無料電子ブック

学習

SQL

Free unaffiliated eBook created from
Stack Overflow contributors.

#sql

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: sql

It is an unofficial and free SQL ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official SQL.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

1: SQL

SQLは、リレーショナル·データベース·システムのデータをするためにされるです。さまざまなベンダーがをし、のさまざまなをっています。

このタグは、 **ISO / ANSI SQL**をにしています 。そののののにはされません。

バージョン

バージョン	い		
1986	SQL-86	ANSI X3.135-1986、ISO 90751987	1986-01-01
1989	SQL-89	ANSI X3.135-1989、ISO / IEC 90751989	1989-01-01
1992	SQL-92	ISO / IEC 90751992	1992-01-01
1999	SQL1999	ISO / IEC 90751999	1999-12-16
2003	SQL2003	ISO / IEC 90752003	20031215
2006	SQL2006	ISO / IEC 90752006	2006-06-01
2008	SQL2008	ISO / IEC 90752008	2008715
2011	SQL2011	ISO / IEC 90752011	2011-12-15
2016	SQL2016	ISO / IEC 90752016	2016-12-01

Examples

Structured Query LanguageSQLは、リレーショナルデータベースシステムRDBMSにされているデータをするためにされたのプログラミングです。 SQLライクなは、リレーショナルデータストリームシステムRDSMSや「SQLNoSQL」データベースでもできます。

SQLは3つのなサブでされています。

1. データDDLデータベースのをおよびする。
2. データDMLデータベースのデータにしてみり、、およびをする。
3. データDCLデータベースにされたデータへのアクセスをします。

WikipediaのSQL

となるDMLは、 `INSERT` 、 `SELECT` 、 `UPDATE`および`DELETE`によってされるCreate、Read、UpdateおよびDeleteCRUDです。

3つのきみINSERT、UPDATE、DELETEをすべてできるされた `MERGE`もあります。

WikipediaのCRUD

くのSQLデータベースはクライアント/サーバーシステムとしてされています。「SQLサーバ」というはそのようなデータベースをする。
に、マイクロソフトは "SQL Server"というのデータベースをします。そのデータベースはSQLのをしていますが、そのデータベースにのはこのタグのトピックではなく、 SQL Serverのドキュメントにしています 。

オンラインでSQLをむ https://riptutorial.com/ja/sql/topic/184/sql

2: ANDとOR

1. SELECT * FROMテーブルWHERE1AND2;
2. SELECT * FROMテーブルWHERE1OR2;

Examples

AND ORの

テーブルをする

		シティ
ボブ	10	パリ
マット	20	ベルリン
メアリー	24	プラハ

```
select Name from table where Age>10 AND City='Prague'
```

る

メアリー

```
select Name from table where Age=10 OR City='Prague'
```

る

ボブ
メアリー

オンラインでANDとORをむ https://riptutorial.com/ja/sql/topic/1386/andとor

3: CREATE FUNCTION

- CREATE FUNCTION function_name[list_of_paramers]りreturn_data_type AS BEGIN function_body RETURN scalar_expression END

パラメーター

function_name	の
list_of_paramenters	がけれるパラメータ
return_data_type	そのがするタイプ。のSQL データ
function_body	のコード
スカラー	によってされるスカラー

CREATE FUNCTIONは、SELECT、INSERT、UPDATE、またはDELETEせをするときにできるユーザーをします。は、のまたはのテーブルをすようにすることができます。

Examples

しいをする

```
CREATE FUNCTION FirstWord (@input varchar(1000))
RETURNS varchar(1000)
AS
BEGIN
    DECLARE @output varchar(1000)
    SET @output = SUBSTRING(@input, 0, CASE CHARINDEX(' ', @input)
        WHEN 0 THEN LEN(@input) + 1
        ELSE CHARINDEX(' ', @input)
    END)

    RETURN @output
END
```

このでは、 **FirstWord**というのをします。このはvarcharパラメータをけれ、のvarcharをします。

オンラインでCREATE FUNCTIONをむ https://riptutorial.com/ja/sql/topic/2437/create-function

4: CREATE TABLE

き

CREATE TABLEステートメントは、データベースにしいテーブルをするためにされます。は、のリスト、そのタイプ、およびからされます。

- CREATE TABLE tableName[ColumnName1] [datatype1] [、[ColumnName2] [datatype2] ...]

パラメーター

パラメータ	
tableName	テーブルの
	テーブルにあるすべてのの「」をみます。については、しいテーブルのをしてください。

テーブルはでなければなりません。

Examples

しいテーブルをする

IDをむな`Employees`テーブル、およびとともにのとをするには、

```
CREATE TABLE Employees(
    Id int identity(1,1) primary key not null,
    FName varchar(20) not null,
    LName varchar(20) not null,
    PhoneNumber varchar(10) not null
);
```

このは、 Transact-SQL

`CREATE TABLE`はデータベースにしいテーブルをし、そのにテーブル`Employees`します

これにいて、カラムとそのプロパティのリストIDなどがきます

```
Id int identity(1,1) not null
```

`Id`	の

`int`	データです。
`identity(1,1)`	のは1からまり、しいごとに1ずつえます。
`primary key`	こののすべてのがのをつことをしています
`not null`	このはNULLをつことはできません

からテーブルをする

テーブルのをすることができます

```
CREATE TABLE ClonedEmployees AS SELECT * FROM Employees;
```

SELECTステートメントののをして、データをしてからしいテーブルにすことができます。しいテーブルのは、されたにってにされます。

```
CREATE TABLE ModifiedEmployees AS
SELECT Id, CONCAT(FName," ",LName) AS FullName FROM Employees
WHERE Id > 10;
```

テーブルをする

テーブルをするには、のをします。

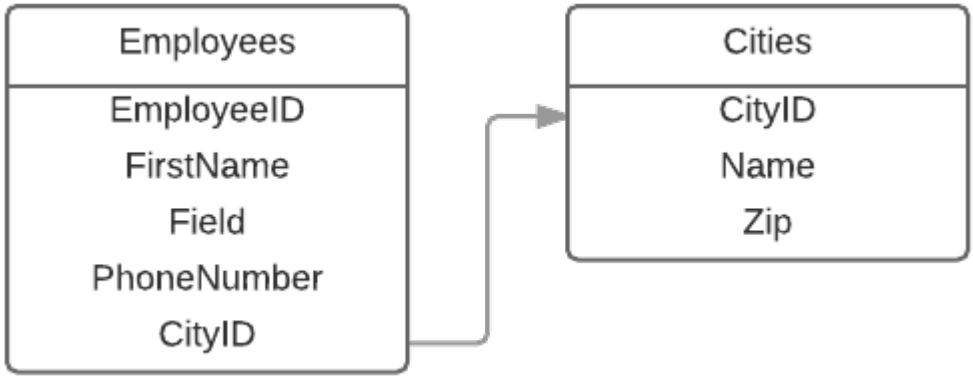
```
CREATE TABLE newtable LIKE oldtable;
INSERT newtable SELECT * FROM oldtable;
```

FOREIGN KEYをしたCREATE TABLE

では、テーブル`Cities`して`Employees`テーブルをつけることができます。

```
CREATE TABLE Cities(
    CityID INT IDENTITY(1,1) NOT NULL,
    Name VARCHAR(20) NOT NULL,
    Zip VARCHAR(10) NOT NULL
);

CREATE TABLE Employees(
    EmployeeID INT IDENTITY (1,1) NOT NULL,
    FirstName VARCHAR(20) NOT NULL,
    LastName VARCHAR(20) NOT NULL,
    PhoneNumber VARCHAR(10) NOT NULL,
    CityID INT FOREIGN KEY REFERENCES Cities(CityID)
);
```

ここでデータベースダイアグラムをつけることができます。



カラム`CityID`の`Employees` 、カラムにする`CityID`テーブルの`Cities` 。 では、これをうためのをつけることができます。

```
CityID INT FOREIGN KEY REFERENCES Cities(CityID)
```

`CityID`	の
`int`	のタイプ
`FOREIGN KEY`	キーをするオプション
`REFERENCES Cities(CityID)`	をする `CitiesCityID`

データベースにしないテーブルへのをできませんでした。にテーブルの`Cities`をし、にテーブルの`Employees`をするソースにしてください。のはエラーになります。

テンポラリ・テーブル またはインメモリ・テーブルの

PostgreSQL と SQLite

セッションにローカルなテーブルをするには

```
CREATE TEMP TABLE MyTable(...);
```

SQL サーバー

セッションにローカルなテーブルをするには

```
CREATE TABLE #TempPhysical(...);
```

がることができるテーブルをするには

```
CREATE TABLE ##TempPhysicalVisibleToEveryone(...);
```

メモリテーブルをするには

```
DECLARE @TempMemory TABLE(...);
```

オンラインでCREATE TABLEをむ https://riptutorial.com/ja/sql/topic/348/create-table

5: CREATEデータベース

- CREATE DATABASE dbname;

Examples

CREATEデータベース

データベースは、のSQLコマンドをしてされます。

```
CREATE DATABASE myDatabase;
```

これにより、テーブルをできるmyDatabaseというのデータベースがされます。

オンラインでCREATEデータベースをむ https://riptutorial.com/ja/sql/topic/2744/createデータベース

6: DROPテーブル

DROP TABLEは、、、、およびトリガーとともにスキーマからをします。

Examples

シンプルなドロップ

```
Drop Table MyTable;
```

のをする

MySQL 3.19

```
DROP TABLE IF EXISTS MyTable;
```

PostgreSQL 8.x

```
DROP TABLE IF EXISTS MyTable;
```

SQL Server 2005

```
If Exists(Select * From Information_Schema.Tables
          Where Table_Schema = 'dbo'
            And Table_Name = 'MyTable')
  Drop Table dbo.MyTable
```

SQLite 3.0

```
DROP TABLE IF EXISTS MyTable;
```

オンラインでDROPテーブルをむ https://riptutorial.com/ja/sql/topic/1832/dropテーブル

7: DROPまたはDELETEデータベース

- MSSQLの
- DROP DATABASE [する] {database_name | database_snapshot_name} [、... n] [;]
- MySQL
- DROP {DATABASE | SCHEMA} [IF EXISTS] db_name

`DROP DATABASE`は、SQLからデータベースをするためにされます。ってがわれないように、データベースをするにずデータベースのバックアップをしてください。

Examples

DROPデータベース

データベースをするのはなのステートメントです。データベースをするとデータベースがされるため、にじてにデータベースのバックアップをるようにしてください。

は、Employeesデータベースをするコマンドです

```
DROP DATABASE [dbo].[Employees]
```

オンラインでDROPまたはDELETEデータベースをむ https://riptutorial.com/ja/sql/topic/3974/dropまたはdeleteデータベース

8: GROUP BY

き

SELECTクエリのは、 `GROUP BY`をして1つのでグループすることができます。グループされたのじをつすべてのがまとめてされます。 1つのではなく、なのがされます。 GROUP BYは、グループされていないをするをする`HAVING`ステートメントをして、とみわせてできます。

- GROUP BY {

  | ROLLUP<group_by_expression> [、... n]
  | CUBE<group_by_expression> [、... n]
  | GROUPING SETS[、... n]
  | - をする
  } [、... n]

- <group_by_expression> :: =

  | [、... n]

- <grouping_set> :: =
  - をする
  | <grouping_set_item>
  | <grouping_set_item> [、... n]

- <grouping_set_item> :: =
  <group_by_expression>
  | ROLLUP<group_by_expression> [、... n]
  | CUBE<group_by_expression> [、... n]

Examples

されたののエントリのを**COUNT**までする

ののにしてカウントまたはをするとします。

このをると、 "Westerosians"

	GreatHouseAllegience
アリア	スターク
Cercei	Lannister

	GreatHouseAllegience
Myrcella	Lannister
ヤラ	グレイジョイ
Catelyn	スターク
サンサ	スターク

GROUP BYをしないと、COUNTはにをします。

```
SELECT Count(*) Number_of_Westerosians
FROM Westerosians
```

る...

ウェステロス
6

しかし、GROUP BYをすることで、されたのをすために、えられたのにしてユーザーをCOUNTすることができます。

```
SELECT GreatHouseAllegience House, Count(*) Number_of_Westerosians
FROM Westerosians
GROUP BY GreatHouseAllegience
```

る...

	ウェステロス
スターク	3
グレイジョイ	1
Lannister	2

GROUP BYとORDER BYをみわせて、をまたはのカテゴリでソートするのはです。

```
SELECT GreatHouseAllegience House, Count(*) Number_of_Westerosians
FROM Westerosians
GROUP BY GreatHouseAllegience
ORDER BY Number_of_Westerosians Desc
```

る...

	ウェステロス
スターク	3
Lannister	2
グレイジョイ	1

HAVINGをしてGROUP BYをフィルタリングする

HAVINGは、GROUP BYのをフィルタリングします。のでは、 ライブラリのサンプルデータベースをしています。

の をいたすべてのをします。

```
SELECT
  a.Id,
  a.Name,
  COUNT(*) BooksWritten
FROM BooksAuthors ba
  INNER JOIN Authors a ON a.id = ba.authorid
GROUP BY
  a.Id,
  a.Name
HAVING COUNT(*) > 1    -- equals to HAVING BooksWritten > 1
;
```

3のがいるすべてのをしますの 。

```
SELECT
  b.Id,
  b.Title,
  COUNT(*) NumberOfAuthors
FROM BooksAuthors ba
  INNER JOIN Books b ON b.id = ba.bookid
GROUP BY
  b.Id,
  b.Title
HAVING COUNT(*) > 3    -- equals to HAVING NumberOfAuthors > 3
;
```

なGROUP BYの

のためにGROUP BYを「それぞれのために」えるのはかもしれません。のクエリ

```
SELECT EmpID, SUM (MonthlySalary)
FROM Employee
GROUP BY EmpID
```

っている

" EmpIDのMonthlySalaryのをえてください"

だからあなたのテーブルがこのようにえたら

```
+-----+-------------+
|EmpID|MonthlySalary|
+-----+-------------+
|1    |200          |
+-----+-------------+
|2    |300          |
+-----+-------------+
```

```
+-+---+
|1|200|
+-+---+
|2|300|
+-+---+
```

1つののがそのなので、Sumはもしないようにえます。、のようにえるは

```
+-----+-------------+
|EmpID|MonthlySalary|
+-----+-------------+
|1    |200          |
+-----+-------------+
|1    |300          |
+-----+-------------+
|2    |300          |
+-----+-------------+
```

```
+-+---+
|1|500|
+-+---+
|2|300|
+-+---+
```

それは、EmpID 1が2つにまっているからです。

ROLAPデータマイニング

SQLには、2つのがされています。これらは、がることができるすべてののセットをすために"ALL"をします。 2つのはのとおりです。

- `with data cube`では、のよりもなすべてのみわせをします。
- `with roll up`ことで、をからへにしてられたを、それらがのにどのようにリストされているかをする。

これらのをサポートするSQLバージョン1999,2003,2006,2008,2011。

のをしてください。

フード	ブランド	
パスタ	ブランド1	100
パスタ	Brand2	250
ピザ	Brand2	300

で

```
select Food,Brand,Total_amount
from Table
group by Food,Brand,Total_amount with cube
```

フード	ブランド	
パスタ	ブランド1	100
パスタ	Brand2	250
パスタ	すべて	350
ピザ	Brand2	300
ピザ	すべて	300
すべて	ブランド1	100
すべて	Brand2	550
すべて	すべて	650

ロールアップで

```
select Food,Brand,Total_amount
from Table
group by Food,Brand,Total_amount with roll up
```

フード	ブランド	
パスタ	ブランド1	100
パスタ	Brand2	250
ピザ	Brand2	300
パスタ	すべて	350

フード	ブランド	
ピザ	すべて	300
すべて	すべて	650

オンラインでGROUP BYをむ https://riptutorial.com/ja/sql/topic/627/group-by

9: IN

Examples

なIN

えられた`id`のどれかをつレコードをするには

```
select *
from products
where id in (1,8,3)
```

のクエリは

```
select *
from products
where id = 1
   or id = 8
   or id = 3
```

サブクエリでINをする

```
SELECT *
FROM customers
WHERE id IN (
    SELECT DISTINCT customer_id
    FROM orders
);
```

は、システムでしたすべてのをします。

オンラインでINをむ https://riptutorial.com/ja/sql/topic/3169/in

10: LIKE

- きのワイルドカード SELECT * FROM [table] WHERE [column_name] 'Value'のように

  _のワイルドカード SELECT * FROM [テーブル] WHERE [] 'V_n'のように

  [charlist]をしたワイルドカード SELECT * FROM [table] WHERE [column_name] 'V [abc] n

WHEREのLIKEは、されたパターンにするのをするためにされます。パターンは、の2つのワイルドカードをしてされます

- Percentage Symbol - ゼロのをすためにされます。
- _アンダースコア - のをすためにされます

Examples

オープンエンドパターンにマッチ

のまたはまたはそのに%ワイルドカードをすると、パターンのまたはの0のをさせることができます。

の "%"をすると、パターンの2つののに0のがすることができます。

このEmployeesテーブルをします

イド	FName	LName		マネージャーID	DepartmentId		Hire_date
1	ジョン	ジョンソン	2468101214	1	1	400	23-03-2005
2	ソフィー	アムゼン	2479100211	1	1	400	11-01-2010
3	ロニー	スミス	2462544026	2	1	600	06-08-2015
4	ジョン	サンチェス	2454124602	1	1	400	23-03-2005
5	ヒルデ	Knag	2468021911	2	1	800	01-01-2000

