

本稿の内容と参考文献

本稿は、データベース書学者が学習した内容をまとめたものである。

誤植だけでなく、内容に不備があるかもしれない事を念頭に入れて読むことを強く推奨する。

参考にした資料を以下に示す。

- ・ 増永良文. 「データベース入門」. 2006年10月10日.
- ・ Spivak, David I. “Functorial Data Migration.” arXiv, February 2, 2013. <https://doi.org/10.48550/arXiv.1009.1166>.
- ・ Spivak, David I. “Category Theory for Scientists (Old Version).” arXiv, September 18, 2013. <https://doi.org/10.48550/arXiv.1302.6946>.

疑問点や考察に関しては、このようにMarkdownの引用の形式を用いる。

本稿の内容自体が上記の資料をまとめたものであることから引用する場面が無いことと、初学者の判断が入っていることを強調するためである。

本稿では（リレーシジョンの表による表現の項目以外）データベースの概念の具体例を挙げる予定は現在はないが、具体的な例は増永良文(2006)を参考にするとよい。

余裕があればSpivak(2013)等を参考に例を挙げていきたいが、その際にはMarkdown形式ではなく $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X}$ で文書作成をしよう。（その場合は構成を一新する可能性もある。SQLや圏論（日本語ではネット上に情報が少ない？）の項目を増やしたい。）

本稿はVSCodeの拡張機能[Markdown Review Enhances](#)を用いて作成している。

始めは[Markdown All in One](#)を用いていたが、より多くのLaTeXの表現を用いられる事と、私の能力ではpdfに変換した際にLaTeXの形式による集合 $\{, \}$ が適切に出力できなかったからだ。

データベースとは何か

データモデル・データベース

データベース：現実世界に起こった現象・事象について、（データモデルによって）機械可読な形でコンピュータに格納し、様々なユーザの問い合わせやデータ処理に提供できるようにしたもの。

データモデリング

一般に、データモデルは以下の3つの要素からなる。

- ・ 構造記述：データベースの構成要素の記述。2章で扱う。
- ・ 意味記述：リレーシジョンだけでは補えない実世界の制約や、一貫性のための制約を扱う。3章でみる。
- ・ 操作記述：検索・更新のためのデータ操作言語

データモデル：現実世界の出来事を記述するための何らかの記号系

データモデルを世に出た順番に分けると、概ね以下のものがある。

1. ネットワークデータモデル, ハイアラキカルデータモデル
 - ネットワークモデル :
 - レコード型と親子集合の2つによって、データやデータ間の関連を表す
 - データはレコード型に格納され、ポインタや鎖を用いて関連付ける
 - ハイアラキカルデータモデル : ネットワークモデルの特殊な場合
 - レコード : 親レコードから子レコードに向けてレコードが張られる
2. リレーショナルデータモデル : 全てのデータを、票として表現
3. オブジェクト指向データモデル : 複合オブジェクトの表現に向いている。

リレーショナルデータモデル・オブジェクト指向データモデルについては、今後見ていく。

データベースとファイルの違い

データベースとファイルの管理方法の違い

- データベース : データが複数個所に散在せず、無矛盾に保ちやすい
- ファイル : データがプログラムごとに管理され、データ間に矛盾が生じたりする。(一方、階層型データベースの一種と見なすこともできる)

データベース管理システム

データベース管理システム (DataBase Management System) :

以下の3つの機能を持ったもの

1. **メタデータ管理機能** : 「データのデータ」を管理
 - メタデータとは : 2つの意味がある
 - 「どの様なデータが、どの様に格納されているか」についての情報。
 - DBMSそのものに関するデータ。データの種類やサイズ, 付与されているindex, アクセス権等についての情報。
2. **質問処理機能** : データベースに対する質問(query)を処理する機能。
 - 特にRDBMSでは、SQL言語を用いて非手続き的・宣言的に書き下される。
 - 手続き的なファイルアクセスレベルのコードに変換する仕事がシステムの優劣を支配する。
3. **トランザクション管理機能** : データベースの一貫性を保持するための機能
 - **トランザクション** : DBMSに対するアプリケーションレベルの仕事の単位。
 - トランザクションの概念を堅持することで、2つの機能を全うする。
 - 障害児回復
 - 同時実行制御
 - トランザクションの処理に異常をきたす原因 :
 - トランザクション自体の不備 : プログラムエラー
 - システム障害 : 電源断
 - メディア障害 : ディスククラッシュなど
 - 例えば、障害によってトランザクションが中途半端に更新したままの場合は、データベースのデータを旧値に戻す。

