【OpenSquare第78回セミナー[みんなで語ろう、データベース その1】

ITエンジニアのための データベース再入門解説

2017年4月27日
株式会社データアーキテクト
真野　正

Contents.

1. 自己紹介
2. 執筆の背景
3. データベース再入門へのいざない
   - ✓ トラブル事例
   - ✓ 誤った使い方
   - ✓ 背景の理論
4. データベース利用の心構え

1. 自己紹介

■経歴

大手SI会社勤務（株式会社シーエーシー他）を経て2005年独立。

産業系での生産管理、販売管理系を中心とした業務システム構築でのリーダー、PMを経験。

DOAによるシステム開発、データモデリング、データベース設計の実務またはコンサルティングに従事する。

リポジトリ、モデリングツール(ERwin等)の普及に関わる。

システム基盤はメインフレーム、C/S、Web系を経験、RDB黎明期より携わり、データモデリングは、概念レベルにとどまることなく、実装を意識した設計を心掛けている。DBA業務、SQL性能改善にも携わり、データアーキテクチャ全般を対応。

大規模開発プロジェクトでのデータモデルレビュー、複数モデル間での共通・個別の調整、データ統制業務に複数携わる。

■外部活動

DOA+コンソーシアム（休眠）

DAMA日本支部

日本データマネジメント協会（JDMC）

日本ガイドシェアー委員会・研究支援

最近の業務

業種	プロジェクト	役割
流通	全社システム再構築システム化計画	論理データモデル、データ管理標準策定
電力	資材．予算実績管理システム	複数データモデリングチームの統轄、論理モデルレビュー
官公庁	決済システム再構築	データモデリング、データ管理
鉄道	次期販売システムのための論理DB構想	現行DB課題分析、新論理DB検討
鉄道	車両管理システム再構築	データ管理業務全般、マルチベンダー統轄
交通	管制支援システム構築・DB支援	DA、DBA業務
製造業	購買システムデータ移行支援	モデルベースの現新データマッピング
鉄道	設備管理システム性能改善	SQL解析、パーティション化
出版卸	基幹再構築データモデリングのためのデータ統制	ドメイン、ネーミングルール評価
金融（銀行）	統合顧客システム構築・概念DB設計	統合顧客データモデリング
金融（損保）	基幹システム再構築データモデリング支援	データモデリング標準化 論理データモデル作成
電力	スマートメータ管理システム	DB設計リファクタリング、DB性能改善
金融（生保）	保険．統合顧客システム．基本計画策定	概念・論理データモデル作成、レビュー
金融	保険.リスク管理システム.要件定義	概念データモデル作成
金融	保険.リスク管理システム.基本設計	DA、DBA業務

2. 執筆の背景

- パフォーマンスチューニングの視点からデータモデリングの重要性を知ってもらいたい
- 実行計画を意識しよう
- モデリングは大事だけど動いてなんぼの世界
- 懸念
  - データ構造・特性を十分に把握しないで設計・製造している
  - RDBが単なるファイルシステムとして使われている
  - 要求される性能を満足できていない
  - 増え続けるデータへの備えができていない
  - データベースが当たり前のように使われているが正しい使われ方が. .
- 目指すところ
  - Web上に氾濫している技術情報を判断できるように
  - 基本に立ちかえるためのトリガーとなれば
  - オンプレシステムを想定しているがクラウドにも適用できる

3. データベース再入門へのいざない

■ データベースに起因するシステムトラブル事例を示し、その原因として誤った使われ方があることをお伝えし、正しく使いこなすための基礎理論の必要性を説き、トラブルを起こさないための解決策を示す。

① データベースに起因するトラブル事例
② 誤ったデータベースの使われ方
③ 背景の理論
④ 解決策

本書の構成

◆第1章と2章は「課題編」です。
第1章では、DBMSをよく知らないために起こった事件等を例示し、問題を提起します。
続く第2章では、ITエンジニアが陥りやすい、誤ったDBの使い方を見ていきます。

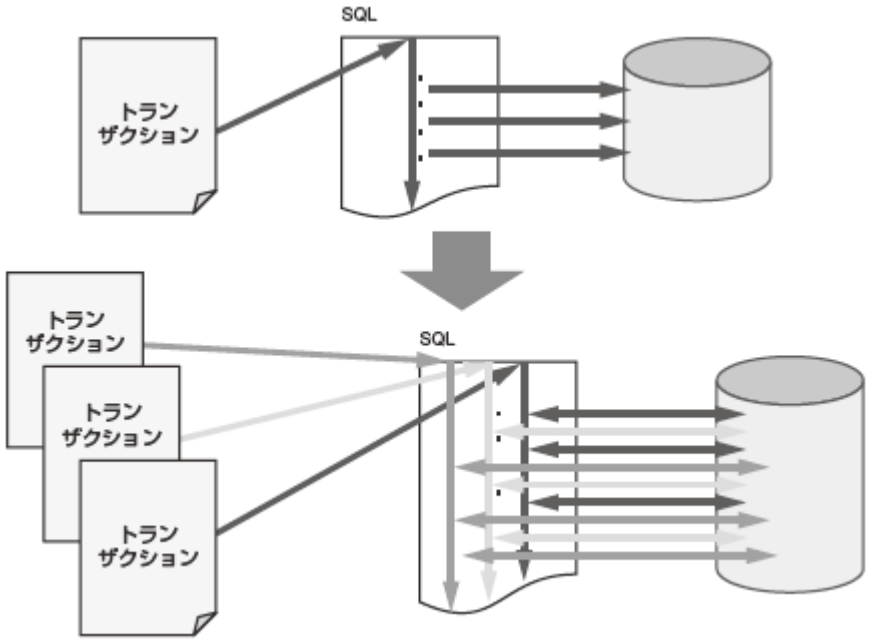
◆第3章と4章は「理論編」です。
第3章では、関係モデルを学び直します。第4章はDBMSの内部構造と操作を学び直します。

