|  |
| --- |
| 博士課程教育リーディングプログラム  実世界データ循環学リーダー人材育成プログラム |
| データツールファースト |
| SQL入門 |

|  |
| --- |
| 名古屋大学  2014/03/02 |

* MySQLは、Oracle Corporationの米国およびその他の国における登録商標あるいは商標です。
* その他の、記載されている会社名、製品名は、各社の登録商標あるいは商標です。

目次

[1. データベースとDBMS 1](#_Toc381558382)

[1.1 データベースとは 1](#_Toc381558383)

[1.2 DBMSの機能 1](#_Toc381558384)

[1.3 MySQLへの接続と切断 2](#_Toc381558385)

[1.4 データベースの作成 2](#_Toc381558386)

[1.5 テーブルの作成 3](#_Toc381558387)

[1.6 キーについて 5](#_Toc381558388)

[1.7 データの挿入 5](#_Toc381558389)

[2. 検索 6](#_Toc381558390)

[2.1 検索の基本 6](#_Toc381558391)

[2.2 条件を指定して検索する 7](#_Toc381558392)

[2.3 between 7](#_Toc381558393)

[2.4 in 8](#_Toc381558394)

[2.5 like 8](#_Toc381558395)

[2.6 is null 9](#_Toc381558396)

[2.7 論理演算子 10](#_Toc381558397)

[2.8 検索結果のソート 11](#_Toc381558398)

[2.9 limit 12](#_Toc381558399)

[2.10 算術演算子 12](#_Toc381558400)

[2.11 関数 13](#_Toc381558401)

[2.11.1 文字列長を取得する 14](#_Toc381558402)

[2.11.2 文字列の連結 14](#_Toc381558403)

[2.11.3 lowerとupper 15](#_Toc381558404)

[2.11.4 substringとtrim 15](#_Toc381558405)

[2.11.5 ceiling、floor、round 16](#_Toc381558406)

[2.11.6 日時の取得 16](#_Toc381558407)

[2.11.7 条件分岐 17](#_Toc381558408)

[2.12 distinct 19](#_Toc381558409)

[2.13 coalesce 19](#_Toc381558410)

[3. 集計関数 21](#_Toc381558411)

[3.1 count 21](#_Toc381558412)

[3.2 sum 22](#_Toc381558413)

[3.3 maxとmin 22](#_Toc381558414)

[3.4 avg 22](#_Toc381558415)

[3.5 group by句 23](#_Toc381558416)

[3.6 having 23](#_Toc381558417)

[4. データの更新 25](#_Toc381558418)

[4.1 レコードの挿入 25](#_Toc381558419)

[4.2 データの更新 25](#_Toc381558420)

[4.3 レコードの削除 26](#_Toc381558421)

[5. テーブルの結合 27](#_Toc381558422)

[5.1 テーブルの結合とは 27](#_Toc381558423)

[5.2 内部結合 27](#_Toc381558424)

[5.3 外部結合 28](#_Toc381558425)

[5.4 3つ以上のテーブルの結合 30](#_Toc381558426)

[6. 副問い合わせ 31](#_Toc381558427)

[7. トランザクション 33](#_Toc381558428)

[7.1 トランザクションとは 33](#_Toc381558429)

[7.2 MySQLでのトランザクション 33](#_Toc381558430)

[7.3 トランザクションの実行 34](#_Toc381558431)

[8. 制約 35](#_Toc381558432)

[8.1 NOT NULL制約 35](#_Toc381558433)

[8.2 一意性制約 35](#_Toc381558434)

[8.3 主キー制約(Primary Key) 36](#_Toc381558435)

[8.4 外部キー制約 36](#_Toc381558436)

[9. 正規化 38](#_Toc381558437)

[9.1 非正規形 38](#_Toc381558438)

[9.2 第1正規化 39](#_Toc381558439)

[9.3 第2正規化 40](#_Toc381558440)

[9.4 第3正規化 40](#_Toc381558441)

[10. 参考文献 42](#_Toc381558442)

[11. 変更履歴 43](#_Toc381558443)

# データベースとDBMS

## データベースとは

「データベース」とは、データを整理統合して格納し、そのデータを活用しやすくした仕組みのことです。データベースを用いることで、情報を集約して効率化を図ることができます。単にデータを集めるだけではなく整理統合して活用しやすくしているところがポイントで、この機能を提供するのがデータベース管理システム(DBMS:DataBase Management System)です。DBMSはコンピュータ上でデータベースを構築するために必要な機能を備えたソフトウェアです。

データベースはどのようにデータを表現するかで「ネットワーク型データベース」「カード型データベース」「リレーショナル型データベース」「オブジェクト型データベース」などの種類があります。この中で現在よく使用されているのが、リレーショナル型データベースです。リレーショナル型データベースでは「表(テーブル)」を基本としてデータを表現し、複数の表を関係づけして処理を行います。

MySQLはこのDBMSに該当するソフトウェアで、リレーショナル型のデータベースです。

## DBMSの機能

DBMSの主な機能には次のようなものがあります。

* データの操作(検索・挿入・更新)
* データの整合性の維持(データ重複の排除)
* 障害復旧
* 機密保護
* 同時実行制御

データベースを用いてデータを操作するときにはSQLとよばれる専用の言語を用います。SQLでデータベースを操作する命令を送信すると、データベースエンジンが送信されたコマンドを解釈し実行し、その結果を返します。SQLは大きく分けてデータ定義言語(DDL:Data Definition Language)・データ操作言語(DML:Data Manipulation Language)・データ制御言語(DCL:Data Control Language)の3つの種類から構成されます。

## MySQLへの接続と切断

コマンドプロンプトからユーザ名、接続先ホスト名を指定してMySQLに接続します。

|  |
| --- |
| > mysql -u ユーザ名 -h 接続先ホスト名 -p <return>  Password: パスワード |

MySQLがインストールされているコンピュータで実行するときには、「-h 接続先ホスト名」を省略することができます。ここで指定するユーザ名とパスワードはOSに登録されているものではなく、MySQLに登録されているユーザ名とパスワードを使用します。