コラム

データと情報 :

- データ：記号の集まり
- 情報：データの受け手がデータを解釈し、得られた意味が受け手の持つ知識を増加した場合に情報が得られる。

リレーショナルデータモデル：構造記述

リレーション

概念の定義

ドメインとその直積

ドメインと呼ばれる集合があるとする。

ドメインの例：

- 集合
 - 有限集合
 - 無限集合
- データ型（で規定される集合）
 - 整数：INTEGER
 - 最大長10の可変長漢字列型：NCHARVARYING(10)

添え字集合を I とする。 n 個のドメインを表現する際、添え字集合に $I = \{1, 2, \dots, n\}$ を用いて D_1, D_2, \dots, D_n と記述する習慣がある。

ドメインの定義の例：

- $D_1 = \{x | x \text{は人名}\}$
- $D_2 = \text{NCHARVARYING}(10)$ （便宜的な表現）

n 個のドメイン D_1, D_2, \dots, D_n の直積 D を以下で定義する。

$$D = D_1 \times D_2 \times \dots \times D_n$$

リレーションの定義

D_1, D_2, \dots, D_n 上のリレーション R を、

$$R \subset D_1 \times D_2 \times \dots \times D_n$$

と定義する。

リレーション R の定義より、**全ての要素はタプルで表現できる**。このことからリレーションは集合ではあるが、**リレーションを（重複の無い）テーブルとして表現できる**。

リレーションはドメインの直積から定義されていることから、カラムの順番（またはドメイン D_i の添え字 i の順番）には意味がある。一方リレーションの行の並び順はデータの集合を並べただけであるから、情報を担っていない。

テーブルで表現するには、「リレーションが有限集合」という条件が必要そう
逆に言えば、リレーションはテーブルの抽象化になっている？

リレーション R に関連する定義：

- 濃度・基数 $|R|$: R のタプルの総数
- 次数 n : 定義で直積をとっているドメインの個数
- n 項リレーション : 次数が n のリレーション

例：リレーション

リレーションを以下のように定義してみる。

- $D_1 = \{x | x \text{ は人名}\}$
- $D_2 = \text{INTEGER}$
- $R \subset D_1 \times D_2$
- $R = \{(\text{太郎}, 25), (\text{一郎}, 25), (\text{花子}, 25), (\text{桃子}, 25)\}$

これを表で表した場合の一例を挙げる。

1	2
太郎	25
一郎	25
花子	25
桃子	25

リレーション名と属性名によるリレーションの定義

リレーションを設計した時点でデータをどのように解釈すべきかは決まっている。

しかし上記の表を挙げただけでは「何のデータであるか」についての解釈が一意的に定まらず、情報としては不足しておりこのままでは設計者の意図を伝えられない。

(実際には「何を観測したデータであるか」という情報を持ったうえで記号系にデータに変換している。しかし上記の方法でリレーションを定義すると、「何のデータであるか」という情報が抜けてしまう。)

そこでリレーション R に対して**リレーション名**を与え、さらに以下を定義する。

- 属性名 A_i ($i = 1, \dots, n$)
- ドメイン関数 $\text{dom} : A_i \mapsto D_i$ ($i = 1, \dots, n$)

これは各ドメインを指定する添え字集合 I に、自然数の集合 $\{1, \dots, n\}$ ではなく属性名 $\{A_i\}_{i \in \{1, \dots, n\}}$ を用いることに等しい。

つまり以下のような表現をすること等価である。

$$D_i = D_{A_i} \quad (i = 1, \dots, n)$$

これらを用いて、リレーション R を次のように再定義できる。

$$R \subset \text{dom}(A_1) \times \text{dom}(A_2) \times \dots \times \text{dom}(A_n)$$

$R(A_1, A_2, \cdots, A_n)$

こうすることでカラムの並び順の意味がなくなっている事に注意する。

例：属性名・リレーション名

上記の例にリレーションリレーション名「友人」を与えれば、属性名とリレーションを以下のように定義する。

- $\text{dom}(\text{名前}) = D_1$
- $\text{dom}(\text{年齢}) = D_2$
- $\text{友人} \subset \text{dom}(\text{名前}) \times \text{dom}(\text{年齢})$

