データベース入門（随時更新）

本稿の内容と参考文献

本稿はデータベース始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、多数の不備が含まれている可能性があることに注意。

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LaTeXで文書作成をするだろう。（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 Rreview 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の表現が使えること，また私の能力ではpdfに変換した際にLaTeXの形式による集合の括弧{，}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える。

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$I = \{1, 2, \cdots, n\}$を用いて$D_1, D_2, \cdots, D_n$と記述する習慣がある。

ドメインの定義の例：

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

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

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

リレーションの定義

$D_1, D_2, \cdots, D_n$上の**リレーション**Rを、

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

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。

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。

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

これを表で表した場合の一例を挙げる。

1	2
太朗	25
一朗	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$A_i \ (i = 1, \cdots, n)$
- ドメイン関数 $\mathrm{dom} : \mathrm{A_i} \mapsto \mathrm{D_i} \quad (\mathrm{i} = 1, \cdots, \mathrm{n})$

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$\{1, \cdots, n\}$ではなく属性名$\{A_i\}_{i \in \{1, \cdots, n\}}$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。

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

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。

$$R \subset \mathrm{dom}(\mathrm{A}_1) \times \mathrm{dom}(\mathrm{A}_2) \times \cdots \times \mathrm{dom}(\mathrm{A_n})$$

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

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。

- $\mathrm{dom}(\text{名前}) = \mathrm{D}_1$

- $\mathrm{dom}(\text{年齢}) = \mathrm{D}_2$
- 友人 $\subset \mathrm{dom}(\text{名前}) \times \mathrm{dom}(\text{年齢})$

これを表で表すと、以下のように記述できる。

名前	年齢
太朗	25
一朗	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$R \subseteq D_1 \times \cdots \times D_n$の全てのドメイン$D_i$に対して制約を設ける。

第 1 正規系（ther first normal form, 1NF）：

1. $\forall D_i \ \neg(\exists \{D'_\lambda\}_{\lambda \in \Lambda} \ \ |\{D'_\lambda\}_{\lambda \in \Lambda}| \neq 1 \Rightarrow D_i = \prod_{\lambda \in \Lambda} D'_\lambda)$
2. $\forall D_i \ \neg(\exists D' \ \ D_i \in D')$

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。

> PostgreSQLのpoint型は非第 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。

リレーションの**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。

以下では、$t[A] \in \mathrm{dom(A)}$をタプル $t \in R(A, \cdots)$ の属性Aの値を表すとする。

主キー

属性Aが**主キー**であるとは、属性Aが**主キー制約**と呼ばれる条件

1. A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を満たす）：

$$\forall t, t' \in R \;\; t[A] \neq t'[A] \Rightarrow t \neq t'$$

2. Aの属性は、空値をとらない：

$$\forall t \in R \;\; t[A] \neq \mathrm{null}$$

を満たすことを指す。

外部キー

リレーション$R(A, \cdots)$の属性Aがリレーション$S(B, \cdots)$に関する**外部キー**であるとは、Rの属性A**外部キー制約**と呼ばれる条件

$$\forall t \in R \;\; t[A] \neq \mathrm{null} \Rightarrow (\exists \mathrm{u} \in \mathrm{S} \;\; \mathrm{t[A] = u[B]})$$

を満たすことを指す。

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。

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。

トリガ（trigger）

トリガ：**ユーザ定義**の意味的制約を記述する仕掛け。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？

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。

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：属性の直積$D_1 \times \cdots \times D_n$を指す。リレーションの操作に対して不変な構造に着目した概念。リレーショの操作によって変わる性質を強調する際は、（リレーションスキーマと対比して）リレーションをインスタンスと呼ぶ。（リレーションにおける属性の依存関係（正規化理論の章を参照）は含む？）
> - 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：（リレーションスキーマと従属性からなる）属性の依存関係を考慮した不変な構造。圏であるスキーマから関手であるインスタンスによってリレーションを作れる。

