データベースS

第7回　リレーションスキーマの設計（1）

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
- リレーションスキーマの設計
- 正規化の必要性
  - なぜ、正規化が必要か？
  - 情報無損失分解と情報損失分解
- 多値従属性、関数従属性

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）
  - 4章（4.1節～4.7節）
- 「データベースシステム」
  北川 博之 著、昭晃堂 出版（3,200円）
  - 4章 55～82ページ

前回の復習

SQLによる問い合わせの記述

- SQLの基本的な書き方
- 条件（WHERE）の書き方
- 出力（SELECT）の書き方
- 順序付け（ORDER BY）
- グループ表（GROUP BY）
- 結合（JOIN）
- 集合演算
- 副問い合わせ（入れ子型質問）

結合の例

Q.学籍番号 00001 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 = '00001'
```

- 科目と履修を科目番号で等結合
  科目（科目番号, 科目名, 単位数）
  履修（科目番号, 学籍番号, 成績）
  （科目番号が同じデータ同士を結合）
  →（科目.科目番号, 科目名, 単位数,
  履修.科目番号, 学籍番号, 成績）

自己結合の例

Q.科目番号 001 の科目に関して、
学籍番号 001001 の学生よりも成績の良かった学生の学籍番号の一覧

```
SELECT   y.学籍番号
FROM     履修 AS x, 履修 AS y
WHERE    x.科目番号 = '001' AND x.学籍番号 = '001001'
         AND y.科目番号 = '001' AND y.成績 > x.成績
```

- x は科目番号001、学籍番号001001の履修データ
- y は x よりも成績の良い履修データ

入れ子型質問

- 入れ子型質問の形
  - 外側の質問の中に、内側の質問がある
  - 内側の質問の結果を、外側の質問で使用
  - 最終的には、外側の質問の結果が出力される

外側
```
SELECT   ・・・・・・
FROM     ・・・・・・
WHERE    ・・・・・・
         内側の問い合わせの結果を使った条件
```
内側
```
  (SELECT   ・・・・・・
   FROM     ・・・・・・
   WHERE    ・・・・・・)
```

相関を有さない入れ子型質問の例

- 相関を有さない入れ子型質問の例

Q.科目番号 001 の成績が平均点よりも高い学生の学籍番号と成績の一覧

```
SELECT   学籍番号, 成績
FROM     履修
WHERE    科目番号 = '001' AND
         成績 >
  (SELECT   AVG(成績)
   FROM     履修
   WHERE    科目番号 = '001')
```

内側の問い合わせは最初に1度だけ行われる

相関を有する入れ子型質問の例

- 相関を有する入れ子型質問の例
  - 学籍番号001001の学生の履修で、成績が平均点よりも高い科目の科目番号、成績の一覧を出力