これを表で表すと、以下のように記述できる。

名前	年齢
太郎	25
一郎	25
花子	25
桃子	25

リレーションスキーマ

リレーションスキーマ：時間に不変なリレーションの性質。

インスタンス：多くの場合、時間の移り変わりとともに変化するリレーションそのものを指す。

リレーションを定義するためのドメイン直積をリレーションスキーマと呼んでいる？
時間的に変化することを強調するときに、リレーションをスキーマと呼んでいる？

リレーションの更新について考察するときにこれらの概念を用いる。

第一正規系

ユーザフレンドリや質問の処理系の単純化のため、リレーションのドメインに対して制約を設ける。

第 1 正規系（ther first normal form, 1NF）：

$$\neg \exists i \in I \ \exists \{D'_\lambda\}_{\lambda \in \Lambda} \quad |\{D'_\lambda\}_{\lambda \in \Lambda}| \neq 1 \Rightarrow D_i = \prod_{\lambda \in \Lambda} D'_\lambda$$

要確認：数式

つまりリレーションの中にリレーションが含まれるような、入れ子型リレーションにならないような制約になっている。

空でない集合の直積は、厳密には結合則が成り立たない ($(A \times B) \times C \neq A \times (B \times C)$) が、自然な全単射 $((a, b), c) \longleftrightarrow (a, b, c) \longleftrightarrow (a, (b, c))$ によって同一視している。
この自然な全単射で同一視された直積を、リレーションに適用した結果が第 1 正規系？

非第1正規形と正規化

非第1正規形

- ドメインに直積を持つような入れ子型リレーション：内部にリレーションを含むリレーション
- ドメインが冪集合であるようなケース：要素として集合をもつドメイン

リレーショナルデータモデル：意味記述

キー制約

リレーショナルデータベースではデータベースを検索・更新をするいずれの場合にも、リレーションのどのタプルにアクセスしようとするかを的確に指定する必要がある。

リレーションは集合として定義されているため、要素は一意的である（重複が無い）。一般には、「リレーションの全属性の集合」の部分集合がタプルの一意識別能力を持つ。

候補キー：リレーション R 上の属性のうち、一意識別能力を持つような最小の集まり。

候補キーは後に関数従属性の概念を用いて厳密に定義する。

主キー

属性名 A_i が**主キー制約**と呼ばれる以下の条件を満たすとき、**主キー**と呼ばれる：

- A_i はタプルの一意識別能力を備えている：候補キーである条件

$$\forall x, y \in \text{dom}(A_i) \quad x \neq y$$

- A_i の属性は、空値をとらない：

$$\text{null} \notin \text{dom}(A_i)$$

外部キー

外部キー：リレーション $R(\dots, A_i, \dots)$ の属性 A_i が**外部キー制約**と呼ばれる以下の条件

- $t[A_i], u[B_1]$ を、それぞれ $t \in R, u \in S$ の A_i 値と B_1 値を表す。
- $\forall t \in R \quad t[A_i] \neq \phi \Rightarrow (\exists u \in S \quad t[A_i] = u[B_1])$

を満たすとき、リレーション $S(B_1, \dots)$ に関する**外部キー**であるという。

圏的スキーマとインスタンス

ここでは属性間・リレーション間の不変な関係を図示する圏的スキーマ(categorical scheme)を定義し、それを基にリレーションを作成するインスタンスを定義する。

準備

グラフと経路

グラフ $G = (V, A, s, t)$:

1. V は集合。
2. A は集合。
3. $s : A \rightarrow V$ は関数。
4. $t : A \rightarrow V$ は関数。

グラフ G 上の長さ n の経路 $p \in \text{Path}_G^{(n)}$:

$$p = (v_0 \xrightarrow{a_1} v_1 \xrightarrow{a_2} v_2 \xrightarrow{a_3} \cdots \xrightarrow{a_n} v_n)$$

ここで、 $\text{Path}_G^{(0)} = V$, $\text{Path}_G^{(1)} = A$ と定める。

グラフ G 上の長さ n の経路の集合 $\text{Path}_G^{(n)}$ は、fiber積 $\times_V := \times_{s,V,t}$ を使って定義できそう :

$$\text{Path}_G^{(n)} = \underbrace{A \times_V A \times_V \cdots \times_V A}_{n\text{個の}A\text{の積}}$$