リレーショナルデータベース設計

実世界のモデル化

データベース構築（実世界のデータベース化）：次の 2 段階で行われる。

0. アクセプタを用いて実世界を認識する。
1. 実世界のデータ構造と制約を記号系によって記述した**概念モデル**を作成する。
   - アクセプタ：測定器，白地図など
   - 概念モデル：実体ー関連モデルが典型例。
2. （概念モデルは必ずしもＢＣＭＳで管理可能な表現とは限らないので、）実動可能な表現にモデルを変換することで**論理モデル**を得る。
   - 論理モデルは概念スキーマとも呼ばれる（データベース管理システムの標準アーキテクチャを定めたANSI/X3/SPARCの術語）。
   - 論理モデル：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，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

概念モデルを作成する理由：

- 論理モデルに振り回されずに実世界の情報構造を的確に捉えられる
- ある論理モデルから別の論理モデルへのデータベース変換のチューニングが行いやすい

概念モデル

いくつかの概念モデルを解説する。

実体ー関連モデル

実体ー関連モデル（entity-relationship model, E-Rモデル）：

- 実世界は**実体**（entity）と，実体間の**関連**（reltionship）で成り立っているとする。
- 個体の実体同士の関連を扱うのではなく、**属性**（attribute）によって特徴づけられた**実体型**（entity type）と**関連型**（reltionship type）を用いて表現する。

各型に関する主キー：

- 実体型：実体を指定する最小の属性の組が主キーとなる。

- 関連型：主キーK, Hからなる実体型をそれぞれ$E_L(K, A), E_R(H, B)$とし，また属性Cを持つE_L, E_R間の関連型を$R(C)$とする。
  i. 1 対 1 関連型：Rの主キーはKまたはH
  ii. 1 対多関連型：Rの主キーはH
  iii. 多対 1 関連型：Rの主キーはK
  iv. 多対多関連型：Rの主キーは和集合$\{K, H\}$

弱実体型（weak entity type）：単独では主キーを持てず、実体型と関連を持つことで一意識別能力が得られる実体型。

- **所有実体型**（owner entity type）：一意識別能力を与える実体型
- **部分キー**（partial key）：所有実体型の主キーと組になることで一意識別能力を持つことができる属性
- **識別関連型**（identifying relationship type, ID関連型）：弱実体型と所有実体型との関連

拡張型実体ー関連モデル

実世界の概念間の階層を表現する。

継承（inheritance）：ある実体型の属性がより具体的な実体型に属性を引き継ぐ場合、実体間で属性の継承があるという。

ISA関連（is-a relationship）：実体型間における属性の継承に関する関連。

> 反対称性を持たない、属性同士の関係

継承の概念を用いて、ある実体型からより抽象的/具体的である新たな実体型を定義できる。

- **特殊化**（specialization）：実体型が、属性を継承されたより具体的な実体型を定義すること。
- **汎化**（generalization）：実体型が、属性を継承するより抽象的な実体型を定義すること。

論理モデル

一意識別能力に着目すれば、論理モデルへの変換を定めることができる。
以下では（第 1 正規形のリレーションからなる）リレーショナルデータモデルへの変換ルールを記す。

実体ー関連モデルのRDBへの変換ルール：実体型$E(K, A), E_R(K, A), E_L(H, B)$の主キーをそれぞれ$K, K, H$とし、所有実体型$E_{\text{owner}}(E, A)$と弱実体型$E(P, D)$の主キーと部分キーをぞれぞれ$K, P$とする。