◆第5章以降は「解決編」です。
第5章は「DBMSの苦手を知る」ことから初めます。誤った使い方を分析し、
理論と照らし合わせて、問題の所在を解明します。
第6章ではDBMSを使いこなす術、第7章でSQLを使いこなすための代表的な手法を詳しく解説。
最後の第8章では「DB利用の心得」をまとめて、本書を締め括ります。

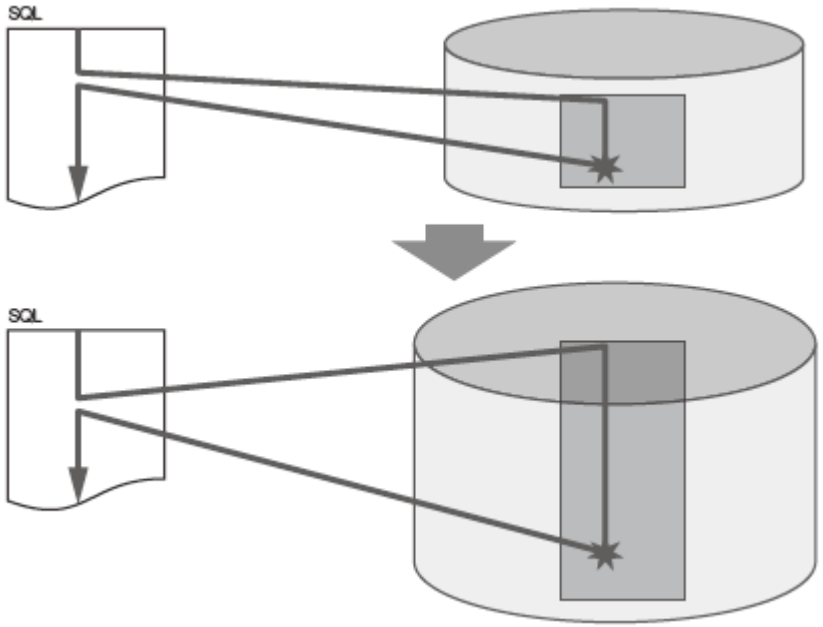
トランザクションの瞬間的な増大

ある時突然、CPU使用率が100%近くに貼りつく
実行計画は「テーブルスキャン」が選択

トラブル事例

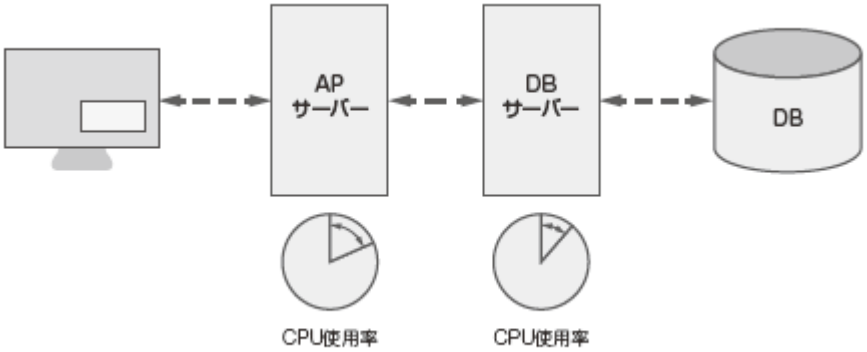


経年蓄積によるデータ量の増大



・販売実績から売れ筋を分析しようとしても、実績データを検索する際に応答が遅くなった、または返ってこない

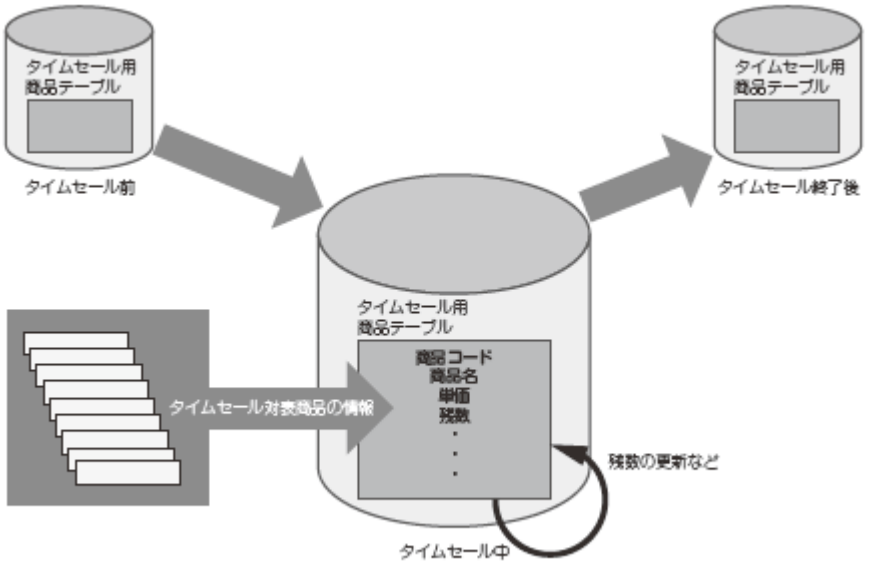