正しくログインできるとMySQLのコマンドプロンプトが表示されます。MySQLとの接続を終了する場合は、コマンドプロンプトから

|  |
| --- |
| > quit; <return> |

と入力します。

## データベースの作成

リレーショナル型のデータベースでは情報を表(テーブル)形式で格納します。データを取り出すときには行(レコード)と列(カラムまたはフィールド)を指定します。

情報をデータベースで管理するには、まず最初にテーブルを管理するデータベースを作成します。データベースを作成するには、MySQLのコマンドプロンプトで

|  |
| --- |
| > create database データベース名; <return> |

と入力します。

現在MySQLに存在するデータベースの一覧を表示するには、

|  |
| --- |
| > show databases; <return> |

と入力します。操作対象のデータベースを選択する場合は

|  |
| --- |
| > use データベース名; <return> |

と入力します。このコマンドの後でテーブルを作成すると、useで指定したデータベース内にテーブルが保存されます。

データベースを削除する場合は

|  |
| --- |
| > drop データベース名; <return> |

と入力します。

## テーブルの作成

テーブルには、書籍テーブルなら書籍の情報、著者テーブルなら著者の情報、のように1種類の情報を格納します。テーブルの作成にはcreate文を使用します。

|  |
| --- |
| > create table テーブル名 (カラム定義); <return> |

「カラム定義」にはカラムの名前と型などを以下の形式で指定します。省略した場合は空のテーブルが作成されます。

|  |
| --- |
| カラム名 型 [DEFAULT デフォルト値] [AUTO\_INCREMENT] [PRIMARY KEY] |

複数のカラムを定義する場合は、コンマ区切りで並べます。

カラム名にはカラムの名前を指定します。型はカラムに格納するデータ型を指定します。MySQLでよく使用されるデータ型には以下のものがあります。

|  |  |
| --- | --- |
| INT | 32bit符号付き整数 |
| INT UNSIGNED | 32bit符号なし整数 |
| FLOAT | 単精度(32bit)浮動小数点数 |
| DOUBLE | 倍精度(64bit)浮動小数点数 |
| DATE | 日付(YYYY-MM-DD) |
| DATETIME | 日付と時刻(YYYY-MM-DD HH:MM:SS) |
| TIME | 時刻(HH:MM:SS) |
| VARCHAR(M) | 最大M文字の可変長文字列 |
| TEXT | 最大65,535文字の可変長文字列 |

デフォルト値は、テーブルにデータを挿入する際に値が指定されなかった場合に挿入される値を指定します(省略可)。AUTO\_INCREMENTは、テーブルの中でひとつの整数型カラムにのみ指定できるもので、データ挿入時にカラムに値を指定しなかった場合は「最大値+1」の値が自動で挿入されます(省略可)。DEFAULTとAUTO\_INCREMENTは同時に指定できません。

PRIMARY KEYは、主にここのレコードを識別する重複のないID(主キーと呼ばれる)として使用するカラムに指定します。PRIMARY KEYはテーブルの中でひとつしか指定できません(省略可)。

例えば、学生の情報(学籍番号、氏名、年齢、出身地)を格納するテーブルを作成する場合は、

|  |
| --- |
| > create table students (  id int primary key,  name varchar(50),  age int,  prefecture varchar(50)); <return> |

のように実行します。

テーブルの削除にはDROP TABLEを使います。

|  |
| --- |
| > drop table テーブル名; <return> |

データベース内に存在するテーブルを確認するには

|  |
| --- |
| > show tables; <return> |

また、特定のテーブルの構造を確認したい場合は

|  |
| --- |
| > describe テーブル名; <return> |

と入力します。

## キーについて

テーブル内のレコードを一意に識別するカラムのことを主キーと呼びます。主キーはひとつのカラムの場合もありますし、複数のカラムの組み合わせから構成されることもあります。一つのテーブルに主キーはひとつだけ設定することができます。

例えば、社員テーブルにある会社の社員情報を格納している場合、主キーとして設定できるものに「社員番号」があります。名前や年齢、所属部署では通常社員を一意に特定することはできませんので、これらは主キーにはなりません。

学生テーブルの場合は出席番号を主キーに出来る場合もありますが、同じ出席番号を持つ学生が複数存在することが一般的です。このような場合は出席番号や学年、クラス番号を単独で主キーにすることができませんが、この3つを組み合わせることで学生を一意に特定することができます。このように複数のカラムを組み合わせた主キーのことを特に「連結キー」と呼びます。

リレーショナルデータベースでは効率的にデータを格納するために、複数のテーブルに分割してデータを管理することが一般的です。この時、あるテーブルを外部のテーブルと関連付けるために設定するキーを外部キーといいます。外部キーは関連付けるテーブルの主キーを設定します。

## データの挿入

作成したテーブルにデータを挿入するにはinsert文を使用します。

|  |
| --- |
| > insert into テーブル名 (カラム名１, カラム名2, …) values (‘値1’, ‘値2’…); <return> |

上記の例ではカラム名1に対して値1を、カラム名2に対して値2が格納されます。文字列や日付などは’で囲む必要があります(INTなどの数値データを囲む必要はありません)。

例えば

|  |
| --- |
| > insert into students (id, name, age, prefecture) values (1, ‘山田太郎’, 20, ‘東京’); <return> |

のように実行すると、id列に1、name列に 山田太郎 、age列に20、prefecture列に東京が格納されたレコードが1件studentsテーブルに追加されます。

# 検索

## 検索の基本

テーブルに格納されているデータを検索するにはselect文を使用します。

ある特定のテーブルに格納されている全てのデータを検索したい場合は、

|  |
| --- |
| > select カラム名1, カラム名2, ... from テーブル名; <return> |

または

|  |
| --- |
| > select \* from テーブル名; <return> |

のように実行します。最初の例の場合はカラム名とカラム名の間をカンマで区切ります。一部のカラムのみを指定すると、そのカラムの情報だけが全レコード分検索できます。2つ目の例の「\*」は「すべてのカラム」を意味する構文で、この場合は指定したレコードの全レコードの全カラムが検索対象となります。

「id, name, age, prefecture」というカラムを持つstudentsテーブルの場合は

|  |
| --- |
| > select id, name, age, prefecture from students; <return> |

または

|  |
| --- |
| > select \* from students; <return> |

と実行すると、studentsテーブルの全ての情報が表示されます。

特定のカラムを取り出す場合は、select文の後に要求するカラムのカラム名を指定します。

|  |
| --- |
| > select カラム名1, カラム名2, ... from テーブル名; <return> |

例えば、上述のテーブルの例で

|  |
| --- |
| > select id, name from students; <return> |

と実行すると、studentsテーブルに含まれるidとnameカラムの情報が全レコード分取得できます。

## 条件を指定して検索する

要求するレコードの条件を指定する場合は、select文にwhere句を指定し、その中で条件を記述します。条件の指定には演算子を用います。

|  |
| --- |
| > select カラム名1, カラム名2,... from テーブル名 where カラム名 演算子 条件値; <return> |

指定できる演算子には以下のようなものがあります。

|  |  |
| --- | --- |
| = | 等しい |
| <>. != | 等しくない |
| < | より小さい |
| <= | より小さいか等しい |
| > | より大きい |
| >= | より大きいか等しい |

例えば、studentsテーブルからageカラムが20以上のレコードだけを取得するには以下のように実行します。

|  |
| --- |
| > select \* from students where age >= 20; <return> |

## between

where句の条件に指定できる演算子にbetweenがあります。この演算子を用いると範囲指定を行って条件を指定することができます。通常、数値型のカラムか日付型のカラムに対して使用します。

|  |
| --- |
| > select カラム名1, カラム名2, ... from テーブル名 where カラム名 between 下限値 and 上限値; <return> |

betweenの後に下限値を、andの後に上限値を指定する必要があります。下限値以上、上限値以下という条件指定にり、検索結果には下限値及び上限値の値が含まれます。