1. 実体型$E(K, A)$は、Kを主キーとするリレーションスキーマ$\mathbf{R_E}(K, A)$に変換される。
2. 実体型$E_L(K, A), E_R(H, B)$間の 1 対 1 関連型$R(C)$は、K, Hのいずれかを主キーとするリレーションスキーマ$\mathbf{R}(K, H, C)$に変換される。
3. 実体型$E_L(K, A), E_R(H, B)$間の 1 対多関連型$R(C)$は、Hを主キーとするリレーションスキーマ$\mathbf{R}(K, H, C)$に変換される。
4. 実体型$E_L(K, A), E_R(H, B)$間の多対 1 関連型$R(C)$は、Kを主キーとするリレーションスキーマ$\mathbf{R}(K, H, C)$に変換される。
5. 実体型$E_L(K, A), E_R(H, B)$間の多対多関連型$R(C)$は、$\{K, H\}$を主キーとするリレーションスキーマ$\mathbf{R}(K, H, C)$に変換される。
6. 弱実体型$E(K, A)$とその所有実体型$E_{\text{owner}}(H, B)$は、$\{K, H\}$を主キーとするリレーションスキーマ$\mathbf{R_E}(K, H, C)$に変換される。

（ID関連型は属性を持たないから、変換不要である。）

コラム：実体ー関連モデル

実体ー関連モデルの実世界の事象の捉え方は一意ではない。
何を実体とし、何が契約であるかは恣意的に決めるものである。

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。
- $\forall i \in \{1, \cdots, n\}\ \mathrm{dom(A_i)} = \mathrm{dom(B_i)}$

和集合$\cup$：リレーションR, Sが和両立とする。

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

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$X = \{A_{i_k} | \forall k \in \mathbf{N},\ \mathbf{1} \leq \mathbf{k_i} \leq \mathbf{n}\} \subset \mathbf{A}$とする。

RのX上の射影：

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

選択演算

θを比較演算子とする。

A_i, A_jが$\theta-$**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$\theta-$**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。

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

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$\theta-$**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。

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

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ョンは、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。

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

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

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$m$次のリレーションとする。

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。

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

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$\mathbf{R} = \mathbf{R}(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$について、
$\mathbf{R}$を2つの射影$\mathbf{R}[\mathbf{X}, \mathbf{Y}], \mathbf{R}[\mathbf{X}, \mathbf{Z}]$に分解したとき、

$$\mathbf{R} = \mathbf{R}[\mathbf{X}, \mathbf{Y}] * \mathbf{R}[\mathbf{X}, \mathbf{Z}]$$

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$\mathbf{R} = \mathbf{R}(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$とする。
$\mathbf{R}[\mathbf{X}, \mathbf{Y}], \mathbf{R}[\mathbf{X}, \mathbf{Z}]$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。

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

（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

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

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、$X \to\to Y|Z$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$X \to\to Y|Z$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

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

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

$$(X \to Y) \ \Rightarrow (X \to\to Y|Z)$$

（証明）
関数従属性$X \to Y$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。

スーパキー：リレーションスキーマ$\mathbf{R}(\mathbf{A_1}, \cdots, \mathbf{A_n})$の属性集合$K$が**スーパキー**であるとは、以下の性質を満たすことである。

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

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$K \to A$が成立する属性集合Kを、スーパキーという。

候補キー：リレーションスキーマ$\mathbf{R}(\mathbf{A_1}, \cdots, \mathbf{A_n})$のスーパキー$K$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$K \to A$が成立する最小の属性集合（スーパキー）Kを、候補キーという。

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。

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

（証明）背理法：$X \to Y \wedge Y \to Z$であるが、$X \to Z$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：$Y \subset X \Rightarrow X \to Y$（自明な従属性）
2. 増加律：$(X \to Y) \Rightarrow \forall Z \subset A\,(X \cup Z \to Y \cup Z)$
3. 推移律：$X \to Y \wedge Y \to Z \Rightarrow X \to Z$

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$F = \{f_i : X \to Y\}_{i=1,\cdots,n}$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$\mathbf{R}(\mathbf{X}, \mathbf{Y}, \cdots, \mathbf{Z})$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$X \to Y$（すなわち$X \to Y \in F^+$）が成立するか」という問題。

