データベース演習

データベースの概要

そもそもデータって何?(1)

・情報の表現である。

・伝達、解釈または処理に適するように形式化されたもの。

・再度情報として解釈できるもの。

(日本工業規格 基本用語より引用)

そもそもデータって何?(2)

データの一例:

住所録

etc...

世の中には扱う情報にあわせて様々なデータ が存在しています。

顧客情報

商品情報

売上情報

データベースとは

様々なデータを管理する仕組みを

「データベース」と呼びます。

データベースとは

「データベース」は、

・登録(CREATE)操作 ・検索(READ) 操作 ・更新(UPDATE)操作 ・削除(DELETE)操作

ができます。

これらはデータベースの基本4操作と言われており、 まとめて「CRUD(クラッド)」と呼ばれます。

データベースとは

関係型DB(表形式)

階層型DB(階層形式)

ネットワーク型DB(網状形式)

ドキュメント型DB(キーと値で管理)

データベースの種類(1)

データベースの種類(1)

現在の主流:

関係型データベース

⇒企業向けアプリやWebサイトで多く利用されています。

ドキュメント型データベース

⇒最近、NoSQLと呼ばれる分野で利用が増えています。 購入商品の提案やTwitter投稿などのデータ分析でも 活用されています。

データベースの種類(2)

Q . 表で管理っていうけど、

関係型データベースと表計算(Excel等)は何が違うの?

A . それぞれ目的が違います。

データベース ⇒ データを貯め込む。

表計算 ⇒ 表を作る。

関係型データベース

関係型データベースについて

関係型データベース・・・

データを表形式で管理するデータベース

※以降、データベースは「関係型データベース」を指すものとして解説します。

データベースの構造

3つの要素から構成されています。

テーブル (表)・・・データを収容する場所

レコード (行)・・・一件分のデータ

フィールド(列)・・データを構成する目的

データベースの構造

データベースの構造

関係型データベースは、 複数のデータ同士の関係を増やしながら扱うことができます。

関係型データベースは、別名リレーショナルデータベース (Relational Database)とも呼ばれます。

データベース管理システム

データベース管理システムとは?

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

データベースを操作する為のソフトウェアです。 DBMSはデータベースを管理します。

ORACLE MySQL Microsoft Access Microsoft SQL Server IBM DB2 PostgreSQL

etc...

MySQLとは何か?

MySQL・・・

DBMS(データベース・マネジメント・システム)の一つ

リレーショナルデータベース(関係型データベース)に特化 している為、RDBMS(Relational Database Management System) の一つとも呼ばれます。

SQLについて

MySQL Structured Query Language DBMSへ指示を与えるための言語です。

ユーザー ⇔ SQL ⇔ DBMS ⇔ DB

ユーザーがSQLを使って命令すると、 DBMSはSQLを解釈してDB操作をおこないます。

SQLとは何か?

SQL文の例

select 名前 from 名簿;

フィールド名 テーブル名 →名簿の表から名前のデータを検索することができます。

SQLとは何か?

SQLの書き方を多く学ぶと、 いろいろなデータ操作ができるようになります!

SQLとは何か?

データベース操作

SQL言語を使った データベース操作

では、ここからSQL言語を使って、データベースの操作をおこなって ゆきましょう。

ここでは、RDBMSであるMySQLにログインして演習を進めてゆきます。

ログイン・ログアウト

ログイン

まず、コマンドプロンプトを起動します。 手順: windowsキー+Rキーで「ファイル名を指定して実行」画面を起動します。 cmdと入力してOKボタンもしくは改行キー(エンターキー)を押下すると、 コマンドプロンプトが起動します。

ログイン

次に、以下のコマンドを入力して実行します。 mysql –u root –p

解説: mysql ・・・ mysqlにログインする為のコマンドです。 -u ・・・ -をつけると、オプション指定(指定した方法でコマンドを実行)

することができます。 -uはUSER(ユーザー)オプションと呼ばれています。 -uの次にユーザー名を指定してログインすることができます。 root ・・・ root(ルート)は最高権限をもつユーザーです。

このユーザーを乗っ取られると、データベースの改ざんなどに つながる為、職場では通常、rootではなく、限定された権限を もつユーザーでログインします。 ただし、研修中は便宜上、rootを使って操作してゆきます。 -p ・・・ -pはPASSWORD(パスワード)オプションと呼ばれています。

指定されたユーザーがパスワードをもつ場合、必ず指定 します。

ログイン

以下の通り、パスワードの入力を求められますので、 mysql と入力してログインします。

※研修所のすべてのパソコンは、上記パスワードを統一して設定を

完了しています。(変更はしないでください。)

ログアウト

次にログアウトします。 MySQLにログインしている状態で、 exit と入力してログアウトします。

quit でもログアウトすることができます。(結果はexitと同じです。)

※MySQLからログアウト後は、コマンドプロンプトの入力待ち状態となります。

mysql>の状態か否かを目安に判断することができます。 ログアウトすると、mysql>の状態ではなくなります。

ログアウト

ここで、コマンドプロンプトを終了してみましょう。

コマンドプロンプトを終了する場合にも、 exit と入力すると終了することができます。

※ Xボタンで終了することも可能ですが、上記のコマンドを入力した方が

コンピュータには親切です。

演習

1.もう一度、コマンドプロンプトを起動してみましょう。

2.MySQLにログインしてみましょう。

データベースの操作

データベースの操作

MySQLは、以下のような仕組みとなっています。

show tables; (テーブル一覧を表示)

※ここではWindows環境を 前提として図説しています。

cmd (コマンドプロンプトを起動する)

mysql -u root -p コマンド

(ログイン)

プロンプト

exit (ログアウト)

show databases; (データベース一覧を表示)

use db2; (使用するDBを指定する) DB1 DB2

RDBMS(MySQL)

データベース一覧を表示する(show)

ここでは、すでにMySQLにログインしているものとして解説して ゆきます。

まず、使用しているRDBMS(MySQL)に作成されているデータ ベース一覧を表示してみます。

show databases;

※データベースへの命令(SQL)は、 必ず;(セミコロン)を最後に使用します。 ;(セミコロン)をSQLでは特に、デリミタ(区切り記号)といいます。 ※databasesは複数形(s)であることに注意してください。

データベース一覧を表示する(show)