```
SELECT   x.科目番号, x.成績
FROM     履修 AS x
WHERE    x.学籍番号 = '001001' AND
         x.成績 >
  (SELECT   AVG(y.成績)
   FROM     履修 AS y
   WHERE    x.科目番号 = y.科目番号)
```

外側の質問のデータが内側の質問で条件として使われている＝相関を有する

- 各履修ごとに、その科目の平均点を計算

SQLの書き方のヒント

- よくある間違いに注意
- 複数のテーブルを用いる場合は結合が必須
- GROUP BY を用いる場合は、SELECT節には、GROUP BY に使った属性か、集約関数しか書けない
- WHERE節には、集約関数は書かない
- HAVINGには、集約関数を使った条件しか書けない

PostgreSQLによるSQL演習（宿題）

- 前回の演習が終わっていない人は、まず、終わらせる（ただし、提出は締め切り済み）
- Moodleから演習内容が書かれたテキストファイルをダウンロード
- 演習のSQLを考えて、実行し、正しいデータが出力されることを確認する
- 演習のテキストファイルに、前回と同様、回答を貼り付けて、Moodleから提出

リレーションスキーマの設計の概要

スキーマとインスタンス（復習）

- リレーショナルモデルのスキーマ・インスタンス
  - スキーマ
    - リレーション名、リレーションの各属性名、ドメイン
    - キー制約、参照整合性制約、属性制約などの制約
  - インスタンス
    - 属性値の組によって表される各データの集合

従業員

従業員番号	部門番号	氏名	年齢
001	1	尾下 真樹	27
002	2	下戸 彩	17
003	3	本村 拓哉	30

スキーマ
※ 実際には各種制約を含む
インスタンス

リレーションスキーマの制約（復習）

従業員

従業員番号	部門番号	氏名	年齢
001	1	尾下 真樹	27
002	2	下戸 彩	17
003	3	本村 拓哉	30
004	1	宇田 ヒカル	20

スキーマ
インスタンス
属性値は各属性のドメイン制約に従う
主キー　外部キー
（超キー、候補キー）
参照整合性制約
リレーションスキーマは第一正規形を満たす

リレーションスキーマの設計

- リレーションスキーマの設計
  - リレーショナルデータベースを作成するときには、最初にスキーマや制約を定義する必要がある
  - どのようにしてスキーマを決めるべきか？

従業員

従業員番号	部門番号	氏名	年齢
001	1	尾下 真樹	27
002	2	下戸 彩	17
004	1	宇田 ヒカル	20

部門

部門番号	部門名
1	開発
2	営業

スキーマの設計

設計と正規化

- 基本的には、データベースで管理したいデータを表すように、スキーマを設計する
- 適当にスキーマを設計すると、使っていくうちに不整合が生じる危険性がある
- 不整合が生じないようなスキーマにする必要がある
  - スキーマを適切に分解することで、不整合が生じないようにすることができる（正規化）

正規化の必要性

第1正規形（復習）

- 第2回の講義で説明
- 第一正規形
  - 属性値は分解不可能な単純な値でなければならない
    - 下のようなリレーションは、第一正規形を満たす

従業員番号	部門番号	氏名	住所	年齢
001	1	尾下 真樹	片島1	27
002	2	下戸 彩	川津3	17
003	3	本村 拓哉	穂波8	30
004	1	宇田 ヒカル	横田2	20
005	1	織口 裕二	花瀬5	35

第一正規形（復習）

- 第一正規形（first normal form）制約
  - ドメイン（属性の取りうる値）は分解不可能な単純な値でなければならないという制約
  - 各属性は、単一の値でなければいけない

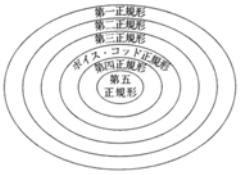
第一正規形を満たさないスキーマの例（ひとつの属性に複数の値がある）

科目番号	科目名	単位数	担当者	実習課題	
			担当者名	課題番号	課題名
001	データベース	2	北山	01	データモデリング
			山田	02	データベース設計
				03	SQL
002	システムプログラム	3	鈴木	01	Cプログラミング
			佐藤	02	システムコール

教科書 図3.2

正規形

- 正規形の種類
  - 第1正規形
  - 第2正規形
  - 第3正規形
  - ボイス・コッド正規形
  - 第4正規形
  - 第5正規形



データベースシステム 図4.7