アプリケーションサーバーの負荷増大



商品検索時に応答が遅い・返ってこない

→調査の結果、APサーバーのCPU使用率が100%近くなっていることがわかった

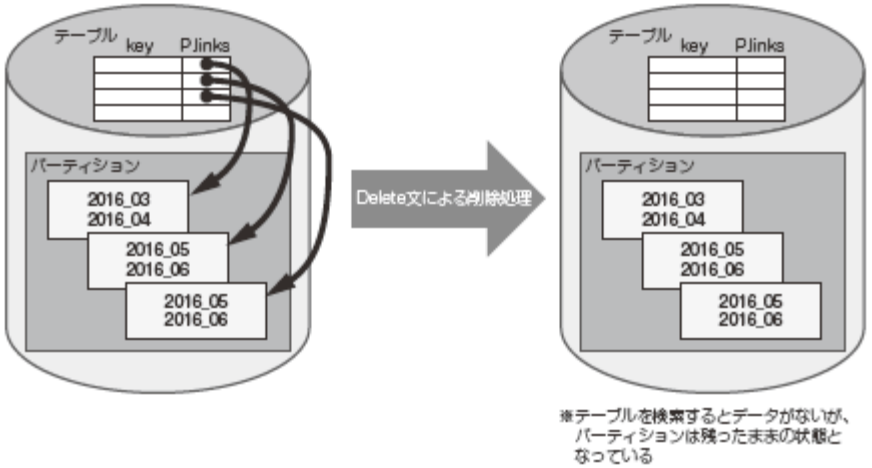
データの挿入・削除があるDB



応答が遅くてタイムセール中に購入できなかった、とのクレーム発生

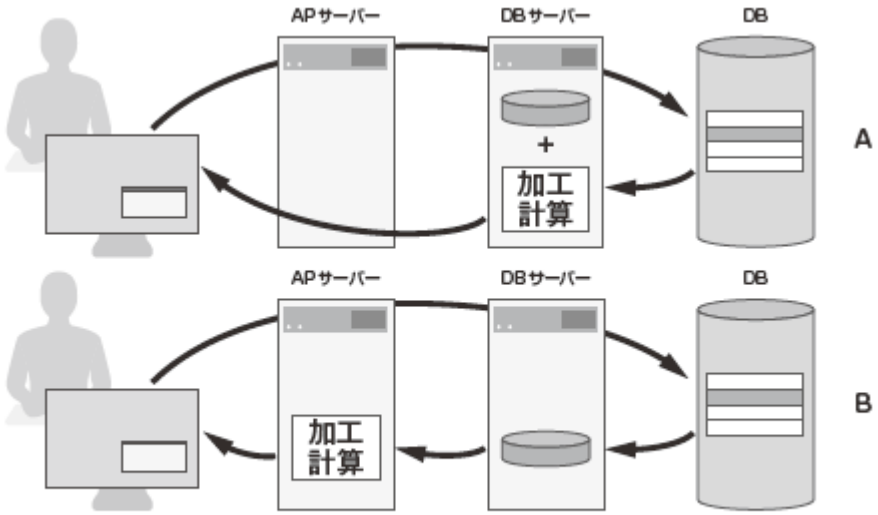
→タイムセール専用テーブルの検索に時間がかかっていたことが判明

蓄積データの削除漏れ



保存期間を過ぎたデータは削除していたが、データ格納領域が不足する事態が発生

SQL一辺倒（全ての処理をSQLで完結させようとする）



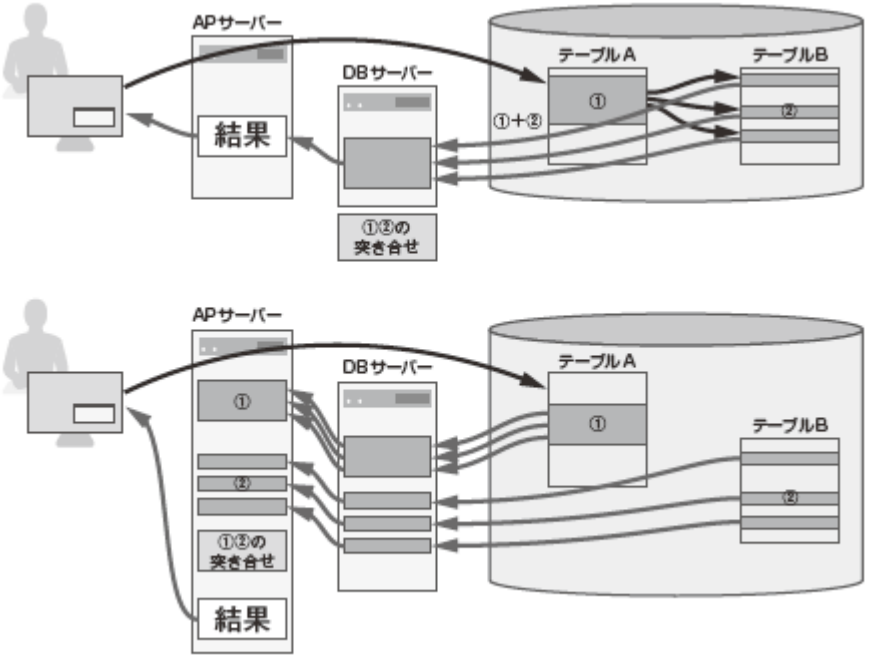
※Aでは加工・計算処理をSQLで行っているため、DBサーバーのCPU負荷が高くなる。