この時、経路 p と矢 a を合成した経路 ap, pa が直積として自然に定義できる。

グラフ G 上の全ての経路の集合 :

$$\text{Path}_G := \bigcup_{n \in \mathbb{N}} \text{Path}_G^{(n)}$$

経路同値

グラフ G とその経路 Path_G が与えられたとき、経路に対する同値関係を定義する。

ここでは $p, q \in \text{Path}_G$, $m, n \in A$ とする。

path equivalence declaration (PED) $\sim : p \sim q \stackrel{\text{def}}{\iff} (s(p) = s(q) \wedge t(p) = t(q))$

経路 Path_G 上のcategorical path equivalence relation (CPER) \simeq :

1. $p, q : a \rightarrow b \wedge m : z \rightarrow a$ のとき、 $p \sim q \Rightarrow mp \sim mq$
2. $p, q : a \rightarrow b \wedge n : b \rightarrow c$ のとき、 $p \sim q \Rightarrow pn \sim qn$

補題 : $(p \simeq q : u \rightarrow v) \wedge (r \simeq s : v \rightarrow w) \Rightarrow pr \simeq qs$

圏的スキーマ

グラフ G と $\text{CPER} \simeq$ の組を (圏的) **スキーマ** (categorical scheme) \mathcal{C} と呼ぶ : $\mathcal{C} := (G, \simeq)$.

V が対象の集合, Path_G を射の集合, 経路の結合を射の合成と見なし、図式の可換性を同値関係である $\text{CPER} \simeq$ とすれば、単位射 $1_v : v \rightarrow v$ ($v \in V$) を定義することでスキーマは圏になる。

本節では単にスキーマと呼が、後に実世界を記述するためにリレーションスキーマ, データベーススキーマを定義する。

インスタンス

スキーマ \mathcal{C} 上のインスタンス $I = (PK, FK) : \mathcal{C} \rightarrow \mathbf{Set}$:

1. $PK : V \rightarrow \mathbf{Set}$ (全ての $v \in V$ に対し、 $I(v)$ は集合である。)
2. $\forall a : v \rightarrow w \in A, \quad FK(a) : PK(v) \rightarrow PK(w)$
3. $p, q : v \rightarrow w \in \text{Path}_G, \quad p = (a_i)_{i=1, \dots, n}, q = (b_i)_{i=1, \dots, m}$ とする。

$$p \simeq q \quad \Leftrightarrow \quad FK(a_n) \circ \dots \circ FK(a_1) = FK(b_m) \circ \dots \circ FK(b_1)$$

スキーマを圏と見なした時、インスタンスは関手と見なせる。

インスタンスの定義とリレーションとの関係を以下に記す :

1. $PK(v)$ はリレーション名 v のリレーションにおける主キーに対応。
2. $a : v \rightarrow v' \in A, \quad PK(a) : PK(v) \rightarrow PK(v')$ はリレーションを定義する際のドメインの直積に対応。
3. 他のリレーションの値を用いる場合、外部キーに対応する属性を用いてリレーションを定義しても本質的な問題は無い。

また、以下が成立する :

- スキーマを対象に、スキーマの変換を射とすれば、スキーマの圏**Sch**が定義できる。
- 任意のスキーマ \mathcal{C} に対し、スキーマを対象、自然変換を射とすれば、 \mathcal{C} 上のインスタンスの圏 $\mathcal{C} - \mathbf{Inst}$ が定義できる。
- **Sch** \simeq **Cat** : 圏同値

今後、圏論と一緒に掘り下げたい。

例 : 圏的スキーマ

圏的スキーマの具体例をみるなら、"[Category Theory for Scientists \(Old Version\)](#)"(Spivak, David I. 2013. p.110~112)を参考にするとよい。

同文献のp.109. Example 3.5.2.15では、文章構造から自然にスキーマを定義出来ている点は面白い。

* この例では、文章がSVO文型だから綺麗な対応が見られるのかもしれない。スキーマが適切に定義できる文章の構造について考えるのは面白そう。

またp.110. Exercise 3.5.3.2では図式だけでなく、言語によって一貫性制約を記述することでデータベーススキーマになっている様に思える。

その他の制約

一貫性制約 (integrity constraint) : データベースが実世界の状態と矛盾せず、一貫していることを保証するための制約。

キー制約以外に、以下の項目がある。

- この章で紹介する制約 : 検査制約, 表明, トリガ
- 正規化理論の章で扱う制約 : 関数従属性, 多値従属性

検査制約(check constraint)