- 更新時に不整合が発生しないようなスキーマの形式を定義するもの
- 第1正規形→第5正規形まで、徐々に条件が厳しくなっていく

正規化

- 正規形を満たさないリレーションスキーマがあるときに、正規形を満たすように分解する

営業

商品番号	顧客番号	社員番号	販売価格
i1	c1	s1	100
i2	c1	s1	200

⇩ 分解

販売

商品番号	顧客番号	販売価格
i1	c1	100
i2	c1	200

営業担当

顧客番号	社員番号
c1	s1
c2	s2

なぜ、正規化が必要か？

- 更新不整合（更新時異常）が生じる
  - 正規化されていないリレーションは、データを更新（挿入、修正、削除）しようとした時に問題（データの不整合）が生じる場合がある
- 正規化を行うことで解決
  - 更新時にデータの整合性が自動的に保たれる（整合性を壊すようなデータの追加・修正ができないような）データベースができる

更新不整合の例

- 更新不整合が起こるスキーマの例
  - 各顧客につき、担当する社員は一人だけ、というルールがあるものとする
  - このスキーマは第1正規形は満たす

営業

商品番号	顧客番号	社員番号	販売価格
i1	c1	s1	100
i1	c2	s2	120
i2	c1	s1	200
i2	c2	s2	210
i3	c1	s1	250
i3	c2	s2	250

主キー

更新不整合

- 更新不整合の種類
  - 修正不整合（修正時異常）
  - 挿入不整合（挿入時異常）
  - 削除不整合（削除時異常）

修正不整合

- 修正不整合（修正時異常）の例
  - ある顧客 c の担当社員が替わると、顧客 c の全てのデータの社員番号 s を変更する必要がある
    - 顧客 c が商品を購入する度に、その顧客の担当社員番号 s が格納されており、冗長

営業

商品番号	顧客番号	社員番号	販売価格
i1	c1	s1	100
i1	c2	s2	120
i2	c1	s1	200
i2	c2	s2	210
i3	c1	s1	250

挿入不整合

- 挿入不整合（挿入時異常）の例
  - ある顧客 c の担当社員 s が決まったとしても、顧客 c との具体的な取引商品が決まっていなければ、その情報を挿入できない
    - 主キーに含まれる商品番号の属性が空値のデータを挿入することは許されないため

商品番号	顧客番号	社員番号	販売価格
–	c	s	–

主キー

削除不整合

- 削除不整合（削除時異常）の例
  - 顧客 c1 の取引商品が一時的に全てなくなった時、顧客 c1 を社員 s1 が担当している、という情報も一緒に失われてしまう

営業

商品番号	顧客番号	社員番号	販売価格
i1	c1	s1	100
i1	c2	s2	120
i2	c1	s1	200
i2	c2	s2	210
i3	c1	s1	250

更新不整合の原因

- さきほどの具体例の問題点
  - 1つのリレーションに複数の異なる事象が含まれていることが原因
    - 具体例では、販売情報と営業担当情報が1つのスキーマで表現されていたことが原因
  - リレーションスキーマが正規形を満たしていない
    - 正規形の定義は、後で説明
- 解決方法
  - 複数のリレーションに分解する必要がある

リレーションスキーマの分解

- 正規形となるようにリレーションを分解
  - 1つのリレーションには1つの事象のみを記録するよう、適切な複数のリレーションに分解する
    - これまで問題となっていた更新異常が起こらない
  - 分解後のリレーションを自然結合することで、分解前の情報が得られる
    - 分解したことによって情報が失われることはない（情報無損失分解）
- 誤った分解を行うと情報損失分解となる

リレーションスキーマの分解

- 分解の具体例

営業

商品番号	顧客番号	社員番号	販売価格
i1	c1	s1	100
i1	c2	s2	120
i2	c1	s1	200
i2	c2	s2	210
i3	c1	s1	250
i3	c2	s2	250
i4	c1	s1	150

↓ 分解

リレーションスキーマの分解

↑ 自然結合することで元に戻る

販売

商品番号	顧客番号	販売価格
i1	c1	100
i1	c2	120
i2	c1	200
i2	c2	210
i3	c1	250
i3	c2	250
i4	c1	150

営業担当

顧客番号	社員番号
c1	s1
c2	s2

※ 分解後のリレーション名は適当に決める

情報損失分解

- 悪い分解の例
  - 分割の仕方によっては、自然結合してももとの情報が戻らないことがある

販売

商品番号	顧客番号	販売価格