誤った使い方

SQLの例

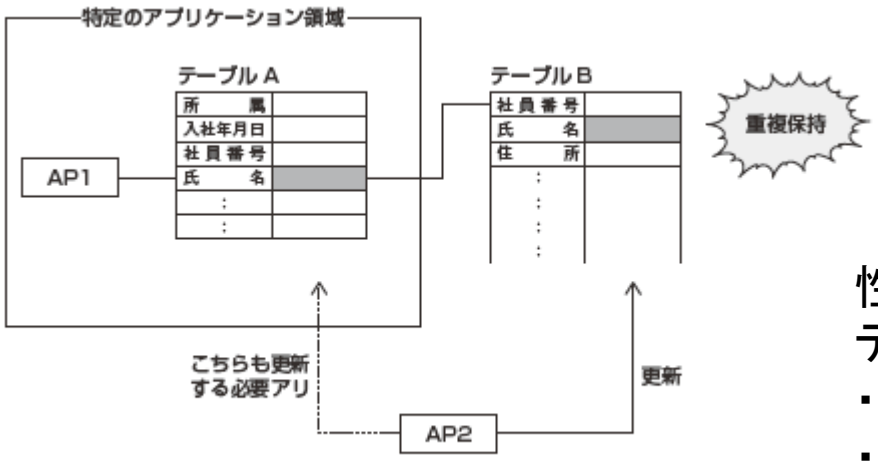
```
SELECT
    a1,
    GREATEST(a2*a3,a3+a4/a5,.....),   ・・・因数の最大値を求める関数
    NVL(a3,xxxx),                     ・・・NULL値の置き換え
    CASE(a5 =xxx THEN ・・・ELSE・・・・・)  ・・・CASE文による加工処理の分岐
    ....

FROM
    T1,
    INNNER JOIN(SELECT ・・・) T2
      ON T1.a1 = T2.B2

....
```

集合演算ではなくレコード取り出し処理としてのSQL使用



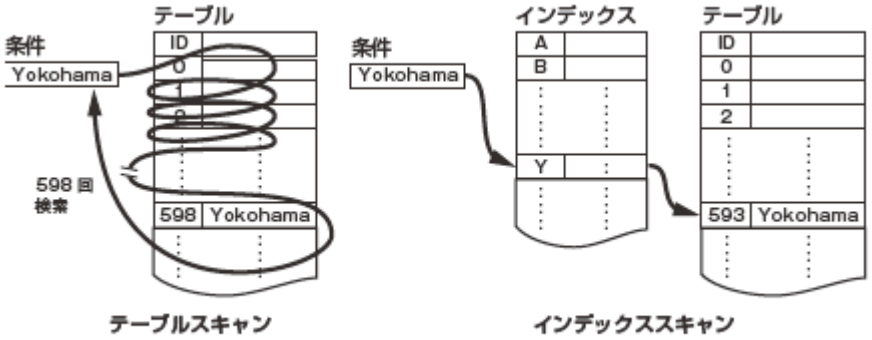
レコード取り出しのためだけにSQL
を発行し、比較・演算をAPで行うと、
・SQLの発行回数分、通信が増える
・APサーバの負荷増大

アプリケーション側の要請が強く反映されたDB構造



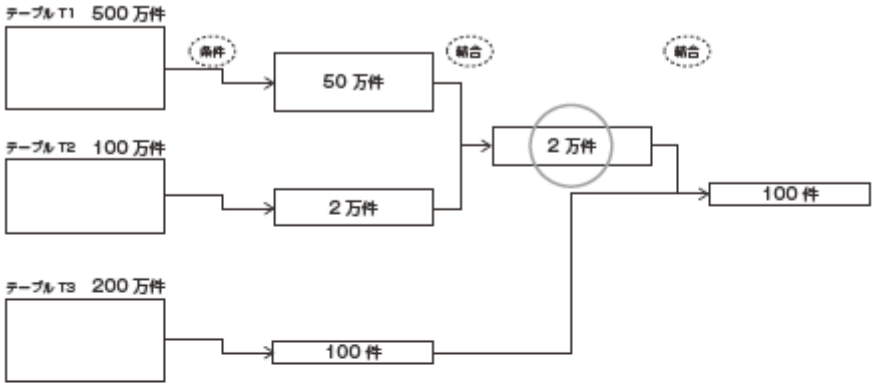
性能向上を優先して正規化を崩し、
データを重複保持した場合、
・二重に更新するための性能劣化
・データ不整合の発生