のステートメントは、Employeesテーブルから 'on' をむ FName をつすべてのレコードをします。

```
SELECT * FROM Employees WHERE FName LIKE '%on%';
```

イド	FName	LName		マネーージャーID	DepartmentId		Hire_date
3	R **on** ny	スミス	2462544026	2	1	600	06-08-2015
4	J **on**	サンチェス	2454124602	1	1	400	23-03-2005

のステートメントは、から '246'でまる PhoneNumberをつすべてのレコードとします。

```
SELECT * FROM Employees WHERE PhoneNumber LIKE '246%';
```

イド	FName	LName		マネーージャーID	DepartmentId		Hire_date
1	ジョン	ジョンソン	**246** 8101214	1	1	400	23-03-2005
3	ロニー	スミス	**246** 2544026	2	1	600	06-08-2015
5	ヒルデ	Knag	**246** 8021911	2	1	800	01-01-2000

のステートメントは、Employeesから '11'でわる PhoneNumberをつすべてのレコードとします。

```
SELECT * FROM Employees WHERE PhoneNumber LIKE '%11'
```

イド	FName	LName		マネーージャーID	DepartmentId		Hire_date
2	ソフィー	アムゼン	24791002**11**	1	1	400	11-01-2010
5	ヒルデ	Knag	24680219**11**	2	1	800	01-01-2000

Fnameの**3**がから 'n'であるすべてのレコード。

```
SELECT * FROM Employees WHERE FName LIKE '__n%';
```

の2をスキップするには、「n」のに2つのアンダースコアがされます

イド	FName	LName		マネ―ジャ―ID	DepartmentId		Hire_date
3	ロニ―	スミス	2462544026	2	1	600	06-08-2015
4	ジョン	サンチェス	2454124602	1	1	400	23-03-2005

のマッチ

SQL-SELECTステートメントのをげるために、ワイルドカード、パーセントおよびアンダースコア_をすることができます。

_ アンダースコアは、パターンマッチのの1にしてワイルドカードとしてできます。

Fnameが 'j'でまり、 'n'でわり、Fnameにに3あるすべてのをします。

```
SELECT * FROM Employees WHERE FName LIKE 'j_n'
```

_ アンダースコアは、パターンをさせるためにワイルドカードとしてすることもできます。

たとえば、このパターンは "jon"、 "jan"、 "jen"などとします。

jn、john、jordan、justin、jason、julian、jillian、joannなどのはされません。クエリでアンダースコアが1つされ、にスキップできます。 1なので、は3のFnameでなければなりません。

たとえば、このパターンは "LaSt"、 "LoSt"、 "HaLt"などとします。

```
SELECT * FROM Employees WHERE FName LIKE '_A_T'
```

またはセットでマッチ

されたのの1をマッチさせます `[af]` またはセットします `[abcdef]`。

このパターンは "gary"にマッチしますが、 "mary"にはマッチしません

```
SELECT * FROM Employees WHERE FName LIKE '[a-g]ary'
```

このパターンは "mary"にしますが、 "gary"にはしません。

```
SELECT * FROM Employees WHERE Fname LIKE '[lmnop]ary'
```

またはセットは、またはセットのに^キャレットをすることですることもできます。

このパターンは "gary"としませんが、 "mary"とします

```
SELECT * FROM Employees WHERE FName LIKE '[^a-g]ary'
```

このパターンは "mary"としませんが、 "gary"とします

```
SELECT * FROM Employees WHERE Fname LIKE '[^lmnop]ary'
```

ANYALLのマッチ

するもの
なくとも1つのにするがあります。このでは、タイプは 'electronics'、 'books'、または 'video'のいずれかでなければなりません。

```
SELECT *
FROM   purchase_table
WHERE  product_type LIKE ANY ('electronics', 'books', 'video');
```

すべてすべてのをたしているがあります。
このでは、「」と「ロンドン」と「」バリエーションをむのがしなければなりません。

```
SELECT *
FROM   customer_table
WHERE  full_address LIKE ALL ('%united kingdom%', '%london%', '%eastern road%');
```

な
ALLをしてすべてのアイテムをします。
このでは、タイプが「エレクトロニクス」ではなく「」ではなく「ビデオ」ではないすべてのがられます。

```
SELECT *
FROM   customer_table
WHERE  product_type NOT LIKE ALL ('electronics', 'books', 'video');
```

のをする

のステートメントは、 Employeesテーブルから AからFまでのでまるFNameをつすべてのレコードとします。

```
SELECT * FROM Employees WHERE FName LIKE '[A-F]%'
```

LIKEクエリのESCAPEステートメント

`LIKE` -queryとしてテキストをする、はのようにします。

```
SELECT *
FROM T_Whatever
WHERE SomeField LIKE CONCAT('%', @in_SearchText, '%')
```

しかし、フルテキストをできるときに`LIKE`するはありませんが50や "a_b"のようなテキストをするとがします。

だからフルテキストにりえるわりに `LIKE` -escapeステ―トメントをってそのをすることができます

```
SELECT *
FROM T_Whatever
WHERE SomeField LIKE CONCAT('%', @in_SearchText, '%') ESCAPE '\'
```

つまり、`\`はESCAPEとしてわれます。これは、するのすべてのに`\`をすることができることをします。ユ―ザ―が`%`または_ようなをしたでも、はしくまります。

えば

```
string stringToSearch = "abc_def 50%";
string newString = "";
foreach(char c in stringToSearch)
     newString += @"\" + c;

sqlCmd.Parameters.Add("@in_SearchText", newString);
// instead of sqlCmd.Parameters.Add("@in_SearchText", stringToSearch);
```

のアルゴリズムはデモンストレ―ションののみです。 1つのがのからなるutf-8はしません。

`string stringToSearch = "Les Mise\u0301rables";`あなたは、ではなく、ごとにこれをうがあります。アジア/アジア/アジアをうは、のアルゴリズムをしないでください。あるいは、なコ―ドをめるには、graphemeClusterごとにうべきです。

Cのインタビュ―のであるReverseStringもしてください。

ワイルドカ―ド

ワイルドカ―ドはSQL LIKEとともにされます。 SQLワイルドカ―ドは、テ―ブルのデ―タをするためにされます。

SQLのワイルドカ―ドは、、_、[charlist]、[^ charlist]

- 0のの

```
   Eg:  //selects all customers with a City starting with "Lo"
        SELECT * FROM Customers
        WHERE City LIKE 'Lo%';

       //selects all customers with a City containing the pattern "es"
      SELECT * FROM Customers
       WHERE City LIKE '%es%';
```

_ - 1のわり

```
Eg://selects all customers with a City starting with any character, followed by "erlin"
SELECT * FROM Customers
WHERE City LIKE '_erlin';
```

[charlist] - するのセットと

```
Eg://selects all customers with a City starting with "a", "d", or "l"
SELECT * FROM Customers
WHERE City LIKE '[adl]%';

//selects all customers with a City starting with "a", "d", or "l"
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

[^ charlist] - にされていないにのみします

```
Eg://selects all customers with a City starting with a character that is not "a", "p", or "l"
SELECT * FROM Customers
WHERE City LIKE '[^apl]%';

or

SELECT * FROM Customers
WHERE City NOT LIKE '[apl]%' and city like '_%';
```

オンラインでLIKEをむ https://riptutorial.com/ja/sql/topic/860/like

11: ORDER BY

Examples

ORDER BYをTOPとみわせてすると、のについてxがされます

このでは、々はののべえをしていない、GROUP BYをすることができますされたが、また、々はセットをするために、TOPをしていることから、され。

のなQAサイトから5のユーザーをすとしましょう。

ORDER BYなし

このせは、デフォルトでけられた5このはにされているではないにもかかわらず、このは「Id」をののとしてします。

```
SELECT TOP 5 DisplayName, Reputation
FROM Users
```

る...

コミュニティ	1
ジェフ･ダルガス	12567
ジャロッド･ディクソン	11739
ジェファトウッド	37628
ジョエル･スピルスキー	25784

ORDER BYで

```
SELECT TOP 5 DisplayName, Reputation
FROM Users
ORDER BY Reputation desc
```

る...

JonSkeet	**865023**
ダーリン･ディミトロフ	**661741**
BalusC	**650237**

ハンス·パサント	**625870**
マーク·グラヴェル	**601636**

のバージョンのSQLMySQLなどでは、に`TOP`わりに`SELECT`のに`LIMIT`をします。たとえば、のようになります。

```
SELECT DisplayName, Reputation
FROM Users
ORDER BY Reputation DESC
LIMIT 5
```

のによるべえ

```
SELECT DisplayName, JoinDate, Reputation
FROM Users
ORDER BY JoinDate, Reputation
```

	JoinDate	
コミュニティ	**2008-09-15**	**1**
ジェフアトウッド	**2008-09-16**	**25784**
ジョエル·スピルスキー	2008-09-16	**37628**
ジャロッド·ディクソン	**20081003**	**11739**
ジェフ·ダルガス	20081003	**12567**

のわりにでソート

のをするのではなく、ののが '1'をして、べえのとなるをすことができます。

Proでをするがいとわれるは、このコードをることはありません。

Conこれはにクエリのをさせます「ORDER BY 14」はらかのカウントをとしますが、スクリーンにはおそらくでされます。

このクエリは、`Reputation`わりにselectからカラム`3`でをソートします。

```
SELECT DisplayName, JoinDate, Reputation
FROM Users
ORDER BY 3
```

	JoinDate	
コミュニティ	2008-09-15	**1**
ジャロッド·ディクソン	20081003	**11739**
ジェフ·ダルガス	20081003	**12567**
ジョエル·スピルスキー	2008-09-16	**25784**
ジェフアトウッド	2008-09-16	**37628**

エイリアス

なクエリのために、エイリアスは、でにすることができます。

```
SELECT DisplayName, JoinDate as jd, Reputation as rep
FROM Users
ORDER BY jd, rep
```

そして、selectののなをすることができます。のをえてください。をするわりに、のようになをしてください。Jdは1、Jdは2などです。

```
SELECT DisplayName, JoinDate as jd, Reputation as rep
FROM Users
ORDER BY 2, 3
```

カスタムのべえ

にこのテーブルの`Employee`をソートするには、`ORDER BY Department`をします。ただし、ソートがアルファベットでないは、 `Department`をしくソートするのにマップするがあります。これはCASEでうことができます

っています	それ
ユスフ	HR
ヒラリー	HR
ジョー	それ
メリー	HR
ケン	

```
SELECT *
```

```
FROM Employee
ORDER BY CASE Department
         WHEN 'HR'         THEN 1
         WHEN 'Accountant' THEN 2
         ELSE                   3
         END;
```

ユスフ	**HR**
ヒラリー	**HR**
メリー	**HR**
ケン	
っています	それ
ジョー	それ

オンラインでORDER BYをむ https://riptutorial.com/ja/sql/topic/620/order-by

12: SQL CURSOR

Examples

データベースごとにインデックスごとにすべてのをするカーソルの

ここでは、カーソルをしてすべてのデータベースをループします。
さらに、SQLのカーソルをして、のカーソルによってされたデータベースをします。
これは、カーソルのをすためです。

```
DECLARE @db_name nvarchar(255)
DECLARE @sql nvarchar(MAX)

DECLARE @schema nvarchar(255)
DECLARE @table nvarchar(255)
DECLARE @column nvarchar(255)



DECLARE db_cursor CURSOR FOR
SELECT name FROM sys.databases


OPEN db_cursor
FETCH NEXT FROM db_cursor INTO @db_name

WHILE @@FETCH_STATUS = 0
BEGIN
    SET @sql = 'SELECT * FROM ' + QUOTENAME(@db_name) + '.information_schema.columns'
    PRINT ''
    PRINT ''
    PRINT ''
    PRINT @sql
    -- EXECUTE(@sql)



    -- For each database

    DECLARE @sqlstatement nvarchar(4000)
    --move declare cursor into sql to be executed
    SET @sqlstatement = 'DECLARE  columns_cursor CURSOR FOR SELECT TABLE_SCHEMA, TABLE_NAME,
COLUMN_NAME FROM ' + QUOTENAME(@db_name) + '.information_schema.columns ORDER BY TABLE_SCHEMA,
TABLE_NAME, ORDINAL_POSITION'



    EXEC sp_executesql @sqlstatement


    OPEN columns_cursor
    FETCH NEXT FROM columns_cursor
    INTO @schema, @table, @column
```

```
    WHILE @@FETCH_STATUS = 0
    BEGIN
        PRINT @schema + '.' + @table + '.' + @column
        --EXEC asp_DoSomethingStoredProc @UserId

    FETCH NEXT FROM columns_cursor --have to fetch again within loop
    INTO @schema, @table, @column

    END
    CLOSE columns_cursor
    DEALLOCATE columns_cursor

    -- End for each database




    FETCH NEXT FROM db_cursor INTO @db_name
END

CLOSE db_cursor
DEALLOCATE db_cursor
```

オンラインでSQL CURSORをむ https://riptutorial.com/ja/sql/topic/8895/sql-cursor

13: SQL Group by vs Distinct

Examples

GROUP BYとDISTINCTのい

`GROUP BY`はとみわせてされます。のをしてください。

ID	ユーザーID		オーダー	
1	43	A	25	20-03-2016
2	57	B	50	22-03-2016
3	43	A	30	25-03-2016
4	82	ストアC	10	26-03-2016
5	21	A	45	29-03-2016

のクエリは`GROUP BY`をしてをします。

```
SELECT
    storeName,
    COUNT(*) AS total_nr_orders,
    COUNT(DISTINCT userId) AS nr_unique_customers,
    AVG(orderValue) AS average_order_value,
    MIN(orderDate) AS first_order,
    MAX(orderDate) AS lastOrder
FROM
    orders
GROUP BY
    storeName;
```

のをします

	total_nr_orders	nr_unique_customers	average_order_value	の	lastOrder
A	3	2	33.3	20-03-2016	29-03-2016
B	1	1	50	22-03-2016	22-03-2016
スト	1	1	10	26-03-	26-03-2016

	total_nr_orders	nr_unique_customers	average_order_value	の	lastOrder
アC				2016	

DISTINCTは、されたのDISTINCTのみわせをリストするのにされます。

```
SELECT DISTINCT
    storeName,
    userId
FROM
    orders;
```

	ユーザーID
A	43
B	57
ストアC	82
A	21

オンラインでSQL Group by vs Distinctをむ https://riptutorial.com/ja/sql/topic/2499/sql-group-by-vs-distinct

14: SQLインジェクション

き

SQLインジェクションは、フォームフィールドにSQLをしてWebサイトのデータベーステーブルにアクセスしようとするみです。 Webサーバーがインジェクションからしない、ハッカーはデータベースをしてのSQLコードをすることができます。ハッカーは、のSQLコードをすることで、アカウントへのアクセスをアップグレードしたり、のユーザーのをしたり、データベースをしたりすることができます。

Examples

SQLインジェクションサンプル

Webアプリケーションのログインハンドラへのびしがのようになっているとします。

```
https://somepage.com/ajax/login.ashx?username=admin&password=123
```

login.ashxでは、のをんでいます。

```
strUserName = getHttpsRequestParameterString("username");
strPassword = getHttpsRequestParameterString("password");
```

データベースにして、そのパスワードをつユーザーがするかどうかをします。

したがって、SQLクエリをします。

```
txtSQL = "SELECT * FROM Users WHERE username = '" + strUserName + "' AND password = '"+
strPassword +"'";
```

これは、ユーザーとパスワードにがまれていないにです。

ただし、パラメータの1つにがまれている、データベースにされるSQLはのようになります。

```
-- strUserName = "d'Alambert";
txtSQL = "SELECT * FROM Users WHERE username = 'd'Alambert' AND password = '123'";
```

これにより、 `d d'Alambert` のがSQLをするため、エラーがします。

あなたはユーザーとパスワードのをエスケープすることでこれをすることができます。

```
strUserName = strUserName.Replace("'", "''");
strPassword = strPassword.Replace("'", "''");
```

ただし、パラメ―タをするがです。

```
cmd.CommandText = "SELECT * FROM Users WHERE username = @username AND password = @password";

cmd.Parameters.Add("@username", strUserName);
cmd.Parameters.Add("@password", strPassword);
```

パラメ―タをせず、いずれかのでもをきえるのをれた、のあるユ―ザ―ハッカ―ともばれますがこれをしてデ―タベ―スでSQLコマンドをできます。

たとえば、がいは、パスワ―ドをのようにします。

```
lol'; DROP DATABASE master; --
```

SQLはのようになります。

```
"SELECT * FROM Users WHERE username = 'somebody' AND password = 'lol'; DROP DATABASE master; --'";
```

ながら、これはなSQLであり、DBはこれをします

このタイプのは、SQLインジェクションとばれます。

すべてのユ―ザ―のメ―ルアドレスをみ、のパスワ―ドもんだり、クレジットカ―ドをんだり、デ―タベ―スのデ―タをんだりするなど、のあるユ―ザ―がうことができるにもたくさんのことがあります。

これは、にあなたのをエスケ―プするがあるです。
そして、かれかれそうすることをいつもれてしまうというは、まさにあなたがパラメ―タをうべきです。パラメ―タをすると、プログラミングフレ―ムワ―クがなエスケ―プをうためです。

なサンプル

SQLがのようにされているとします。

```
SQL = "SELECT * FROM Users WHERE username = '" + user + "' AND password ='" + pw + "'";
db.execute(SQL);
```

に、ハッカ―は`pw' or '1'='1`ようなパスワ―ドをえてデ―タをすることができます。のSQLはのようになります。

```
SELECT * FROM Users WHERE username = 'somebody' AND password ='pw' or '1'='1'
```

これは、`'1'='1'`がにであるため、`Users`テ―ブルのすべてののパスワ―ドチェックをパスします。

これをぐには、SQLパラメ―タをします。