i1	c1	100
i1	c2	120
i2	c1	200
i2	c2	210
i3	c1	250
i3	c2	250
i4	c1	150

商品担当

商品番号	社員番号
i1	s1
i1	s2
i2	s1
i2	s2
i3	s1
i3	s2
i4	s1

情報損失分解

- 悪い分解の例

販売

商品番号	顧客番号	販売価格
i1	c1	100
i1	c2	120
i2	c1	200
i2	c2	210
i3	c1	250
i3	c2	250
i4	c1	150

商品担当

商品番号	社員番号
i1	s1
i1	s2
i2	s1
i2	s2
i3	s1
i3	s2
i4	s1

情報損失分解

- 悪い分解の例
  - 自然結合した時、もともとなかったタプルが増える

営業

商品番号	顧客番号	社員番号	販売価格
i1	c1	s1	100
i1	c1	s2	100
i1	c2	s1	120
i1	c2	s2	120
i2	c1	s1	200
i2	c1	s2	200
i2	c2	s1	210
i2	c2	s2	210
…	…	…	…

情報無損失分解と情報損失分解

営業

商品番号	顧客番号	社員番号	販売価格

主キー

※ 各顧客につき、担当する社員は一人だけ

- 情報無損失分解（正しい分解）

販売

商品番号	顧客番号	販売価格

営業担当

顧客番号	社員番号

- 情報損失分解（悪い分解）

販売

商品番号	顧客番号	販売価格

商品担当

商品番号	社員番号

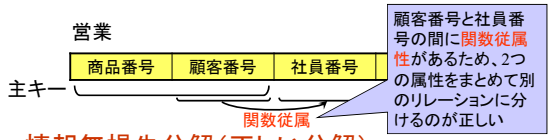
リレーションスキーマ分解のルール

- リレーションスキーマの分解
  - 更新による不整合が生じるのを防ぐ
  - 基本的には1つのリレーションに、複数の事象の情報を混在させないようにすることが必要
- 適当に分解すると、情報損失分解になってしまうため、きちんとしたルールに基づいて分解することが必要

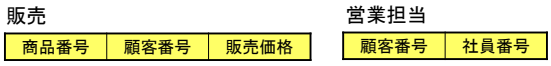
リレーションスキーマ分解のルール

- 正規形を定義するための従属性の定義
  - 多値従属性
  - 関数従属性
    - これらはリレーションスキーマ中の属性同士の依存関係を定義するためのもの
- 正規形の定義
  - 上で定義した従属性を使って正規形を定義
  - 正規形を満たさないリレーションスキーマがあれば、満たすように分割する

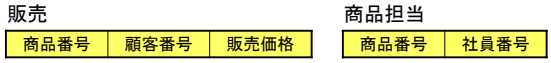
情報無損失分解と情報損失分解



- 情報無損失分解（正しい分解）

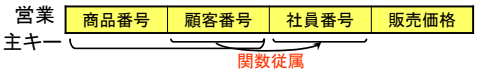


- 情報損失分解（悪い分解）

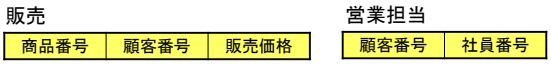


ここまでのまとめ

- 悪いリレーションスキーマの例
  - 各顧客につき、担当する社員は一人だけ、というルールがあるものとする



- このままでは更新不整合が生じてしまう
- 関数従属性・多値従属性に注目して分解する



関数従属性 と 多値従属性

関数従属性と多値従属性

- 関数従属性 X→Y
  - 属性（の組）X が決まれば、属性（の組）Y が一意に決まる
- 多値従属性 X→→Y
  - ある属性（の組） Xについて、いくつかの属性（の組）Yが存在すれば、必ず全ての XY（RS－XY）の組み合わせが存在する
    - RSはリレーションの全ての属性

関数従属性

- 関数従属性の条件

リレーションスキーマ RS のリレーション R が以下の条件を満たす時、Y は X に関数従属 $(X \rightarrow Y)$

$$(\forall t \in R)(\forall u \in R)(t[X] = u[X] \rightarrow t[Y] = u[Y])$$

- リレーションRにおいて、2つのタプルの属性Xの値が同じであれば、属性Yの値も同じ値になる
  - 属性Xが決まれば、属性Yも決まる
- 情報無損失分解の十分条件になる（必要条件ではないことに注意）（教科書「システム」p.75 定理 4.2）

関数従属性の例

- 関数従属性の例
  - 各顧客につき、担当する社員は一人だけとする

営業

商品番号	顧客番号	社員番号	販売価格
i1	c1	s1	100
i1	c2	s2	120
i2	c1	s1	200
i2	c2	s2	210
i3	c1	s1	250
i3	c2	s2	250
i4	c1	s1	150

主キー

関数従属性の例

- 関数従属性の例
  - 各顧客につき、担当する社員は一人だけとする