例えば、studentsテーブルからageカラムが20以上25以下のレコードだけを取得するには

|  |
| --- |
| > select \* from students where age between 20 and 25; <return> |

と実行します。

## in

where句の条件に指定できる演算子にinがあります。この演算子を用いると、複数の値を条件に指定することが可能です。

|  |
| --- |
| > select カラム名1, カラム名2 from テーブル名 where カラム名 in (値リスト); <return> |

値リストでは指定したカラムが取り得る値をカンマで区切って指定します。

例えば、studentsテーブルからprefectureカラムが osaka または tokyo のレコードだけを取得するには、

|  |
| --- |
| > select \* from students where prefecture in (‘osaka’, ‘tokyo’); <return> |

と実行します。

## like

where句の条件に指定できる演算子にlikeがあります。この演算子を用いると、文字列パターンを使用したあいまい検索が可能です。

|  |
| --- |
| > select カラム名1, カラム名2 from テーブル名 where カラム名 like ‘文字列パターン’; <return> |

文字列パターンには以下の様なワイルドカードを用いることができます。

|  |  |
| --- | --- |
| % | 任意の0文字以上の文字列 |
| \_ | 任意の一文字 |

例えば、studentsテーブルからnameカラムに naka が含まれるレコードだけを取得するには

|  |
| --- |
| > select \* from students where name like ‘%naka%’; <return> |

と実行します。

ワイルドカード文字を % や \_ という文字としてマッチさせたい場合は\文字を使ってエスケープを行います。(厳密には「\」(バックスラッシュ)ですが、「\」と表示されることが多いようです。)

|  |  |
| --- | --- |
| \% | %にマッチする |
| \\_ | \_にマッチする |

また\を文字としてマッチさせたい場合にも、エスケープを行い

|  |
| --- |
| \\ |

のように指定します。

\以外をエスケープ文字として使用する場合はescape節を使用します。

|  |
| --- |
| > select カラム名1, カラム名2 from テーブル名 where カラム名 like ‘文字列パターン’ escape ‘エスケープ文字’; <return> |

例えば、@をエスケープ文字として使用する場合は、

|  |
| --- |
| > select \* from students where name like ‘%@\_%’ escape ‘@‘; <return> |

のように実行します。エスケープ文字の直後のワイルドカード文字(\_)のみが通常の文字として扱われます。上記の例では、nameに\_が含まれるレコードを検索します。

## is null

カラムにデータが存在しないという状態を表すのがnullです。カラムがnullかどうかは=演算子では調べられません。is null演算子を用いることで、nullを条件にした検索が可能です。

|  |
| --- |
| > select カラム名1, カラム名2 from テーブル名 where カラム名 is null; <return> |

例えば、studentsテーブルからprefectureカラムに値が入っていない(nullな)レコードだけを検索する場合は、

|  |
| --- |
| > select \* from students where prefecture is null; <return> |

と実行します。

「nullではない」という条件を指定する場合はis not nullと指定します。

## 論理演算子

複数の条件を組み合わせて検索を行う場合はandやor演算子を用います。

|  |
| --- |
| > select カラム名1, カラム名2 from テーブル名 where 条件式1 and 条件式2; <return> |

論理演算子には、以下のものがあります。

|  |  |
| --- | --- |
| A and B | AかつB |
| A or B | AまたはB |
| not A | Aではない |

例えば、studentsテーブルからprefectureがtokyoまたはosakaのレコードだけを取得するには、

|  |
| --- |
| > select \* from students where prefecture = ‘tokyo’ or prefecture = ‘osaka’; <return> |

と実行します。また、studentsテーブルからprefectureがkyotoでかつageが25以上のレコードだけを取得するには、

|  |
| --- |
| > select \* from students where prefecture = ‘kyoto’ and age >= 25; <return> |

と実行します。

元の条件を否定した検索を行う場合は、論理演算子notを用います。

|  |
| --- |
| > select カラム名1, カラム名2 from テーブル名 where not 条件式; <return> |

この場合はwhere句で指定された条件式を否定した条件で検索が行われます。例えば、studentsテーブルからprefectureがtokyo以外のレコードだけを取得するには、

|  |
| --- |
| > select \* from students where not prefecture = ‘tokyo’; <return> |

と実行します。

今までに出てきた「between」や「in」はandやorを使って書き換えることができます。

これら3つの論理演算子の優先順位はnotが一番高く、orが一番低い「not, and, or」の順です。そのため、

|  |
| --- |
| > select \* from students where age >= 25 and prefecture = ‘tokyo’ or prefecture=‘osaka’; <return> |

と実行すると、

1. ageが25以上でprefectureがtokyo

2. prefectureがosaka

のいずれかの条件を満たすレコードが検索結果となります。ここで

1. ageが25以上でprefectureがtokyo

2. ageが25以上でprefectureがosaka

のいずれかの条件を満たすレコードを検索結果とするためには()をつけて演算子の優先度を変えます。

|  |
| --- |
| > select \* from students where age >= 25 and (prefecture=‘tokyo’ or prefecture=‘osaka’); <return> |

## 検索結果のソート

order by句を用いると、指定したカラムでレコードをソートした結果を取得できます。

|  |
| --- |
| > select カラム名1, カラム名2 from テーブル名 where 条件式 order by カラム名 [asc ¦  desc]; <return> |

ascを指定すると昇順で、descを指定すると降順で、order byで指定したカラムを元にソートを行い、結果を出力します。ascまたはdescの指定を行わない場合は、ascを指定したとみなされます。whereと同時に用いる場合はorder by句は最後に記述します。

例えば、studentsテーブルからprefectureがtokyoのレコードをageの降順でソートした結果を取得するには、

|  |
| --- |
| > select \* from students where prefecture = ‘tokyo’ order by age desc; <return> |

と実行します。

## limit

取得した結果から何件目のデータを取り出すかを指定する場合はlimit句を使います。

|  |
| --- |
| > select カラム名1, カラム名2 from テーブル名limit [取得開始位置,]件数; |

指定した件数分のデータを取得できます。取得開始位置は0からスタートし、指定しない場合は0件目を指定したものとして扱われます。通常はorder byと合わせて使われます。

studentsテーブルからageの順でソートした結果の2件目から3つのデータを取得する場合は、

|  |
| --- |
| > select \* from students oder by age limit 1, 3; |

のように実行します。

## 算術演算子

数値や日付データの計算を行う場合は、算術演算子を用います。

ageに10加算した値を得たい場合は、

|  |
| --- |
| > select age + 10 from students; <return> |

のように実行します。指定できる演算子には

|  |  |
| --- | --- |
| + | 加算 |
| - | 減算 |
| \* | 乗算 |
| / | 除算 |
| DIV | 除算（結果は整数部のみ取得） |
| % | 剰余 |