```
SQL = "SELECT * FROM Users WHERE username = ? AND password = ?";
db.execute(SQL, [user, pw]);
```

オンラインでSQLインジェクションをむ https://riptutorial.com/ja/sql/topic/3517/sqlインジェクション

15: SQLのクリーンコード

き

れたみやすいSQLクエリのとれた

Examples

キーワードとのとスペル

テーブル/カラム

フォーマットテーブル/カラムの2つのながあるCamelCaseとsnake_case

```
SELECT FirstName, LastName
FROM Employees
WHERE Salary > 500;
```

```
SELECT first_name, last_name
FROM employees
WHERE salary > 500;
```

は、オブジェクトにされているものをするがあります。これは、がはでなければならないことをします。テーブルがかかをわなければならないかどうかは、よくされているですが、にはのテーブルをするがです。

tblやcolようなやをするとがするため、けてください。ただし、SQLキーワードとのをけるためにされることがあります。は、トリガーやインデックスははクエリではされませんでされます。

キーワード

SQLキーワードはとをしません。ただし、でするのがです。

SELECT *

SELECT *は、でされているのとじですべてのをします。

SELECT *をすると、クエリによってされるデータは、テーブルがされるたびにされるがあります。これにより、なるバージョンのアプリケーションまたはデータベースがいにがないというリスクがします。

さらに、にくのをみると、ディスクとネットワークI / Oのがえるがあります。

したがって、にするをににするがあります。

```
--SELECT *                                  don't
  SELECT ID, FName, LName, PhoneNumber  -- do
  FROM Emplopees;
```

ィンタラクティブクエリをする、これらのはされません。

ただし、EXISTSはのデータをしますなくとも1つのがつかったにのみチェックしますので、`SELECT *`はEXISTSのサブクエリでにはなりません。じで、EXISTSののをすることはがありません。したがって、 `SELECT *`にがあります。

```
-- list departments where nobody was hired recently
SELECT ID,
       Name
FROM Departments
WHERE NOT EXISTS (SELECT *
                  FROM Employees
                  WHERE DepartmentID = Departments.ID
                    AND HireDate >= '2015-01-01');
```

インデント

くけれられているはありません。もがめていることは、すべてを1にることがいことです。

```
SELECT d.Name, COUNT(*) AS Employees FROM Departments AS d JOIN Employees AS e ON d.ID =
e.DepartmentID WHERE d.Name != 'HR' HAVING COUNT(*) > 10 ORDER BY COUNT(*) DESC;
```

なくとも、すべてのをしいにれ、そうでなければすぎるようにするならば、をする

```
SELECT d.Name,
       COUNT(*) AS Employees
FROM Departments AS d
JOIN Employees AS e ON d.ID = e.DepartmentID
WHERE d.Name != 'HR'
HAVING COUNT(*) > 10
ORDER BY COUNT(*) DESC;
```

には、をするSQLキーワードののすべてがじにげされます。

```
SELECT   d.Name,
         COUNT(*) AS Employees
FROM     Departments AS d
JOIN     Employees AS e ON d.ID = e.DepartmentID
WHERE    d.Name != 'HR'
HAVING   COUNT(*) > 10
ORDER BY COUNT(*) DESC;
```

これは、SQLキーワードをさせながらうこともできます。

のなスタイルは、なキーワードをのにすることです。

```
SELECT
    d.Name,
    COUNT(*) AS Employees
FROM
    Departments AS d
JOIN
    Employees AS e
    ON d.ID = e.DepartmentID
WHERE
    d.Name != 'HR'
HAVING
    COUNT(*) > 10
ORDER BY
    COUNT(*) DESC;
```

したのをにさせると、みやすさがします。

```
SELECT Model,
       EmployeeID
FROM Cars
WHERE CustomerID = 42
  AND Status     = 'READY';
```

のをすると、SQLコマンドをのプログラミングにめむことがしくなります。しかし、くのでは、Cの`@"..." """..."""`、Pythonの`"""..."""` `R"(...)"`、C ++の`R"(...)"`、ののみがあります。

なはにするがあります。 なにはいくつかのがあります。

- はWHEREのどこかにあり、のフィルタとしています。これにより、どのテーブルがされているか、どのようにされているかがわかりにくくなります。
- のため、いのリスクがくなり、でつかるがくなります。
- SQLでは、なだけでをできます 。

  ```
  SELECT d.Name,
         e.Fname || e.LName AS EmpName
  FROM      Departments AS d
  LEFT JOIN Employees   AS e ON d.ID = e.DepartmentID;
  ```

- なでは、USINGをできます。

  ```
  SELECT RecipeID,
         Recipes.Name,
         COUNT(*) AS NumberOfIngredients
  FROM      Recipes
  LEFT JOIN Ingredients USING (RecipeID);
  ```

  これにより、のテーブルがじをするがあります。

USINGはからしたをにします。たとえば、このクエリのはの`RecipeID`をします。

オンラインでSQLのクリーンコードをむ https://riptutorial.com/ja/sql/topic/9843/sqlのクリーンコード

16: TRUNCATE

き

TRUNCATEは、テーブルからすべてのデータをします。これはフィルタなしのDELETEにていますが、データベースソフトウェアによっては、のとがあります。

- TRUNCATE TABLE table_name;

TRUNCATEはDDLデータコマンドであり、DELETEデータ、DML、コマンドとのにはきないがあります。 TRUNCATEはデータベースからのレコードをすばやくするですが、TRUNCATEコマンドをしてのにしているかどうかをするには、これらのをするがあります。

- TRUNCATEはデータページです。したがって、TRUNCATEをすると、にけられたDMLトリガーON DELETEはしません。のにはがかかりますが、でデータをするがあります。
- TRUNCATEはされたでされるディスクをし、DELETEはをします
- りてられるテーブルがIDカラムMS SQL Serverをする、シードはTRUNCATEコマンドによってリセットされます。これによりのがするがあります
- されているセキュリティロールとされているSQLのにじて、TRUNCATEコマンドをするためになが ないがあります

Examples

Employeeテーブルからすべてのをする

```
TRUNCATE TABLE Employee;
```

truncateテーブルをすると、インデックスとトリガをすべてしてすべてをするだけなので、DELETE TABLEをするがいがあります。

テーブルのはベースのです。つまり、がされます。 Truncate tableは、データページのデータをりてします。 100のテーブルがあるは、テーブルをするほうがテーブルのステートメントをうほうがずっとくなります。

DELETEをしてのをすることはできますが、のをTRUNCATEすることはできません。すべてのレコードをにTRUNCATEすることしかできません。すべてのをしてからしいレコードをすると、にされたからAuto Incremented Primary Keyがききされます.Truncateでは、Auto Incrementalキーもリセットされ、1からします。

テーブルをりてるときには、キーがしていなければなりません 。それのは、エラーがします。

オンラインでTRUNCATEをむ https://riptutorial.com/ja/sql/topic/1466/truncate

17: TRY / CATCH

TRY / CATCHは、MS SQL ServerのT-SQLのです。

これはT-SQLのエラーをにします。これは.NETコードにています。

Examples

TRY / CATCHでの

これは、なdatetimeのためにのをロールバックします。

```
BEGIN TRANSACTION
BEGIN TRY
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, 'not a date', 1)
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    THROW
    ROLLBACK TRANSACTION
END CATCH
```

これはのをコミットします

```
BEGIN TRANSACTION
BEGIN TRY
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    THROW
    ROLLBACK TRANSACTION
END CATCH
```

オンラインでTRY / CATCHをむ https://riptutorial.com/ja/sql/topic/4420/try---catch

18: UNION / UNION ALL

き

SQLの***UNION***キーワードは、のない***SELECT***のとするためにされます。 UNIONをしてをするには、のSELECTがじでじデータのをつがありますが、のさはなるがあります。

- SELECT column_1 [、column_2] FROM table_1 [、table_2] [WHERE]
  ユニオン| **UNION ALL**
  SELECT column_1 [、column_2] FROM table_1 [、table_2] [WHERE]

`UNION`および`UNION ALL`は、2つののSELECTのセットをの/テーブルにします。

`UNION` / `UNION ALL`がするためには、クエリのとのがしているがあります。

`UNION`と`UNION ALL`せのいは、 `UNION`が`UNION ALL`にないのをすることです。

このなレコードのは、よりされたクエリのためににするまたはにしないデフォルトがに`UNION ALL`にならないことがわかっているに、このためにされるべきのがなくても、クエリをしくくするがあります。

Examples

な**UNION ALL**クエリ

```
CREATE TABLE HR_EMPLOYEES
(
    PersonID int,
    LastName VARCHAR(30),
    FirstName VARCHAR(30),
    Position VARCHAR(30)
);

CREATE TABLE FINANCE_EMPLOYEES
(
    PersonID INT,
    LastName VARCHAR(30),
    FirstName VARCHAR(30),
    Position VARCHAR(30)
);
```

たちがからすべての`managers`をしたいとしましょう。

`UNION`をすることで、 `manager position`をしているHRとのすべてのをることができます

```
SELECT
    FirstName, LastName
FROM
```

```
    HR_EMPLOYEES
WHERE
    Position = 'manager'
UNION ALL
SELECT
    FirstName, LastName
FROM
    FINANCE_EMPLOYEES
WHERE
    Position = 'manager'
```

UNIONステートメントは、からをします。のでじとをつ々をつことがなので、をしないためにUNION ALLをしています。

にエイリアスをするは、のようにのselectにエイリアスをくだけです。

```
SELECT
    FirstName as 'First Name', LastName as 'Last Name'
FROM
    HR_EMPLOYEES
WHERE
    Position = 'manager'
UNION ALL
SELECT
    FirstName, LastName
FROM
    FINANCE_EMPLOYEES
WHERE
    Position = 'manager'
```

なと

なで

- UNIONは2つのセットをし、セットからをします
- UNION ALLはをしようとせずに2つのセットをします

  くのがっているのは、をするがないときにUNIONをすることです。なセットにするのパフォーマンスコストはにです。

UNIONがな

2つのなるにしてをフィルタリングするがあるとし、にして々のクラスタインデックスをしたとします。 UNIONすると、をぎながらのインデックスをできます。

```
SELECT C1, C2, C3 FROM Table1 WHERE C1 = @Param1
UNION
SELECT C1, C2, C3 FROM Table1 WHERE C2 = @Param2
```

これにより、これらのクエリをにするためになインデックスのみがとなるため、パフォーマンスチューニングがになります。ソーステーブルにするなきみパフォーマンスをさせる、クラスタインデックスのをかなりなくすることもです。

UNION ALLがな

2つのにしてテーブルをフィルタリングするがあるとしますが、レコードをフィルタリングするはありませんデータモデルの、ユニオンにデータがしてされることはないためです。

```
SELECT C1 FROM Table1
UNION ALL
SELECT C1 FROM Table2
```

これは、のテーブルににされるようにされたデータをするビューをするパフォーマンスのから、レコードをロールアップしたいなどににです。データがすでにされているため、データベースエンジンでをするとがされず、クエリにがされます。

オンラインでUNION / UNION ALLをむ https://riptutorial.com/ja/sql/topic/349/union---union-all

19: WHEREおよびHAVINGをしてをフィルタリングする

- SELECT column_name
  FROM table_name
  WHERE column_nameの
- SELECT column_name、aggregate_functioncolumn_name
  FROM table_name
  GROUP BY column_name
  column_nameのをする

Examples

WHEREは、にするのみをします

Steamはページの10ドルのゲームをっています。らのシステムののどこかに、らくのようなクエリがあります

```
SELECT *
FROM Items
WHERE Price < 10
```

INをして、リストにまれるをつをします。

このでは、サンプルデータベースのCar Tableをしています。

```
SELECT *
FROM Cars
WHERE TotalCost IN (100, 200, 300)
```

このクエリは、コスト200のCar2と、コスト100のCar3をします。これは、 ORをむのをすることにします。

```
SELECT *
FROM Cars
WHERE TotalCost = 100 OR TotalCost = 200 OR TotalCost = 300
```

LIKEをしてするとをする

LIKEのなドキュメントをしてください。

このでは、サンプルデータベースのEmployeesテーブルをしています。

```
SELECT *
FROM Employees
WHERE FName LIKE 'John'
```

このクエリは、のが「John」とにするEmployee1をします。

```
SELECT *
FROM Employees
WHERE FName like 'John%'
```

%すると、をすることができます

- John% - が'John'でまり、そのにのがくEmployeeをします
- %John - が'John'でわり、のだけんだEmployeeをします
- %John% - のどこにでも'John'がまれているEmployeeをします

この、クエリは、が「John」である2と、が「Johnathon」である4をします。

NULL / NOT NULLをつWHERE

```
SELECT *
FROM Employees
WHERE ManagerId IS NULL
```

このステートメントは、 ManagerIdのがNULL EmployeeレコードをすべてNULL。

はのようになります。

```
Id    FName    LName    PhoneNumber    ManagerId    DepartmentId
1     James    Smith    1234567890     NULL         1
```

```
SELECT *
FROM Employees
WHERE ManagerId IS NOT NULL
```

このステートメントは、 ManagerIdのがNULLでないすべてのレコードをしNULL。

はのようになります。

```
Id    FName       LName     PhoneNumber    ManagerId    DepartmentId
2     John        Johnson   2468101214     1            1
3     Michael     Williams  1357911131     1            2
4     Johnathon   Smith     1212121212     2            1
```

WHEREをWHERE ManagerId = NULLまたはWHERE ManagerId <> NULLすると、じでがされません。

でHAVINGをする

`WHERE`とはなり、 `HAVING`はでできます。

> は、ののが、よりなまたは Wikipedia ののをするために、のでとしてグループされるです。

なには、 `COUNT()` 、 `SUM()` 、 `MIN()` 、 `MAX()`ます。

このでは、サンプルデータベースのCar Tableをしています。

```
SELECT CustomerId, COUNT(Id) AS [Number of Cars]
FROM Cars
GROUP BY CustomerId
HAVING COUNT(Id) > 1
```

このクエリは`Number of Cars`する`CustomerId`および`Number of Cars`をします。この、のをつのは1です。

はのようになります。

ID	の
1	2

BETWEENをしてをフィルタリングする

のでは、 Item SalesおよびCustomersサンプル·データベースをしています。

> BETWEENはインクルーシブです。

Numbersで**BETWEEN**をう

```
SELECT * From ItemSales
WHERE Quantity BETWEEN 10 AND 17
```

このクエリは、1017のをつすべての`ItemSales`レコードをします。はのようになります。

イド	セール	ItemId		
1	2013-07-01	100	10	34.5
4	2013-07-23	100	15	34.5
5	2013724	145	10	34.5

に**BETWEEN**をする

```
SELECT * From ItemSales
WHERE SaleDate BETWEEN '2013-07-11' AND '2013-05-24'
```

このクエリは20137112013524の`SaleDate`をつすべての`ItemSales`レコードをします。

イド	セール	ItemId		
3	2013-07-11	100	20	34.5
4	2013-07-23	100	15	34.5
5	2013724	145	10	34.5

のわりにのをするは、のをのにするか、しいをるために24をまたはするがあります。

テキストでの**BETWEEN**の

```
SELECT Id, FName, LName FROM Customers
WHERE LName BETWEEN 'D' AND 'L';
```

SQL fiddle

このクエリは、がアルファベット「D」と「L」のにるをすべてします。この、1と3がされます。が「M」でまる2はまれません。

イド	FName	LName
1	ウィリアム	ジョーンズ
3	リチャード	ディビス

```
SELECT * FROM Employees
```

このステートメントは、`Employees`テーブルからすべてのをします。

```
Id    FName      LName     PhoneNumber    ManagerId    DepartmentId    Salary  Hire_date
CreatedDate    ModifiedDate
1     James      Smith     1234567890     NULL         1               1000    01-01-2002    01-01-
2002     01-01-2002
2     John       Johnson   2468101214     1            1               400     23-03-2005    23-03-
2005     01-01-2002
3     Michael    Williams  1357911131     1            2               600     12-05-2009    12-05-
2009     NULL
4     Johnathon  Smith     1212121212     2            1               500     24-07-2016    24-07-
2016     01-01-2002
```

`SELECT`ステートメントのに`WHERE`をすると、されたをにすることができます。この、 `=`をしてにするはのとおりです。

```
SELECT * FROM Employees WHERE DepartmentId = 1
```

`DepartmentId`が`1`だけをします。

```
Id   FName     LName    PhoneNumber   ManagerId   DepartmentId    Salary  Hire_date
CreatedDate   ModifiedDate
1    James     Smith    1234567890    NULL        1               1000    01-01-2002    01-01-
2002    01-01-2002
2    John      Johnson  2468101214    1           1               400     23-03-2005    23-03-
2005    01-01-2002
4    Johnathon Smith    1212121212    2           1               500     24-07-2016    24-07-
2016    01-01-2002
```

ANDとOR

また、のをみわせて、よりな`WHERE`をすることもできます。のでは、 `Employees`テーブルをしています。

```
Id   FName     LName    PhoneNumber   ManagerId   DepartmentId    Salary  Hire_date
CreatedDate   ModifiedDate
1    James     Smith    1234567890    NULL        1               1000    01-01-2002    01-01-
2002    01-01-2002
2    John      Johnson  2468101214    1           1               400     23-03-2005    23-03-
2005    01-01-2002
3    Michael   Williams 1357911131    1           2               600     12-05-2009    12-05-
2009    NULL
4    Johnathon Smith    1212121212    2           1               500     24-07-2016    24-07-
2016    01-01-2002
```

そして