これをFに関するXの閉包$X^+ = \{A | A \in \{X, \cdots, Z\} \wedge (X \to A) \in F^+\}$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. $X^{(1)} = X^{(i-1)} \cup \{z | Y \to z \in F, Y \subset X^{(i-1)}\} \quad (i \geq 1)$
3. $X(i) = X^{(i-1)}$ であれば $X^+ = X^{(i-1)}$ とし、そうれなければ 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、コッドも述べている。
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

非キー属性：いかなる候補キーに属していない属性をいう。

以下では更新時以上を解消するため、候補キーと非キー属性との関数従属性の関係に着目し、高次の正規化を定義する。
データベースを作成する際は、高次の正規形を満足するようにリレーションを分解する。

第２正規形

完全関数従属：関数従属性$X \to Y$について、$\forall X' \subset X,\ \neg(X' \to Y)$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。

第２正規形：以下の２つの条件を満たすリレーションスキーマ$\mathbf{R}$を、**第２正規形**（2NF）という。

1. $\mathbf{R}$は1NFである。
2. $\mathbf{R}$の全ての非キー属性は、$\mathbf{R}$の各候補キーに完全関数従属性している。

第３正規形

推移的関数従属性：主キーA，非キー属性B, Cからなるリレーションスキーマ$\mathbf{R}(\mathbf{A}, \mathbf{B}, \mathbf{C})$について、以下の条件を満たすとき、$C$は$A$に**推移的に関数従属している**という。またこのとき$\mathbf{R}$は推移的関数従属性を持つという。
A \rarr C \land (A,B) \rarr C$$

第３正規形：以下の２つの条件を満たすリレーションスキーマ$\mathbf{R}$を、**第３正規形**（3NF）という。

1. $\mathbf{R}$は2NFである。
2. $\mathbf{R}$の全ての非キー属性は、$\mathbf{R}$のいかなる候補キーにも、推移的に関数従属しない。

これらの条件は、

ボイスーコッド正規形

ボイスーコッド正規形：以下の条件の**いずれかを満たす**リレーションスキーマ$\mathbf{R}(\mathbf{X}, \cdots, \mathbf{Y})$を、**ボイスーコッド正規形**（BCNF）という。

- 関数従属性$X \to Y$は**自明な関数従属性**（$Y = \phi \vee Y \subset X$）である。
- Xは$\mathbf{R}$のスーパキーである。

第４正規形

第４正規形：以下の２つの条件を満たすリレーションスキーマ$\mathbf{R}$を、**第４正規形**（4NF）という。

1. 多値従属性$X \twoheadrightarrow Y | Z$は**自明な多値従属性**（$Z = \phi \vee Y \subset X$）である。
2. Xは$\mathbf{R}$のスーパキーである。

より高次の正規形

Wikipedia（2023年10月５日 確認）によれば、他にも第５次・６次の正規形や、Domain-key normal formがある。

一方、3NFに正規化した時点で高次の正規形である場合も多いとの話もある。高次の正規形は一部の情報が失われる可能性があるので注意が必要であるそうだ。

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。

圏的な正規形

データベースについて、圏的なスキーマを定義するための条件（圏的な正規形）を定義する。

圏的な正規形：以下の条件を満たすとき、データベースは圏的な正規形（categorical noraml form）であるという。

1. 全てのテーブルtは１つの主キーである列ID_tを持つ。これをID列（rod-ids）と呼ぶ。
2. テーブルtの全ての列cについて、列cの各値を参照するtarget table（的テーブル）t'を持つ。これを
$$c : t \to t'$$
と記述する。
3. 特に、テーブルtの行dが文字列や整数値などの純粋データ（pure data）を含むとき、それらの的テーブルt'は単純に１行のテーブルである。これを
$$d : t \to t'$$
と書く。
4. テーブルtからt'へのデータベース上の２つの経路$p, q : t \to t'$が存在し、スキーマの作成者が同じID列に対応すると知っている場合、このパス同値をスキーマの一部として宣言しなければならない。この経路同値を
$$p \simeq q$$

