタは確かに存在し得るし、実際にそうだろう。しかし、すでに説明したように、そのような物理レベルの記憶の詳細はすべて、関係型システムにおいては**ユーザから隠蔽されている**。

——クリス・デイト『データベースシステム概論　第6版』

Ref.

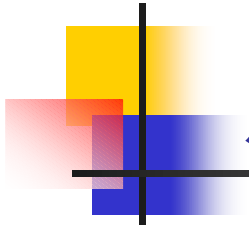
このRDBの「アドレス嫌い」の性格は、最近までオートナンバリング機能が標準SQLに取り入れられなかったこととも関係してる（それ自体が意味を持たない連番は、実質的にアドレスとしてしか機能しないから）。



面倒な手続きも無い方がいい

とりあえず、ループ、変数、代入をなくしてみました。

→ でも、それで機能的に貧弱になっては本末転倒



ループなんて嫌いだ

- 手続き型から宣言型への移行

  “プログラマでなくても DB を扱えるようにしたい”

  → そのためには、手続き型言語の躓きの石である「ループ」をなくさなければならない。

  → レコードじゃなくて、最初からレコードの「集合」を操作対象にすれば、ループをなくせるじゃないか。

Ref.

「関係操作では、関係全体をまとめて操作の対象とする。目的は**繰返し（ループ）をなくすことである**。いやしくも末端利用者の生産性を考えようというのであれば、この条件を欠くことはできないし、応用プログラマの生産性向上に有益であることも明らかである。」（E.F.コッド「関係データベース：生産性向上のための実用的基盤」）

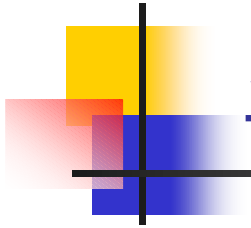
SQLに見る集合指向の考え方

手続き型と集合指向の対比 (1)

- Ex.以下のテーブルから全員が「待機」状態のチームを選択してください。(『指南書』1-10より)

Teams

member（隊員）	team_id（チームID）	status（状態）
ジョー	1	待機
ケン	1	出動中
ミック	1	待機
カレン	2	出動中
キース	2	休暇
ジャン	3	待機
ハート	3	待機
ディック	3	待機
ベス	4	待機
アレン	5	出動中
ロバート	5	休暇
ケーガン	5	待機



手続き型言語の場合

1. 「status」を表すフラグ用の変数 flg_status を用意する。
2. flg_status を 0 で初期化する
3. テーブルを一行ずつループして、「status」列を参照する。

   もし「出動中」 または「休暇」なら、flg_status を 1 に更新する。

   もし「待機」なら、何もしない。
4. チームが変わったら、2. へ戻る。

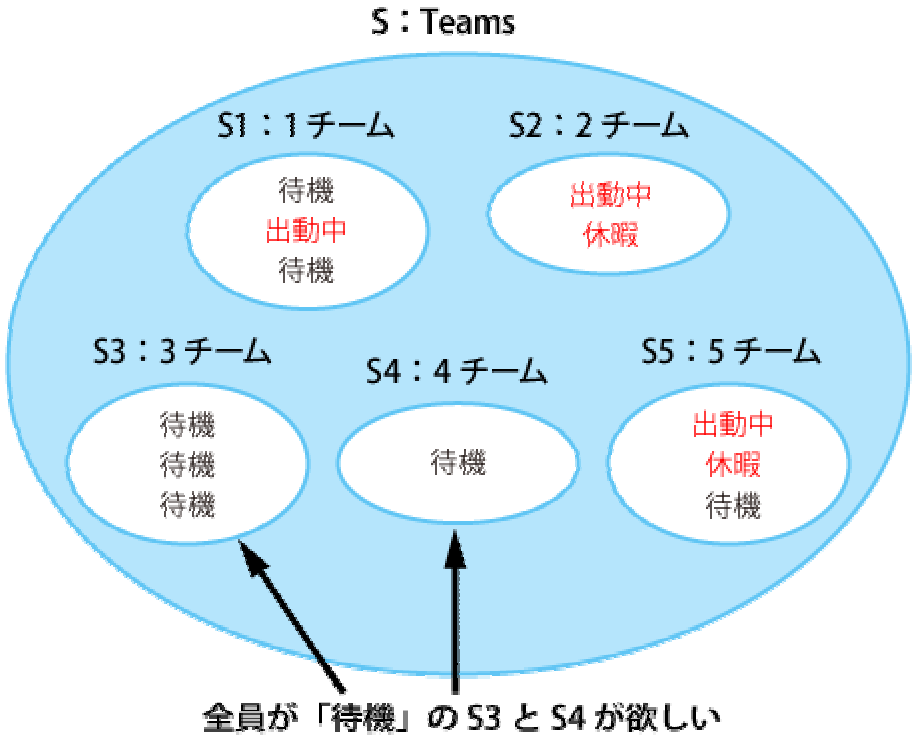
集合指向言語の場合：パーティション・カット

集合のパーティション・カット

```
SELECT team_id
  FROM Teams
GROUP BY team_id
HAVING COUNT(*) =
            SUM(CASE WHEN status = '待機'
                          THEN 1
                           ELSE 0 END);
```