```
SELECT * FROM Employees WHERE DepartmentId = 1 AND ManagerId = 1
```

ります

```
Id   FName     LName    PhoneNumber   ManagerId   DepartmentId    Salary  Hire_date
CreatedDate   ModifiedDate
2    John      Johnson  2468101214    1           1               400     23-03-2005    23-03-
2005    01-01-2002
```

または

```
SELECT * FROM Employees WHERE DepartmentId = 2 OR ManagerId = 2
```

ります

```
Id   FName     LName    PhoneNumber   ManagerId   DepartmentId    Salary  Hire_date
CreatedDate   ModifiedDate
3    Michael   Williams 1357911131    1           2               600     12-05-2009    12-05-
```

```
2009    NULL
4    Johnathon Smith    1212121212    2           1               500     24-07-2016    24-07-
2016    01-01-2002
```

HAVINGをして、グループののをチェックする

ID			
1	2	5	100
1	3	2	200
1	4	1	500
2	1	4	50
3	5	6	700

ProductID 2と3のをしたをするには、HAVINGをできます

```
select customerId
from orders
where productID in (2,3)
group by customerId
having count(distinct productID) = 2
```

り

ID
1

せでは、のproductIDをつレコードのみをし、HAVINGをして2つのproductIdsをつグループをチェックします。

のは

```
select customerId
from orders
group by customerId
having sum(case when productID = 2 then 1 else 0 end) > 0
   and sum(case when productID = 3 then 1 else 0 end) > 0
```

このクエリは、productID 2のレコードとproductID 3のレコードのなくとも1つをつグループのみをします。

どこにいるの

のレコードをしますTableNameにするレコードっTableName1。

```
SELECT * FROM TableName t WHERE EXISTS (
    SELECT 1 FROM TableName1 t1 where t.Id = t1.Id)
```

オンラインでWHEREおよびHAVINGをしてをフィルタリングするをむ
https://riptutorial.com/ja/sql/topic/636/whereおよびhavingをしてをフィルタリングする

20: XML

Examples

XMLデータからのクエリ

```
DECLARE @xmlIN XML = '<TableData>
<aaa Main="First">
  <row name="a" value="1" />
  <row name="b" value="2" />
  <row name="c" value="3" />
</aaa>
<aaa Main="Second">
  <row name="a" value="3" />
  <row name="b" value="4" />
  <row name="c" value="5" />
</aaa>
<aaa Main="Third">
  <row name="a" value="10" />
  <row name="b" value="20" />
  <row name="c" value="30" />
</aaa>
</TableData>'

SELECT t.col.value('../@Main', 'varchar(10)') [Header],
t.col.value('@name', 'VARCHAR(25)') [name],
t.col.value('@value',  'VARCHAR(25)') [Value]
FROM    @xmlIn.nodes('//TableData/aaa/row') AS t (col)
```

```
Header    name    Value
First      a        1
First      b        2
First      c        3
Second     a        3
Second     b        4
Second     c        5
Third      a        10
Third      b        20
Third      c        30
```

オンラインでXMLをむ https://riptutorial.com/ja/sql/topic/4421/xml

21: インサート

- INSERT INTO table_namecolumn1、column2、column3、...VALUES1、2、3、...;
- INSERT INTO table_namecolumn1、column2 ...SELECT value1、value2 ... from other_table

Examples

しいを

```
INSERT INTO Customers
VALUES ('Zack', 'Smith', 'zack@example.com', '7049989942', 'EMAIL');
```

このステートメントは、 `Customers`テーブルにしいをします。 `Id`にはがされていないので、にされることにしてください。ただし、そののののはすべてするがあります。

されたのみをする

```
INSERT INTO Customers (FName, LName, Email, PreferredContact)
VALUES ('Zack', 'Smith', 'zack@example.com', 'EMAIL');
```

このステートメントは、 `Customers`テーブルにしいをします。データはされたにのみされます - `PhoneNumber`にがされていないことにしてください。ただし、 `not null`マークされたすべてのをめるがあります。

SELECTをしてのテーブルのデータをINSERTする

```
INSERT INTO Customers (FName, LName, PhoneNumber)
SELECT FName, LName, PhoneNumber FROM Employees
```

このでは、すべてのがCustomersテーブルにされます。 2つのテーブルはなるフィールドをち、すべてのフィールドをしたくないので、するフィールドとするフィールドをするがあります。フィールドはじものとばれるはありませんが、じデータであるがあります。このでは、Idフィールドにアイデンティティがされており、にインクリメントされることをとしています。

にじフィールドをち、すべてのレコードをしたいテーブルが2つあるは、のようにします。

```
INSERT INTO Table1
SELECT * FROM Table2
```

にのをする

1つのコマンドでのをすることができます。

```
INSERT INTO tbl_name (field1, field2, field3)

VALUES (1,2,3), (4,5,6), (7,8,9);
```

のデータをにするには、DBMSのとがです。

MySQL - LOAD DATA INFILE

MSSQL - BULK INSERT

オンラインでインサートをむ https://riptutorial.com/ja/sql/topic/465/インサート

22: インデックス

き

は、デ―タベ―スのせをするために、のでされたのへのポインタをむデ―タです。それらはブックのにています。ここでは、ペ―ジのはペ―ジでけされています。

いくつかのタイプのがし、にすることができます。クエリのWHERE、JOIN、またはORDER BYでされるにインデックスがする、クエリのパフォ―マンスがにします。

は、にづいてのをソ―トすることによってりをするです。

のようなさなデ―タベ―スでは、のはにえませんが、がいとパフォ―マンスがにします。テ―ブルのすべてのをチェックするわりに、サ―バ―はインデックスにしてバイナリをできます。

インデックスをするのトレ―ドオフは、きみとデ―タベ―スサイズです。インデックスをするにはスペ―スがです。また、INSERTがするかがされるたびに、をするがあります。これは、SELECTクエリでテ―ブルをスキャンするのとじくらいなではありませんが、それはまだしておくべきことです。

Examples

インデックスの

```
CREATE INDEX ix_cars_employee_id ON Cars (EmployeeId);
```

これにより、テ―ブル*Cars*の*EmployeeId*のがされます。このインデックスは、のように、サ―バ―に*EmployeeId*のをソ―トまたはするようにするクエリのをさせます。

```
SELECT * FROM Cars WHERE EmployeeId = 1
```

には、のようにのをめることができます。

```
CREATE INDEX ix_cars_e_c_o_ids ON Cars (EmployeeId, CarId, OwnerId);
```

この、は、のがじでけされているは、すべてのをソ―トまたはするようにするにちます。つまり、デ―タをするときに、テ―ブルをべるのではなく、インデックスをしてするをつけることができます。

たとえば、のは2のインデックスをします。

```
SELECT * FROM Cars WHERE EmployeeId = 1 Order by CarId DESC
```

ただし、がなる、インデックスにはのようながありません。

```
SELECT * FROM Cars WHERE OwnerId = 17 Order by CarId DESC
```

`OwnerId = 17`アイテムをつけるために、データベースがEmployeeIdとCarIDのすべてのにわたってインデックスをしなければならないため、インデックスはそれほどではありません。

インデックスはまだされているがありますが、クエリオプティマイザがインデックスをして`OwnerId`でフィルタリングし、なのみをすることが、テーブルがきいはになテーブルをするよりもです

クラスタされた、の、ソートみのインデックス

には、またはのをすることによってできるいくつかのがあります。

```
CREATE CLUSTERED INDEX ix_clust_employee_id ON Employees(EmployeeId, Email);
```

のSQLは、Employeesにしいクラスタインデックスをします。クラスタインデックスは、テーブルののをするインデックスです。テーブルがインデックスのにするようにソートされます。これは、にクラスターされたが1つしかないことをします。クラスター化インデックスがすでにテーブルにする、のステートメントはします。 クラスタされたインデックスのないテーブルは、ヒープともばれます。

```
CREATE UNIQUE INDEX uq_customers_email ON Customers(Email);
```

これにより、テーブル*Customers*の*Email*ののインデックスがされます。このインデックスは、のインデックスのようにクエリをするとともに、そののすべてのメールアドレスをにするようします。がでないメールでまたはされた、またははデフォルトでします。

```
CREATE UNIQUE INDEX ix_eid_desc ON Customers(EmployeeID);
```

これにより、EmployeeIDがでなければならないテーブルもされる、Customersのインデックスがされます。 がでないはします。このは、IDをするがいるです。

```
CREATE INDEX ix_eid_desc ON Customers(EmployeeID Desc);
```

これにより、でソートされたがされます。デフォルトでは、インデックスはMSSQLサーバではなくともですが、することができます。

のインデックスをする

```
UPDATE Customers SET Email = "richard0123@example.com" WHERE id = 1;
```

*Customers*の *Email*にユニークなインデックスがされていると、これはします。ただし、この、

のをできます。

```
UPDATE Customers SET Email = "richard0123@example.com" WHERE id = 1 ON DUPLICATE KEY;
```

SAP ASEドロップインデックス

このコマンドは、テーブルのインデックスをします。これは`SAP ASE`サーバでします。

```
DROP INDEX [table name].[index name]
```

```
DROP INDEX Cars.index_1
```

べえられたインデックス

をするでソートされたをすると、に`SELECT`はソートをいません。

```
CREATE INDEX ix_scoreboard_score ON scoreboard (score DESC);
```

クエリをすると

```
SELECT * FROM scoreboard ORDER BY score DESC;
```

データベースシステムは、そのでインデックスルックアップをうことができるので、ソートはいません。

の、またはのと

```
DROP INDEX ix_cars_employee_id ON Cars;
```

`DROP`コマンドをってインデックスをすることができます。このではします`DROP`テーブルのに*ix_cars_employee_id*とばれるを。

これにより、インデックスがにされ、インデックスがクラスタされているは、クラスタリングがされます。をせずにすることはできません。くてコストがかかるがあります。また、インデックスをにすることもできます。

```
ALTER INDEX ix_cars_employee_id ON Cars DISABLE;
```

これにより、にするメタデータとともにをすることができます。

なとして、これはをしているため、をにすることができます。があれば、はでにするのではなく、することができます。

```
ALTER INDEX ix_cars_employee_id ON Cars REBUILD;
```

NULLをする

```
CREATE UNIQUE INDEX idx_license_id
   ON Person(DrivingLicenseID) WHERE DrivingLicenseID IS NOT NULL
GO
```

このスキ―マは0..1のをにします - 々はゼロまたは1つのをつことができ、は1にしかしません

インデックスの

Bツリ―は、デ―タの//のためにすることがあります。 SQLServerのでは、インデックスペ―ジはですとペ―ジはにしていませんをつことができます。のは、のとににによくています。

インデックスをするには

```
ALTER INDEX index_name REBUILD;
```

デフォルトでは、のはオフラインであり、をロックしてDMLをしますが、くのRDBMSではオンラインがです。また、のDBベンダ―は、 `REORGANIZE` SQLServerや`COALESCE` / `SHRINK SPACE` Oracleなどのインデックスのをしています。

クラスタインデックス

クラスタインデックスをする、テ―ブルのは、クラスタインデックスがされているによってソ―トされます。したがって、2つのなるでをべえることができないため、にはクラスタされたは1つしかできません。

に、きなデ―タ･テ―ブルでのみをするは、クラスタ―ド･インデックスをするのがです。クラスタ―ド･インデックスのは、にきむにデ―タをするがあるりになることです。

にクラスタ―ド･インデックスをするEmployee_Surnameの

```
CREATE CLUSTERED INDEX ix_employees_name ON Employees(Employee_Surname);
```

クラスタされていないインデックス

ノンクラスタ―ド･インデックスは、テ―ブルとはにされます。こののインデックスは、それがすテ―ブルのへのポインタをみます。

このポインタはロケ―タとばれます。ロケ―タ―のは、デ―タ･ペ―ジがヒ―プまたはクラスタ―にされているかどうかによってなります。ヒ―プの、ロケ―タはへのポインタです。クラスタされたの、ロケ―タはクラスタされたキ―です。

EmployeesおよびEmployee_Surnameにクラスタインデックスをする

```
CREATE NONCLUSTERED INDEX ix_employees_name ON Employees(Employee_Surname);
```

テーブルにのノンクラスタード・インデックスがすることがあります。みりは、クラスタインデックスよりもクラスタされていないインデックスでは、にインデックスをするがあるため、テーブルよりもくなります。ただし、きみにははありません。

インデックスまたはフィルタリングインデックス

SQL ServerとSQLiteでは、のサブセットだけでなく、のサブセットもむをできます。

`order_state_id`が`finished2`に`order_state_id equal 、 order_state_id equal`が`started1`に`order_state_id equal`したをつがにしているとえてください。

あなたのビジネスがのようなクエリをしている

```
SELECT id, comment
  FROM orders
 WHERE order_state_id =  1
   AND product_id = @some_value;
```

なインデックスでは、のだけをめてインデックスをできます。

```
CREATE INDEX Started_Orders
          ON orders(product_id)
       WHERE order_state_id = 1;
```

このインデックスは、フィルタリングされていないインデックスよりもさくなり、スペースがされ、インデックスのコストがされます。

オンラインでインデックスをむ https://riptutorial.com/ja/sql/topic/344/インデックス

23: ウィンドウ

Examples

すべてのにされたをする

```
SELECT your_columns, COUNT(*) OVER() as Ttl_Rows FROM your_data_set
```

id		Ttl_Rows
1		5
2	foo	5
3	バー	5
4	バズ	5
5	クックス	5

2つのクエリをしてをするわりに、をウィンドウとしてし、なセットをウィンドウとしてできます。
これはなのさなしに、さらなるのためのベースとしてすることができます。

のがのプロパティをつにフラグをする

はこのデータをっているとしましょう

テーブル

id		タグ
1		unique_tag
2	foo	
42	バー	
3	バズ	こんにちは
51	クックス	

はすべてのをし、タグがのでされているかどうかをりたい

```
SELECT id, name, tag, COUNT(*) OVER (PARTITION BY tag) > 1 AS flag FROM items
```

はのようになります。

id		タグ	フラグ
1		unique_tag	
2	foo		
42	バー		
3	バズ	こんにちは	
51	クックス		

データベースにOVERとPARTITIONがない、これをしてじをることができます

```
SELECT id, name, tag, (SELECT COUNT(tag) FROM items B WHERE tag = A.tag) > 1 AS flag FROM
items A
```

のをする

このデータがえられた

2016-03-12	200
2016-03-11	-50
2016-03-14	100
2016-03-15	100
2016-03-10	-250

```
SELECT date, amount, SUM(amount) OVER (ORDER BY date ASC) AS running
FROM operations
ORDER BY date ASC
```

あなたをえるだろう

		ランニング
2016-03-10	-250	-250
2016-03-11	-50	-300

		ランニング
2016-03-12	200	-100
2016-03-14	100	0
2016-03-15	100	-100

のグループにまたがってNののをする

このデータ

ユーザーID	
1	2016-07-20
1	2016-07-21
2	2016-07-20
2	2016-07-21
2	2016-07-22

```
;with CTE as
(SELECT *,
        ROW_NUMBER() OVER (PARTITION BY User_ID
                           ORDER BY Completion_Date DESC) Row_Num
FROM    Data)
SELECT * FORM CTE WHERE Row_Num <= n
```

n = 1をすると、 `user_id`ごとにのが1つされ`user_id`。

ユーザーID		Row_Num
1	2016-07-21	1
2	2016-07-22	1

LAGをして「シーケンス」レコードをする

これらのサンプルデータがえられた

ID		STATUS_TIME	STATUS_BY
1	1	2016-09-28-19.47.52.501398	USER_1
3	1	2016-09-28-19.47.52.501511	USER_2

ID		STATUS_TIME	STATUS_BY
1		2016-09-28-19.47.52.501517	USER_3
3		2016-09-28-19.47.52.501521	USER_2
3		2016-09-28-19.47.52.501524	USER_4

`ID`でされるは、ステータスをスキップすることなく、 `STATUS` 'ONE'から 'TWO'に 'THREE'ににするがあります。は、ルールにし、すぐに 'ONE'から 'THREE'にするユーザー `STATUS_BY` のをつけることです。

`LAG()`は、ののののをすことでをするのにちます。

```
SELECT * FROM (
 SELECT
  t.*,
  LAG(status) OVER (PARTITION BY id ORDER BY status_time) AS prev_status
  FROM test t
) t1 WHERE status = 'THREE' AND prev_status != 'TWO'
```

あなたのデータベースにLAGがない、これをってじをることができます

```
SELECT A.id, A.status, B.status as prev_status, A.status_time, B.status_time as
prev_status_time
FROM Data A, Data B
WHERE A.id = B.id
AND   B.status_time = (SELECT MAX(status_time) FROM Data where status_time < A.status_time and
id = A.id)
AND   A.status = 'THREE' AND NOT B.status = 'TWO'
```

オンラインでウィンドウをむ https://riptutorial.com/ja/sql/topic/647/ウィンドウ

24: カスケード

Examples

カスケードをする

をするアプリケーションがあるとします。
さらに、アプリケーションがクライアントごとにするとしますテナント。
のクライアントがあります。
したがって、データベースにはクライアントのテーブルとルームのテーブルがまれます。

、すべてのクライアントにはNのルームがあります。

これは、クライアントテーブルをして、ルームテーブルにキーがあることをします。

```
ALTER TABLE dbo.T_Room  WITH CHECK ADD  CONSTRAINT FK_T_Room_T_Client FOREIGN KEY(RM_CLI_ID)
REFERENCES dbo.T_Client (CLI_ID)
GO
```

クライアントがのソフトウェアにしたとすると、ソフトウェアのデータをするがあります。しかし、もしあなたが