検査制約 : 1つのリレーション内で適切な関係を保証するための制約を、**検査制約**という。設計者の視点からの制約記述。

例：

- 社員の所属する部署の値は'K55', 'K41', 'その他'のいずれかでなければならない。
- 社員の平均給与は20以上でなければならない。

表明 (assertion)

表明：異なるリレーション間で定義される制約を**表明**によって記述する。設計者の視点からの制約記述。

例：リレーション「社員」，「部門」を定義したとする。

「上司よりも高給をとっている社員がいてはいけない」という制約は2つのリレーション間を用いて表現される。

トリガ (trigger)

ユーザのアプリケーションの視点からの制約。

ユーザ定義とは？ここの理解は、先にSQL触った方が良さそう。

あるリレーションの更新をきっかけに、他のリレーションに対して変更をするための仕掛けが**トリガ**である。

例：社員一部門データベースがあるとする。

リレーション「社員」において、新入社員を挿入したり、辞める人を削除する際には、リレーション「部門」の部員数を増減する必要がある。

データベーススキーマ

データベーススキーマ：リレーションスキーマに対し、さらに一貫性制約を課した不変な構造。

リレーションスキーマ：リレーションの不変な構造。おそらく属性の直積 $D_1 \times \cdots \times D_n$ を指しており、（その部分集合である）リレーションが操作によって変わる事を強調する際に（リレーションスキームと対比して）インスタンスと呼んでいる。

データベーススキーマ：リレーションスキーマに一貫性制約を含めた不変な構造。

圏的スキーマ：リレーション間の不変な構造。圏的スキーマからインスタンスによってリレーションを作れる。

["Category Theory for Scientists \(Old Version\)"](#) (Spivak, David I. 2013. p.110. Exercise 3.5.3.2) では図式だけでなく、言語によって一貫性制約を記述することでデータベーススキーマになっている様に思える。

リレーショナル代数

リレーショナル代数：リレーショナルデータベースを操作するための演算。リレーションについて閉じている集合操作である。

- 集合論的な演算：和，差，共通部分，直積
- リレーショナルの演算：射影，選択，結合，商

これらのうち和，差，直積，射影，選択の5つの演算は独立であり、他の演算はこれらを用いて定義することができる。また射影，選択，結合，商はドメインを用いて定義するため、リレーショナルらしい演算であると言える。

本章の参考文献：[石川 佳治, "データベース 【4：リレーショナルデータモデル（2）】"](#)

集合論論的な演算

以下の演算は、リレーションを単に集合として見なした演算である。

和集合

先ず、和集合を定義するため、和両立を定義する。

和両立 : $R(A_1, \cdots, A_n), S(B_1, \cdots, B_m)$ について、

- $n = m$: R, S の次数が等しい。
- $\forall i \in \{1, \cdots, n\} \text{ dom}(A_i) = \text{dom}(B_i)$

和集合 \cup : リレーション R, S が和両立とする。

$$R \cup S := \{t | t \in R \vee t \in S\}$$

差集合

差集合 :

$$R \setminus S := \{t | t \in R \wedge t \notin S\} = \{t \in R | t \notin S\} \subset R$$

共通集合

共通集合 :

$$R \cap S := R \setminus (R \setminus S) = \{t | t \in R \wedge t \in S\} \subset R, S$$

直積集合

直積集合 :

$$R \times S := \{(r, s) | r \in R \wedge s \in S\}$$

リレーショナルの演算

本項目では、 $A = \{A_i\}_{i=1, \dots, n}, B = \{B_i\}_{i=1, \dots, m}, C = \{C_i\}_{i=1, \dots, k}$ を属性の集合とする。

射影演算

$R(A_1, \cdots, A_n)$ の全属性集合 A の部分集合を $X = \{A_{i_k} | \forall k \in \mathbf{N}, 1 \leq k \leq n\} \subset A$ とする。

R の X 上の射影 :

$$\sigma_X(R) \equiv R[X] := \{t[X] | t \in R\}$$

集合論の射影演算子 π を用いて、 $\sigma_X(R) := \prod_{A_i \in X} \pi_{A_i}(R) = \prod_{A_i \in X} \text{dom}(A_i)$ と記述できそう

選択演算

θ を比較演算子とする。

A_i, A_j が **θ −選択可能**：次の命題の真偽が定まる： $\forall t \in R, t[A_i] \theta t[A_j]$

$R(\cdots, A_i, \cdots, A_j, \cdots)$ の A_i と A_j 上の **θ −選択**： R の要素のうち、 $A_i \theta A_j$ を満たすタプルの集合を返す。

$$\sigma_{A_i \theta A_j}(R) \equiv R[A_i \theta A_j] := \{t \mid t \in R \wedge (t[A_i] \theta t[A_j])\} \subset R$$