※表示されるデータベース一覧は、使用されているパソコンによって異なります。

これは過去の受講生が作成されたデータベースなどが含まれている為ですので、 ここでは気にされなくても大丈夫です。

新しくデータベースを作成する(create)

早速、新しいデータベースを作成します。

create database sample;

と入力して、データベースを作成してみましょう。

※赤字はSQL文 ※青字はデータベース名

新しくデータベースを作成する(create)

このように、データベースを作成する際には、

create database データベース名;

※赤字はSQL文 ※青字はデータベース名

のように命令します。 作成されたか否かは、show databases;で確認することができます。

新しくデータベースを作成する(create)

データベースの削除(drop)

次に、作成したデータベースを削除してみます。

drop database sample;

と入力して、データベースを削除してみましょう。

※赤字はSQL文 ※青字はデータベース名

※データベースを削除すると、中のテーブルやデータは 全て消えてしまうので注意して下さい。

データベースの削除(drop)

このように、データベースを削除する際には、

drop database データベース名;

※赤字はSQL文 ※青字はデータベース名

のように命令します。 削除されたか否かは、show databases;で確認することができます。

データベースの削除(drop)

演習

1.データベースsampleを作成してみましょう。

2.データベースsampleが作成されたか確認してみましょう。

テーブルの操作

使用するデータベースを 決定する(use)

次に、使用するデータベースを決定してみます。

use sample;

と入力して、使用するデータベースを決定してみましょう。

※赤字はSQL文 ※青字はデータベース名

使用するデータベースを 決定する(use) このように、使用するデータベースを決定するには、

use データベース名;

※赤字はSQL文 ※青字はデータベース名

のように命令します。 決定手続きが完了すると、Database changedと表示されます。

※データベースを使用する際には、このように必ずデータベース を決定する必要がありますので、注意しましょう。

使用するデータベースを 決定する(use)

テーブル一覧を表示する(show)

次に、決定したデータベースに含まれるテーブル一覧を表示 してみます。

show tables;

と入力して、テーブル一覧を表示してみましょう。

テーブル一覧を表示する(show)

もし決定したデータベース内にテーブルが存在しない場合には、 Empty set と表示されます。

Emptyとは、空っぽという意味です。 つまり、Empty setは、何も設定されていないことを示します。

テーブル一覧を表示する(show)

もし決定したデータベース内にテーブルが存在する場合には、 以下のような形式で表示されます。 ※ここでは、参考として表示しています。

テーブルを作成する(create)

次に、決定したデータベースにテーブルを作成してみます。 ※ここではテーブルlessonを作成します。

create table lesson( id int, name varchar(100), point int, team enum('red','blue','green') );

と入力して、テーブルを作成してみましょう。

テーブルを作成する

テーブルを作成する このように、テーブルを作成するには、 create table テーブル名( フィールド1 データ型, フィールド2 データ型 ); のように入力します。 フィールドは、テーブル内に作成される目的を示します。 カラムや列とも呼ばれます。 データ型は、フィールド単位で指定されるデータの種類(数字や文字列)を 示します。代表的な例は、次のページを参照してください。

なお、SQL文が長くなる場合には、,(カンマ)などの区切りで改行しても構い ません。また、1行で入力しても構いません。 データベースは、;が入力されたタイミングでSQL文が完結したと判断します。 (Windowsでは、もし入力中に誤記などがあった場合、Ctrlキー+Cキーで入 力待ち状態に戻すことができます。ただし、取りやめたSQL文は、再度 入力が必要となります。)

データ型 SQLには、以下のようなデータ型があります。 データ型とは、プログラミング言語(ここではSQL)が扱うデータの種類(数字 や文字など)を分類し、データの性質を指定する為の仕組みです。 代表的なものには、以下のようなものがあります。

int型 double型

整数を扱うことができる型 浮動小数点数を扱うことができる型

varchar型

文字列を扱うことができる型 text型

大量の文字を扱うことができる型 enum型

指定した値(列挙した値)のみ扱うことができる型

等々。MySQLのデータ型は、体系化すると主に数値型、日時型、文字列型 に分類することができます。 今後のサイト構築の際には、日付や時間を扱うdate型、datetime型や、 金額(特に外貨を含む)を扱うfloat型、decimal型なども登場します。 次のページ以降、それぞれを一覧にしています。 今後、少しずつ使えるデータ型を増やしてゆきましょう。

データ型 数値型

データ型 日時型

※1. 「YEAR(2)」は、5.5.27の時点で廃止されました。それ以降のバージョン で「YEAR(2)」を指定するとwarningが発生し、自動的に「YEAR(4)」になりま す。ちなみに「YEAR(1)」でも「YEAR(4)」でも「YEAR」でもすべて「YEAR(4)」 になります。

※2. MySQL 5.6.4以降では、TIME, DATETIME, TIMESTAMPで最大6桁 の「Fractional Seconds」つまり「秒の小数部」を付加できます。指定は 「DATETIME(6)」などとします。「DATETIME」「DATETIME(0)」はどちらも 「DATETIME」として定義されます。

データ型 文字列型

データ型 このように、データベースには多くのデータ型が準備されています。 なお、データベースで浮動小数点数を扱う際には、以下のような点に留意が 必要です。

データ型による値の変化(浮動小数点数)

例えば、金額(特に外貨を含む)等を扱う際、float型、decimal型がよく利用さ れます。 これは、米国通貨であれば、ドル、セントという単位を浮動小数点数で扱う必 要がある為です。 データベースでは、全てのレコードに同じ値を入力した場合、データ型によっ て値が変化する場合があります。

これを次ページにまとめます。

データ型 データ型による値の変化(浮動小数点数)

図のように、データ型によって値が変化する場合があります。 データ型を選択する際のポイントを次ページにまとめます。

データ型 データ型による値の変化(浮動小数点数)

・小数部を含んで6桁までであれば、float型を使ってよい。 ・小数点以下桁数を揃えて正確に扱う場合には、decimal型またはdouble型

を使う(例:緯度経度情報など)。 ・値をそのまま登録したい場合には、varchar型を使う。

なお、float型、double型、decimal型それぞれが表現可能なサイズは以下の 通りです。 float型 ・・・4バイト double型 ・・・8バイト decimal型・・・65桁