と記述する。

> ID列が主キー（あるいは実体），グラフの矢が依存関係，グラフ上のパスが従属関係の推移に対応すると捉える

このアイディアをもとに圏的なスキーマを定義する。

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. $s : A \to V$は関数。
4. $t : A \to V$は関数。

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

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

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

> グラフG上の長さ$n_{(>0)}$の経路の集合 $\mathrm{Path}_{\mathrm{G}}^{(\mathrm{n})}$は、fiber積$\times_V := \times_{s,V,t}$を使って定義できそう：
>
> $$\mathrm{Path}_{\mathrm{G}}^{(\mathrm{n})} = \underbrace{\mathrm{A} \times_{\mathrm{V}} \mathrm{A} \times_{\mathrm{V}} \cdots \times_{\mathrm{V}} \mathrm{A}}_{\text{n個のAの積}}$$
>
> この時、$(a, (b, c)) = (a, b, c)$とすれば経路pと矢aを合成した経路apがfiber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：

$$\mathrm{Path}_{\mathrm{G}} := \bigcup_{\mathrm{n} \in \mathbf{N}} \mathrm{Path}_{\mathrm{G}}^{(\mathrm{n})}$$

経路同値

グラフGとその経路$\mathrm{Path}_{\mathrm{G}}$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 $p, q \in \mathrm{Path}_{\mathrm{G}}$，$m, n \in A$ とする。

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

経路$\mathrm{Path}_{\mathrm{G}}$上のcategorical path equivalence relation (CPER) $\simeq$：

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

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

圏的スキーマ

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

> Vが対象の集合，$\mathrm{Path_G}$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CPER$\simeq$とすれば、単位射$1_v : v \to v \ (v \in V)$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ぶが、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**$I = (PK, FK) : \mathcal{C} \to \mathrm{Set}$とは、以下を満たす写像である：

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

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

> 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. $PK(v)$はリレーション名がvであるリレーションの主キーに対応。
2. $a : v \to v' \in A,\ FK(a) : PK(v) \to PK(v')$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、対応する外部キーを用いてリレーションを定義しても本質的な問題はない。

また、以下が成立する：

- スキーマを対象，スキーマの変換を射とすれば、スキーマの圏$\mathbf{Sch}$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（関手の射）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。
- $\mathbf{Sch} \simeq \mathbf{Cat}$：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？

データベース管理システム（以下、勉強中）

DBMSの標準アーキテクチャ

ANSI/X3/SPRCが提案した標準アーキテクチャ：

基本方針：「標準とは、**機能**（facility）間の**インタフェース**（interface）を規定する」

- 機能：管理者・変換器・データ辞書
- インタフェース：機能間のインタフェースとしての各種言語

管理者

管理者：特定の責任を持つ人

1. **組織体管理者**（enterprise administrator）：データベースの概念スキーマを定義する責任を持つ
2. **データベース管理者**（database administrator）：組織体管理者が定義した概念スキーマを、データベースの内部スキーマに変換する責任を持つ
3. アプリケーションシステム管理者（application system administrator）：組織体管理者が定義した概念スキーマを基にして、データベースの様々な外部スキーマを定義する責任を持つ

処理機能

概念スキーマプロセッサ（conceptual schema processor）：（内部スキーマプロセッサ，外部スキーマプロセッサの機能も同様に規定）

1. 組織体管理者によって定義された概念スキーマの構文的・意味的チェック
2. その概念スキーマをコンピュータが理解できる形式にコード化
3. 出来上がる概念スキーマ記述をデータ辞書に格納

データベース変換：

1. **外部／内部データベース変換**：データ辞書に格納されているアプリケーションシステム管理者が定義した外部スキーマと概念スキーマ間の変換情報をもとに、両スキーマ間のオブジェクトの対応を規定し外部データベースをユーザに提供。
2. **内部データベース／内部記憶変換**：

データ辞書

DBMSの３層スキーマ構造

物理的データ独立性

論理的データ独立性

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