結合演算

結合

$R(\cdots, A_i, \cdots)$ と $S(\cdots, B_j, \cdots)$ の A_i と B_j 上の **θ −結合** $\bowtie_{A_i \theta B_j}$ ： R と S の直積のうち、 $A_i = B_j$ を満たすタプルのみからなる集合を返す。

$$R \bowtie_{A_i \theta B_j} S \equiv R[A_i \theta B_j]S := \sigma_{A_i \theta B_j}(R \times S) = \{(t, s) \mid t \in R \wedge u \in S \wedge t[A_i] \theta u[B_j]\} \subset R \times S$$

等結合

等結合：比較演算子 θ に等号 $=$ を用いた結合： $R \bowtie_{A_i = B_j} S \equiv R[A_i = B_j]S$

等結合 $R[A_i = B_j]S$ で得られるリレーシオンは、各タプルの属性 A_i, B_j が重複する。そこで自然な結合を定義する。

自然結合

自然結合：結合の操作時に、重複する属性を除外しながら結合する演算。

$R(A_1, \cdots, A_n, B_1, \cdots, B_m), S(B_1, \cdots, B_m, C_1, \cdots, C_k)$ をリレーシオンとする。

$$R \bowtie S \equiv R * S := \pi_{A_1, \cdots, A_n, B_1, \cdots, B_m, C_1, \cdots, C_k}(\sigma_{R[B]=S[B]}(R \times S))$$

商演算

$R(A_1, \cdots, A_{n-m}, B_1, \cdots, B_m)$ を n 次の、 $S(B_1, \cdots, B_m)$ を m 次のリレーシオンとする。

R を S で割った商： R の属性のうち、 S にも属性として含まれているものを除く。

$$R \div S := \pi_{A_1, \cdots, A_n}(R) - \pi_{A_1, \cdots, A_n}((\pi_{A_1, \cdots, A_n}(R) \times S) - R) = \{t \mid t \in R[A_1, \cdots, A_{n-m}] \wedge \forall u \in S ((t, u) \in R)\}$$

$$\text{Prop. } (R \times S) \div S = R$$

リレーショナル論理

集合論に基づいたリレーショナル代数とは別のデータ操作言語として、述語論理に基づいた**リレーショナル論理**なるものがあるらしい。

リレーショナル論理：

- タプルリレーショナル論理
- ドメインリレーショナル論理

3つの大系は問い合わせの記述能力に関して等価である（記述できること・できないことが同じ）。逆に言うと、いずれもリレーショナルデータベースのデータ操作言語として取りこぼしが無い（リレーショナル完備である）。

正規化理論：更新時異常と無損失分解

リレーションが第1正規形であっても、リレーションの更新時に異常が発生する場合がある。これを解決するために高次の正規化を定義したい。

本章はその準備として、情報無損失分解の理論と関数従属性について述べる。

更新時異常

第一正規形であっても、リレーションの更新する際にキー属性の制約に抵触し、異常が発生する場合がある。

更新時異常：タプルの挿入・削除・修正時に発生する異常

- タプル挿入時異常：キー属性に該当する値がnullであるタプルを挿入することでキー制約に抵触する。
- タプル削除時異常：あるキー属性の値が唯一つのタプル t にしか格納されていない場合、そのタプル t の削除することはキー制約に抵触する。
- タプル修正異常：あるキー属性 A の値が唯一つのタプル t にしか格納されていない場合、そのタプルの属性値 $t[A]$ を修正することはキー制約に抵触する。

従属性

情報無損失分解

意図しない更新時異常は、しばしば、異なる事象を同一のリレーションで管理した場合に引き起こされる（one fact in one relation ポリシーの抵触）。