があります。

## 関数

関数を用いることで、データを加工することができます。

MySQLで予め用意されている代表的な関数は以下のとおりです。

|  |  |
| --- | --- |
| LENGTH(str) | 文字列長（バイト数） |
| CONCAT(str1, str2, …) | 文字列連結 |
| LOWER(str) | 大文字を小文字に変換 |
| UPPER(str) | 小文字を大文字に変換 |
| SUBSTRING(str, pos[, len]) | 文字列str中のpos文字目からlen文字を取得 |
| TRIM(str) | strの前後の空白を除去 |
| CEILING(num) | 小数点以下の切り上げ |
| FLOOR(num) | 小数点以下の切り捨て |
| ROUND(num[, spec]) | 数値の丸め（specは小数点以下の桁数） |
| NOW() | 現在日時 |
| CURDATE() | 現在日付 |
| CURTIME() | 現在時刻 |
| DATEDIFF(date1, date2) | 日付の差 |
| CASE | 条件による処理分岐 |

関数はネストして用いることができます。その場合は内側の関数から処理が行われます。例えば、

|  |
| --- |
| > select upper(substring(first\_name, 1, 2)) from students; |

のように実行すると、まずsubstringが行われてからその結果に対してupperが実行されます。

### 文字列長を取得する

length関数を用いると、文字列長を取得することができます。

|  |
| --- |
| > select length(文字列); |

例えば、 hello の文字列長を表示する場合は、

|  |
| --- |
| > select length(‘hello’); |

のように実行します。また、studentsテーブルのfirst\_nameカラムの値の文字列長を表示する場合は、

|  |
| --- |
| > select length(first\_name) from students; |

のように実行します。

lengthはバイト数を値として返すため、マルチバイト文字を指定した場合は文字数と一致しません。例えば「やまだ」という文字列の場合は6と表示されます。マルチバイト文字で文字数を取得したい場合はlengthの代わりにchar\_length関数を用います。

### 文字列の連結

concat関数を用いると、文字列の連結が行えます。

|  |
| --- |
| > select concat(文字列1, 文字列2, ...); <return> |

concatの引数には連結したい文字列をカンマ区切りで指定します。例えば、 hello と world を続けて表示するには

|  |
| --- |
| > select concat(‘hello’, ‘world’); <return> |

のように実行します。studentsテーブルにlast\_nameカラムとfirst\_nameカラムがあり、繋げて表示したい場合は、

|  |
| --- |
| > select concat(last\_name, first\_name) from students; <return> |

のように実行します。

また、select文には文字列定数を指定することもできます。上記の例で「名前: yamada taro」のように表示したい場合は、

|  |
| --- |
| > select concat(‘名前：’, last\_name, ‘ ‘, first\_name) from students; <return> |

のように実行します。上記のように実行すると、表示される列名は指定した関数が表示されます。実行結果の列名を指定する場合は、asキーワードを使います。

|  |
| --- |
| > select 列名 as 別名 from テーブル名; <return> |

たとえば、前述の例で実行結果の列名を「名前」にする場合は、

|  |
| --- |
| > select concat(‘名前：’, last\_name, ‘ ‘, first\_name) as 名前 from students; <return> |

のように実行します。

asキーワードは表定義とは異なる名前で実行結果の表示を行いたい場合にも使えます。また、MySQLではasは省略可能です。

### lowerとupper

lower関数は大文字を小文字に、upper関数は小文字を大文字に変換します。

|  |
| --- |
| > select lower(文字列); |

のように実行します。studentsテーブルのfirst\_nameカラムの値を大文字で表示したい場合は、

|  |
| --- |
| > select upper(first\_name) from students; |

のように実行します。

### substringとtrim

substring関数を用いると文字列の一部を取得することができます。

|  |
| --- |
| > select substring(文字列, pos [,len]); |

と実行すると、文字列かのpos文字目からlen文字を取得します。lenを指定しない場合はpos文字目から最後までを取得できます。

|  |
| --- |
| > select substring(‘Hello’, 3, 2); |

と実行すると結果は’ll’が得られます。studentsテーブルのlast\_nameカラムの値から先頭の2文字だけを取得したい場合は

|  |
| --- |
| > select substring(last\_name, 1, 2) from students; |

のように実行します。

### ceiling、floor、round

ceiling関数、floor関数、round関数は小数点の丸めを行いますが、少しずつ動作が異なります。ceiling関数は指定値以上の最小の整数を返しますが、floor関数は指定値以下の最大の数を返します。round関数は、デフォルトで小数点以下を四捨五入します。

例えば、2.3を指定すると、ceilingは3を返しますがfloorは2を返します。また-2.3を指定した場合は、ceilingは-2をfloorは-3を返します。つまり、ceilingは常に正方向への丸め、floorは負方向への丸めを行います。

roundは小数点数の他に、どの桁で四捨五入を行うかという指定を行うことが可能です。

|  |
| --- |
| > select round(123.456);  > select round(123.456, 2);  > select round(123.456, -2); |

のように実行すると、一つ目の例では小数点以下の四捨五入で123となりますが、2つめの例では小数点以下の指定桁数を四捨五入の対象として123.46となります。3つめの例では整数部2桁目で四捨五入を行い、100となります。

### 日時の取得

日時の取得の代表的な関数にnowがあります。now関数は

|  |
| --- |
| > select now(); |

のように実行すると、現在の日時を取得することができます。また、現在の日付だけを取得す場合はcurdate関数、現在の時刻だけを取得する場合はcurtime関数を利用します。

|  |
| --- |
| > select curdate();  > select curtime(); |

また、2つの日付の差を求めるにはdatediff関数を用います。

|  |
| --- |
| > select datediff(日付1, 日付2); |

例えば、

|  |
| --- |
| > select datediff(‘2014-03-19’, ‘2014-03-10’); |

のように実行すると2つの日付の差である9を取得することができます。

### 条件分岐

SQL文の中で条件分岐の指定を行う場合はcase関数を使います。

|  |
| --- |
| case 式  when 条件1 then 結果1  [when 条件2 then 結果2  when 条件n then 結果n  else 値]  end |

case関数では、与えられた式の値とwhen句で指定された条件(比較値)を比較し、初めて合致したところで対応する結果(値)を返します。合致する値がなかった場合は、else句の値を返します。when句は必要な数だけ列挙でき、elseは省略可能です。式の値がNULLである場合、条件としてNULLを指定しても処理が行われません。NULLの場合の条件を指定する場合は、coalesceを用いるか「case when val is null」などで対応します。

例えば、studentsテーブルにclub\_idというカラムがあり、値が1の場合はテニス、2の場合は水泳、それ以外の場合は野球と返す場合は以下のように実行します。

|  |
| --- |
| > select first\_name,  case club\_id  when 1 then ‘テニス’  when 2 then ‘水泳’  else ‘野球’  end  from students; |

