

テーブル設計、Viewの作成

1日目

2日目

3日目

4日目

5日目

6日目

7日目

8日目

9日目

10日目

11日目

12日目

13日目

14日目

15日目

16日目

17日目

18日目

19日目

20日目

21日目

テーブル設計の目的

- データの冗長性を排除して、使用するディスクスペースを減らす
- テーブル内のデータの正確さを維持する
- 適切にテーブル分割して、効率的にアクセスできるようにする

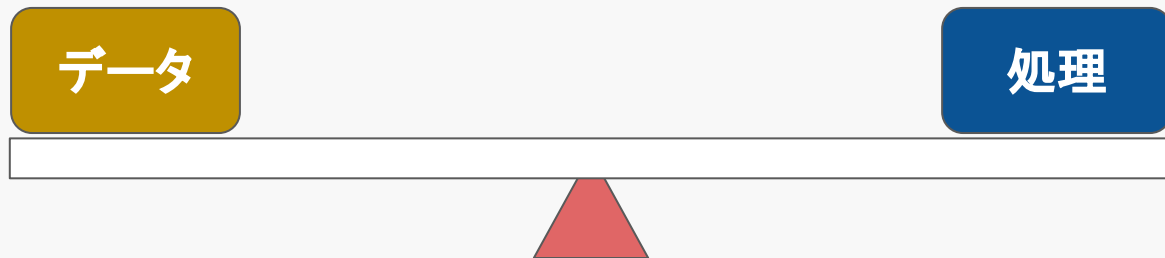
テーブル設計

データ中心アプローチ(DOA)とは

システム設計の考え方の1つ。まず、**どういうデータを利用するのか**を洗い出す。次に、どのようにデータを扱うのかを考えて、処理を書き出すアプローチ

現在、**主流のシステム設計の方法**

他方で、処理を先に洗い出して、その後にデータを設計するアプローチが**プロセス中心アプローチ(POA)**という。



- 処理は、データと比較して実装の段階で **変更が加えられやすい**ため、処理を先に定義してその結果に基づいてデータを決めるのは **非効率**
- 処理内容の変更は容易だがデータ定義の変更は難しく、**データ定義はシステムのパフォーマンス(性能)に直結する**。そのため、データの設計は最優先される

3層スキーマについて

スキーマとは、概要や図解という意味で、データベースの設計範囲を3つに分けて考える。
それぞれ、外部スキーマ、概念スキーマ、内部スキーマという

①外部スキーマ

ユーザー、アプリケーションから見た DB
のデータ構造(ビューなど)

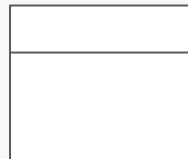
ユーザー



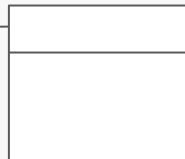
②概念スキーマ

DBの保持するデータ要素データ間の関
係などのデータ構造(テーブルなど)

table_a



table_b



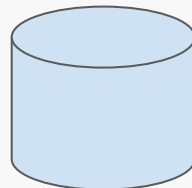
③内部スキーマ

テーブルの細部。テーブルなどのデータ
ファイルの詳細な物理的配置(カラムの
タイプ、インデックスDBファイルの配置
場所,RAIDなど)

データ、インデックス
ファイル



DB



テーブル設計と正規化

概念スキーマを設計する上でやること

概念スキーマ設計では、必要なデータと格納するテーブルを決める。テーブルを分割して外部キーを作成し、どのテーブルとどのテーブルを紐づけるのか決める
このとき効率的にテーブルを分割していくが、この手順を正規化という。

第1 正規形

カラムの繰り返し項目がある場合、繰り返しのカラムを削除してカラムを減らし、削除文を格納する行を増やす

学生番号	...	受験科目名1	点数1
1001	...	英語	90

...

受験科目名N	点数N
数学	80



学生番号	...	受験科目名	点数
1001	...	英語	90
1001	...	数学	80

第2 正規形

候補キー(テーブル内の行を一意に識別するカラム)の一部に 従属しているカラムを分割する

学生番号	科目コード	受験科目名	点数	...
1001	EN	英語	90	...
1001	MA	数学	80	...
1002	EN	英語	80	...
1002	JA	国語	70	...

候補キー: テーブルの行を一意に識別する



受験科目名は、科目コードに従属する

学生番号	科目コード	点数	...
1001	EN	90	...
1001	MA	80	...
:	:	:	...