※float型とdouble型を使った値はコンピュータの浮動小数点数演算の近似 計算に依存しますが、decimal型は少数を正確に格納することができます。

データ型 decimal型

浮動小数点数を扱うデータ型のうち、decimal型は特に「パック無し浮動小数 点」と呼ばれています。 ここではdecimal型の記述方法についてまとめます。

decimal型は以下のようなルールにて記述をおこないます。

decimal(M,D) ここでは、 M・・・扱う少数の最大桁数 D・・・小数点以下の桁数 を示します。

つまり、decimal(10,3)とした場合には、全体で10桁、うち小数点以下は3桁 という解釈になります。

データ型 geometory型

MySQLでは、バージョン5.0以降、位置情報(緯度、経度)を扱えるようになり ました。ここでは位置情報を扱うgeometory型についてまとめます。 (geometory型は特殊なDB作成が必要なので、今回のgeometory型の範囲は 作業を割愛して構いません。) geometory型は以下のような使い方が可能です。

まず、以下のようなテーブルを作成したとします。

create table spot (

spot\_id int not null, latlon geometory not null, primary key (point\_id), spatial key (latlon) );

位置情報はgeometory型で宣言すると扱うことができ ます。 また、spatial keyを使うと、()内のレコードについて、あ る地点の近くのデータを取得するというSQLを記述す ることができるようになります。

データ型 geometory型

次に先程作成したspotテーブルに値を登録します。 geometory型を使用する場合、以下のような記述をおこないます。

insert into spot (spot\_id, latlon) values (1,GeomFromText('POINT(137.10 35.20)'));

GeomFormTextは、MySQLに定義された関数で、文字列表現からGeometory型で 値を読み取る為の仕組みです。

POINTは、POINT(経度 緯度)の形式で記述します。 かなら経度を最初に記述し、緯度を次に記述します。 また、経度と緯度のあいだは、カンマ(,)ではなく、スペースを使用します。

データ型 geometory型

次にspotテーブルの値を検索します。 以下のような検索をおこなってみます。

select \* from spot;

geometory型はそのまま検索すると、文字化けして表示されます。 その為、MySQLに定義された関数を使って検索をおこないます。 検索方法を、次のページにまとめます。

データ型 geometory型

あらためてspotテーブルの値を検索します。 今度はMySQLに定義された関数を使って検索をおこなってみます。

select spot\_id, X(latlon), Y(latlon), ASTEXT(latlon) from spot;

X、Y、ASTEXTはそれぞれMySQLに定義された関数です。 Xは経度、Yは緯度、ASTEXTは文字列表現をそれぞれ表示します。

なお、MySQLでは他にも定義された関数があります。 geometory型では、2点間の距離を計測する為の関数や方法なども準備されていま す(MBRContains、LINESTRING)。 また、多角形のエリアを表現するPOLYGON関数などもあります。

データ型 geometory型

次にspotテーブルの値を検索します。 geometory型を使用する場合、以下のような記述をおこないます。

insert into spot (spot\_id, latlon) values (1,GeomFromText('POINT(137.10 35.20)'));

GeomFormTextは、MySQLに定義された関数で、文字列表現からGeometory型で値を読み 取る為の仕組みです。

POINTは、POINT(経度 緯度)の形式で記述します。 必ず経度を最初に記述し、緯度を次に記述します。 また、経度と緯度のあいだは、カンマ(,)ではなく、スペースを使用します。

テーブル定義を確認する(desc) ここまでデータ型についてまとめました。

作成したテーブルは、テーブル定義としてその構造を確認できます。 では、テーブル定義を確認してみます。

desc lesson;

と入力して、テーブル定義を確認してみましょう。

※なお、descはdescribeの省略命令です。 describe lesson; でも同じ結果を表示することができます。

テーブル定義を確認する(desc)

テーブル定義を確認する(desc) このようにテーブル定義を確認するには、

desc テーブル名; もしくは describe テーブル名;

のように入力します。

※入力方法は、上記のいずれでも構いません。

テーブルの削除(drop)

次に、作成したデータベースを削除してみます。

drop table lesson;

と入力して、テーブルを削除してみましょう。

また、削除ができたら、 show tables; と入力して、lessonテーブルが削除されたことを確認してみましょう。

※ 一旦削除命令をすると、やり直しはできません。

ご注意ください。

テーブルの削除(drop)

このようにテーブルを削除するには、

drop テーブル名;

のように入力します。

フィールドのオプション設定

テーブルを作成する際には、フィールドにオプションを付与することで 役割や制限を設けることができます。 代表的なオプションには、以下のようなものがあります。 これらについて次のページ以降、解説してゆきます。

必須入力 主キー 自動連番 ユニークキー 索引付与 キー

not null primary key auto\_increment unique index key

フィールドのオプション設定

not null 必須入力

フィールドに対して、not nullを定義すると、指定されたフィールドに、NULLを 登録することができなくなります。 つまり、必ず何らかの値を登録しなければなりません。

また、フィールドに対してnullを定義することもできます。 この場合、指定されたフィールドにはNULLを登録することができるようになりま す。(MySQLではnot nullやnull指定を明記していない場合、自動的にnull指定 がなされます。つまり、nullもしくはnot nullを指定していない場合には、NULLを 登録することができる状態となっています。)

フィールドのオプション設定

primary key 主キー

データを扱っていると、社員番号を重複したくない、商品番号を重複したくない、 同じメールアドレスの登録を避けたい、など、制限を設けたいフィールドが発生 します。

primary keyを定義すると、重複禁止のインデックス(index)を作成することがで きます。 ※インデックス(index)は、後述しますが検索を早める為の仕組みです。

また、primary keyを定義するとnot null制約が自動的に付与されます。 つまり、primary keyを定義したフィールドは、必ず値を登録しなければいけま せん。

※primary keyはテーブル毎にidを定義し、テーブルの目的に応じたid同士を 紐付ける場合にひろく利用されます。(商品テーブルにおける商品IDなど)

フィールドのオプション設定

primary key 主キー

なお、primary keyは複数のフィールドに対して、ひとつのprimary keyを定義す ることができます。

これを複合プライマリーキー(multi primary key)と言います。

例)

create table test(

gakunen int not null, kumi varchar(10) not null, name varchar(20), primary key(gakunen, kumi) );