ここではリレーションの更新時の情報の損失が無いような分解を考えるため、（インスタンスではなく）リレーションスキーマの分割について考える。

情報無損失分解：

X, Y, Z が互いに素な属性集合であるリレーションスキーマ $R = R(X, Y, Z)$ について、 R を2つの射影 $R[X, Y], R[X, Z]$ に分解したとき、

$$R = R[X, Y] * R[X, Z]$$

が成立するならば、この分解は**情報無損失**であるという。

定理【情報無損失分解】：

X, Y, Z が互いに素な属性集合であるリレーションスキーマを $R = R(X, Y, Z)$ とする。

$R[X, Y], R[X, Z]$ への分解が情報無損失であるための必要十分条件は以下が成立することである。

$$R = R[X, Y] * R[X, Z] \Leftrightarrow \forall R \subset R, \forall t, t' \in R (t[X] = t'[X] \Rightarrow (t[X, Y], t'[Z]), (t'[X, Y], t[Z]) \in R)$$

(証明の概要) 自然結合の定義より $R[X, Y] * R[X, Z]$ は、 $t[X] = t'[X]$ を満たすタプル t, t' から、タプル $(t[X], t[Y], t'[Z]), (t'[X], t'[Y], t[Z])$ からなる直積集合を作る演算である。これは任意のインスタンスにおいて成立する。

多値従属性

多値従属性：リレーションスキーマ $\mathbf{R}(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ に対して情報無損失分解と等価な条件

$$\forall R \subset \mathbf{R}, \forall t, t' \in \mathbf{R} (t[\mathbf{X}] = t'[\mathbf{X}] \Rightarrow (t[\mathbf{X}, \mathbf{Y}], t'[\mathbf{Z}]), (t'[\mathbf{X}, \mathbf{Y}], t[\mathbf{Z}]) \in \mathbf{R})$$

が成立するとき、 \mathbf{R} に**多値従属性** (multi-valued dependency, MVD) が存在するといい、 $X \twoheadrightarrow Y|Z$ とかく。

Y と Z が、X に対して従属性を持っているから多値従属？

直交：リレーションスキーマ \mathbf{R} が多値従属性 $X \twoheadrightarrow Y|Z$ を満たすとき、Y と Z は**直交**するという。

関数従属性

関数従属性：リレーションスキーマ $\mathbf{R}(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ が

$$\forall R \subset \mathbf{R}, \forall t, t' \in \mathbf{R} (t[\mathbf{X}] = t'[\mathbf{X}] \Rightarrow t[\mathbf{Y}] = t'[\mathbf{Y}])$$

を満たすとき、**関数従属性** (functional dependency, FD) が存在するといい、 $X \rightarrow Y$ とかく。

定理：任意のリレーションスキーマ $\mathbf{R}(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ について、

$$(X \rightarrow Y) \Rightarrow (X \twoheadrightarrow Y|Z)$$

(証明)

関数従属性 $X \rightarrow Y$ が成立するならば $t[X] = t'[X] \Rightarrow t[Y] = t'[Y]$ であるから、 $t[X, Y] = t'[X, Y]$ が成立する。
よって $(t[X, Y], t'[Z]) = (t'[X, Y], t'[Z]) = t'[X, Y, Z] \in R$ である。
 $(t'[X, Y], t[Z]) \in R$ も同様にして示せる。

これより、関数従属性は、多値従属性の特殊な場合と理解できる。

候補キー

関数従属性を用いて、候補キーを定義する。

スーパーキー：リレーションスキーマ $\mathbf{R}(\mathbf{A}_1, \dots, \mathbf{A}_n)$ の属性集合 K が**スーパーキー**であるとは、以下の性質を満たすことである。

$$\forall R \subset \mathbf{R}, \forall t, t' \in \mathbf{R} (t[\mathbf{K}] = t'[\mathbf{K}] \Rightarrow t = t')$$

つまり、リレーションスキーマの全属性集合 A に対して、関数従属性 $K \rightarrow A$ が成立する属性集合 K を、スーパーキーという。

候補キー：リレーションスキーマ $\mathbf{R}(\mathbf{A}_1, \dots, \mathbf{A}_n)$ のスーパーキー K が**候補キー**であるとは、以下の性質を満たすことである。

$$\forall H \subsetneq K \neg (\forall t, t' \in R (t[H] = t'[H] \Rightarrow t = t'))$$

つまり、リレーションスキーマの全属性集合 A に対して、関数従属性 $K \rightarrow A$ が成立する最小の属性集合（スーパーキー） K を、候補キーという。

完全関数従属

完全関数従属：関数従属性 $X \rightarrow Y$ について、 $\forall X' \subset X \neg(X' \rightarrow Y)$ であるとき、 Y は X に**完全関数従属**（fully functionally dependent）しているという。