上記のように実行するとstudentsテーブルからfirst\_nameの値とclub\_idに該当するクラブ名が取得できます。

whenには条件式を指定することもできるので、scoreカラムの値が80以上なら優、60以上なら可、60未満なら不可と返す場合は、

|  |
| --- |
| > select first\_name,  case  when score >= 80 then ‘優’  when score >= 60 then ‘可’  else ‘不可’  end  from students; |

のように実行します。

## distinct

取得したデータの中に重複する行が含まれていた場合は、distinctを用いて重複を取り除くことができます。

|  |
| --- |
| > select distinct 列名1 [, 列名2, ...] from テーブル名; |

selectで取得した結果の中に重複する行が含まれる場合に、その行を取り除きます。studentsテーブルからtaroというfirst\_nameを持つレコードのlast\_nameを取得する場合、

|  |
| --- |
| > select last\_name from students where first\_name=‘taro’; |

と実行すると、同姓同名の学生がいる場合は同じ値を持つlast\_nameの結果が複数得られますが、

|  |
| --- |
| > select distinct(last\_name) from students where first\_name=‘taro’; |

と実行すると、重複が無い結果を得ることができます。

## coalesce

coalesceはNULL値でない最初の引数を返す関数です。複数の引数を持ち、その引数を最初から評価して、最初にNULL値でない引数を返します。

|  |
| --- |
| coalesce(a, b, c, ...) |

coalesceは以下のように動作します。

|  |
| --- |
| > select col1, col2, col3, coalesce(col1, col2, col3) from test;  col1 col2 col3 coalesce(col1, col2, col3)  ————————————————————————-  (null) apple lemon apple  lemon apple banana lemon  (null) (null) lemon lemon |

カラムの値を用いて計算を行う場合、NULL値が含まれていると結果がNULLになります。そこで、coalesce関数を用いてNULLの場合は0に置き換えて計算を行うというような使い方をすることがあります。

|  |
| --- |
| > select coalesce(age, 0) + 10 from students; |

# 集計関数

集計関数は、複数行を処理して合計や平均などの集計処理を行う関数です。代表的な集計関数には以下のものがあります。

|  |  |
| --- | --- |
| sum(列名) | 合計を出力 |
| avg(列名) | 平均値を出力 |
| max(列名) | 最大値を出力 |
| min(列名) | 最小値を出力 |
| count(列名) | データ件数を出力 |

集計関数はNULLを除外して集計します。count関数を用いてデータ件数を数える場合、その列にNULL値のレコードが存在していると全レコード数と一致しません。

また、集計関数はネストして使用することができ、その場合は内側の関数から順番に処理されます。

|  |
| --- |
| > select avg(coalesce(age, 0)) from students; |

と実行すると、age列にNULLを持つレコードが存在する場合は、その値を0に変換してから平均値を求めます。

## count

count関数を用いるとレコード数を取得できます。studentsテーブルからageのレコード数を数えるには

|  |
| --- |
| > select count(age) from students; |

のように実行します。countはNULLのレコードは存在しないものとして扱います。列名に\*を指定した場合は行の値が全てNULLでもカウントされます。

|  |
| --- |
| > select count(\*) from students; |

のように実行すると、studentsテーブルに含まれるレコード件数が分かります。また、distinctと組み合わせると、その列に含まれる値の種類の数を調べることができます。

## sum

selectで取得したレコードの指定列の合計値を求めるにはsum関数を用います。

studentsテーブルから東京出身者のage列の合計を求めるには

|  |
| --- |
| > select sum(age) from students where prefecture=‘東京’; |

のように実行します。NULL値を持つレコードが含まれている場合、NULLは存在しないものとして扱われます。

## maxとmin

selectで取得したレコードの指定列の最大値および最小値を求めるにはmax関数やmin関数を用います。

studentsテーブルから、最も小さいageの値を取得するには、

|  |
| --- |
| > select min(age) from students; |

のように実行します。

## avg

selectで取得したレコードの指定列の平均値を求めるにはavg関数を用います。studentsテーブルからageの平均値を求めるには、

|  |
| --- |
| > select avg(age) from students; |

と実行します。countと同様にNULLのデータが存在する場合は集計の対象としません。例えば、age列の値が「100, NULL, 200 」である場合、avgの結果は150になります。NULLも集計の対象とする場合はcoalesce関数を用います。

|  |
| --- |
| > select avg(coalesce(age, 0)) from students; |

上記のように実行すると、NULLを0として扱いavgの結果は100になります。

## group by句

集計関数を用いてカラムの値を集計する場合に、特定のカラムを対象にグループ化を行いグループ単位で集計を行うことができます。どのカラムを対象にグループ化するのかを指定するにはgroup by句を使用します。

|  |
| --- |
| > select 列名1, 列名2, ... from テーブル名 group by 列名3[, 列名4]; |

group by句の後に指定したカラムに格納されている値が同じデータをグループとしてまとめます。複数のカラムを指定した場合は、複数のカラムの値の組み合わせが同じデータをグループとしてまとめます。

studentsテーブルから同じclub\_idを持つ行のage列の平均を求める場合は、

|  |
| --- |
| > select club\_id, avg(age) from students group by club\_id; |

のように実行します。

group by句とwhere句を同時に指定した場合、where句のほうが先に処理されます。また、集計関数をwhere句に指定することはできません。例えば、

|  |
| --- |
| > select club\_id, avg(age) from students where age >= 20 order by club\_id; |

と実行した場合、

1. ageが20以上のレコードのみをstudentsテーブルから取得

2. club\_idでグループ化してage列の平均値を求める

という順序で処理されます。

## having

集計関数を用いて条件指定を行う場合は、having句を用います。

|  |
| --- |
| select 列名1, 列名2, ... from テーブル名 having 条件; |

club\_idでグループ化して集計した場合にageの平均が22以上となるclub\_idを求める場合は、

|  |
| --- |
| > select club\_id from students group by club\_id having avg(age) >= 22; |

のように実行します。また、where句とhaving句を同時に指定した場合は、where句が先に処理されます。

|  |
| --- |
| > select club\_id from students where age >= 20 group by club\_id having avg(age) >= 25; |

のように実行すると、

1. ageが20以上のレコードをstudentsテーブルから抽出

2. club\_idでグループ化してageの平均が25以上のレコードを抽出

の順で処理されます。

# データの更新

## レコードの挿入

テーブルにレコードを挿入するにはinsert文を用います。

|  |
| --- |
| > insert into テーブル名 (カラム名1, カラム名2, ...) values (値1, 値2, ...); <return> |

カラム名には値を格納するカラム名のリストをカンマ区切りで指定します。値はカラム名のリストと対になるように指定します。上記の場合はカラム名1に値1が、カラム名2に値2が格納されたレコードが挿入されます。