```
DELETE FROM T_Client WHERE CLI_ID = x
```

それで、があるときにクライアントをすることはできないので、キーがします。

これで、クライアントをするにクライアントルームをするコードをアプリケーションにすることになりました。アプリケーションのがされるため、さらにくのキーがデータベースにされるとしてください。ろしい。データベースのすべてのについて、アプリケーションのコードをNにするがあります。おそらく、のアプリケーションえば、のシステムとのインターフェースでもコードをさせるがあります。

コードでうよりもれたがあります。
キーに`ON DELETE CASCADE`をすることができます。

```
ALTER TABLE dbo.T_Room  -- WITH CHECK -- SQL-Server can specify WITH CHECK/WITH NOCHECK
ADD  CONSTRAINT FK_T_Room_T_Client FOREIGN KEY(RM_CLI_ID)
REFERENCES dbo.T_Client (CLI_ID)
ON DELETE CASCADE
```

あなたはうことができる

```
DELETE FROM T_Client WHERE CLI_ID = x
```

クライアントがされると、はにされます。

- アプリケーションコードのなし。

の1つのMicrosoft SQL Serverでは、それをするテーブルがある、これはしません。したがって、のように、ツリーでカスケードをしようとすると、

```
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_T_FMS_Navigation_T_FMS_Navigation]') AND parent_object_id =
OBJECT_ID(N'[dbo].[T_FMS_Navigation]'))
ALTER TABLE [dbo].[T_FMS_Navigation]  WITH CHECK ADD  CONSTRAINT
[FK_T_FMS_Navigation_T_FMS_Navigation] FOREIGN KEY([NA_NA_UID])
REFERENCES [dbo].[T_FMS_Navigation] ([NA_UID])
ON DELETE CASCADE
GO

IF  EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_T_FMS_Navigation_T_FMS_Navigation]') AND parent_object_id =
OBJECT_ID(N'[dbo].[T_FMS_Navigation]'))
ALTER TABLE [dbo].[T_FMS_Navigation] CHECK CONSTRAINT [FK_T_FMS_Navigation_T_FMS_Navigation]
GO
```

Microsoft-SQLサーバーでは、なツリーで`ON DELETE CASCADE`してキーをすることができないため、しません。このの1つは、ツリーがするがあり、それがデッドロックにつながるがあるということです。

これにし、PostgreSQLはこれをうことができます。
がであることがである。
ツリーがなは、エラーがします。
そのは、でdeleteをするだけです。

の
つまり、クライアントテーブルをにしてすることはできません。これをうと、「T_Room」...のすべてのエントリがされるためですデルタはもうなくなります

オンラインでカスケードをむ https://riptutorial.com/ja/sql/topic/3518/カスケード

25: クロス、

Examples

CROSS APPLYとOUTER APPLYの

のでテーブルがされているときにがされます。

にするをするテーブルをします。その、にするをするEmployeeテーブルをします。はにしているため、テーブルにはDepartmentテーブルとのがあります。

のクエリはテーブルからデータをし、CROSS APPLYをしてDepartmentテーブルのレコードのEmployeeテーブルをします。 2のクエリは、DepartmentテーブルをEmployeeテーブルにするだけで、すべてのするレコードがされます。

```
SELECT *
FROM Department D
CROSS APPLY (
    SELECT *
    FROM Employee E
    WHERE E.DepartmentID = D.DepartmentID
) A
GO
SELECT *
FROM Department D
INNER JOIN Employee E
  ON D.DepartmentID = E.DepartmentID
```

したをると、まったくじセットになります。 JOINとどのようにうのですかまた、よりなクエリをくでどのようにちますか

スクリプト2ののクエリは、Departmentテーブルのデータをし、OUTER APPLYをしてDepartmentテーブルのレコードのEmployeeテーブルをします。 Employeeテーブルにしないについては、5と6のにられるように、これらのにNULLがまれます。2のクエリは、DepartmentテーブルとEmployeeテーブルのにLEFT OUTER JOINをするだけです。どおり、クエリはテーブルからすべてのをします。 Employeeテーブルにするものがないであってもです。

```
SELECT *
FROM Department D
OUTER APPLY (
    SELECT *
    FROM Employee E
    WHERE E.DepartmentID = D.DepartmentID
) A
GO
SELECT *
FROM Department D
LEFT OUTER JOIN Employee E
  ON D.DepartmentID = E.DepartmentID
```

```
GO
```

の2つのクエリがじをしても、はしなります。しかし、コストはあまりいはありません。

、APPLYがになをするがる。 Script3では、DepartmentIDをパラメータとしてけり、このにするすべてのをすテーブルをしています。のクエリは、Departmentテーブルからデータをし、CROSS APPLYをして、したとします。テーブルここではDepartmentテーブルからのDepartmentIDをし、サブクエリにたのをします。のは、CROSS APPLYのわりにOUTER APPLYをするため、データのみをしたCROSS APPLYとはなり、OUTER APPLYはのデータもし、したにNULLをれます。

```
CREATE FUNCTION dbo.fn_GetAllEmployeeOfADepartment (@DeptID AS int)
RETURNS TABLE
AS
  RETURN
  (
  SELECT
    *
  FROM Employee E
  WHERE E.DepartmentID = @DeptID
  )
GO
SELECT
  *
FROM Department D
CROSS APPLY dbo.fn_GetAllEmployeeOfADepartment(D.DepartmentID)
GO
SELECT
  *
FROM Department D
OUTER APPLY dbo.fn_GetAllEmployeeOfADepartment(D.DepartmentID)
GO
```

にっているは、のクエリのわりになをできますかのクエリのCROSS / OUTER APPLYをINNER JOIN / LEFT OUTER JOINにきえ、ON1 = 1をしてクエリをすると、「マルチパート」がられます。 D.DepartmentIDは「バインドできませんでした」エラー。 JOINでは、クエリのコンテキストがまたはテーブルのコンテキストとなるため、クエリの/をパラメータとしてにバインドすることができないためです。したがって、そのようなにはAPPLYがです。

オンラインでクロス、をむ https://riptutorial.com/ja/sql/topic/2516/クロス-

26: コメント

Examples

1コメント

のコメントのには--があり、のわりにく

```
SELECT *
FROM Employees -- this is a comment
WHERE FName = 'John'
```

のコメント

のコードコメントは/* ... */ラップされ/* ... */

```
/* This query
   returns all employees */
SELECT *
FROM Employees
```

このようなコメントをのにすることもできます

```
SELECT /* all columns: */ *
FROM Employees
```

オンラインでコメントをむ https://riptutorial.com/ja/sql/topic/1597/コメント

27: サブクエリ

せは、せのなるで、またはsetでできます。

それらはかっこ()むがあります。サブクエリのがのものとされるは、のがするがあります。テンポラリテーブルにをけるには、FROMのサブクエリにテーブルエイリアスがです。

Examples

WHEREのせ

サブクエリをしてセットをフィルタリングします。たとえば、にしいをつすべてのをします。

```
SELECT *
FROM Employees
WHERE Salary = (SELECT MAX(Salary) FROM Employees)
```

FROMのいわせ

FROMのせは、せのにされたにわれるとにします。

```
SELECT Managers.Id, Employees.Salary
FROM (
  SELECT Id
  FROM Employees
  WHERE ManagerId IS NULL
) AS Managers
JOIN Employees ON Managers.Id = Employees.Id
```

SELECTのいわせ

```
SELECT
  Id,
  FName,
  LName,
  (SELECT COUNT(*) FROM Cars WHERE Cars.CustomerId = Customers.Id) AS NumberOfCars
FROM Customers
```

FROMのサブクエリ

サブクエリをして、テーブルをし、それを「」クエリのFROMでできます。

```
SELECT * FROM (SELECT city, temp_hi - temp_lo AS temp_var FROM weather) AS w
WHERE temp_var > 20;
```

は、のが20よりきいからをします 。はのとおりです。

シティ	temp_var
セントルイス	21
ロサンゼルス	31
ロサンゼルス	23
ロサンゼルス	31
ロサンゼルス	27
ロサンゼルス	28
ロサンゼルス	28
ロサンゼルス	32

。

WHEREのサブクエリ

のでは、がなをる の をしますサブqqueryで。

```
SELECT name, pop2000 FROM cities
WHERE pop2000 < (SELECT avg(pop2000)  FROM cities);
```

ここでは、サブクエリSELECT avgpop2000FROM citiesをしてWHEREのをします。はのとおりです。

	ポップ2000
サンフランシスコ	776733
セントルイス	348189
カンザスシティー	146866

SELECTのサブクエリ

せは、せのSELECTでもできます。のクエリでは、 weatherテーブルのすべてのとcitiesテーブルのするがされます 。

```
SELECT w.*,  (SELECT c.state FROM cities AS c WHERE c.name = w.city ) AS state
FROM weather AS w;
```

なるテーブルのクエリをしてクエリをフィルタリングする

このクエリは、スーパーバイザテーブルにないすべてのをします。

```
SELECT *
FROM Employees
WHERE EmployeeID not in (SELECT EmployeeID
                         FROM Supervisors)
```

LEFT JOINをしてもじがられます。

```
SELECT *
FROM Employees AS e
LEFT JOIN Supervisors AS s ON s.EmployeeID=e.EmployeeID
WHERE s.EmployeeID is NULL
```

サブクエリ

SynchronizedまたはCoordinatedともばれますサブクエリは、クエリののをするネストされたクエリです。

```
SELECT EmployeeId
    FROM Employee AS eOuter
    WHERE Salary > (
       SELECT AVG(Salary)
       FROM Employee eInner
       WHERE eInner.DepartmentId = eOuter.DepartmentId
    )
```

せ`SELECT AVG(Salary) ...`は、そのせから`EmployeeeOuter`するため、 にあります。

オンラインでサブクエリをむ https://riptutorial.com/ja/sql/topic/1606/サブクエリ

28: シーケンス

Examples

シーケンスの

```
CREATE SEQUENCE orders_seq
START WITH     1000
INCREMENT BY   1;
```

が1000でシーケンスが1つえたシーケンスをします。

シーケンスの

seq_name .NEXTVALへのは、シーケンスののをするためにされます。 1つのでは、1つのシーケンスしかできません。ステートメントにNEXTVALへのがある、それらはじをします。

INSERTにはNEXTVALをできます

```
INSERT INTO Orders (Order_UID, Customer)
        VALUES (orders_seq.NEXTVAL, 1032);
```

UPDATESにできます

```
UPDATE Orders
SET Order_UID = orders_seq.NEXTVAL
WHERE Customer = 581;
```

SELECTSにもできます

```
SELECT Order_seq.NEXTVAL FROM dual;
```

オンラインでシーケンスをむ https://riptutorial.com/ja/sql/topic/1586/シーケンス

29: ジョイン

き

JOINは、2つのテーブルのをするです。は、タイプINNER / OUTER / CROSSおよびLEFT / RIGHT / FULL、でしますとのテーブルのがどのようにしているかによってされるのテーブルのステッチセットです。

テーブルは、それまたはのテーブルにすることができます。 3つのテーブルのにアクセスするがあるは、のをFROMでできます。

- `[ { INNER | { { LEFT | RIGHT | FULL } [ OUTER ] } } ] JOIN`

ジョインは、そのがすように、のテーブルのデータをクエリするであり、にはのテーブルからしたがされます。

Examples

な

「」ともばれるは、2つののデータを`join`でされたでします。

のでは、Employeesテーブルからのファーストネームを、Departmentsテーブルから Nameするのをします。

```
SELECT Employees.FName, Departments.Name
FROM   Employees
JOIN   Departments
ON Employees.DepartmentId = Departments.Id
```

これはサンプルデータベースからのものをします

.FName	Departments.Name
ジェームス	HR
ジョン	HR
リチャード	

な

は、 `from`にのをち、カンマでり, `where`でそれらのをすることによってもできます。このはとばれますには`join`はまれていない`join`。

すべてのRDBMSがそれをサポートしていますが、ははされていません。このをすることがいえであるはのとおりです。

- ったをしてしまうのクロスをすることはです。に、クエリにのがあるは、
- クロスジョインをしていたは、からからかではなくわりにCROSS JOINをきます、メンテナンスにかがそれをするがあります。

のでは、のファーストネームと、がいているのをします。

```
SELECT e.FName, d.Name
FROM   Employee e, Departments d
WHERE  e.DeptartmentId = d.Id
```

これはサンプルデータベースからのものをします

e.FName	d.Name
ジェームス	HR
ジョン	HR
リチャード	

またはともばれますは、テーブルのすべてのがにされるです。ののするがしない、するフィールドは`NULL`です。

のでは、すべてのとそのでくのファーストネームをします。のいないはとしてにされますが、はNULLになります。

```
SELECT          Departments.Name, Employees.FName
FROM            Departments
LEFT OUTER JOIN Employees
ON              Departments.Id = Employees.DepartmentId
```

これはサンプルデータベースからのものをします

Departments.Name	.FName
HR	ジェームス
HR	ジョン
HR	ジョンナトン
	マイケル
テック	ヌル

それではどうやってくの

FROMには2つのテーブルがあります。

イド	FName	LName		マネージャーID	DepartmentId		HireDate
1	ジェームス	スミス	1234567890	ヌル	1	1000	01-01-2002
2	ジョン	ジョンソン	2468101214	1	1	400	23-03-2005
3	マイケル	ウィリアムズ	1357911131	1	2	600	12-05-2009
4	ジョンナトン	スミス	1212121212	2	1	500	24-07-2016

そして

イド	
1	HR
2	
3	テック

に、テーブルをえる2つのテーブルからデカルトがされる。
をたすレコード *Departments.Id = Employees.DepartmentId* はでされています。これらはクエリののにされます。

これはLEFT OUTER JOINであるため、すべてのレコードはのLEFTDepartmentsからされますが、RIGHTのレコードはとしないはNULLマーカーがえられます。のでは、これは`NULL` **Tech**を`NULL`

イド		イド	FName	LName		マネージャーID	DepartmentId		HireDate
1	**HR**	**1**	ジェームス	スミス	**1234567890**	ヌル	**1**	**1000**	**01-01-2002**
1	**HR**	**2**	ジョン	ジョンソン	**2468101214**	**1**	**1**	**400**	**23-03-2005**
1	HR	3	マイケル	ウィリアムズ	1357911131	1	2	600	12-05-2009
1	**HR**	**4**	ジョンナトン	スミス	**1212121212**	**2**	**1**	**500**	**24-07-2016**
2		1	ジェームス	スミス	1234567890	ヌル	1	1000	01-01-2002
2		2	ジョン	ジョンソン	2468101214	1	1	400	23-03-2005
2		**3**	マイケル	ウィリアムズ	**1357911131**	**1**	**2**	**600**	**12-05-2009**
2		4	ジョンナトン	スミス	1212121212	2	1	500	24-07-2016
3	テック	1	ジェームス	スミス	1234567890	ヌル	1	1000	01-01-2002
3	テック	2	ジョン	ジョンソン	2468101214	1	1	400	23-03-2005
3	テック	3	マイケル	ウィリアムズ	1357911131	1	2	600	12-05-2009
3	テック	4	ジョンナトン	スミス	1212121212	2	1	500	24-07-2016

に、 **SELECT**でされるは、テーブルをすようにされます。

Departments.Name	.FName
HR	ジェームス
HR	ジョン
	リチャード
テック	ヌル

テーブルは、それにされていてもよい。このでは、の2つのをするためにエイリアスをするがあります。

のでは、 サンプルデータベースEmployeesテーブルの Employeeごとに、のマネージャのするファーストネームとともにのをむレコードがされます。マネージャーもであるため、テーブルはです。

```
SELECT
    e.FName AS "Employee",
    m.FName AS "Manager"
FROM
    Employees e
JOIN
    Employees m
    ON e.ManagerId = m.Id
```

このクエリは、のデータをします。

	マネージャー
ジョン	ジェームス
マイケル	ジェームス
ジョンナトン	ジョン

それではどうやってくの

のにはのレコードがまれています。

イド	FName	LName		マネージャーID	DepartmentId		HireDate
1	ジェーム	スミス	1234567890	ヌル	1	1000	01-01-

イド	FName	LName		マネージャーID	DepartmentId		HireDate
	ス						2002
2	ジョン	ジョンソン	2468101214	1	1	400	23-03-2005
3	マイケル	ウィリアムズ	1357911131	1	2	600	12-05-2009
4	ジョンナトン	スミス	1212121212	2	1	500	24-07-2016

のアクションは、 **FROM**でされるテーブルのすべてのレコードのデカルトをすることです。この、Employeesテーブルが2あるので、テーブルはのようになりますこのではされていないフィールドはすべてしています。

e.Id	e.FName	e.ManagerId	m.Id	m.FName	m.ManagerId
1	ジェームス	ヌル	1	ジェームス	ヌル
1	ジェームス	ヌル	2	ジョン	1
1	ジェームス	ヌル	3	マイケル	1
1	ジェームス	ヌル	4	ジョンナトン	2
2	ジョン	1	1	ジェームス	ヌル
2	ジョン	1	2	ジョン	1
2	ジョン	1	3	マイケル	1
2	ジョン	1	4	ジョンナトン	2
3	マイケル	1	1	ジェームス	ヌル
3	マイケル	1	2	ジョン	1
3	マイケル	1	3	マイケル	1
3	マイケル	1	4	ジョンナトン	2
4	ジョンナトン	2	1	ジェームス	ヌル
4	ジョンナトン	2	2	ジョン	1

e.Id	e.FName	e.ManagerId	m.Id	m.FName	m.ManagerId
4	ジョンナトン	2	3	マイケル	1
4	ジョンナトン	2	4	ジョンナトン	2

のアクションは、 **JOIN**をたすレコードのみをすることです。エイリアスされた`e`テーブルの`ManagerId`がエイリアスされた`m`テーブル`Id`としいレコード

e.Id	e.FName	e.ManagerId	m.Id	m.FName	m.ManagerId
2	ジョン	1	1	ジェームス	ヌル
3	マイケル	1	1	ジェームス	ヌル
4	ジョンナトン	2	2	ジョン	1

に、 **SELECT**でされるがされ、このがされます。

e.FName	m.FName
ジョン	ジェームス
マイケル	ジェームス
ジョンナトン	ジョン

に、`e.FName`と`m.FName`は、 AS でりてられたエイリアスにきえられます。

	マネージャー
ジョン	ジェームス
マイケル	ジェームス
ジョンナトン	ジョン

クロスジョイン

クロス·ジョインは、2つのメンバーのデカルトをいます。デカルトは、1つののがの2ののとされることをします。たとえば、 `TABLEA`に20、 `TABLEB`に20がある、は`20*20 = 400`になります。

サンプルデータベースの