営業

商品番号	顧客番号	社員番号	販売価格
i1	c1	s1	
i1	c2	s2	
i2	c1	s1	
i2	c2	s2	210
i3	c1	s1	
i3	c2	s2	
i4	c1	s1	

例えば、顧客番号が c1 であれば、担当する社員番号は常に s1 になる

このスキーマには、
顧客番号 → 社員番号
という関数従属性がある

主キー

関数従属

（関数従属の属性で分解できる）

情報無損失分解と情報損失分解

営業

商品番号	顧客番号	社員番号

主キー

関数従属

顧客番号と社員番号の間に関数従属性があるため、2つの属性をまとめて別のリレーションに分けるのが正しい

- 情報無損失分解（正しい分解）

販売

商品番号	顧客番号	販売価格

営業担当

顧客番号	社員番号

- 情報損失分解（悪い分解）

販売

商品番号	顧客番号	販売価格

商品担当

商品番号	社員番号

関数従属性とキー属性の関係

- ある属性（属性の集合）→ リレーションの他の全ての属性　の関数従属性があれば、その属性はキー属性（超キー属性）になる

営業

商品番号	顧客番号	社員番号	販売価格

主キー

関数従属

商品番号, 顧客番号 → 社員番号, 販売価格
商品番号, 顧客番号 → 社員番号
商品番号, 顧客番号 → 販売価格

- 上記の例のような自明な関数従属性が存在

キー制約（復習）

- 超キー
  - 複数のインスタンスが同一の属性値をもつことがないような属性、あるいは、属性の集合
  - キーの条件を満たす、全ての属性、属性の集合
- 候補キー（単にキーと呼ぶこともある）
  - 最小の超キー（候補キーの部分集合が超キーとならない）
- 主キー
  - 候補キーのうち、属性値が空値にならず管理上適当なもの
  - 例：学生（学籍番号、氏名、住所、学科）

超キー
主キー
候補キー

※どの属性が超キーになるかはデータに依存
※超キーは他にも存在

多値従属性

- 情報無損失分解の必要十分条件
- 多値従属性の定義

リレーションスキーマ RS のリレーション R が、以下の条件を満たす時、X は Y に多値従属 $(X \rightarrow\rightarrow Y)$

$$(\forall t \in R)(\forall u \in R)(t[X]=u[X] \rightarrow (\exists v \in R)(\exists w \in R)$$
$$((t[X]=u[X]=v[X]=w[X]) \wedge$$
$$t[Y]=v[Y] \wedge t[RS-XY]=w[RS-XY] \wedge$$
$$u[Y]=w[Y] \wedge u[RS-XY]=v[RS-XY]))$$

多値従属性

- 情報無損失分解の必要十分条件
- 多値従属性の定義

リレーションスキーマ RS のリレーション R が、以下の条件を満たす時、X は Y に多値従属 $(X \rightarrow\rightarrow Y)$

t(x, y1, z1), u(x, y2, z2) というタプルが存在すれば 、必ず v(x, y1, z2), w(x, y2, z1) のようなタプルも存在する

多値従属性の例

- プロジェクトごとに社員とミーティング日が決まっているものとする

プロジェクト

プロジェクト番号	社員番号	ミーティング日
p1	e1	月曜日
p1	e2	月曜日
p1	e1	木曜日
p1	e2	木曜日
p2	e1	月曜日
p2	e3	月曜日
p2	e1	金曜日
p2	e3	金曜日

主キー

多値従属性の例

- プロジェクトごとに社員とミーティング日が決まっているものとする

プロジェクト

プロジェクト番号	社員番号	ミーティング日	
p1	e1	月曜日	t
p1	e2	月曜日	w
p1	e1	木曜日	v
p1	e2	木曜日	u

プロジェクト p1 には、社員 e1, e2 が所属
プロジェクト p1 のミーティング日は、月曜日と木曜日
プロジェクトに所属する社員とミーティング日の情報をリレーションにすると例のように4つのデータとなる

多値従属性の例

- プロジェクトごとに社員とミーティング日が決まっているものとする

プロジェクト

プロジェクト番号	社員番号	ミーティング日	
p1	e1	月曜日	t
p1	e2	月曜日	w
p1	e1	木曜日	v
p1	e2	木曜日	u

t(x, y_1, z_1), u(x, y_2, z_2) というタプルが存在すれば 、必ず v(x, y_1, z_2), w(x, y_2, z_1) のようなタプルも存在する

X, Y は複数の属性の組でも良く、Z＝RS－XY であることに注意