カラム名リストに指定しなかったカラム入る値は、テーブル定義で決まります。テーブル定義時にdefaultが指定されている場合は、そこで指定された値が格納されます。auto incrementが指定されていれば、最大値+1の値が格納されます。しかしテーブル定義で何も指定していない場合はnullになります。また、nullをカラムに入れる場合は値リストにnullを指定します。

|  |
| --- |
| > insert into students (id, name, age, prefecture) values (10, ‘taro’, 20, null); <return> |

と実行すると、idとして10、nameとしてtaro、ageとして20, prefectureにnullが格納された新しいレコードが1件、studentsテーブルに挿入されます。

## データの更新

既存のデータを更新するにはupdate文を用います。

|  |
| --- |
| > update テーブル名 set カラム名1=値1, カラム名2=値2, ... where 条件式; <return> |

カラム名1に値1が、カラム名2に値2が設定されます。値にはnullも指定できます。whereで指定される条件に合致するレコードが更新対象レコードとして処理されます。where句を省略すると全レコードが対象となります。

|  |
| --- |
| > update students prefecture = ‘tokyo’ where age = 20; <return> |

と実行すると、studentsテーブルに格納されているageが20の全レコードのprefectureがtokyoに更新されます。

## レコードの削除

既存のレコードを削除するにはdelete文を用います。

|  |
| --- |
| > delete from テーブル名 where 条件式; <return> |

条件式に合致するレコードが全て削除されます。where句を省略するとテーブル内に含まれる全レコードが削除されます。

|  |
| --- |
| > delete from students where age >= 25; <return> |

と実行すると、ageが25以上の全レコードがstudentsテーブルから削除されます。

# テーブルの結合

## テーブルの結合とは

リレーショナルデータベースでは、関連するデータを無駄なく格納するためにいくつかのテーブルに分割してデータを管理するのが一般的です。この分割したテーブルをキー同士で関連付けて1つのデータに統合する処理を結合といいます。

結合には、内部結合、外部結合などさまざまな種類があります。

## 内部結合

内部結合は、テーブル間の主キーと外部キーを紐付けてキー同士の値が一致するデータだけを取得する結合です。内部結合を行うにはinner join句を使います。

|  |
| --- |
| > select 列名1, 列名2, ... from テーブル名1 inner join テーブル名2 on テーブル名1.外部キー = テーブル名2.主キー; |

例えば、以下の様な学生テーブルとクラブテーブル、施設テーブルが存在していたとします。

<students>

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| student\_id | name | grade | height | club\_id |
| 1 | taro | 2 | 165 | 1 |
| 2 | jiro | 2 | 168 | 2 |
| 3 | hanako | 1 | 170 | 1 |
| 4 | ken | 4 | 164 | 4 |
| 5 | mike | 3 | 175 | 5 |

<clubs>

|  |  |  |
| --- | --- | --- |
| club\_id | club\_name | location\_id |
| 1 | 野球 | 2 |
| 2 | テニス | 3 |
| 3 | サッカー | 2 |
| 4 | 水泳 | 1 |

<clubs>

|  |  |
| --- | --- |
| location\_id | location\_name |
| 1 | ３号棟プール |
| 2 | 北グラウンド |
| 3 | 南グラウンド |

学生テーブルには外部キーとしてclub\_idが設定されています。この2つを内部結合するには、

|  |
| --- |
| > select students.student\_id, students.name, students.club\_id, clubs.club\_name  from students  inner join clubs  on students.club\_id = clubs.club\_id; |

のように実行します。

結合を行う場合は、カラム名がどのテーブルに属するものであるかが分かるように「テーブル名.カラム名」の形式で記述します。ただし、テーブル間でカラム名が重複していない場合は、「カラム名」として指定することができます。

また、結合を行う際にはテーブル名に別名をつけることができます。

|  |
| --- |
| > select s.student\_id, s.name, s.club\_id, c.club\_name  from students as s  inner join clubs as c  on s.club\_id = c.club\_id; |

## 外部結合

内部結合は、結合するテーブルの双方に存在するデータだけを取得します。これに対して外部結合は、結合する片方のテーブルにしか存在しないデータを取り出すことができます。結合するデータがない部分はNULLとして表示されます。外部結合を行い場合はleft join句もしくはright join句を用います。

|  |
| --- |
| > select 列名1, 列名2, ... from テーブル名1 left join テーブル名2 on テーブル名1.外部キー = テーブル名2.主キー; |

上記の例では、テーブル名2に該当データがない場合でもテーブル名1の情報はすべて表示されます。leftではなくrightを指定した場合は、テーブル名1に該当データがない場合でもテーブル名2の情報はすべて表示されます。

学生テーブルとクラブテーブルの例では、

|  |
| --- |
| > select students.student\_id, students.name, students.club\_id, clubs.club\_name  from students  left join clubs  on students.club\_id = clubs.club\_id; |

実行すると、内部結合では表示されなかったmikeのデータが表示されます。mikeの所属するクラブの情報がクラブテーブルに存在していないため、mikeのクラブ名はNULLになります。

|  |
| --- |
| > select students.student\_id, students.name, students.club\_id, clubs.club\_name  from students  right join clubs  on students.club\_id = clubs.club\_id; |

と実行すると、同様に内部結合では表示されなかったサッカー部の情報が表示されます。しかし、サッカー部に所属する学生が存在しないため、学生情報はNULLで表示されます。

## 3つ以上のテーブルの結合

3つ以上のテーブルを結合する場合も、2つのテーブルを結合する時と同じく「join テーブル名 on 結合条件」で結合します。students, clubs, locationsを結合する場合は以下のように実行します。

|  |
| --- |
| > select students.student\_id, students.name, students.club\_id, clubs.club\_name,  locations.location\_name  from students  inner join clubs  on students.club\_id = clubs.club\_id  inner join locations  on clubs.location\_id = locations.location\_id; |

# 副問い合わせ

SELECT文などのSQL文の中に更に別のSELECT文を入れることができます。この、SQL文の中に埋め込まれた問い合わせのことを副問い合わせといいます。副問い合わせを用いると、SELECT文の実行結果を用いて別のSQL文を実行することができます。

副問い合わせはwhere句の中で用いられることが一般的です。

|  |
| --- |
| > SQL文 where 条件式 SELECT文; |

たとえば、student\_idが2の学生が所属しているクラブ名を取得する場合は

|  |
| --- |
| > select club\_name from clubs where club\_id = (select club\_id from students where student\_id = 2); |

のように実行することができます。

副問い合わせには、結果を最大1行しか戻さないものと複数行戻すものがあります。

メインのSQL文で以下のような演算子を用いたwhere句を記述する場合は、副問い合わせは1行だけを返す必要があります。

|  |  |
| --- | --- |
| = | 等しい |
| > | より大きい |
| < | より小さい |
| >= | 以上 |
| <= | 以下 |
| <> | 等しくない |