テーブルを作成後、テーブル定義を確認すると、次のページのように表示され ます。

このように、複数のフィールドに対してprimary keyが設定されていることがわか ります。

なお、複合プライマリーキーは、データ管理が複雑になりやすい仕組みなので、 あまり使用しないことをお勧めします。

フィールドのオプション設定

primary key 主キー

フィールドのオプション設定

auto\_increment 自動連番

auto\_incrementが定義されたフィールドには、自動的に連番値が登録されてゆ きます。auto\_incrementを定義するフィールドは、必ず整数のデータ型を宣言し ます。

なお、auto\_incrementが設定されたフィールドは、登録する値がNULLもしくは0 だった場合には、既にレコード登録されている最大値に1を加えた値が登録さ れます。

フィールドのオプション設定

unique ユニークキー

uniqueが定義されたフィールドは、重複した値を登録できなくなります。 重複した値が登録できない仕組みはprimary keyと同じですが、uniqueと primary keyの違いは、uniqueの場合NULL登録ができる点です。

フィールドにuniqueを定義した場合、そのレコードにはindexが自動設定されま す。これをユニークインデックス(unique index)と言います。

フィールドのオプション設定

index 索引付与 データベースでは通常、テーブル内の全てのデータを1件ずつ順番にチェック し、条件に合うデータのみ抽出する、という動作が行われます。 これを「フルテーブルスキャン」といいます。

この場合、データベースに対する負荷が大きい為、データを保存した際、「指定 したレコードの値」と、「そのデータの保存位置情報」を本の索引のような構造で 保存しておき、目的のデータを抽出する際に、その索引を使って検索の効率化 を図る仕組みが用意されました。 この時作成される索引の仕組みをindexといいます。

フィールドのオプション設定

index 索引付与 なお、indexが定義されたフィールドは、以下の場合、検索速度の向上が効果でき ます。 1.テーブル内のデータ量が多く、少量のレコードを検索したい場合。 2.WHERE句の条件、結合の条件、ORDER BY句の条件として頻繁に利用する

可能性が高い場合。 3.NULL値が多いデータから、NULL値以外を検索したい場合。

また、以下のような場合にはindexを定義すべきではありません。 1.表の規模が小さいか、表の大部分のレコードを検索する場合 2.WHERE句等の条件としてあまり使用されない場合 3.列の値が頻繁に登録、更新、削除される場合

※これはindexが定義されたフィールドについて、データに変更が発生すると、

indexが都度、再作成される為です。

フィールドのオプション設定

key キー keyには定義方法によって2つの使い方があります。 1.primary key(主キー)として使う場合 2.index(索引付与)として使う場合

1.primary key(主キー)として使う場合

テーブルを作成する際、フィールド宣言と同じタイミングでkeyを宣言すると そのフィールドはprimary keyとして定義されます。 つまり、primary keyはkeyと省略することが可能です。 例)

CREATE TABLE test(

id int key ); ただし、混乱を防ぐ、可読性を向上する点で、通常はprimary keyとする方が 親切です。

フィールドのオプション設定

key キー 2.index(索引付与)として使う場合

テーブルを作成する際、各フィールド宣言のあとで、keyを宣言すると、()内で 宣言されたフィールドに対してindexが定義されます。 例) create table test( id int not null, key (id) );

また、key キー名(フィールド名)の形式で宣言すると、指定されたフィールド名に 対してindexが定義され、このindex名としてkey名が更に定義されます。 例) key key\_name(point)

例えば上記では、pointに対してkey\_nameという名前でindexが定義されます。

フィールドのオプション設定

先程lessonテーブルを削除しましたので、 次にフィールドのオプション設定を使ってlessonテーブルを作成してみます。

create table lesson( id int not null primary key auto\_increment, name varchar(100) unique, point int, team enum(‘red’,’blue’,’green’), index(name), key point(point) );

と入力して、新しくlessonテーブルを作成してみましょう。

フィールドのオプション設定

フィールドのオプション設定

★enumについて

enumを使用すると、()内に記述したデータのみ扱うことができます。 つまり、()内に記述されていないデータを登録することはできません。

create table lesson2(

team enum('red','blue','green') );

★デフォルト値を追加する

defaultを使用すると、データを登録する際に該当フィールドの値を指定 しなかった場合、自動的にdefaultで指定された値が登録されます。

create table lesson3(

team enum('red','blue','green') default 'red' );

フィールドのオプション設定

lessonテーブルが作成できましたら、desc(もしくはdescribe)を使って、テーブル 定義を確認してみます。

desc lesson;

と入力して、テーブル定義を確認してみましょう。

このように、フィールドのオプション設定を使った場合、テーブル定義にオプション 設定の内容が表示されます。

フィールドのオプション設定

データを登録する

データを登録する

それでは次に、作成したlessonテーブルにデータを登録してみます。

insert into lesson( id,name,point,team ) values( 1,'Tanaka',80,'red' );

と入力してデータを登録してみましょう。

データを登録する

データを登録する

このようにデータを登録するには、

insert into テーブル名( フィールド1,フィールド2,フィールド3... ) values( フィールド1のデータ,フィールド2のデータ, フィールド3のデータ... );

のように入力します。

データを登録する

また、テーブルに含まれるすべてのフィールドに対して、データを登録する 場合には、フィールド宣言を省略することができます。 具体的には、以下のような記述ができます。

insert into テーブル名 values( フィールド1のデータ,フィールド2のデータ, フィールド3のデータ... );

データを検索する

データを検索する

それでは次に、作成したlessonテーブルのデータを検索してみます。

select \* from lesson;

と入力して、lessonテーブルのデータを検索してみましょう。

データを検索する

データを検索する

このように、テーブルのデータを検索するには、

select \* from テーブル名;

のように入力します。

\*(アスタリスク)の記号は、指定したテーブルから全データを検索することを 意味します。

データを検索する

また、テーブルから指定したフィールドについてのデータを検索するには、

select 列名 from テーブル名;

のように入力します。

例えば、 select name, point from lesson; のように入力すると、nameとpointのデータのみ検索することができます。

データを検索する

データベースでは、以下のような検索ができます。 それぞれ入力して確認してみましょう。

★全ての列を表示 select \* from lesson;

★指定した列だけを表示 select point from lesson;

★結果を縦に表示 select \* from lesson \G (\Gオプションを付けた場合には、;(セミコロン)記号をつける必要はありません。

演習

1.これまで作成したテーブルにデータをたくさん登録してみましょう。

2.登録したデータを検索してみましょう。

条件付き検索

条件付き検索

SQLを使うと、特定の条件を満たす値について検索することができます。 これを条件付き検索といいます。

条件指定の方法はさまざまですが、以下のような形式となります。

select フィールド名 from テーブル名 where 条件;

※どのような条件で、どのテーブルから、どのフィールドのデータを取り出すか?