結果:

```
team_id
-------
      3
      4
```

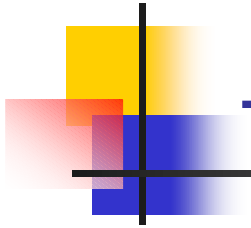


集合指向言語の場合：特性関数

特性関数：ある値が集合の中に含まれるかどうかを決定する

member（隊員）	team_id（チームID）	status（状態）	特性関数のフラグ
ジョー	1	待機	1
ケン	1	出動中	0
ミック	1	待機	1
カレン	2	出動中	0
キース	2	休暇	0
ジャン	3	待機	1
ハート	3	待機	1
ディック	3	待機	1
ベス	4	待機	1
アレン	5	出動中	0
ロバート	5	休暇	0
ケーガン	5	待機	1

1なら待機中
0ならそれ以外



テーブルはホールケーキです

・SQLはテーブルをレコードの集まりだとは考えていない。

それは、幾らでも自由に切り分けが可能なケーキ。

SQLでケーキをカットする道具：

・GROUP BY

・PARTITION BY

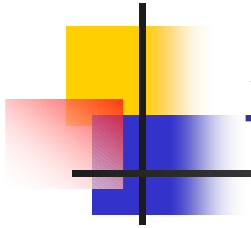
（宿題①：この二つの道具の微妙な違いは何でしょう？）

手続き型と集合指向の対比 (2)

- Ex.以下のテーブルの連番に歯抜けがあるかチェックしてください。(『指南書』1-4より)

SeqTbl

連番（seq）	名前（name）
1	ディック
2	アン
3	ライル
5	カー
6	マリー
9	ベン



手続き型言語の場合

1. 連番列の値を格納する変数 intSeq を宣言する。
2. 一行目の seq 列の値を intSeq へ代入する。
3. 残りのレコードについてもループして、当該レコードの seq 列と intSeq +1 を比較して、異なっていれば歯抜けがあると分かる。

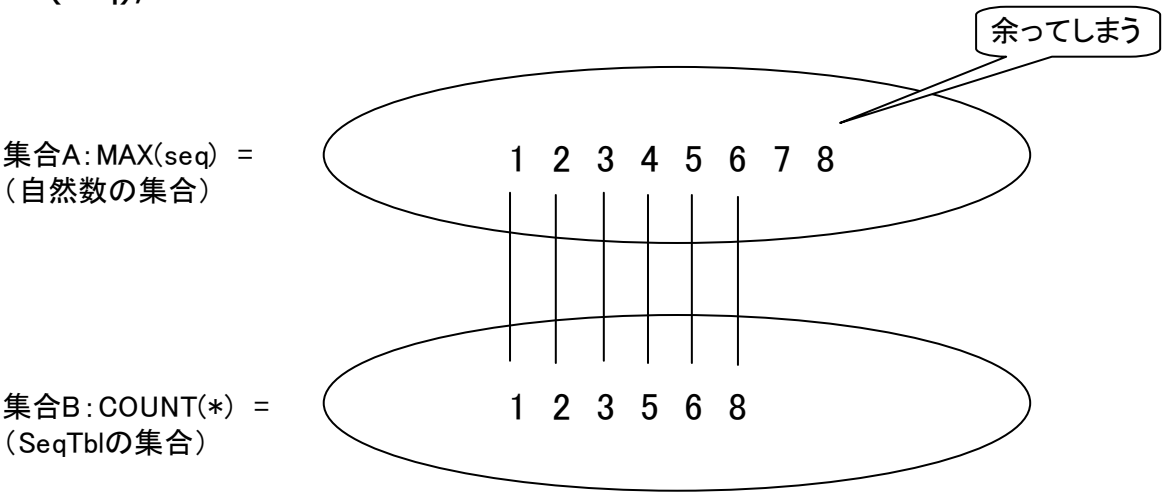
集合指向言語の場合：全単射

集合間の全単射（一対一対応）

```
-- 結果が返れば歯抜けあり
SELECT '歯抜けあり' AS gap
  FROM SeqTbl
HAVING COUNT(*) <> MAX(seq);
```

結果

```
gap
-------------
'歯抜けあり'
```



手続き型と集合指向の対比 (3)