多値従属性にもとづく分解の例の例

- 多値従属性の書き方
  - この例の場合、以下の多値従属性があると言える
    - どの書き方でも同じ意味になる

プロジェクト番号 →→ 社員番号

プロジェクト番号 →→ ミーティング日

プロジェクト番号 →→ 社員番号 | ミーティング日

プロジェクト

プロジェクト番号	社員番号	ミーティング日

多値従属性にもとづく分解の例

プロジェクト

プロジェクト番号	社員番号	ミーティング日

主キー

プロジェクト番号 →→ 社員番号 | ミーティング日

分解

参加社員

プロジェクト番号	社員番号
p1	e1
p1	e2
p2	e1
p2	e3

ミーティング日

プロジェクト番号	ミーティング日
p1	月曜日
p1	木曜日
p2	月曜日
p2	金曜日

多値従属性と関数従属性の関係

- 関数従属性は多値従属性の特殊な形
- 関数従属性を満たす場合は、多値従属性も満たす
  - $X \to Y$ ならば $X \to\to Y$
  - 関数従属性は、多値従属性の Y が一通りしかない（一意に決定する）ケース
    - 具体例では、関数従属性を満たしているため、同時に多値従属性も満たす
  - 逆は成り立たないことに注意

多値従属性と関数従属性の例

- 関数従属性が成立すれば、多値従属性も成立

顧客番号 → 社員番号 ➡ 顧客番号 →→ 社員番号

顧客番号ごとに社員番号が一意に決まる特殊な例

営業

商品番号	顧客番号	社員番号	販売価格
i1	c1	s1	100
i1	c2	s2	120
i2	c1	s1	200
i2	c2	s2	210
i3	c1	s1	250
i3	c2	s2	250

関数従属性と多値従属性のまとめ

- 関数従属性 X → Y
  - 属性（の組）X が決まれば、属性（の組）Y が一意に決まる
- 多値従属性 X →→ Y
  - ある属性（の組）Xについて、いくつかの属性（の組）Yが存在すれば、必ず全ての XY（RS－XY）の組み合わせが存在する
    - RSはリレーションの全ての属性
  - 関数従属性は多値従属性の特殊なものになる
    - Yが常に1種類のみ存在するもの

関数従属性の性質

- その属性にどのような値が入るかという条件によって決まる
  - スキーマだけからは分からず、データベースに格納されるデータについての予備知識が必要
    - 今回の例では、1件の顧客につき担当者1人というきまりなど
  - スキーマ設計を行う時には、どのような関数従属性が存在するかを調べて、書き出す必要がある

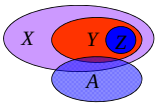
関数従属性の理論

- 関数従属性の導出
  - 与えられた関数従属性から全ての関数従属性を求める
- 論理的に含意する（logically imply）
  - $\{A \to B, B \to C\}$ ならば $A \to C$ となる

    $\{A \to B, B \to C\} \models A \to C$
- 閉包（closure）
  - ある関数従属性の集合 F から導出される全ての関数従属性の集合 F^+

関数従属性の理論

- アームストロングの公理系
  - 反射律
    - Y が X の部分集合であれば、$X \rightarrow Y$
    - 例：学生番号、氏名 → 氏名
  - 増加律
    - $X \rightarrow Y$ であれば、$X \cup A \rightarrow Y \cup A$
    - 例：学生番号 → 学科 なら、学生番号、氏名 → 学科、氏名
  - 推移律
    - $X \rightarrow Y$ かつ $Y \rightarrow Z$ であれば、$X \rightarrow Z$
    - 例：学生番号 → 学科番号 かつ 学科番号 → 学科名 なら、学生番号 → 学科名
  - 関数従属性 $X \rightarrow Y$ に、アームストログの公理を適用した結果も、必ず関数従属性になる（＝健全で完全（sound and complete））



まとめ

- 前回の復習
- リレーションスキーマの設計
- 正規化の必要性
  - なぜ、正規化が必要か？
  - 情報無損失分解と情報損失分解
- 多値従属性、関数従属性

次回予告

- 次回：スキーマの正規形と正規化
  - 第2正規形、第3正規形、ボイス・コッド正規形、第4正規形、第5正規形
- 次々回：スキーマの設計
  - 実体関連モデルからのスキーマの論理設計
  - 正規形を満たすようにスキーマを分解