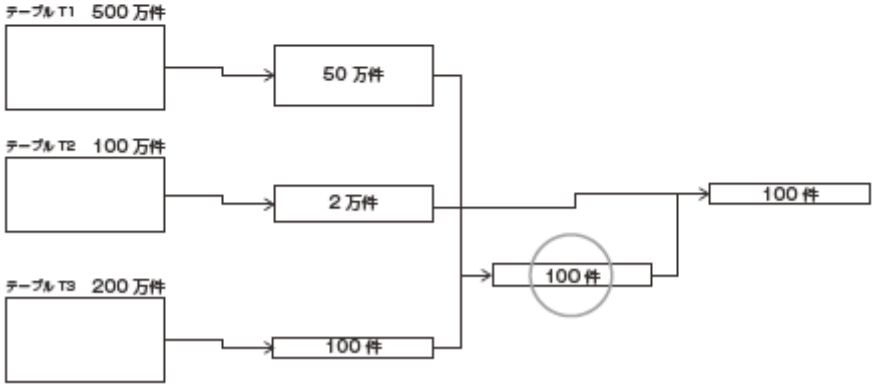
インデックスが使われないSQL



SQLの記述によっては、インデックスを指定していてもインデックス検索が実行されないことがある。
→DBサーバーの負荷増大

検索結果の件数収束が遅いSQL





早い段階で検索結果を小数に
絞り込んだほうが、検索コストが
かからない
結合種類

局所的（部分最適）なSQLチューニング

・ インデックスの強制使用
テーブル T1 のインデックス A,B,C を使用するアクセスパスを選択する

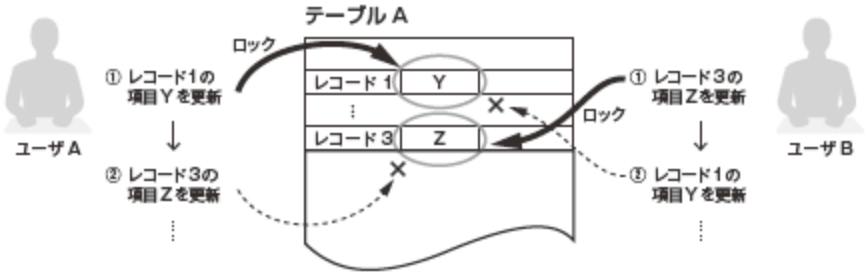
```
SELECT ····  /* INDEX(T1 A,B,C) */ FROM ····
```

・ 結合種類・順番の指定
テーブル T1,T2,T3 の順でネステッドループを行うアクセスパスとする

```
SELECT ····  /* USE_NL(T1,T2,T3) */ FROM ····
```

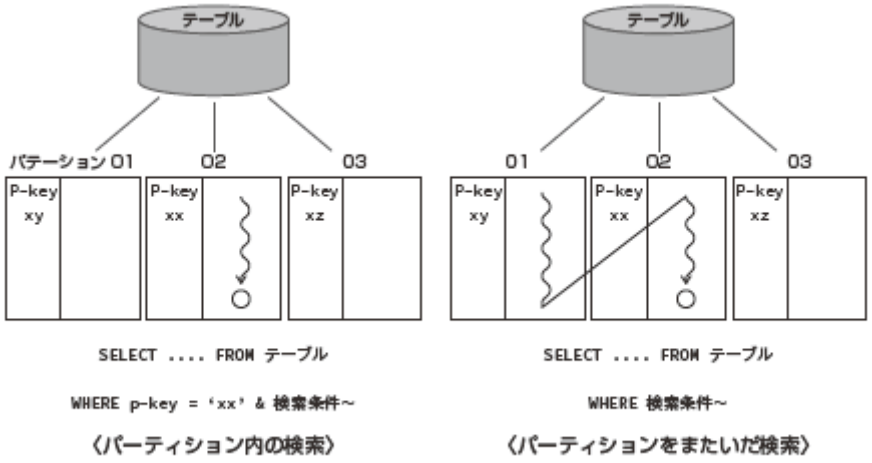
SQLを狙い通りに動かすためにヒント句を使用
→データ量が変わった場合など、ヒント句を用いたSQLが最適ではなくなることがある

後手に回るデッドロック対策



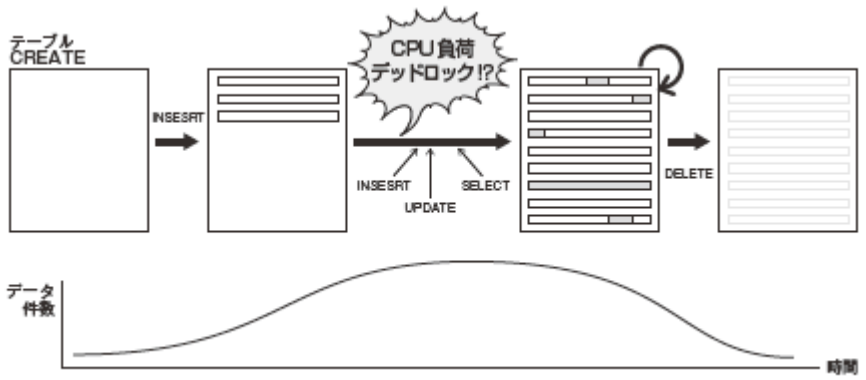
デッドロックを防ぐためにはアクセス順を守ることが基本だが、プログラマ任せになりがち
→開発の下流段階でアクセス順を決めることも…