関数従属性の公理化

データベース設計者はリレーションスキーマを定義する際に、実世界の状況を理解したうえで「成立している"根本的"な関数従属性は見落とさない」という信念をもって、成立すべき関数従属性を列挙する（関数従属性の集合の定義）必要がある。

しかし、抽出した関数従属性が他の関数従属性の集合から導けたり、新たな関数従属性を見つけるかもしれない。このことを定式化する。

関数従属性の推移律

定理：リレーションスキーマ \mathbf{R} の関数従属性は推移律を満たす。

$$(X \rightarrow Y) \wedge (Y \rightarrow Z) \Rightarrow X \rightarrow Z$$

（証明）背理法： $X \rightarrow Y \wedge Y \rightarrow Z$ であるが、 $X \rightarrow Z$ が成立しないと仮定する。

あるインスタンス $R \subset \mathbf{R}$ について、関数従属性より以下の条件を満たすを満たすタプルが少なくとも2つのタプル t, t' が存在する。

$$t[X] = t'[X] \Rightarrow t[Y] = t'[Y], \quad t[Y] = t'[Y] \Rightarrow t[Z] = t'[Z]$$

これより $t[X] = t'[X] \Rightarrow t[Z] = t'[Z]$ が成立するが、これは仮定に反する。

この定理より、関数従属性がいくつか存在すると、新しい関数従属性が**導出**する（derive）ことができる事が分かる。

アームストロングの公理系

アームストロングの公理系：属性集合 A のリレーションスキーマを $R(A)$ とする。

1. 反射律： $Y \subset X \Rightarrow X \rightarrow Y$ （自明な従属性）
2. 増加律： $(X \rightarrow Y) \Rightarrow \forall Z \subset A (X \cup Z \rightarrow Y \cup Z)$
3. 推移律： $X \rightarrow Y \wedge Y \rightarrow Z \Rightarrow X \rightarrow Z$

アームストロングの公理系の性質：

- 健全性（sound）：アームストロングの公理系によって導出される関係は、リレーションスキーマ \mathbf{R} 上の関数従属性である。
- 完全性（complete, 完備性とも？）： F^+ 中の全ての関数従属性が、公理系の適用によって導出可能である。

つまり、関数従属性の集合 $F = \{f_i : X \rightarrow Y\}_{i=1, \dots, n}$ が与えられたとき、この公理系を適用することで（ F の下で）実現可能な全ての関数従属性の集合 F^+ を導くことができる。

閉包：関数従属性 F から導かれる関数従属性の全体 F^+ を、**閉包**（closure）という。

含意問題

含意問題：リレーションスキーマ $\mathbf{R}(\mathbf{X}, \mathbf{Y}, \dots, \mathbf{Z})$ の関数従属性の集合 F が与えられたとする。

「 \mathbf{R} の属性からなる集合 X, Y について、関数従属性 $X \rightarrow Y$ （すなわち $X \rightarrow Y \in F^+$ ）が成立するか」という問題。

これを F に関する X の閉包 $X^+ = \{A \mid A \in \{X, \dots, Z\} \wedge (X \rightarrow A) \in F^+\}$ を求めることで実現する。

X の閉包 X^+ を求めるアルゴリズム：

1. $X(0) = X$ とおく。
2. $X^{(1)} = X^{(i-1)} \cup \{z \mid Y \rightarrow z \in F, Y \subset X^{(i-1)}\}$ ($i \geq 1$)
3. $X(i) = X^{(i-1)}$ であれば $X^+ = X^{(i-1)}$ とし、そうしなければ 2. にいく。

関数従属性の閉包 F^+ を求めるアルゴリズムは計算量が指数関数オーダーである一方、属性集合の閉包 X^+ は多項式オーダーで計算できる。

関数従属性の重要性

データモデルが構造記述，意味記述，操作記述の3要素からなることは、コードも述べている。

関数従属性は意味記述は興味深い（実際、リレーション更新時の異常は主キーが満たすべきキー制約が原因になっている）。

高次の正規形は、全て関数従属性で記述される。

正規化理論：高次の正規化（以下、勉強中）

第2正規形

第3正規形

ボイスーコッド正規形

第4正規形

第5正規形

リレーショナルデータベース設計

実世界のモデル化

概念モデル

論理モデル

データベース構築：

1. 世界を
2. 実体－関連モデル（E-Rモデル）

データベース管理システム

質問処理の最適化

トランザクション

障害時回復

同時実行制御

オブジェクト指向データベース