科目コード	受験科目名
EN	英語
MA	数学
:	:

第3 正規形

候補キー以外のカラムに 従属しているカラムを分割する

学生番号	科目コード	受験科目名	点数	受験地コード	受験地名	...
1001	EN	英語	90	0001	東京	...
1001	MA	数学	80	0002	大阪	...
1002	EN	英語	80	0003	名古屋	...
1002	JA	国語	70	0004	福岡	...



受験地は、受験地コードに従属する

...	受験地コード	...
...	0001	...
...	0002	...
:	:	...

受験地コード	受験地名
0001	東京
0002	大阪
:	:

第3正規形よりあとの正規形について

正規形は第3正規形以降も存在するが(ボイスコード正規形、第4正規形、第5正規形)、実際に使われることは少なく、実務では **第3正規形まで行われることが多い**。

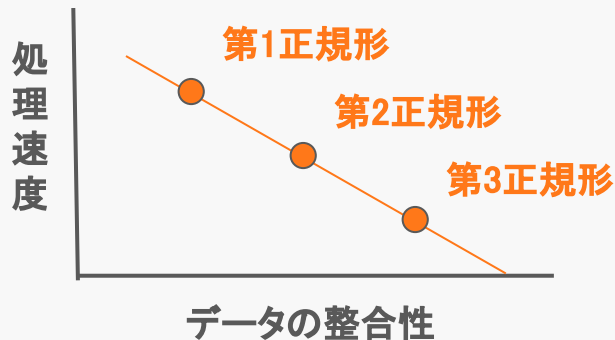
ただし、**あえて第3正規形まで行わないパターンもある** ので、以下で説明する

第3正規形での問題点

第3正規形まで進めると、テーブルを効率的に分割することができ **テーブル構造がわかりやすくなる**。また、総データ量も抑えることもできる。

ただし、**テーブルの結合が多く発生するため、処理時間が増える可能性がある** *)

*) **テーブル結合は一般的に処理時間が長い**。レコード量が少ない(数10万以下)ではあまり気にせず第3正規形まで行ってよい



- **原則として、まずは第3正規形まで 行う**
- **ただし、パフォーマンスの問題が起きた場合は、正規化を無視してカラムを追加する**
(詳細は、20日のSQLチューニングの章でご説明します)

その他のテーブル定義のコツ

1. 主キーは必ずつける。主キーには、数値型(INT等)または固定長文字列(CHAR)を用いる

主キーは、レコードの絞込みや他のテーブルと連結する場合に用いることができます。主キーのないテーブルでも、特定のレコードを表示したり削除する場面はあるので**必ず主キーは作成しましょう**。

主キーには、数値か固定文字列を使いましょう。主キーは他のテーブルとの連結に使うこともあるため、可変文字列では誤って空白が入るなどで連結に失敗したり値が変更される可能性があります。

主キーの例

名前	型	例	確認するところ
student_id	UNSIGNED INT	1, 2, 3, 4, 5, ...	INTで数値の数は足りるか
student_id	CHAR(8)	00000001, 00000002	CHARのサイズは足りるか
author_id, book_id	CHAR(8), CHAR(8)	A0000001, B0000001 A0000001, B0000002	CHARのサイズは足りるか

2. レコードが挿入された時刻を表す(create_at)とレコードが更新された時刻を表す(update_at)は必ずつける

create_atとupdate_atを作成すると、そのレコードがいつ作成されたのか、いつ更新されたのかあとで調べることができ、デバッグをする際に便利です。必ずつけましょう。

DEFAULT値にCURRENT_TIMESTAMPを使い、現在時刻が挿入されるようにする
update_atには、ON UPDATE CURRENT_TIMESTAMPをオプションで付与する
データタイプには、DATETIMEかTIMESTAMPを用いる

studentsテーブル

student_id(PK)
name VARCHAR : create_at TIMESTAMP update_at TIMESTAMP

3. 文字列型や数値型は、大きすぎる型を設定しない

文字列型や数値型は無駄に大きすぎるものを設定すると、データ使用量が多くなるため必要十分なサイズに設定する(ギリギリに設定すると、挿入するデータに対してサイズが足らなくなる可能性があるので注意) 例えば年齢カラムに、INT(-2147483648 ~ 2147483647)を設定しても、そこまでの数値は必要でないので TINYINT UNSIGNED(0 ~ 256)で十分です。

数値型(必要な容量)

TINYINT(1バイト), SMALLINT(2バイト), MEDIUMINT(3バイト), INT(4バイト), BIGINT(8バイト)

文字列型: 必要な容量

CHAR(M): Mバイト

VARCHAR(M): L + 1バイト(Lとは、格納される文字長)