を指定します。

次のページ以降、いろいろな書き方をご紹介してゆきます。 順番に試しながら書き方を押さえてゆきましょう。

条件付き検索

★ point が 50 のデータ select \* from lesson where point = 50; ★ point が 80 以上のデータ select \* from lesson where point >= 80; ★ team が red でないデータ(!=) select \* from lesson where team != 'red'; ★ name の最初が Tで始まるデータ(like) select \* from lesson where name like 'T%'; ★name の最後が a で終わるデータ(like) select \* from lesson where name like '%a'; ★name の途中に ana が含まれているデータ(like) select \* from lesson where name like '%ana\_\_';

条件付き検索

★ point が 60 以上 かつ 80 以下のデータ(between) select \* from lesson where point between 60 and 80; ★ team が red , green なデータ(in) select \* from lesson where team in ('red','green') ; ★ point が 80 以上で team が blue のデータ(and) select \* from lesson where point >= 80 and team = 'blue'; ★ point が 80 以上 もしくは team が blue のデータ(or) select \* from lesson where point >= 80 or team = 'blue';

データの並び替え

データの並び替え

SQLを使うと、データの並び替えも可能です。 以下、いろいろな書き方をご紹介してゆきます。 順番に試しながら書き方を押さえてゆきましょう。

★昇順で並び替える(order by) select \* from lesson order by point;

★降順で並び替える(order by desc) select \* from lesson order by point desc;

★指定した件数だけ表示する(limit) select \* from lesson limit 3;

★途中から指定した件数だけ表示する(limit) select \* from lesson limit 2,2;

データの集計

データの集計

さらにSQLを使うと、データの集計も可能です。 以下、いろいろな書き方をご紹介してゆきます。 順番に試しながら書き方を押さえてゆきましょう。

★データの件数を表示する(count) select count(\*) from lesson; ★指定したカラムの最高値を表示する(max) select max(point) from lesson; ★指定したカラムの合計値を表示する(sum) select sum(point) from lesson; ★指定したカラムの平均を表示する(avg) select avg(point) from lesson; ★チームごとの平均を表示する(group by) select team, avg(point) from lesson group by team;

データの集計

★指定したカラムの文字列数を表示する。(length) select name, length(name) from lesson;

★指定したカラムの文字列を連結する。(concat) select concat (name,'(',team,')') from lesson;

データの更新・削除

データの更新・削除

ここでは、作成したlessonテーブルのデータを更新や削除してみます。

以下のように入力して、lessonテーブルのデータを更新や削除してみましょう。

★idが1のpointのデータを100に更新する。(update set) update lesson set point = 100 where id = 1;

★pointが60点未満のデータを削除する。(delete) delete from lesson where point < 60;

★指定したテーブルのデータを全て削除する。 (delete) delete from lesson; (※drop コマンドと違いテーブルそのものは削除されない)

データの更新・削除

このように、テーブルのデータを更新や削除するには、 データの更新 条件を満たすフィールドの値を更新する場合(単一のフィールド): update テーブル名 set フィールド名=値 where 条件句; 条件を満たすフィールドの値を更新する場合(複数のフィールド): update テーブル名 set フィールド名=値,フィールド名=値・・・ where 条件句; データの削除 条件を満たすフィールドの値を削除する場合: delete from テーブル名 where 条件句; テーブルに登録されたすべてのフィールドの値を削除する場合: delete from テーブル名;

のように入力します。

テーブル構造の変更

テーブル構造の変更

ここでは、作成したlessonテーブルのテーブル構造を変更してみます。 既に作成されたテーブルに対してテーブル構造の変更をおこなう場合には、 alterを使います。

以下のように入力して、lessonテーブルのテーブル構造を変更してみましょう。

★pointフィールドの後に電話番号フィールドを追加する。 alter table lesson add telnumber int(11) after point;

★電話番号用のフィールド名を変更する。 alter table lesson change telnumber cellphone\_number int (11); (ここではtelnumberを、cellphone\_numberに名称変更しています。)

※それぞれ実行後、descコマンドで結果を確認してみましょう。

外部キー制約

外部キー制約

SQLでは、特定のテーブルに登録できるデータを、他テーブルに登録されている データ以外は登録できなくなるよう定義することができます。 これを外部キー制約といいます。

次のページ以降、2つのテーブルを作成して、外部キー制約を定義してみます。

外部キー制約

指定した列に対して、他テーブルの列に登録された値以外は 入力出来ないようにする。

foreign key(自分の列名) references 他テーブル名(他テーブルの列名)

外部キー制約

まず、以下の内容でgoods(商品)テーブルを作成しましょう。

create table goods( id int, name varchar(10), index(name) );

外部キー制約

次に、sales(購入)テーブルを作成します。 また、ここでは商品名に外部キー制約を付与します。

create table sales( id int, name varchar(10), day date, index(name), foreign key(name) references goods(name) );

ここではforign keyにて、salesテーブルのうち、外部キー制約をおこないたい フィールド名を宣言しています。また、referencesにて、制約をおこなう為のデー タをもつテーブルとフィールド名を宣言しています。 次のような形式となります。 foreign key(name) references goods(name)

外部キー制約するフィールド名 外部キー制約するフィールドのデータをもつテーブル名(同じくデータをもつフィールド名)

この時、テーブル sales の name に 登録できるデータは他テーブル goods のname に登録されているデータ以外は登 録できなくなります。

※ 例えば、goodsのname に ‘aaa’ というデータだけが挿入されていた場合、外部 キー制約を設定したselesのnameに’bbb’は登録できません。

外部キー制約

SQLファイルを コマンドで読み込む

SQLファイルをコマンドで読み込む

企業では、数多くのデータベースを扱う機会が多々あります。 また、開発向け、試験向け、本番向けなど、いくつかのパソコン(サーバー) に同じ仕組みをもつデータベースを準備する機会も多々あります。