メインのSQL文で以下のような演算子を用いたwhere句を記述する場合は、副問い合わせは複数行を返すものでも問題ありません。

|  |  |
| --- | --- |
| IN | 副問い合わせの結果のいずれかと等しい |
| NOT IN | 副問い合わせの結果全てと等しくない(<>ALLと同じ） |
| ANY | 副問い合わせの結果のいずれか１つとの間で条件を満たす |
| ALL | 副問い合わせの結果の全てとの間で条件を満たす |

たとえば、「3年生以上の学生が所属しているクラブ」を取得する場合は

|  |
| --- |
| > select club\_name from clubs where club\_id in (select club\_id from students where grade >= 3); |

のように実行します。ここで、inの代わりにnot inを使うと、3年生の誰も所属していないクラブが取得できます。

anyは副問い合わせが戻す値リストのうち、どれか1つとの間で条件を満たすことができるレコードの評価がtrueとなります。

|  |
| --- |
| > select name, height from students  where height <any  (select height from students grade = 2); |

のように実行すると、副問い合わせは165と168を返します。<anyは「いずれか1つの値より小さい」という条件になりますので、taroとkenの情報が取得できます。

allは副問い合わせが戻す値リストの全てに対して条件を満たすことができるレコードの評価がtrueとなります。

|  |
| --- |
| > select name, height from students  where height <all  (select height from students grade = 2); |

のように実行すると、副問い合わせは165と168を返します。<allは「全ての値より小さい」という条件になりますので、kenの情報のみが取得できます。

# トランザクション

## トランザクションとは

一般的な業務処理では、いくつかの処理をひとまとまりの処理として扱いたい場合があります。データベースでは、このような論理的な処理の単位をトランザクションと呼びます。

たとえば、銀行での振込を想定します。振込の処理は

1. 振込元の口座残高から振込金額を引く(出金)

2. 振込先の口座残高に振込金額を加える(入金)

の2つから構成されます。システム障害で1の処理は成功したがその後の2の処理が失敗した場合、振込元の残高は減り、振込先の残高は増えないという状況になります。こうした不整合はシステム的にあってはならない問題です。振込の処理では1と2は「両方成功」するか「両方失敗」しなければなりません。

トランザクション処理は複数の処理をひとまとまりとして扱うための仕組みです。振込の例では1と2の入金処理と出金処理を1つのトランザクションとして扱います。

トランザクションは「ここからひとまとまりの処理である」ことをDBMSに通知してから始めます。そして、トランザクションの全ての処理が正常に終了したタイミングで、処理の確定を行います。この確定処理のことをコミットといいます。また、トランザクション処理の途中で処理が失敗した場合は、トランザクションの全ての処理を取り消します。この取消処理のことをロールバックといいます。このようにしてトランザクションのすべての処理が「すべて成功」か「すべて失敗」であることを保証できます。

振込処理の場合は出金処理を行う前に、トランザクションを開始します。そして出金と入金が両方共正常に終了したタイミングでコミットを行います。出金処理が成功したが、入金処理が失敗した場合は、まだトランザクションは仮登録の状態になっています。この仮登録の状態をもとに戻すためにロールバックを行います。

## MySQLでのトランザクション

MySQLにはストレージエンジンが複数用意されています。ストレージエンジンとは、データ操作の指示を受けて実際にデータの出し入れを行う部分のことです。MySQLの主なストレージエンジンにMyISAMとInnoDBがあります。MyISAMはデフォルトのエンジンですが、トランザクションには対応していません。InnoDBはトランザクションに対応しています。

MySQLではトランザクションを使用するにはInnoDBを使用する必要があります。使用するストレージエンジンを指定するには、テーブルを作成する時にengineオプションを指定します。

|  |
| --- |
| > create table テーブル名 (カラム定義) engine = InnoDB; |

## トランザクションの実行

トランザクション処理を開始するには、最初に開始を通知する命令であるbeginを実行します。

|  |
| --- |
| > begin; |

beginでトランザクションが開始されると、以降のSQLによる更新はcommit命令で明示的に確定されるまでは、データベースには反映されません。トランザクションを確定するには

|  |
| --- |
| > commit; |

と実行します。また、トランザクションを開始してからの処理を取り消す場合は

|  |
| --- |
| > rollback; |

と実行します。一度コミットした更新処理を取り消すことはできません。

|  |
| --- |
| > begin;  > update account set balance = balance - 100 where id = 10;  > update account set balance = balance + 100 where id = 20;  > commit; |

のように実行すると、2つのupdate処理がデータベースに反映されます。commitの代わりにrollbackを実行すると2つのupdate処理は取り消され、データベースの内容はトランザクション前の状態になります。

# 制約

データベースに格納するデータの整合性を維持するための重要な機能に「制約」があります。データベースには、入力可能なデータの条件を設定し、その条件に基づいてデータが正しいかどうかを検証する機能があります。これが制約です。

制約を利用すると、更新されるデータが正しいかどうかをデータベースが自動的にチェックし、整合性を維持することができます。

## NOT NULL制約

列の値としてNULL地を禁止するための制約です。NULLは「未定義」であることを表し、列になにも値が設定されていないことを示します。

MySQLでNOT NULL制約を設定するためには、テーブル作成時のカラム定義の中で指定します。

|  |
| --- |
| > create table テーブル名(カラム名1 データ型 NOT NULL, ....); |

この制約は、単一の列に対して定義され、ひとつのテーブルに複数設定することが可能です。

## 一意性制約

一意性制約は、カラムに重複した値を設定することを禁止するための制約です。

MySQLで一意性制約を設定するためには、テーブル作成時のカラム定義の中で指定します。

|  |
| --- |
| > create table テーブル名(列名 データ型 UNIQUE, ...); |

この制約は、単一の列、または複数の列の組み合わせに対して定義され、ひとつのテーブルに複数設定することが可能です。

## 主キー制約(Primary Key)

主キーは、テーブル内のレコードを一意に特定するためのカラムです。主キー制約が設定されたカラムには、重複する値を登録することはできません。またNULLも格納できません。

MySQLで主キー制約を設定するためには、テーブル作成時のカラム定義の中で指定します。

|  |
| --- |
| > create table テーブル名 (列名1 データ型 primary key, ... ); |

また複数のカラムを指定して主キー制約を設定することもできます。

|  |
| --- |
| > create table テーブル名(カラム定義..., primary key(カラム名1, カラム名2, ...)); |

この場合は、指定したカラムの組み合わせが重複したデータは格納できません。

主キー制約は、ひとつの表にひとつしか設定できません。また、主キー制約が指定された列または列の組み合わせに対しては、NOT NULL制約と一意性制約が自動的に付加されます。