日付型、時刻型(必要な容量)

DATE(3バイト), DATETIME(8バイト), TIMESTAMP(4バイト), TIME(3バイト)

4. カラムの制約、デフォルト値、コメントはキチンとする

制約やデフォルト値、コメント文の記述をしましょう。

要件と照らし合わせて、制約(NOT NULL, UNIQUE, CHECK)やデフォルト値を入れることで、システムの信頼性の向上につながり、コメント文をいれることで保守性の向上に繋がります

Viewの作成

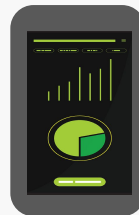
Viewついて

Viewとは、テーブルとテーブルを紐づけたり、特定のカラムだけ取り出したりして作成された仮想的なテーブルのこと。3層スキーマのうち**外部スキーマ**にあたる。

①外部スキーマ

ユーザー、アプリケーションから見た DB
のデータ構造(ビューなど)

ユーザー



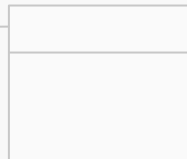
②概念スキーマ

DBの保持するデータ要素データ間の関係などのデータ構造(テーブルなど)

table_a



table_b



③内部スキーマ

テーブルの細部。テーブルなどのデータファイルの詳細な物理的配置(カラムのタイプ、インデックスDBファイルの配置場所,RAIDなど)

データ、インデックス
ファイル



DB



Viewを作成する

paymentsテーブル

id
customer_id
check_number
amount
payment_date

customersテーブル

id
name
phone_number
country
address



paymentsテーブルとcustomersテーブルを紐づけてレコードを取得

```
SELECT
    ct.name, ps.check_number,
    ps.payment_date, ps.amount
FROM payments AS ps
INNER JOIN
    customers AS ct
ON ps.customer_id = cs.id
```

このSQLを仮想的なテーブルとして用意する
(ショートカットのようなもの)

Viewを作成する

```
CREATE VIEW customers_payments_view AS
```

```
SELECT
```

```
    ct.name, ps.check_number, ps.payment_date, ps.amount
```

```
FROM payments AS ps
```

```
INNER JOIN
```

```
    customers AS ct
```

```
ON ps.customer_id = cs.id
```

INNER JOINの結果をビュー
customers_paymentsにする

Viewを利用する

```
SELECT * FROM customers_payments_view; # Viewから値を取り出す
```

これは、customers_payments_viewに値が入っているのではなく、内部的にINNER JOINをしてテーブル結合をしている

```
SHOW TABLES; # 作成したViewがTableとともに表示される
```

```
SELECT * FROM INFORMATION_SCHEMA.VIEWS WHERE TABLE_SCHEMA='db_name' # Viewのみ表示
```

```
SHOW CREATE VIEW view_name; # VIEWの定義を表示
```

Viewを利用する

```
SELECT * FROM customers_payments_view WHERE column = “〇〇”; # 絞り込みをする
```

```
SELECT * FROM customers_payments_view ORDER BY column; # 並び替えをする
```

```
DROP VIEW view_name; # Viewを削除する
```

```
ALTER VIEW view_name AS  
SELECT ...; # Viewを再定義する
```

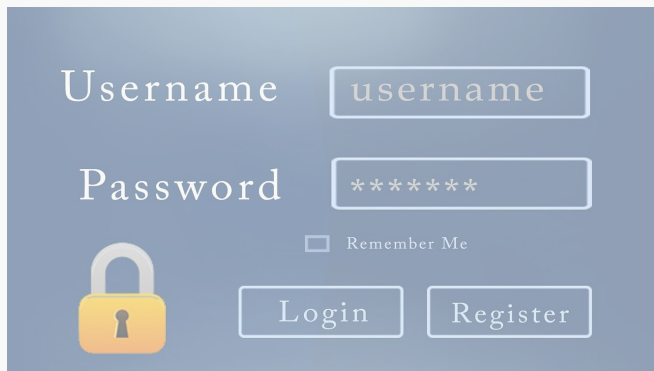
```
RENAME TABLE view_name TO new_view_name; # Viewの名前を変更する
```

Viewを利用する上での注意点

- Viewであることがわかる名前をつける(_view 等)
- (副問い合わせの中などは除き)WHEREやORDER BYなどはつけない
(Viewを利用する際にもWHEREやORDER BYをつけて実行することはできる)
- パフォーマンスが落ちるため、Viewの中に含んだViewを定義しない。必ずテーブルに分解してViewを作成する

Twitterデモアプリのテーブル設計をしよう

これまで学習した内容を用いてTwitter(デモ)のテーブル定義をします



A mockup of a Twitter login/register form. It has a blue background. At the top, it says 'Username' next to a text input field containing 'username'. Below that, it says 'Password' next to a text input field containing '*****'. There is a 'Remember Me' checkbox with the text 'Remember Me' next to it. At the bottom, there is a yellow padlock icon, a 'Login' button, and a 'Register' button.