この時、それぞれのパソコン(サーバー)に、同じSQLを命令することになりま すが、これを手入力してゆくのは、大変非効率であり、ミスもおこりやすくなり ます。 また、パソコン(サーバー)によっては、入力ミスによって、構造が異なるデー タベースが出来てしまう可能性が高まってしまいます。

その為、データベースでは、SQLを記述したファイルを準備して、これをコマ ンドでデータベースに登録する仕組みが準備されています。 これによって、いくつものパソコン(サーバー)に対して、同じファイルを使い まわすことができるようになります。また、作業の効率化やミスの防止にもつ ながります。

SQLファイルをコマンドで読み込む

ログアウト後、

notepad command.sql とコマンドプロンプトに入力すると、メモ帳が開きます。 タイトル部分にはcommand.sqlと表示されているか確認して下さい。 メモ帳にて開いたファイルに以下のSQLを入力して下さい。

drop table if exists gaibu; create table gaibu(id int,name varchar(255)); insert into gaibu(id,name) value (1,'MySQL');

入力後、ファイルを保存し、ファイルを閉じます。 次のページでは、drop table if exists gaibu;について解説します。

SQLファイルをコマンドで読み込む

drop table if exists gaibu; 今回作成したSQLファイル内で、上記のような宣言が登場しました。 SQLでは、 特定のデータベースやテーブルが存在する時には、これを削除したい。 もしくは 特定のデータベースやテーブルが存在しない時には、これを作成したい。 のような宣言をおこなうことができます。 例) drop database if exists test\_database; drop table if exists test\_table; create database if not exists test\_database; create table if not exists test\_table(id int,name varchar(255));

SQLファイルをコマンドで読み込む

なお、例)に記載したそれぞれの意味は、以下の通りです。

drop database if exists test\_database; もし、test\_databaseが存在していたら、これを削除しなさい。

drop table if exists test\_table; もし、test\_tableが存在していたら、これを削除しなさい。

create database if not exists test\_database; もし、test\_databaseが存在しなければ、これを作成しなさい。

create table if not exists test\_table(id int,name varchar(255)); もし、test\_tableが存在しなければ、(id int,name varchar(255))の内容でこれを作成しなさい。

SQLファイルをコマンドで読み込む

では、先程作成したSQLファイルを使って、コマンドで読み込みをおこないます。 以下のコマンドを入力してみましょう。

mysql –u root –p sample < command.sql ※ 「<」の向きを間違えやすいのでご注意ください。

エラーメッセージが表示されなければ、読み込みは完了しています。

SQLファイルをコマンドで読み込む

このように、以下のように入力するとSQLファイルをコマンドで読み込むことが できます。 mysql –u ユーザ名 –p データベース名 < 外部ファイル名

では次に、MySQLにログインして、テーブルが作成されているか確認してみます。

mysql -u root -p use sample; show tables; select \* from gaibu; 上記のような流れで作成されているか確認してみましょう。

SQLファイルをコマンドで読み込む

バックアップ、復元

バックアップ、復元

作成したデータベースは、バックアップしたり、復元することができます。 データベースをバックアップすると、この命令をおこなった時の場所(フォルダ) 内に、新しくSQLファイルが生成されます。

まず、データベースからログアウトしましょう。 ログイン中の場合、 exit と入力して、データベースからログアウトしてください。

バックアップ、復元

では、次にバックアップをおこなってみます。

mysqldump -u root -p sample > sampledump.sql と入力してみましょう。 ※ 「>」の向きを間違えやすいのでご注意ください。

作成したデータベースは、バックアップしたり、復元することができます。 データベースをバックアップすると、この命令をおこなった時の場所(フォルダ) 内に、新しくSQLファイルが生成されます。

すると、上記の場合、C:\Users\internousの中にSQLファイルが生成されます。 次に、生成されたか否か、 dir と入力すると確認することができます。 ※dirはコマンドプロンプトの命令です。いま現在のフォルダの中にある ファイルやフォルダを表示することができます。

バックアップ、復元

このように、mysqldumpコマンドを使うと、データベースのバックアップをおこなう ことができます。

バックアップでは、以下のような形式で、ユーザ名、データベース名、出力したい SQLファイル名(これがバックアップファイルとなります)を指定します。 mysqldump –u root –p データベース名 > ファイル名.sql

バックアップの際には、必ずデータベースをログアウトしておくことを忘れないよう にしましょう。 実行後、コマンドを使った際のフォルダにSQLファイルが生成されます。 生成されたSQLファイルは、dirコマンドで確認することができます。

バックアップ、復元

では、次にバックアップしたSQLファイルを使って、データベースやテーブル を復元してみます。

まず、バックアップしたデータベースやテーブルを復元する前に、既に存在し ているデータベース内のテーブルを削除してみます。 データベースにログインして、以下のようにsampleデータベース内に存在する lessonテーブルとgaibuテーブルを削除してみましょう。

mysql -u root -p use sample; drop table lesson; drop table gaibu; 次に show tables; を使って、テーブルが削除されたことを確認しておきましょう。

では、先ほど生成したSQLファイルを使って、復元をおこなってみます。

復元するには、SQLファイルの読み込みと同じ方法を使います。 まず、データベースからログアウトして、以下のコマンドを入力してみましょう。

次に、データベースにログインして、sampleデータベースにlessonテーブルと gaibuテーブルが復元されているか確認してみましょう。

★ mysql –u root –p データベース名 < ファイル名.sql

バックアップ、復元

バックアップ、復元

このように復元するには、SQLファイルの読み込みと同じ方法を使います。 復元の際には、必ずデータベースをログアウトしておくことを忘れないようにしま しょう。

復元は、SQLファイルの読み込みと同じ方法なので、以下のような形式となります。 mysql –u root –p データベース名 < ファイル名.sql

なお、SQLファイル内に use データベース名; の記述があれば、以下のような形式で復元することもできます。 mysql –u root –p < ファイル名.sql

これは、バックアップに対する復元だけではなく、普段おこなうSQLファイルの読 み込みについても同じです。

では、最後に、データベースにログインして、復元されていることを確認しておき ましょう。

バックアップ、復元

パスワード設定

パスワード設定 セキュリティ担保の観点から、企業では定期的にパスワード変更を実施するよう 推進されています。 パスワードを変更するには、以下のような形式で入力します。

set password for root@localhost = password('mysql');