パーティション化しただけでは性能向上は望めない



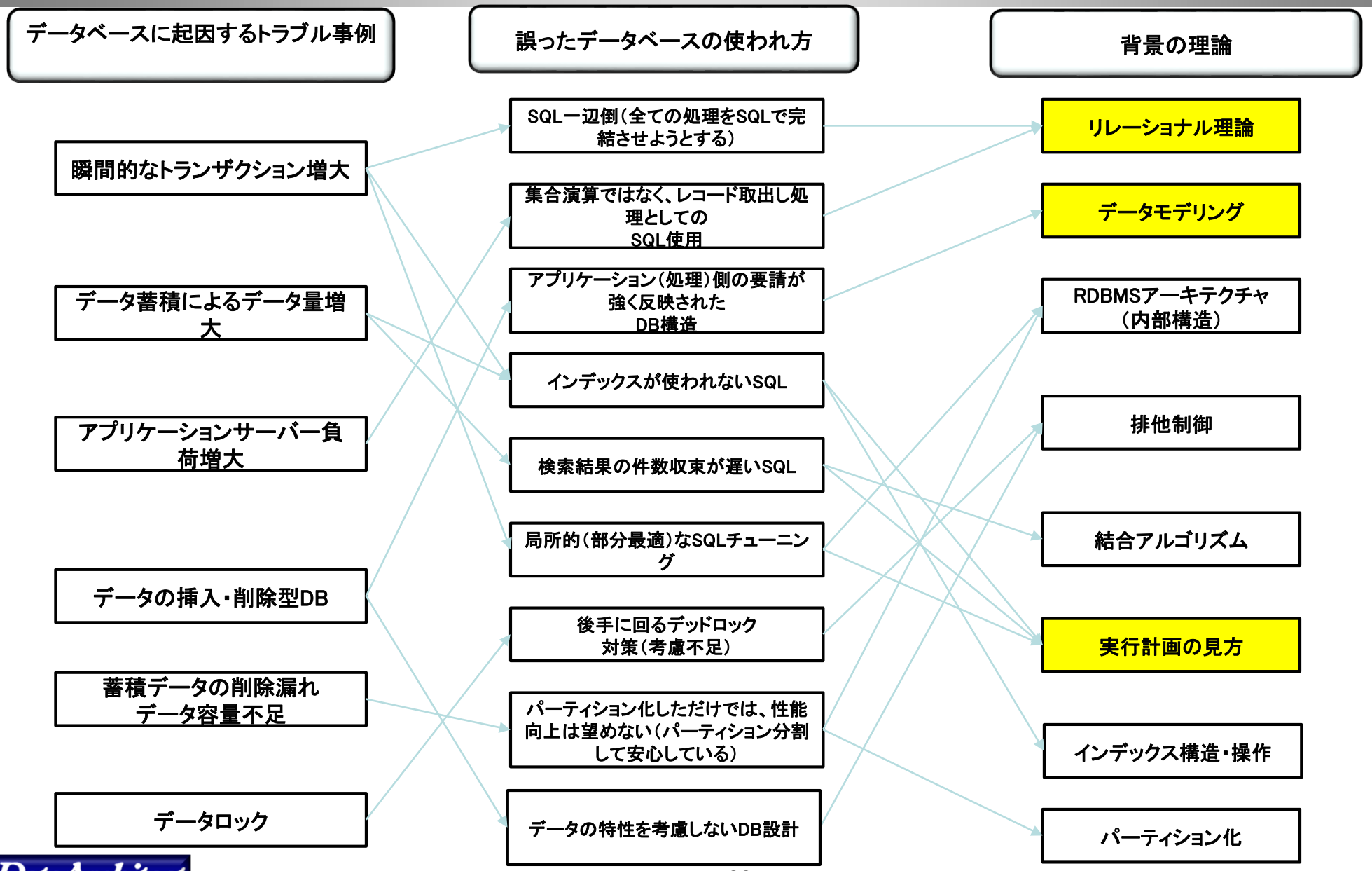
パーティション分割しただけで安心してはならない
→検索条件が適正か

データの特性を考慮しないDB設計



短いサイクルでInsert→Update・Select→Deleteが行われるDBの考慮点
・Insertが集中した場合のCPU負荷
・InsertやUpdateの競合によるデッドロックの発生
・実行計画の取得のタイミング

理論への因果関係



リレーショナル理論の適用

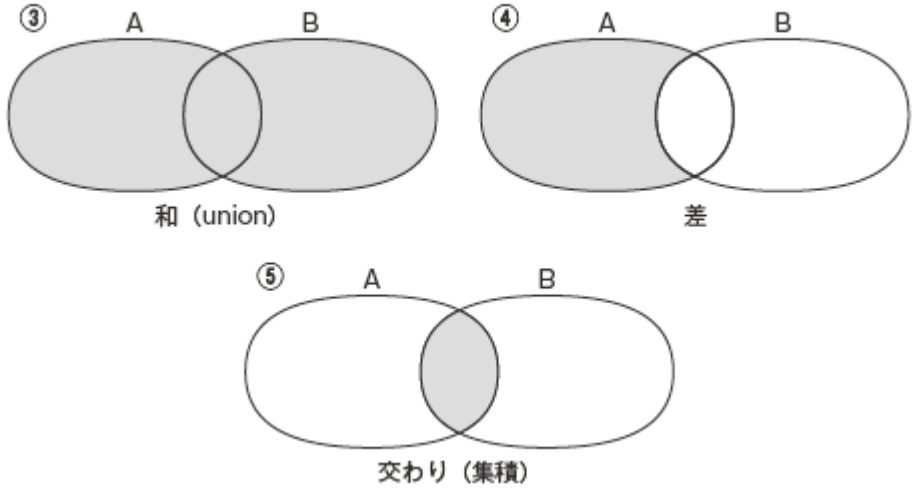
- リレーショナルモデルと操作
  - ◆実店舗、web店舗顧客の例
- データモデルによる裏付け
- 実行計画

背景の理論

リレーショナルモデル操作

- Insert
- Update
- Delete
- Select
- Restrict
- Projection
- Union
- Diference　except/minus
- Intersect
- Product
- Join
- Divide