```
SELECT d.Name, e.FName
FROM   Departments d
```

```
CROSS JOIN Employees e;
```

すもの

d.Name	e.FName
HR	ジェームス
HR	ジョン
HR	マイケル
HR	ジョンナトン
	ジェームス
	ジョン
	マイケル
	ジョンナトン
テック	ジェームス
テック	ジョン
テック	マイケル
テック	ジョンナトン

デカルトをいたいは、なCROSS JOINをくことをおめします。

サブクエリへの

サブクエリのは、/テーブルからデータをし、それをテーブルまたはヘッダーテーブルのレコードとともにするによくされます。たとえば、レコードの、レコードのの、またはまたはフィールドにづくまたはのをすることができます。このでは、エイリアスをしています。これにより、のテーブルがわっているに、クエリがみやすくなります。かなりなサブクエリのようなものがあります。この、テーブルPurchase Ordersからすべてのをし、テーブルPurchaseOrderLineItemsのレコードののののみをしています。

```
SELECT po.Id, po.PODate, po.VendorName, po.Status, item.ItemNo,
  item.Description, item.Cost, item.Price
FROM PurchaseOrders po
LEFT JOIN
     (
       SELECT l.PurchaseOrderId, l.ItemNo, l.Description, l.Cost, l.Price, Min(l.id) as Id
       FROM PurchaseOrderLineItems l
```

```
      GROUP BY l.PurchaseOrderId, l.ItemNo, l.Description, l.Cost, l.Price
    ) AS item ON item.PurchaseOrderId = po.Id
```

されているされていない

にいタイプのJOINは、LATERAL JOINPostgreSQL 9.3のです。
これはSQL-ServerOracleのCROSS APPLY / OUTER APPLYともばれます。

なえは、テーブルまたはインラインサブクエリが、するすべてのにされるということです。

これにより、たとえば、にするエントリだけをのテーブルにすることができます。
のとののいは、にしたサブクエリにしたをできるというにあります。

PostgreSQL 9.3+

||JOIN **LATERAL**

SQLサーバー

クロス| OUTERがされます

`INNER JOIN LATERAL`は`CROSS APPLY`とじです
`LEFT JOIN LATERAL`は`OUTER APPLY`とじです

PostgreSQL 9.3

```
SELECT * FROM T_Contacts

--LEFT JOIN T_MAP_Contacts_Ref_OrganisationalUnit ON MAP_CTCOU_CT_UID = T_Contacts.CT_UID AND
MAP_CTCOU_SoftDeleteStatus = 1
--WHERE T_MAP_Contacts_Ref_OrganisationalUnit.MAP_CTCOU_UID IS NULL -- 989


LEFT JOIN LATERAL
(
    SELECT
         --MAP_CTCOU_UID
         MAP_CTCOU_CT_UID
        ,MAP_CTCOU_COU_UID
        ,MAP_CTCOU_DateFrom
        ,MAP_CTCOU_DateTo
   FROM T_MAP_Contacts_Ref_OrganisationalUnit
   WHERE MAP_CTCOU_SoftDeleteStatus = 1
   AND MAP_CTCOU_CT_UID = T_Contacts.CT_UID

    /*
    AND
    (
        (__in_DateFrom <= T_MAP_Contacts_Ref_OrganisationalUnit.MAP_KTKOE_DateTo)
        AND
        (__in_DateTo >= T_MAP_Contacts_Ref_OrganisationalUnit.MAP_KTKOE_DateFrom)
    )
    */
   ORDER BY MAP_CTCOU_DateFrom
```

```
   LIMIT 1
) AS FirstOE
```

SQL Serverの

```
SELECT * FROM T_Contacts

--LEFT JOIN T_MAP_Contacts_Ref_OrganisationalUnit ON MAP_CTCOU_CT_UID = T_Contacts.CT_UID AND
MAP_CTCOU_SoftDeleteStatus = 1
--WHERE T_MAP_Contacts_Ref_OrganisationalUnit.MAP_CTCOU_UID IS NULL -- 989

-- CROSS APPLY -- = INNER JOIN
OUTER APPLY    -- = LEFT JOIN
(
    SELECT TOP 1
         --MAP_CTCOU_UID
         MAP_CTCOU_CT_UID
        ,MAP_CTCOU_COU_UID
        ,MAP_CTCOU_DateFrom
        ,MAP_CTCOU_DateTo
   FROM T_MAP_Contacts_Ref_OrganisationalUnit
   WHERE MAP_CTCOU_SoftDeleteStatus = 1
   AND MAP_CTCOU_CT_UID = T_Contacts.CT_UID

    /*
    AND
    (
        (@in_DateFrom <= T_MAP_Contacts_Ref_OrganisationalUnit.MAP_KTKOE_DateTo)
        AND
        (@in_DateTo >= T_MAP_Contacts_Ref_OrganisationalUnit.MAP_KTKOE_DateFrom)
    )
    */
   ORDER BY MAP_CTCOU_DateFrom
) AS FirstOE
```

フル・ジョイン

あまりられていないJOINの1つのタイプは、FULL JOINです。
FULL JOINは2016のMySQLではサポートされていません

FULL OUTER JOINは、ののすべてのと、ののすべてのをします。

のテーブルに、のテーブルにするがないや、のテーブルにするがないは、それらのもリストされます。

1

```
SELECT * FROM Table1

FULL JOIN Table2
     ON 1 = 2
```

2

```
SELECT
     COALESCE(T_Budget.Year, tYear.Year) AS RPT_BudgetInYear
    ,COALESCE(T_Budget.Value, 0.0) AS RPT_Value
FROM T_Budget

FULL JOIN tfu_RPT_All_CreateYearInterval(@budget_year_from, @budget_year_to) AS tYear
      ON tYear.Year = T_Budget.Year
```

ソフトをしているは、WHEREでソフトステータスをチェックするがありますFULL JOINは UNIONのようにするため。
あなたはAP_SoftDeleteStatus = 1をjoinにくので、このさなをとすのはです。

また、FULL JOINをしているは、、WHEREにNULLをするがあります。にNULLをするのをれると、INNERとじがあります。これは、FULL JOINをしているにましくないものです。

```
SELECT
     T_AccountPlan.AP_UID
    ,T_AccountPlan.AP_Code
    ,T_AccountPlan.AP_Lang_EN
    ,T_BudgetPositions.BUP_Budget
    ,T_BudgetPositions.BUP_UID
    ,T_BudgetPositions.BUP_Jahr
FROM T_BudgetPositions

FULL JOIN T_AccountPlan
    ON T_AccountPlan.AP_UID = T_BudgetPositions.BUP_AP_UID
    AND T_AccountPlan.AP_SoftDeleteStatus = 1

WHERE (1=1)
AND (T_BudgetPositions.BUP_SoftDeleteStatus = 1 OR T_BudgetPositions.BUP_SoftDeleteStatus IS
NULL)
AND (T_AccountPlan.AP_SoftDeleteStatus = 1 OR T_AccountPlan.AP_SoftDeleteStatus IS NULL)
```

な

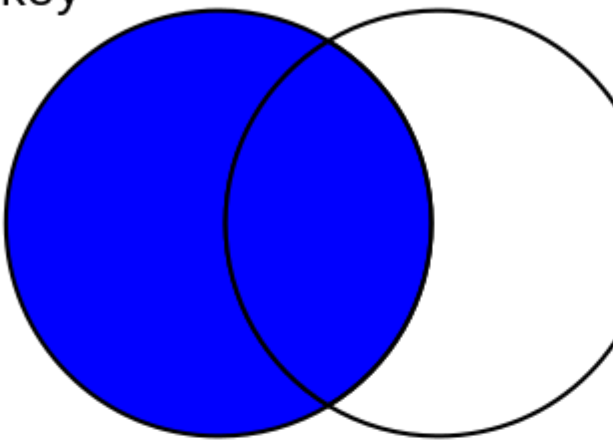
は、データをするためによくされます。 SQLでは、これらはなテーブルでされます。たとえば、のようになります。

```
WITH RECURSIVE MyDescendants AS (
    SELECT Name
    FROM People
    WHERE Name = 'John Doe'

    UNION ALL

    SELECT People.Name
    FROM People
    JOIN MyDescendants ON People.Name = MyDescendants.Parent
)
SELECT * FROM MyDescendants;
```

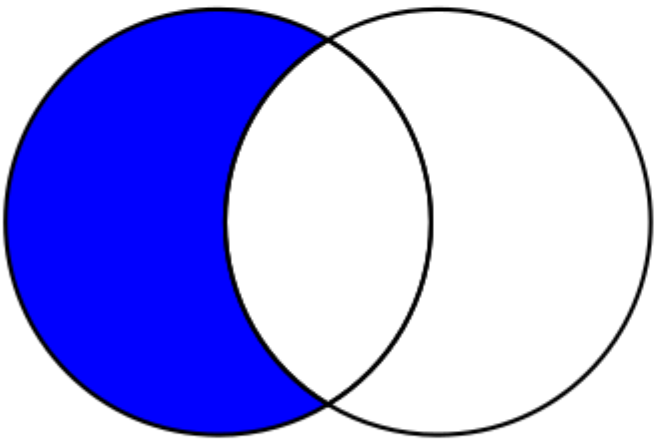
ののい

SQLには、するがにまれるかどうか `INNER JOIN`、 `LEFT OUTER JOIN`、 `RIGHT OUTER JOIN`、および `FULL OUTER JOIN` `INNER`および`OUTER`キーワードはオプションをするための々なタイプがあります。のは、これらのタイプののいをしています。のはによってされるをし、いはがらないをします。

```
SELECT <fields>
FROM TableA  A
LEFT JOIN TableB  B
ON A.key = B.key
```



```
SELECT <fields>
FROM TableA  A
LEFT JOIN TableB  B
ON A.key = B.key
WHERE B.key IS NULL
```



えます。

```
select * from a INNER JOIN b on a.a = b.b;
select a.*,b.* from a,b where a.a = b.b;

a | b
--+--
3 | 3
4 | 4
```

は、Aのすべてのと Bのをえます。

```
select * from a LEFT OUTER JOIN b on a.a = b.b;

a |  b
--+-----
1 | null
2 | null
3 |    3
4 |    4
```

に、により、Bのすべてのと、Aのがられます。

```
select * from a RIGHT OUTER JOIN b on a.a = b.b;

a    |  b
-----+----
3    |  3
4    |  4
null |  5
null |  6
```

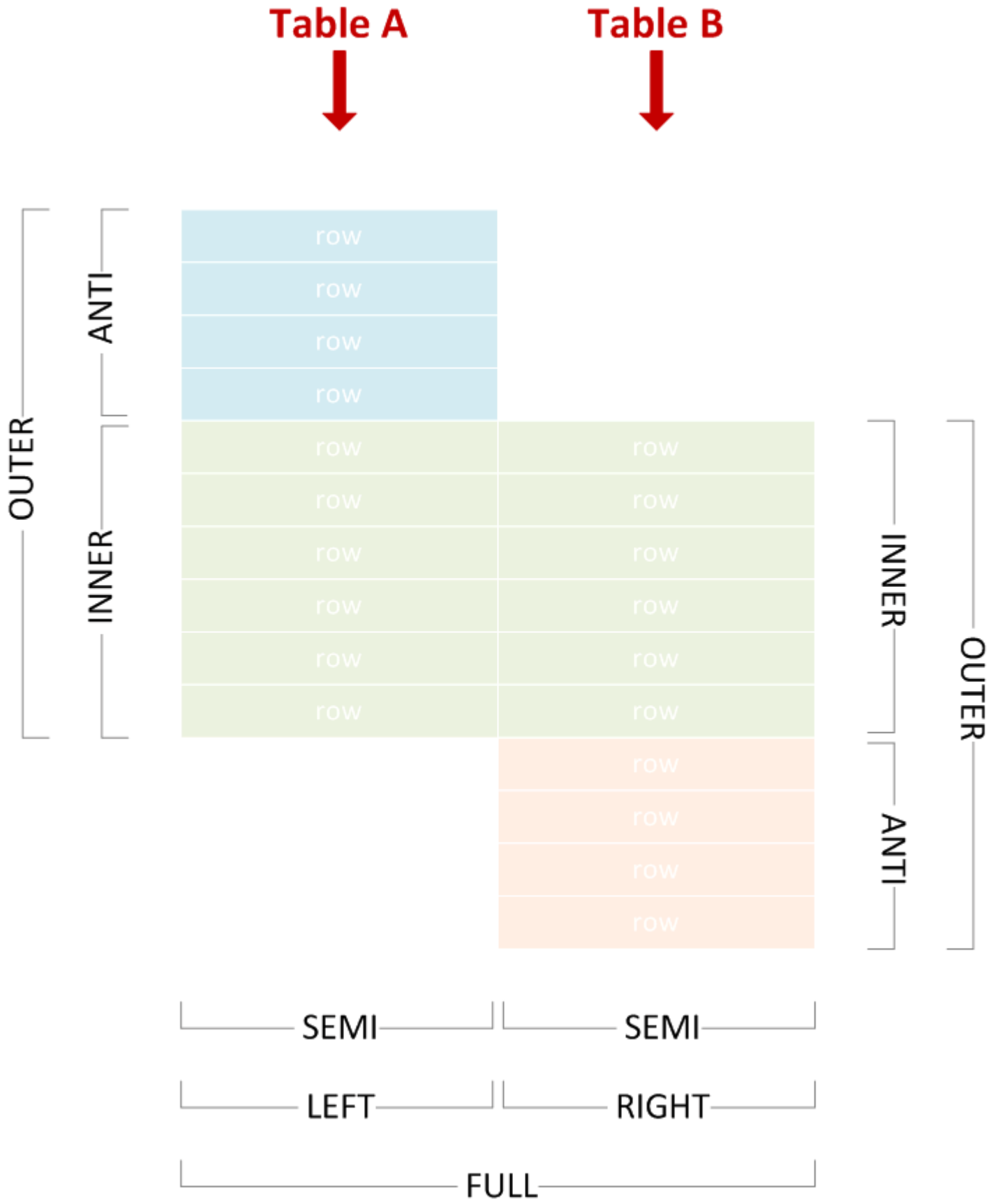
なにより、AとBの、つまりAのすべてのとBのすべてのがられます。Aのかが Bのするデータをたない、Bは nullです。に。

```
select * from a FULL OUTER JOIN b on a.a = b.b;

 a   |  b
-----+-----
   1 | null
   2 | null
   3 |    3
   4 |    4
null |    6
null |    5
```

JOIN、、、アンチ...

たとえば、2つのテーブルAとBとそののがしているとしますのJOINとので、のケースではでもい

ません。

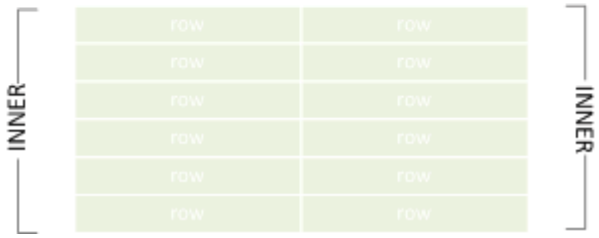


さまざまなタイプをして、またはのをいずれかのからめたりしたり、のからするをしてしくをけることができます。

のでは、のテストデータをしています。

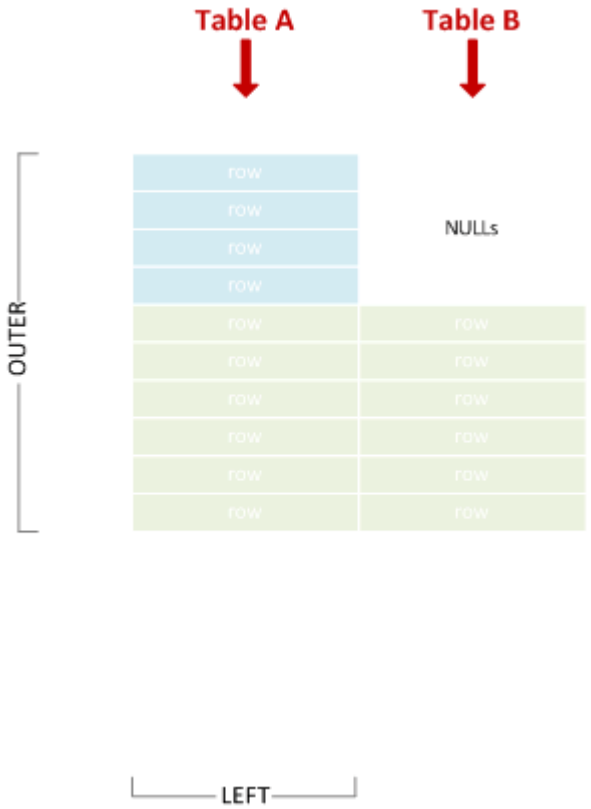
```
CREATE TABLE A (
    X varchar(255) PRIMARY KEY
);