## 外部キー制約

テーブルを結合する際に、他のテーブルの主キーを参照するキーのことを外部キーと呼びます。この対応関係を保つための制約が外部キー制約です。

MySQLで外部キー制約を設定するためには、テーブル作成時のカラム定義の中で指定します。

|  |
| --- |
| > create table テーブル名(カラム定義...,  foreign key (外部キー列名, ...) references 参照先テーブル名 (参照先列名, ... )); |

この時、作成されるテーブルを「子テーブル」、参照先テーブルを「親テーブル」と呼びます。

外部キー制約を設定することで、主キーと外部キーの関係にあるテーブルに対して更新を行う際に以下の様なチェックを行い、対応する主キーが存在しない外部キーができてしまうという不具合を防ぎます。

* 子テーブルの外部キーの値が、親テーブルの主キー値として存在するか
* 親テーブルのレコードを更新・削除する場合に、この主キー値を参照している外部キーが存在しないか

# 正規化

正規化とは、データを効率よく格納するためにテーブルをどのように分割するかを決めるための作業です。

## 非正規形

たとえば、以下の様な学生の成績一覧があったとします。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 工学部  1301023  田中一郎   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 科目コード | 科目名 | 満点 | 得点 | 担当教員コード | 担当教員名 | | MAT-01 | 基礎数学 | 150 | 80 | 1234 | 鈴木 | | SCI-01 | 力学 | 200 | 150 | 2345 | 山田 | | LAN-03 | 仏語 | 100 | 70 | 3456 | 高田 | | IT-02 | アルゴリズム | 80 | 35 | 2233 | 谷口 | |

この情報をこのままデータベースに格納しようとすると、

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学籍番号 | 氏名 | 学部 | 試験科目１ | | |
| 科目コード | 科目名 | 満点 |
| 1301023 | 田中一郎 | 工学部 | MAT-01 | 基礎数学 | 150 |
|  | | | 試験科目２ | | |
| 得点 | 担当教員コード | 担当教員名 | 科目名 | 満点 | 得点 |
| 80 | 1234 | 鈴木 | 力学 | 200 | 150 |

のような形になります。他の学生が同じ科目を受験した場合、このテーブルには同じ情報があちこちに保存されてしまいます。また、情報に更新が発生した場合も効率が悪くなります。例えば科目名に変更があった場合は、同一科目を受験している学生が複数いる場合はその人数分の修正が必要になります。また、一人の学生しか受験していない科目がある場合、その学生の受験データを削除すると、その科目の情報が無くなってしまいます。

このような不具合が発生しないように、リレーショナルデータベースではテーブルを適切に分割してデータのムダや矛盾をなくす作業を行います。これが正規化の作業です。

正規化の作業が行われていない、データの形式を非正規形と言います。

## 第1正規化

第1正規化で行う作業は

* テーブルに含まれる繰り返し項目をなくす
* 導出可能項目をなくす

です。

導出可能項目とは他の項目から計算などにより求めることが可能な項目です。例えば、「満点」と「得点」の項目があれば、正答率は導出可能です。検索効率などを考えて導出可能項目を残すこともありますが、削除することが一般的です。

繰返し項目とは、上述の例では試験科目の情報です。繰返し項目を持つと、同一テーブルに含まれる複数のレコードの長さが一定に保たれません。受験科目が多い学生のレコードは長く、受験科目の少ない学生のレコードは短くなってしまいます。受験科目の多い学生に合わせてテーブルを設計すると、受験科目が少ない学生のレコードには無駄な領域ができてしまいます。また、現在の最大受験科目数より多く受験する学生が将来出てくるかもしれません。そこで、このような繰り返し部分は別のテーブルとして分離してしまいます。

＜学生情報＞

|  |  |  |
| --- | --- | --- |
| 学籍番号 | 氏名 | 学部 |
| 130123 | 田中一郎 | 工学部 |

<受験科目>

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 学籍番号 | 科目コード | 科目名 | 満点 | 得点 | 担当教員コード | 担当教員名 |
| 130123 | MAT-01 | 基礎数学 | 150 | 80 | 1234 | 鈴木 |
| 130123 | SCI-01 | 力学 | 200 | 150 | 2345 | 山田 |

このように第1正規化を行うことによりできたデータ構造を「第一正規形」と呼びます。

また、分割してできた新しいテーブルには、科目情報だけではなく、学生情報テーブルの主キーである学籍番号をカラムとして持たせます。そして学籍番号と科目コードの2つの組み合わせで1つのレコードが特定できるようにします。

上記のようなテーブル構造は、以下のように表記することができます。

学生情報=学籍番号+氏名+学部

受験科目=学籍番号+科目コード+科目名+満点+得点+担当教員コード+担当教員名

## 第2正規化

第2正規化では第1正規形から部分関数従属性を取り除きます。

1つのテーブルにおいて、ある属性Xの値を決めた時、他の属性Yの値が一意に決まる時「YはXに関数従属である」と言います(X→Yと表記する)。例えば、学生情報テーブルにおいて主キーである「学籍番号」の値を決めると非キー属性である学生の氏名が決まります。この時、氏名は学籍番号に関数従属します。部分関数従属とはキーの一部に非キー属性が関数従属している状態のことをいいます。

学生情報テーブルの主キーはひとつのカラムでのみ構成されているので、部分関数従属はありません。受験科目テーブルは科目コードと学籍番号の2つのカラムから構成される連結キーを持ちます。

そして「科目名」と「満点」「担当教員コード」と「担当教員名」は科目コードのみに従属しますので、この部分を別テーブルに分離します。

学生情報=学籍番号+氏名+学部

科目=科目コード+科目名+満点+担当教員コード+担当教員名

受験科目=学籍番号+科目コード+得点

## 第3正規化

第3正規化では第2正規形から推移関数従属を取り除きます。推移関数従属とは、「Aが決まるとBが決まり、その結果Cが決まる」という関係で「A→B→C(ただしB→Aは不成立)」と表記します。ここで、「A→B」はキーから非キー属性の関数従属性を表し、「B→C」は非キー属性間の関数従属性を表しています。

科目テーブルのキーである「科目コード」が決まると他の非キー属性の値が一意に決まるという関数従属性を持っていますが、非キー属性の「担当教員名」は「担当教員コード」が決まれば一意に決まるという関数従属性を含んでいます。これを別テーブルに分離すると第3正規形が得られます。

学生情報=学籍番号+氏名+学部

科目=科目コード+科目名+満点+担当教員コード

受験科目=学籍番号+科目コード+得点

教員=担当教員コード+担当教員名

# 参考文献

* Alan Beaulieu 著, 株式会社クイープ 訳: 「初めてのSQL」オライリージャパン,2006
* [SQL]データベース | TECHSCORE

<http://www.techscore.com/tech/sql/index.html/>

# 変更履歴

|  |  |  |
| --- | --- | --- |
| **版数** | **変更内容** | **年月日** |
| 1.0 | 新規作成 | 2014/MAR/2 |