図3.4-2　リレーショナルモデルの操作



例示　実店舗、web店舗顧客の例

■ 和・差の関係など、データモデルだけでは表現がし難いものもある

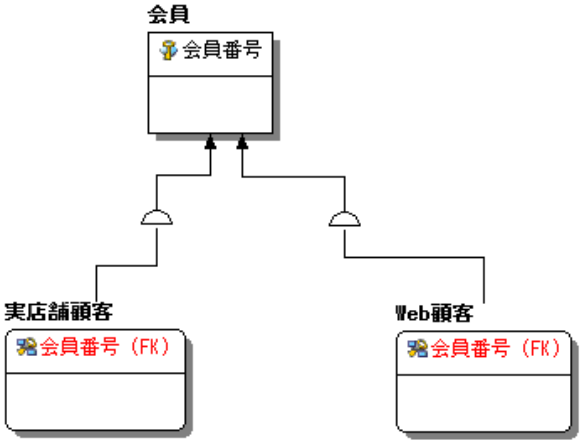
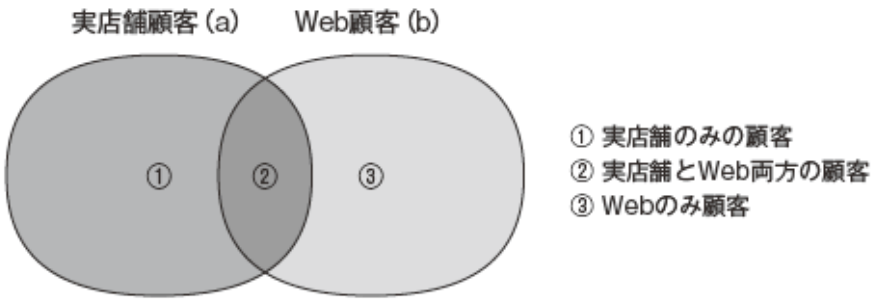


図3.10　ベン図



・実店舗のみで購入している顧客 …… ①
集合aから実店舗とWebの両方を利用している顧客（②）を除いたものとなります。

・Webのみを利用している顧客 …… ③

・実店舗とWebの両方を利用している顧客 …… ②
aとbの結合として求めることができ、①よりも先に求めておきます。

・実店舗またはWebを利用している顧客（重複を除く）…… ①+②+③
aとbの和として求めることができます（UNIONによりaとbの重複データは排除されます）。

①実店舗のみで購入している顧客

A.

```
SQL> SELECT a.会員番号,a.氏名 FROM 実店舗顧客A a
2      MINUS
3      (SELECT c.会員番号,c.氏名 FROM 実店舗顧客A c INNER JOIN WEB
       顧客A b ON  c.会員番号 = b.会員番号);
```

B.

```
SQL> SELECT a.会員番号,a.氏名 FROM 実店舗顧客A a WHERE
2      NOT EXISTS (SELECT c.会員番号,c.氏名 FROM WEB顧客A c WHERE
       a.会員番号 =c.会員番号) ;
```

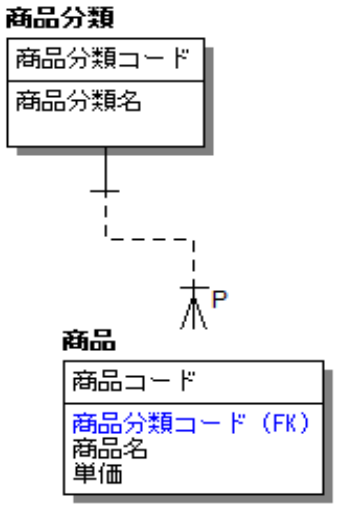
実行計画

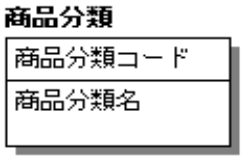
```
SQL> SELECT a.会員番号,a.氏名 FROM 実店舗顧客A a
  2   MINUS
  3    (SELECT c.会員番号,c.氏名 FROM 実店舗顧客A  c INNER JOIN WEB顧客A b ON
       c.会員番号 = b.会員番号);

実行計画
----------------------------------------------------------
Plan hash value: 2901267341

-------------------------------------------------------------------------------------
| Id  | Operation            | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |               | 10007 |   537K|    30  (57)| 00:00:01 |
|   1 |  MINUS               |               |       |       |            |          |
|   2 |   SORT UNIQUE        |               | 10007 |   234K|    15  (14)| 00:00:01 |
|   3 |    TABLE ACCESS FULL | 実店舗顧客A   | 10007 |   234K|    13   (0)| 00:00:01 |
|   4 |   SORT UNIQUE        |               | 10005 |   302K|    16  (19)| 00:00:01 |
|   5 |    NESTED LOOPS      |               | 10005 |   302K|    14   (8)| 00:00:01 |
|   6 |     TABLE ACCESS FULL| 実店舗顧客A   | 10007 |   234K|    13   (0)| 00:00:01 |
|*  7 |     INDEX UNIQUE SCAN| SYS_C0015376  |     1 |     7 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

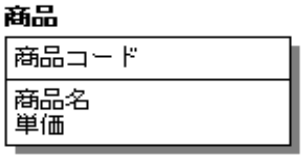
```
SQL> SELECT a.会員番号,a.氏名 FROM 実店舗顧客A a WHERE
  2    NOT EXISTS (SELECT c.会員番号,c.氏名 FROM WEB顧客A  c WHERE a.会員番号  =
       c.会員番号) ;

実行計画
----------------------------------------------------------
Plan hash value: 3617630561