ここでは、自身のパソコンという意味を示すlocalhost(ローカルホスト)に対して rootユーザーのパスワードをmysqlに変更する記述となっています。

※ @の後はパソコンの名前であるホスト名やパソコンを特定するためにつけられ

たIPアドレス(192.168.1.255のような形式で、パソコンや通信機器で使われる 電話番号のようなもの)を指定することができます。

研修所では、rootユーザーのパスワードは必ずmysqlとしています。 他のパスワードは使わないようにしてください。

ユーザー作成

ユーザー作成

MySQLをインストールした直後では、「root(ルート)」ユーザーのみ作成され ています。

「root」ユーザーは、データベースにおける最高権限をもつユーザーです。 つまり、rootユーザーはデータベースの削除など、全ての操作が実行できて しまう権限を持っています。

このユーザーが乗っ取られると、データの改ざんや削除が簡単に実現され てしまう為、企業では通常、作業者向けアカウントが発行されます。

作業者が自由に削除操作などをおこなえない様、予め制限されたユーザー を使用するなど、企業では目的・用途に応じたユーザーを作成して、操作を おこなってゆきます。

※root ユーザー:全ての操作を実行できるスーパーユーザー

ユーザ作成

ユーザーを作成するには、以下のような命令をおこないます。

grant all on sample.\* to dbuser@localhost identified by 'mysql';

(意味) ホスト名:localhostに、ユーザ名:dbuserを作成し、データベース名:sampleの 全てのテーブルに対して、GRANT OPTION(グラントオプション)以外の権限 all(全操作権限)を付与します。

※GRANT OPTION権限とは、他のユーザーに権限を付与する為の権限です。 例えばgrant宣言の末尾にwith grant optionを記述すると、宣言された操作権 限について、宣言されたユーザーに付与することができます。 ここではパスワードはmysqlとしています。

ユーザ作成

作成したユーザでログインできる事を確認して完了。

テーブル結合

テーブル結合 テーブル結合とは・・・ データベースではテーブル内の情報を結合してひとつにまとめることが出来 ます。 これを「テーブル結合」といいます。 テーブル結合には、3種類の結合方法があります。 結合方法によって、出力結果は変化します。 3種類のテーブル結合

結合の種類 SQL

内部結合 INNER JOIN

外部結合

左外部結合 LEFT OUTER JOIN

右外部結合 RIGHT OUTER JOIN

完全外部結合 FULL OUTER JOIN

交差結合(クロス結合) CROSS JOIN

テーブル結合

「内部結合」と「外部結合」は、SQLの一部を省略することができます。

また職場では、省略で可否や書き方を「開発標準規約」で統一しています。

システム開発では、ルールブックが定められており、これに従って開発をおこないます。 (具体的には、コーディング規約、命名規則、設計書フォーマット、開発ワークフローなどがあります。)

結合の種類 SQL(省略した場合)

内部結合 JOIN

外部結合

左外部結合 LEFT JOIN

右外部結合 RIGHT JOIN

完全外部結合 FULL JOIN

交差結合(クロス結合) CROSS JOIN

内部結合

内部結合とは

内部結合は、結合するテーブルの両方にて一致するレコードのみ表示することが できます。 例)ここでは、user\_tableのuser\_idとteam\_tableのidを紐付けテーブル結合を

user\_table

おこないます。

user\_id name

1 tanaka

2 yamada

3 kimura

team\_table

id team

1 red

2 blue

3 green

テーブル結合

内部結合

name team

tanaka red

yamada blue

kimura green

内部結合

内部結合(INNER JOIN)の使い方

※内部結合では、以下のような書き方が出来ます。

内部結合の使い方(その1)

SELECT テーブルA.カラム1,テーブルB.カラム1 FROM テーブルA INNER JOIN テーブルB ON テーブルA.カラム2 = テーブルB.カラム2;

内部結合の使い方(その2)

SELECT (取得するカラム) FROM テーブルA INNER JOIN テーブルB ON 結合条件;

データベースを作成しよう!(1)

データベース「sample」作成しましょう。

データベース「sample」を作成

テーブル「sample」に入る

内部結合

内部結合

データベースを作成しよう!(2)

内部結合

userテーブルを作成しよう!(1)

「user」テーブルを作成しよう。

以下の内容を参考に「user」テーブルを作成しましょう。

カラム名 データ型 KEY NULL EXTRA

user\_id int primary key not null auto\_increment

name varchar(100)

内部結合

userテーブルを作成しよう!(2)

内部結合

テーブルに値を入れよう!(1)

次は「user」テーブルに値を入れましょう。

\*user\_id「3」は省いてます。

user\_id name

1 tanaka

2 yamada

4 kimura

5 suzuki

テーブルに値を入れよう!(2)

内部結合

内部結合

teamテーブルを作成しよう!(1)

では次に「team」テーブルを作成しましょう。

以下の内容を参考に「team」テーブルを作成しましょう。

カラム名 データ型 KEY NULL EXTRA

id int

primary key

not null auto\_increment

team enum('red','blue','green')

内部結合

teamテーブルを作成しよう!(2)

内部結合

テーブルに値を入れよう!(1)

次は「team」テーブルに値を入れましょう。

\*あえてid「4」は省いてます。

id team

1 red

2 green

3 blue

5 green

6 red

テーブルに値を入れよう!(2)

内部結合

内部結合しよう!(1)

内部結合

結合

「userテーブル」と「teamテーブル」を内部結合します。 内部結合すると一致するレコードのみ表示します。

内部結合

内部結合しよう!(2)

select user.name,team.team from user inner join team on user.user\_id = team.id;

内部結合

内部結合しよう!(3)

「userテーブル」の「user\_id」と「teamテーブル」の「id」が一致するもののみ表示されます。

内部結合

内部結合しよう!(4)

select user.name,team.team

from user inner join team

on user.user\_id = team.id ;

(SELECT文:説明)

表示するカラムを入力 「user」テーブルの中の「name」と「team」テーブルの中の「team」を選択している

内部結合の基準となるテーブルは今回「user」テーブルとします。 そして「INNER JOIN」で結合するテーブルは「team」とします。

「ON」の後に結合条件を入力 「userテーブル」の「user\_id」と「teamテーブル」の「id」と紐付けます。

内部結合しよう!(5)