CREATE TABLE B (
    Y varchar(255) PRIMARY KEY
);

INSERT INTO A VALUES
    ('Amy'),
    ('John'),
    ('Lisa'),
    ('Marco'),
    ('Phil');

INSERT INTO B VALUES
    ('Lisa'),
    ('Marco'),
    ('Phil'),
    ('Tim'),
    ('Vincent');
```

するのをします。



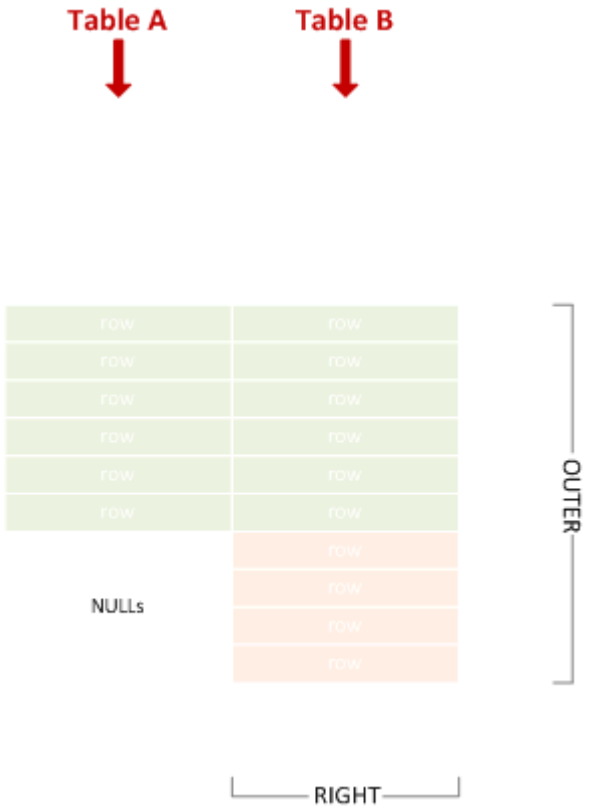

```
SELECT * FROM A JOIN B ON X = Y;

X      Y
------ -----
Lisa   Lisa
Marco  Marco
Phil   Phil
```

「」とされることもあります。するのをし、しないのをみます。




```
SELECT * FROM A LEFT JOIN B ON X = Y;

X      Y
-----  -----
Amy    NULL
John   NULL
Lisa   Lisa
Marco  Marco
Phil   Phil
```

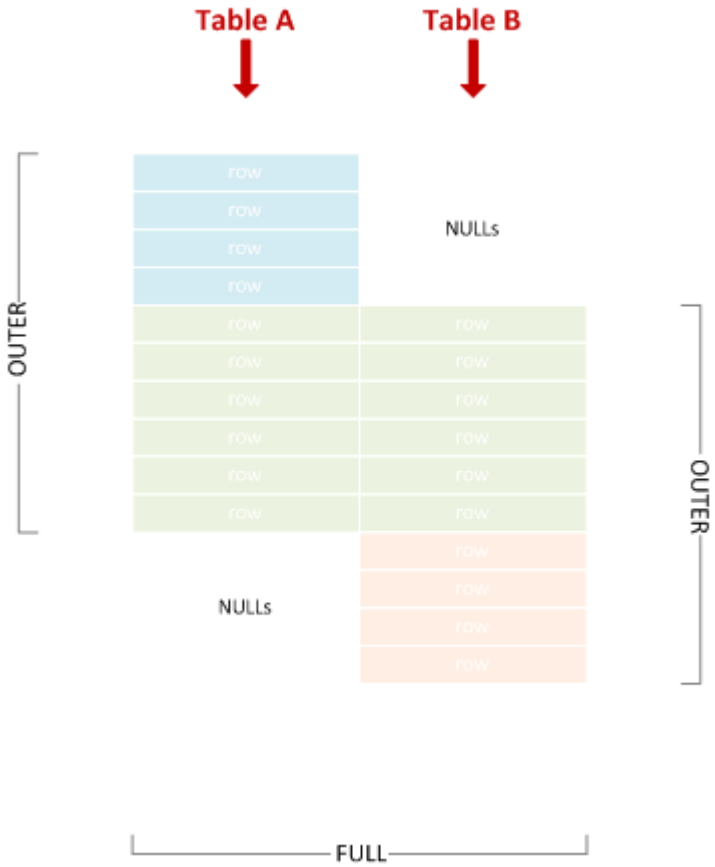
「」とされることもあります。するのをし、しないのをみます。




```
SELECT * FROM A RIGHT JOIN B ON X = Y;

X      Y
-----  -------
Lisa   Lisa
Marco  Marco
Phil   Phil
NULL   Tim
NULL   Vincent
```

な

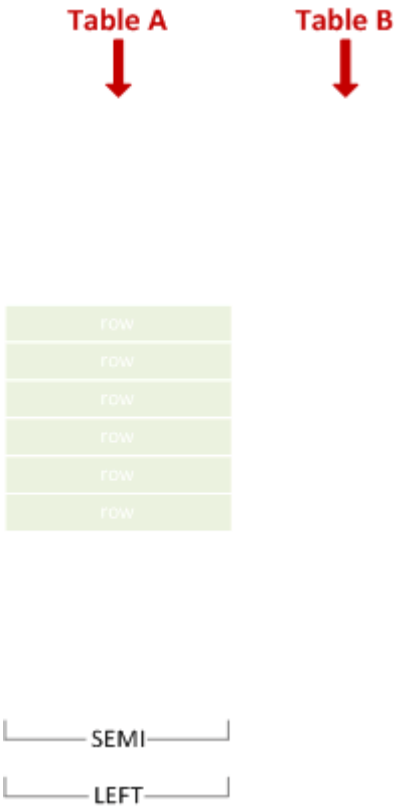
には「」とされることもある。の。




```
SELECT * FROM A FULL JOIN B ON X = Y;

X       Y
-----   -------
Amy     NULL
John    NULL
Lisa    Lisa
Marco   Marco
Phil    Phil
NULL    Tim
NULL    Vincent
```

セミ

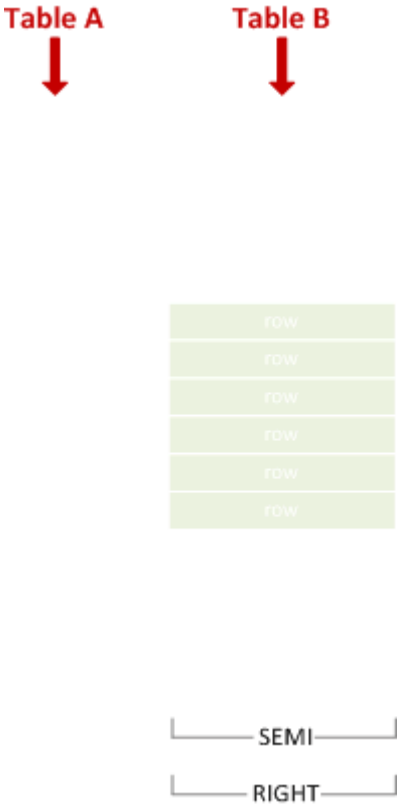
のにするのをみます。




```
SELECT * FROM A WHERE X IN (SELECT Y FROM B);

X
-----
Lisa
Marco
Phil
```

セミ

のにするのがまれます。

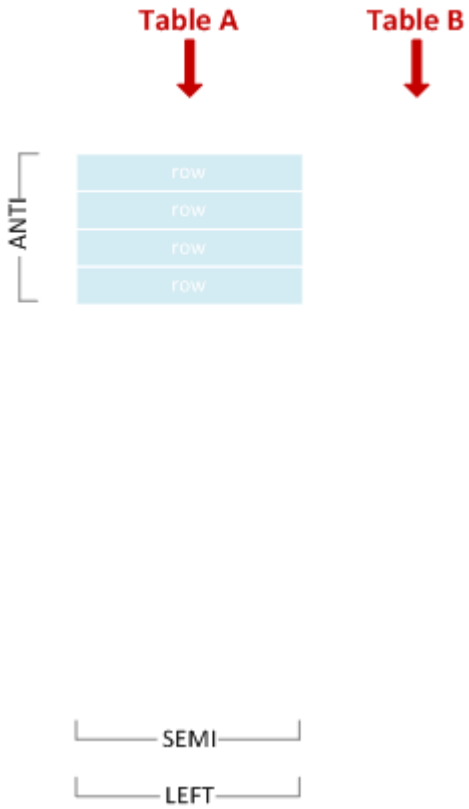



```
SELECT * FROM B WHERE Y IN (SELECT X FROM A);

Y
-----
Lisa
Marco
Phil
```

ごのとおり、とのセミジョインにはのINがありません.SQLテキストのテーブルをりえるだけでがられます。

のアンチ･セミ･ジョイン

のとしないのがまれます。

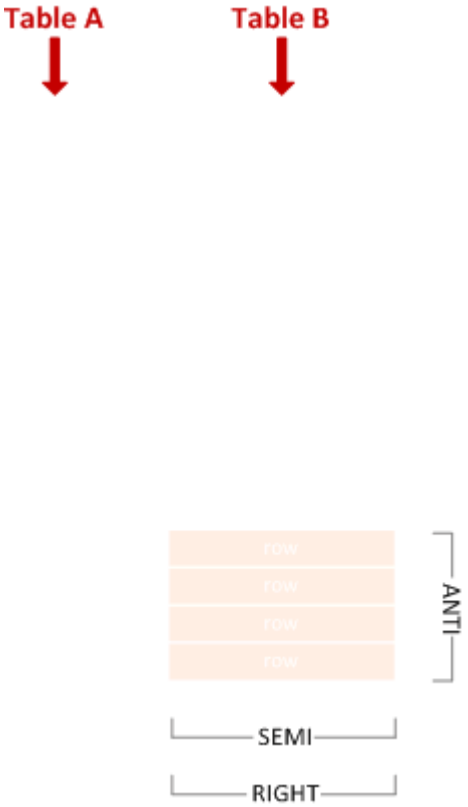



```
SELECT * FROM A WHERE X NOT IN (SELECT Y FROM B);

X
----
Amy
John
```

NULLカラムでNOT INをしているはしてくださいはこちら 。

アンチ-セミ-ジョイン

のとしないのがまれます。




```
SELECT * FROM B WHERE Y NOT IN (SELECT X FROM A);

Y
-------
Tim
Vincent
```

ごのとおり、とのアンチ･セミ･ジョインにはのNOT INがありません - SQLテキストのテーブルをりえるだけでがられます。

クロス

すべてののとすべてののデカルト。

```
SELECT * FROM A CROSS JOIN B;

X      Y
-----  -------
Amy    Lisa
John   Lisa
Lisa   Lisa
Marco  Lisa
Phil   Lisa
Amy    Marco
John   Marco
Lisa   Marco
Marco  Marco
Phil   Marco
```

```
Amy    Phil
John   Phil
Lisa   Phil
Marco  Phil
Phil   Phil
Amy    Tim
John   Tim
Lisa   Tim
Marco  Tim
Phil   Tim
Amy    Vincent
John   Vincent
Lisa   Vincent
Marco  Vincent
Phil   Vincent
```

クロスは、にするをつにします。したがって、のクエリはじをします。

```
SELECT * FROM A JOIN B ON 1 = 1;
```

これは、にそれとしているテーブルをします。は、でしたのにすることができます。たとえば、これはです。

```
SELECT * FROM A A1 JOIN A A2 ON LEN(A1.X) < LEN(A2.X);

X     X
----  -----
Amy   John
Amy   Lisa
Amy   Marco
John  Marco
Lisa  Marco
Phil  Marco
Amy   Phil
```

オンラインでジョインをむ https://riptutorial.com/ja/sql/topic/261/ジョイン

30: スキップページネーション

Examples

からいくつかのをスキップする

ISO / ANSI SQL

```
SELECT Id, Col1
FROM TableName
ORDER BY Id
OFFSET 20 ROWS
```

MySQL

```
SELECT * FROM TableName LIMIT 20, 42424242424242;
-- skips 20 for take use very large number that is more than rows in table
```

Oracle

```
SELECT Id,
   Col1
FROM (SELECT Id,
           Col1,
           row_number() over (order by Id) RowNumber
      FROM TableName)
WHERE RowNumber > 20
```

PostgreSQL

```
SELECT * FROM TableName OFFSET 20;
```

SQLite

```
SELECT * FROM TableName LIMIT -1 OFFSET 20;
```

のをする

ISO / ANSI SQL

```
SELECT * FROM TableName FETCH FIRST 20 ROWS ONLY;
```

MySQL; PostgreSQL; SQLite

```
SELECT * FROM TableName LIMIT 20;
```

Oracle

```
SELECT Id,
   Col1
FROM (SELECT Id,
           Col1,
           row_number() over (order by Id) RowNumber
      FROM TableName)
WHERE RowNumber <= 20
```

SQLサーバー

```
SELECT TOP 20 *
FROM dbo.[Sale]
```

スキップしていくつかのをるページネーション

ISO / ANSI SQL

```
SELECT Id, Col1
FROM TableName
ORDER BY Id
OFFSET 20 ROWS FETCH NEXT 20 ROWS ONLY;
```

MySQL

```
SELECT * FROM TableName LIMIT 20, 20; -- offset, limit
```

オラクルSQLサーバー

```
SELECT Id,
   Col1
 FROM (SELECT Id,
           Col1,
           row_number() over (order by Id) RowNumber
      FROM TableName)
WHERE RowNumber BETWEEN 21 AND 40
```

PostgreSQL; SQLite

```
SELECT * FROM TableName LIMIT 20 OFFSET 20;
```

オンラインでスキップページネーションをむ https://riptutorial.com/ja/sql/topic/2927/スキップ-ページネーション-

31: ストアドプロシージャ

ストアドプロシージャは、クエリでまたはびすことができるデータベースにされたSQLです。ストアドプロシージャをすると、またはにされるロジックをカプセルし、キャッシュされたクエリプランをしてクエリのパフォーマンスをさせることができます。なクエリがすことができるをすことができます。

SQLにるそののは、 Wikipeidaにされています 。

Examples

ストアドプロシージャのとびし

ストアド･プロシージャは、データベースGUI SQL Serverの をして、またはSQLをしてのようにできます。

```
-- Define a name and parameters
CREATE PROCEDURE Northwind.getEmployee
    @LastName nvarchar(50),
    @FirstName nvarchar(50)
AS

-- Define the query to be run
SELECT FirstName, LastName, Department
FROM Northwind.vEmployeeDepartment
WHERE FirstName = @FirstName AND LastName = @LastName
AND EndDate IS NULL;
```

プロシージャをびす

```
EXECUTE Northwind.getEmployee N'Ackerman', N'Pilar';

-- Or
EXEC Northwind.getEmployee @LastName = N'Ackerman', @FirstName = N'Pilar';
GO

-- Or
EXECUTE Northwind.getEmployee @FirstName = N'Pilar', @LastName = N'Ackerman';
GO
```

オンラインでストアドプロシージャをむ https://riptutorial.com/ja/sql/topic/1701/ストアドプロシージャ

32: セレクト

き

SELECTステートメントは、ほとんどのSQLのです。どのセットがクエリによってされるべきかをし、ほとんどの、データベースのどのをするかをするFROMとともにされます。

- SELECT [DISTINCT] [column1] [、[column2] ...]
  FROM [table]
  [WHERE]
  [GROUP BY [1] [、[2] ...]

  [HAVING [1] [、[2] ...]

  [オーダーby ASC | DESC]

SELECTは 、のテーブルからすカラムのデータと、そのカラムのをします。

```
SELECT Name, SerialNumber
FROM ArmyInfo
```

`Name`と`Serial Number`ののみをしますが、 `Rank`というのはしません。

```
SELECT *
FROM ArmyInfo
```

すべてのがされることをします。ただし、にはテーブルのすべてのをすので、 `SELECT *`をすることはです。

Examples

ワイルドカードをして、クエリのすべてのをします。

の2つのテーブルをつデータベースをえてみましょう。

テーブル

イド	FName	LName	DeptId
1	ジェームス	スミス	3
2	ジョン	ジョンソン	4

テーブル

イド	
1	
2	マーケティング
3	ファイナンス
4	それ

な**select**ステートメント

*は、テーブルのなすべてのをするためにされるワイルドカードです。

なのとしてすると、クエリがFROMしているすべてののすべてのがFROMます。このエフェクトは、クエリがJOINをしてアクセスするすべてのテーブルにされます。

のクエリをえてみましょう。

```
SELECT * FROM Employees
```

Employeesテーブルのすべてののすべてのフィールドがされます。

イド	FName	LName	DeptId
1	ジェームス	スミス	3
2	ジョン	ジョンソン	4

ドット

のテーブルからすべてのをするには、ワイルドカードをドットでテーブルにできます。

のクエリをえてみましょう。