-------------------------------------------------------------------------------------
| Id  | Operation            | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |               | 10007 |   302K|    14   (8)| 00:00:01 |
|   1 |  NESTED LOOPS ANTI   |               | 10007 |   302K|    14   (8)| 00:00:01 |
|   2 |   TABLE ACCESS FULL  | 実店舗顧客A   | 10007 |   234K|    13   (0)| 00:00:01 |
|*  3 |   INDEX UNIQUE SCAN  | SYS_C0015376  |     1 |     7 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------
```

データモデルによる裏付け

■ER図として関連が描ける　===　リレーショナル理論の適用　===　SQL記述可







商品コード体系
□□○○○○○○　（8桁）
□□：商品分類コード
○○○○：商品識別番号

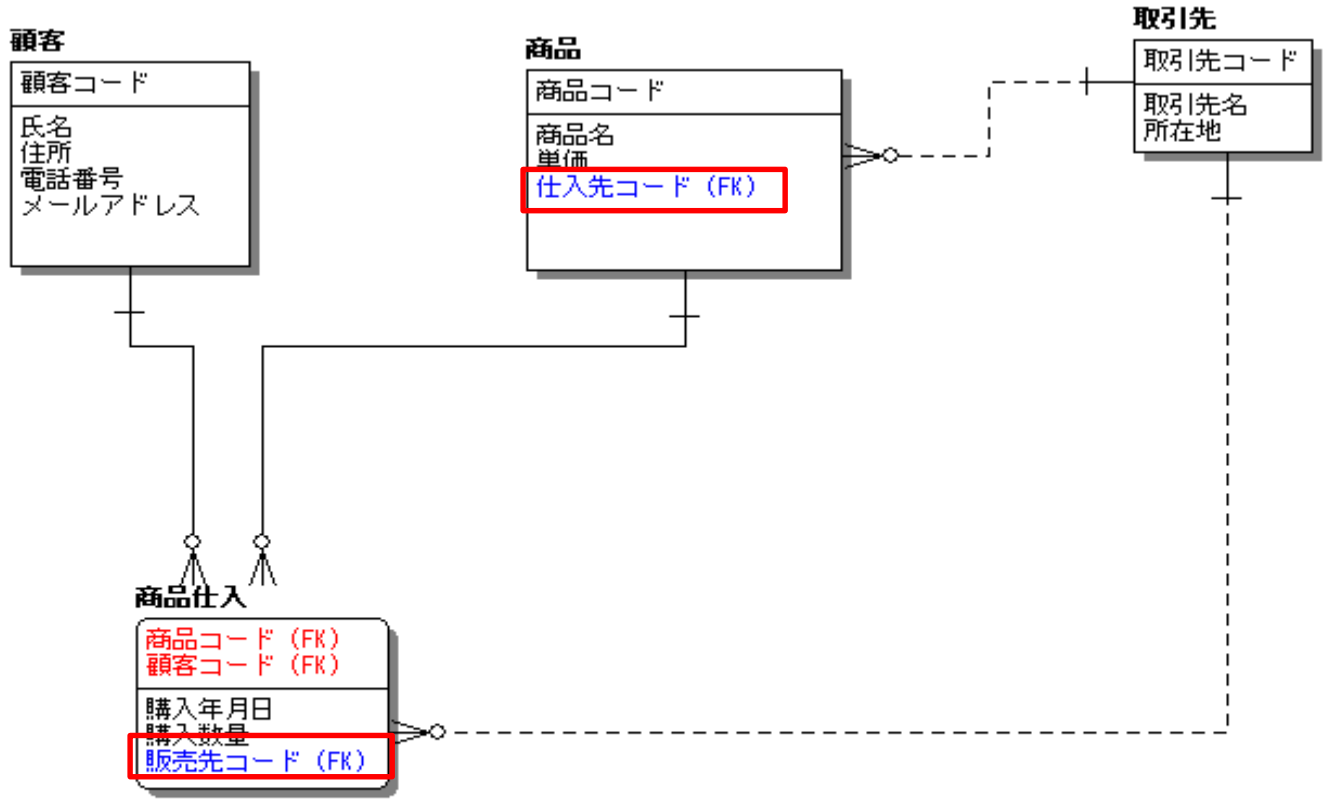
カラムの一部が結合キーとなっており、型変換が発生。

```
select t1.商品コード、t1.商品名、t2.商品分類名
  from  商品 t1
 left outer join 商品分類 t2
  on t2.商品分類コード = t1.商品分類コード
;
```

```
select t1.商品コード、t1.商品名、t2.商品分類名
  from  商品 t1
 left outer join 商品分類 t2
  on t2.商品分類コード = t1substr(商品コード,1,2)
;
```

データモデルによる裏付け

■リレーションシップにより、商品の「仕入先」と「販売先」の情報を誤り無く取得することができる。
■仕入先、販売先は、取引先のロール名と呼ばれデータモデリングの基本作法です。



4. データベース利用の心構え

■ まとめ

- ◆データベース間の関連がデータモデルに定義されていること
- ◆インデックスが有効に機能していること
- ◆適正なパーティションに分割されていること
- ◆適切なトランザクションに分離されていること
- ◆適正なSQL記述となっていること
- ◆実行計画の監視を怠らないこと

お問い合わせ先

■ データモデリングに関しては、以下の書籍で詳しく解説しています

◆ 実践的データモデリング入門

◆ 独習データベース設計

株式会社データアーキテクト
代表取締役
ITコンサルタント　　真野　正

E-mail.mano@dataarch.co.jp
URL:http://dataarch.co.jp