ログインと登録をするユーザー情報を格納するテーブル(users)を作成します

- ユーザーはメールアドレスで登録して、同じメールアドレスは登録できない
- パスワードは8以上

usersテーブル

id INT UNSIGNED PK

name VARCHAR(50)
password VARCHAR(50)
CHECK(LENGTH(password)>=8)
email VARCHAR(50) UNIQUE,
create_at TIMESTAMP,
update_at TIMESTAMP

```
CREATE TABLE users(  
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,  
  name VARCHAR(50),  
  password VARCHAR(50) CHECK(LENGTH(password) >=8),  
  email VARCHAR(50) UNIQUE,  
  create_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  update_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
  CURRENT_TIMESTAMP  
);
```

つぶやきを格納するテーブル

ユーザーがつぶやいたメッセージを格納する

- usersテーブルと紐づく
- メッセージの文字サイズは上限が140文字

tweetsテーブル

id INT UNSIGNED PK

message VARCHAR(140)
user_id INT UNSIGNED FK
create_at TIMESTAMP
update_at TIMESTAMP

```
CREATE TABLE tweets(  
  id INT UNSIGNED PRIMARY KEY AUTO_INCREMENT,  
  message VARCHAR(140),  
  user_id INT UNSIGNED,  
  create_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  update_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
  CURRENT_TIMESTAMP,  
  CONSTRAINT fk_users_tweets FOREIGN KEY(user_id)  
  REFERENCES users(id)  
);
```

いいねを格納するテーブル

ユーザーがつぶやいたメッセージに対していいねをつける

- どのつぶやきかわかる(tweetsテーブルと紐づく)
- 誰がいいねしたかわかる(usersテーブルとも紐づく)

likesテーブル

user_id INT UNSIGNED FK PK tweet_id INT UNSIGNED FK PK
create_at TIMESTAMP update_at TIMESTAMP

```
CREATE TABLE likes(  
  user_id INT UNSIGNED,  
  tweet_id INT UNSIGNED,  
  create_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  update_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
  CURRENT_TIMESTAMP,  
  CONSTRAINT fk_users_likes FOREIGN KEY(user_id)  
  REFERENCES users(id),  
  CONSTRAINT fk_tweets_likes FOREIGN KEY(tweet_id)  
  REFERENCES tweets(id),  
  PRIMARY KEY(user_id, tweet_id)  
);
```


フォロー情報を格納するテーブル

どのユーザーがどのユーザーにフォローしているか格納する

- フォローしているユーザーがわかるカラム(follower_id)とフォローされているユーザーがわかるカラム(followee_id)を作成する
- follower_idとfollowee_idはusersテーブルと紐づく外部キーになる

followsテーブル

follower_id INT UNSIGNED FK PK
followee_id INT UNSIGNED FK PK
create_at TIMESTAMP
update_at TIMESTAMP

```
CREATE TABLE follows(  
  follower_id INT UNSIGNED,  
  followee_id INT UNSIGNED,  
  create_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,  
  update_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE  
  CURRENT_TIMESTAMP,  
  CONSTRAINT fk_users_follower FOREIGN KEY(follower_id)  
  REFERENCES users(id),  
  CONSTRAINT fk_tweets_followee FOREIGN KEY(followee_id)  
  REFERENCES tweets(id),  
  PRIMARY KEY(follower_id, followee_id)  
);
```

テーブルのER図

usersテーブル

id INT UNSIGNED PK
name VARCHAR(50) password VARCHAR(50) CHECK(LENGTH(password)>=8) email VARCHAR(50) UNIQUE create_at TIMESTAMP update_at TIMESTAMP

followsテーブル

follower_id INT UNSIGNED FK PK followee_id INT UNSIGNED FK PK
create_at TIMESTAMP, update_at TIMESTAMP

tweetsテーブル

id INT UNSIGNED PK
message VARCHAR(140) user_id INT UNSIGNED FK create_at TIMESTAMP update_at TIMESTAMP

likesテーブル

user_id INT UNSIGNED FK PK tweet_id INT UNSIGNED FK PK
create_at TIMESTAMP, update_at TIMESTAMP

1 N

1

1

N

N

1

N