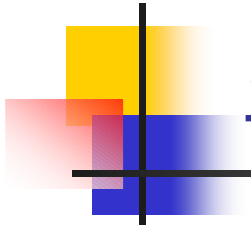
- Ex.以下の二つのテーブルのデータが完全一致しているか調べてください。(『指南書』1-7より)

tbl_A

key	col_1	col_2	col_3
A	2	3	4
B	0	7	**9**
C	5	1	6

tbl_B

key	col_1	col_2	col_3
A	2	3	4
B	0	7	**8**
C	5	1	6



手続き型言語の場合

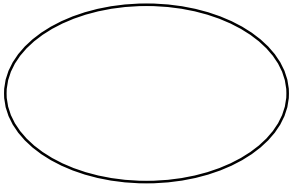
1. tbl_A と tbl_B のレコードを key をキーとしてソートする
2. それぞれのテーブルについて、一行ずつループして、各列の値を比較する。どこか一つでも異なっていれば不一致と分かる。

集合指向言語の場合：冪等性

UNIONの冪等性（idempotency）

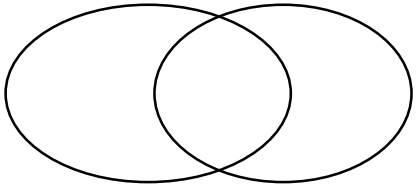
```
--このクエリの結果が tbl_Aとtbl_Bの行数と一致すれば、
--両者は等しいテーブル
SELECT COUNT(*) AS row_cnt
  FROM ( SELECT *
           FROM   tbl_A
         UNION
         SELECT *
           FROM   tbl_B ) TMP;
```

二つの集合が同一なら、和集合も同一



A UNION B = A = B

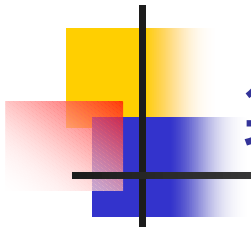
二つの集合が同一でなければ、和集合が変化する



A UNION B ≠ A ≠ B

結果

```
row_cnt
-----------
          4
```



集合指向は難しいのか (1):集合ってなに？

- SQLの集合指向に多くの人が違和感を感じるのはなぜか

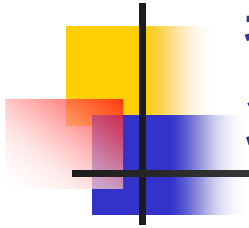
  - 集合論を学校で習う機会がない。

    基数と序数の区別、単射、全射、全単射、冪等など基本的概念でも、大学で集合論の講義に出ないと知る機会がない。

    一方、関数や順序（序数）については、中学と高校で習うので、誰にでも馴染み深い。

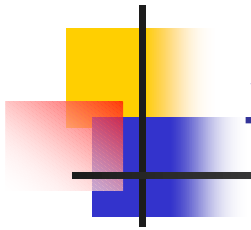
- そもそも集合論が難しいのではないか。

  - New Math 運動の失敗。「子供に集合論を教えるのは無理がある。」



集合指向は難しいのか (2): 欠陥言語としてのSQL

- SQL自身が、完全にきれいな言語設計にはなっていないことによる混乱。
  - 集合と多重集合
  - NULL と3値論理
  - 真理値演算の貧弱さ
  - 表のネストなどの高階表現が認められていない



手続き型との和解

- 順序を意識した機能の追加。
  - ウィンドウ関数
  - オートナンバリング
  - シーケンス・オブジェクト
  - PSM

データベースの未来について

パフォーマンスは金で買う

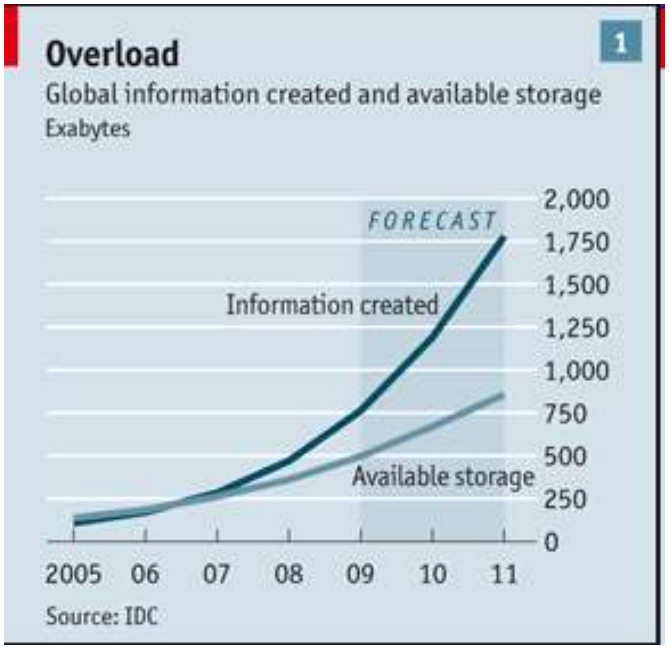
- 今後DBが扱うデータ量は増加の一途を辿る。

  -> その場合問題になるのはパフォーマンス。

- ハードウェアとアーキテクチャによる解決

  -> 並列処理

  -> ボトルネックであるディスクI/Oをいかに軽減するか。



出典：”Data, data everywhere”, *Economist,* Feb 25th 2010

今後のデータベース界の予想

- 汎用性の高いRDBを中心に、一つの軸に特化した専門DBが割拠する。