各テーブルのカラムを全て取得し内部結合してみましょう。

内部結合

内部結合は結合条件が一致するもののみ表示されます。 どちらかのテーブルのみ値が登録されている場合は、表示されません。

結合条件

「user\_id」と「id」が一致するもののみ表示

外部結合

外部結合

外部結合の種類

種類 正式名 略称名

左外部結合 LEFT OUTER JOIN LEFT JOIN

右外部結合 RIGHT OUTER JOIN RIGHT JOIN 完全外部結合 FULL OUTER JOIN FULL JOIN

内部結合では、結合条件と一致したデータのみ表示されました。 外部結合では、一致したデータに加えて、どちらかのテーブルのみ 存在するデータについても表示されます。

左外部結合

左外部結合とは

user table

user\_id name 1 tanaka 2 yamada 4 kimura 5 suzuki

team\_table 表示

id team 1 red 2 green 3 blue 5 green 6 red

左外部結合

user\_id name id team

1 tanaka 1 red 2 yamada 2 blue

4 kimura NULL NULL

5 suzuki 5 green

左外部結合では、FROMのあとに書かれたテーブル(左) のみ値が登録されている場合でも、表示されます。

左外部結合

左外部結合には LEFT OUTER JOIN を使おう!

SQL文としては

SELECT テーブルA.カラム1,テーブルB.カラム1 FROM テーブルA LEFT OUTER JOIN テーブルB ON テーブルA.カラム2 = テーブルB.カラム2;

簡略化すると

SELECT (取得するカラム) FROM テーブルA LEFT OUTER JOIN テーブルB ON 結合条件;

左外部結合しよう!(1)

ここでは、内部結合にて作成した「userテーブル」と「teamテーブル」を使用します。

左外部結合

左外部結合を使うと、内部結合の時と表示結果が変わりました。

左外部結合

左外部結合しよう!(2)

内部結合では表示されなかったレコードが 表示されました。

左外部結合

左外部結合は、FROMの後ろに宣言されたテーブルに従って表示されます。 今回の場合、「userテーブル」にある値に沿って「teamテーブル」の値が表示されます。

左外部結合について

左外部結合について

SELECT\* FROM テーブルA LEFT OUTER JOIN テーブルB

左 右

左には親テーブルを入力して左外部結合をおこないます。 親テーブルには、必ず主キー(primary key)が宣言されていなけれ ばなりません。

左外部結合

・ここでは「userテーブル」が左に表示されて います(主キーが宣言されているテーブル)。

・「teamテーブル」にid 4の登録はされていま せんが、左外部結合では 親テーブル のカラムに合わせて表示されます。

右外部結合

右外部結合とは

user table

user\_id name 1 tanaka 2 yamada 4 kimura 5 suzuki

team table

id team 1 red 2 green 3 blue 5 green 6 red

右外部結合

user\_id name id team

1 tanaka 1 red 2 yamada 2 blue

5 suzuki 5 green

NULL NULL 3 blue

NULL NULL 6 red

表示

右外部結合では、右のテーブルに従って値を取得 します。 左のテーブルに存在しないレコードも表示されます。

右外部結合

右外部結合には RIGHT OUTER JOIN を使おう!

SQL文としては

SELECT テーブルA.カラム1,テーブルB.カラム1 FROM テーブルA RIGHT OUTER JOIN テーブルB ON テーブルA.カラム2 = テーブルB.カラム2;

簡略化すると

SELECT (取得するカラム) FROM テーブルA RIGHT OUTER JOIN テーブルB ON 結合条件;

右外部結合しよう!(1)

内部結合で作った「userテーブル」と「teamテーブル」を使用していきます。

右外部結合

右外部結合

右外部結合しよう!(2)

左外部結合の時と表示が変わったはずです。

「teamテーブル」に基いて表示された

右外部結合について

SELECT\* FROM テーブルA RIGHT OUTER JOIN テーブルB

左(親) 右(子)

左外部結合の時と同じく、左には親テーブルを入力 右外部結合の際は、子テーブルに合わせて値が出力される

・「userテーブル」にはid 3,6は ないが、子に合わせて表示される

右外部結合

完全外部結合

完全外部結合とは

user table

user\_id name 1 tanaka 2 yamada 4 kimura 5 suzuki

team table

id team 1 red 2 green 3 blue 5 green 6 red

完全外部結合

表示

user\_id name id team

1 tanaka 1 red 2 yamada 2 blue

4 kimura

NUL L

NULL

5 suzuki 5 green NULL NULL 3 blue NULL NULL 6 red

両方のテーブルの全データを取得し、どちら かのテーブルにしかない値でも全て表示する。

完全外部結合

完全外部結合には FULL OUTER JOIN を使おう!

SQL文としては

SELECT テーブルA.カラム1,テーブルB.カラム1 FROM テーブルA FULL OUTER JOIN テーブルB ON テーブルA.カラム2 = テーブルB.カラム2;

\*全て小文字で入力しても反映します。

簡略化すると

SELECT (取得するカラム) FROM テーブルA FULL OUTER JOIN テーブルB ON 結合条件;

完全外部結合

FULL OUTER JOINはMySQLで使えない?!

MySQLではFULL OUTER JOINは未対応です。

OracleやPostgreSQL、DB2の場合、FULL OUTER JOINを使用できます。

\*実際に入力すると、以下の文になります。

\*MySQLだとERRORになります。

完全外部結合

MySQLの完全外部結合はUNION句を使おう!

UNION句は複数のSELECT文によってデータをそれぞれ取得し、 その結果を結合した上で、1つのデータとして取得する場合に使います。

user\_id name id team

1 tanaka 1 red

2 yamada 2 blue

4 kimura NULL NULL

5 suzuki 5 green

NULL NULL 3 blue

NULL NULL 6 red

FULL OUTER JOINを 使わずに完全外部結合 と同じ表示にすることが 可能となります。

MySQLの完全外部結合はUNION句を使おう!

「LEFT OUTER JOIN」と「RIGHT OUTER JOIN」の結果を UNION句を使って結合し表示する。

SELECT \* FROM user LEFT OUTER JOIN team ON user.user\_id = team.id UNION SELECT \* FROM user RIGHT OUTER JOIN team ON user.user\_id = team.id ;

完全外部結合

改行しても1行で入力してもどちらでも大丈夫です。 単語の間に半角スペースを入れ忘れないよう気をつけてね!

MySQLの完全外部結合はUNION句を使おう!

完全外部結合

以上