```
SELECT
    Employees.*,
    Departments.Name
FROM
    Employees
JOIN
    Departments
    ON Departments.Id = Employees.DeptId
```

これにより、 Employeeテーブルのすべてのフィールドをむデータセットがされ、そのにDepartmentsテーブルのNameフィールドだけがされます。

イド	FName	LName	DeptId	
1	ジエームス	スミス	3	ファイナンス
2	ジョン	ジョンソン	4	それ

の

であれば、プロダクションコードで*をすることはされることがにされています。

1. データベースエンジンがなデータをみんでフロントエンドコードにするため、IO、ネットワーク、メモリがです。これは、いやファイルをするのにされるようなきなフィールドがするににです。
2. `SELECT <columns> FROM <table>`よりもなクエリののとして、データベースがをディスクにスプールするがあるは、さらになIOがします。
3. なのがのようなは、なおよび/またはさらにIOがします。
   - それらをサポートするデータベースのカラム
   - ビューからする、クエリオプティマイザがのでできるテーブル/ビューの
4. テーブルやビューにでされたがあいまいなになった、しないエラーがするがあります。たとえば、 `SELECT * FROM orders JOIN people ON people.id = orders.personid ORDER BY displayname` という- `displayname` 、ユーザーがののののためにのあるをけることができるようにテーブルにされた、がされます`ORDER BY`があいまいでエラーをきこすがあるためのMS SQL Serverバージョンではあいまいな、このではない、アプリケーションにがされるコードがされることがありますしいがされたのであるなどのためにされています。

あなたは*することができますかをしていますか

プロダクションコードではけるのがですが、やプロトタイプのためにデータベースにしてクエリーをするは、 `*`をするのがです。

々 、このようなでは、むそれはやむをないるアプリケーションでのを`tablealias.*`わずか*な。

`SELECT A.col1, A.Col2 FROM A WHERE EXISTS (SELECT * FROM B where A.ID = B.A_ID)`などの`EXISTS`をする、Bからのデータはされません。したがって、はで、エンジンはBからのがされないことをっているので、 `*`をしたときのパフォーマンスはしません。に、 `COUNT(*)`はにはをさないため、フィルタリングでされるものをみんでするだけでみます。

でする

SELECT WHEREのはのとおりです。

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

*[condition]*には、>、<、=、<>、> =、<=、LIKE、NOT、IN、BETWEENなどのまたはをしてした

のSQLをできます。

のは、ステータスが 'READY'の 'Cars'テーブルからすべてのをします。

```
SELECT * FROM Cars WHERE status = 'READY'
```

よりくのについては、 WHEREとHAVINGをしてください。

々のを

```
SELECT
    PhoneNumber,
    Email,
    PreferredContact
FROM Customers
```

このステートメントは、 `Customers`テーブルのすべてのから`PhoneNumber` 、 `Email` 、および`PreferredContact`というをします。また、は`SELECT`にれるでされます。

はのようになります。

	Eメール	PreferredContact
3347927472	william.jones@example.com	
2137921892	dmiller@example.net	Eメール
ヌル	richard0123@example.com	Eメール

のがされている、のにをすることにより、のからをできます`[table_name].[column_name]`

```
SELECT
    Customers.PhoneNumber,
    Customers.Email,
    Customers.PreferredContact,
    Orders.Id AS OrderId
FROM
    Customers
LEFT JOIN
    Orders ON Orders.CustomerId = Customers.Id
```

* `AS OrderId`は、 `Orders`テーブルの`Id`フィールドが`OrderId`というのとしてされることを`OrderId`ます。については、エイリアスのをしてください。

いテーブルをしないようにするには、テーブルエイリアスをできます。これにより、ジョインでしたフィールドのいテーブルをきむがされます。 じの2つのインスタンスのをするは、のをしてをするがあります。 `Customers c`や`Customers AS c`ようなテーブルエイリアスをくことができます。ここで`c`は`Customers`エイリアスとしてし、のような`Email`します `c.Email`。

```
SELECT
    c.PhoneNumber,
    c.Email,
    c.PreferredContact,
    o.Id AS OrderId
FROM
    Customers c
LEFT JOIN
    Orders o ON o.CustomerId = c.Id
```

エイリアスをしたSELECT

エイリアスはにコードをし、をよりみやすくするためにされます。

テーブルがく、なのたとえば、テーブルに2つのIDがありますが、1つだけがされているほど、コードはくなります。 テーブルのエイリアスにえて、これにより、データベースでよりわかりやすいをして、そのをにすることができます。

さらに、されたにをけるために、ビューなどでになることがあります。

すべてのバージョンのSQL

エイリアスは、すべてのバージョンのSQLで " をしてできます。

```
SELECT
    FName AS "First Name",
    MName AS "Middle Name",
    LName AS "Last Name"
FROM Employees
```

なるバージョンのSQL

Microsoft SQL Serverでエイリアスをするには、 ' 、 " 、 [] をできます。

```
SELECT
    FName AS "First Name",
    MName AS 'Middle Name',
    LName AS [Last Name]
FROM Employees
```

どちらもになります

ファーストネーム	ミドルネーム	
ジェームス	ジョン	スミス
ジョン	ジェームス	ジョンソン
マイケル	マーカス	ウィリアムズ

このは、され`FName`と`LName`えられたのを。これは、エイリアスがにく`AS`をするか、にのろにエイリアスをきむことでされます。つまり、のクエリはとじになります。

```
SELECT
    FName "First Name",
    MName "Middle Name",
    LName "Last Name"
FROM Employees
```

ファーストネーム	ミドルネーム	
ジエームス	ジョン	スミス
ジョン	ジエームス	ジョンソン
マイケル	マーカス	ウィリアムズ

しかし、なバージョンつまり、 `AS`をするは、よりみやすくなります。

エイリアスにではないのがあるは、、、またはなしでできます。

```
SELECT
    FName AS FirstName,
    LName AS LastName
FROM Employees
```

ファーストネーム	
ジエームス	スミス
ジョン	ジョンソン
マイケル	ウィリアムズ

とりわけ、MS SQL Serverでなのバリエーションは`<alias> = <column-or-calculation>`です。たとえば、のようになります。

```
SELECT FullName = FirstName + ' ' + LastName,
       Addr1    = FullStreetAddress,
       Addr2    = TownName
FROM CustomerDetails
```

これはとです

```
SELECT FirstName + ' ' + LastName As FullName
       FullStreetAddress          As Addr1,
       TownName                   As Addr2
FROM CustomerDetails
```

どちらもになります

フルネーム	Addr1	Addr2
ジェームズスミス	123 AnyStreet	TownVille
ジョンジョンソン	668 MyRoad	Anytown
マイケル･ウィリアムズ	999ハイ･エンド･ドクター	ウィリアムズバーグ

いくつかは、してつける`=`のわりに`As`くは、それがではないなはとてもく、すべてのデータベースでサポートされていない、このフォーマットにしておめしますが、みやすいです。 `=`をのとするがあります。

すべてのSQLバージョン

また、をするがあるは、またはをしてエスケープすることもできます。

```
SELECT
    FName as "SELECT",
    MName as "FROM",
    LName as "WHERE"
FROM Employees
```

なるバージョンのSQL

に、MSSQLでキーワードをエスケープするには、さまざまながあります。

```
SELECT
    FName AS "SELECT",
    MName AS 'FROM',
    LName AS [WHERE]
FROM Employees
```

セレクト	から	どこに
ジェームス	ジョン	スミス
ジョン	ジェームス	ジョンソン
マイケル	マーカス	ウィリアムズ

また、エイリアスは、 `ORDER BY`などのじクエリののいずれかにできます。

```
SELECT
    FName AS FirstName,
    LName AS LastName
```

```
FROM
    Employees
ORDER BY
    LastName DESC
```

ただし、することはできません

```
SELECT
    FName AS SELECT,
    LName AS FROM
FROM
    Employees
ORDER BY
    LastName DESC
```

これらの `SELECT`および`FROM` からエイリアスをするには

これにより、にのエラーがします。

ソートされたをむ

```
SELECT * FROM Employees ORDER BY LName
```

このステートメントは、 `Employees`テーブルからすべてのをします。

イド	FName	LName	
2	ジョン	ジョンソン	2468101214
1	ジェームス	スミス	1234567890
3	マイケル	ウィリアムズ	1357911131

```
SELECT * FROM Employees ORDER BY LName DESC
```

または

```
 SELECT * FROM Employees ORDER BY LName ASC
```

このはソートをします。

のソートをすることもできます。えば

```
SELECT * FROM Employees ORDER BY LName ASC, FName ASC
```

このでは、まず`LName`をソートし、じ`LName`をつレコードについては`LName`でソートし`FName`。 これにより、でのとのがられます。

`ORDER BY`にをするのをするために、わりにのをすることができます。は1からまることにしてください。

```
SELECT Id, FName, LName, PhoneNumber FROM Employees ORDER BY 3
```

`ORDER BY`に`CASE`をめむこともできます。

```
SELECT Id, FName, LName, PhoneNumber FROM Employees ORDER BY CASE WHEN LName='Jones` THEN 0
ELSE 1 END ASC
```

これによりがソートされ、 "Jones"の`LName`をつすべてのレコードがにされます。

されたキーワードのにがけられたをする

がされたキーワードとする、SQLではでむがあります。

```
SELECT
    "ORDER",
    ID
FROM ORDERS
```

カラムはとをします。

DBMSによっては、をするのがあります。たとえば、SQL Serverはこのにをします。

```
SELECT
    [Order],
    ID
FROM ORDERS
```

MySQLとMariaDBはデフォルトでバッククォートをいます

```
SELECT
    `Order`,
    id
FROM orders
```

したレコードをする

SQL 2008では、されるレコードをするために`FETCH FIRST`がされています 。

```
SELECT Id, ProductName, UnitPrice, Package
FROM Product
ORDER BY UnitPrice DESC
FETCH FIRST 10 ROWS ONLY
```

このは、のバージョンのRDMSでのみサポートされています。ベンダーのがのシステムでされています。OpenEdge 11.xは`FETCH FIRST <n> ROWS ONLY`もサポートしています。

さらに、 `FETCH FIRST <n> ROWS ONLY`に`OFFSET <m> ROWS`すると、をフェッチするにをスキップすることができます。

```
SELECT Id, ProductName, UnitPrice, Package
FROM Product
ORDER BY UnitPrice DESC
OFFSET 5 ROWS
FETCH FIRST 10 ROWS ONLY
```

SQL ServerとMS Accessでのクエリがサポートされています 。

```
SELECT TOP 10 Id, ProductName, UnitPrice, Package
FROM Product
ORDER BY UnitPrice DESC
```

MySQLまたはPostgreSQLでじことをうには、 `LIMIT`キーワードをするがあります。

```
SELECT Id, ProductName, UnitPrice, Package
FROM Product
ORDER BY UnitPrice DESC
LIMIT 10
```

Oracleでは`ROWNUM`でもじことができます

```
SELECT Id, ProductName, UnitPrice, Package
FROM Product
WHERE ROWNUM <= 10
ORDER BY UnitPrice DESC
```

10レコード。

```
Id    ProductName              UnitPrice            Package
38    Côte de Blaye            263.50               12 - 75 cl bottles
29    Thüringer Rostbratwurst  123.79               50 bags x 30 sausgs.
9    Mishi Kobe Niku           97.00                18 - 500 g pkgs.
20    Sir Rodney's Marmalade   81.00                30 gift boxes
18    Carnarvon Tigers         62.50                16 kg pkg.
59    Raclette Courdavault     55.00                5 kg pkg.
51    Manjimup Dried Apples    53.00                50 - 300 g pkgs.
62    Tarte au sucre           49.30                48 pies
43    Ipoh Coffee              46.00                16 - 500 g tins
28    Rössle Sauerkraut        45.60                25 - 825 g cans
```

ベンダーニュアンス

Microsoft SQLの`TOP`は`WHERE`のにし、テーブルのどこにでもするはされたのをすことにして`ROWNUM`は`WHERE`のとしてします。テーブルのにされたをすると、つかるがあるはゼロのがられます。

テーブルエイリアスでする

```
SELECT e.Fname, e.LName
```

```
FROM Employees e
```

Employeesテーブルには、テーブルのに「e」がけられます。これにより、のテーブルがじフィールドをち、データをすテーブルをするがあるに、あいまいさをりくのにちます。

```
SELECT e.Fname, e.LName, m.Fname AS ManagerFirstName
FROM Employees e
    JOIN Managers m ON e.ManagerId = m.Id
```

エイリアスをすると、テーブルをできなくなります。すなわち、

```
SELECT e.Fname, Employees.LName, m.Fname AS ManagerFirstName
FROM Employees e
JOIN Managers m ON e.ManagerId = m.Id
```

エラーがします。

`INNER JOIN`によってきこされたのをするために、よりには「」というのエイリアスがSQLにされました。 1992のSQLでは、 `NATURAL JOIN` mySQL、PostgreSQL、Oracleではされていますが SQL Serverではされていませんをすることにより、こののをしました。のは、テーブルがなる `Id`および`ManagerId` をつでされているが、じ `LName` 、 `FName` のにされていないため、するのをするがある

```
SELECT Fname, LName, ManagerFirstName
FROM Employees
     NATURAL JOIN
     ( SELECT Id AS ManagerId, Fname AS ManagerFirstName
       FROM Managers ) m;
```

dervedテーブルにしてエイリアス/をしなければなりませんがそうでないとSQLはエラーをげます、クエリでにすることはしてできません。

のからをする

```
SELECT *
FROM
    table1,
    table2
```

```
SELECT
    table1.column1,
    table1.column2,
    table2.column1
FROM
    table1,
    table2
```

これはSQLのクロスプロダクトとばれ、セットのクロスプロダクトとじです

これらのステートメントは、1つのクエリでのテーブルからしたをします。

からされるにはのはありません。

による

`AVG()`は、されたのをします。

```
SELECT AVG(Salary) FROM Employees
```

をwhereとみわせることもできます。

```
SELECT AVG(Salary) FROM Employees where DepartmentId = 1
```

はgroup byとみわせることもできます。

がのにされ、ごとにをめたいは、のクエリをできます。

```
SELECT AVG(Salary) FROM Employees GROUP BY DepartmentId
```

`MIN()`は、されたをします。

```
SELECT MIN(Salary) FROM Employees
```

`MAX()`はされたをします。

```
SELECT MAX(Salary) FROM Employees
```

カウント

`COUNT()`は、されたのをします。

```
SELECT Count(*) FROM Employees
```

のをたすのをするとみわせることもできます。

```
SELECT Count(*) FROM Employees where ManagerId IS NOT NULL
```

のをして、ののをすることもできます。 `NULL`はカウントされないことにしてください。

```
Select Count(ManagerId) from Employees
```

Countはdistinct countとdistinctキーワードでみわせることもできます。

```
Select Count(DISTINCT DepartmentId) from Employees
```

SUM()は、すべてのにしてされたのをします。

```
SELECT SUM(Salary) FROM Employees
```

ヌルでする

```
SELECT Name FROM Customers WHERE PhoneNumber IS NULL
```

NULLをしたは、なるをとります。 =をしないでください。わりにIS NULLまたはIS NOT NULLしてください。

CASEでする

にあるロジックを 'オンザフライで'するがある、CASEをしてそれをできます。

```
SELECT CASE WHEN Col1 < 50 THEN 'under' ELSE 'over' END threshold
FROM TableName
```

また、チェーンすることもできます

```
SELECT
    CASE WHEN Col1 < 50 THEN 'under'
         WHEN Col1 > 50 AND Col1 <100 THEN 'between'
         ELSE 'over'
    END threshold
FROM TableName
```

CASEをのCASEのにれることもできます

```
SELECT
    CASE WHEN Col1 < 50 THEN 'under'
         ELSE
            CASE WHEN Col1 > 50 AND Col1 <100 THEN Col1
            ELSE 'over' END
    END threshold
FROM TableName
```

テーブルをロックせずにする

々テーブルがみみのためにまたはのされるとき、けはもはやけにはならず、さなビットがカウントされると、パフォーマンスをするためにLOCKのないselectをすることがあります。

SQLサーバー

```
SELECT * FROM TableName WITH (nolock)
```

MySQL

```
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM TableName;
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

オラクル

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM TableName;
```

DB2

```
SELECT * FROM TableName WITH UR;
```

ここで、 `UR`は「コミットされていないみり」をします。

レコーのがわれているテーブルですると、しないがじるがあります。

ののみを

```
SELECT DISTINCT ContinentCode
FROM Countries;
```

このクエリは、 `Countries`テーブルの`ContinentCode`からすべての`DISTINCT` のなるをします

ContinentCode
OC
EU
として
NA
AF

SQLFiddleデモ

からのので

```
SELECT * FROM Cars  WHERE status IN ( 'Waiting', 'Working' )
```

これはに

```
SELECT * FROM Cars  WHERE ( status = 'Waiting' OR status = 'Working' )
```

すなわち、 `value IN ( <value list> )` は `OR` のです。

グループのをる

ののにづいてをえる

```
SELECT category, COUNT(*) AS item_count
FROM item
GROUP BY category;
```

の

```
SELECT department, AVG(income)
FROM employees
GROUP BY department;
```

なのは、 `GROUP BY` でされたのみをするか、 でするだけをすることです。

そこ `WHERE` はしてもすることができ `GROUP BY` が、 `WHERE` のグループがわれるにレコードをフィルタリングします。

```
SELECT department, AVG(income)
FROM employees
WHERE department <> 'ACCOUNTING'
GROUP BY department;
```

たとえば、が1000をえるのみをするなど、グループがしたにをフィルタリングするがあるは、 `HAVING` をするがあります。

```
SELECT department, AVG(income)
FROM employees
WHERE department <> 'ACCOUNTING'
GROUP BY department
HAVING avg(income) > 1000;
```

のでする。

`AND` キーワードは、クエリにをするためにされます。

サム	18	M

ジョン	21	M
ボブ	22	M
メアリー	23	F

```
SELECT name FROM persons WHERE gender = 'M' AND age > 20;
```

これはされます

ジョン
ボブ

ORキーワードをして

```
SELECT name FROM persons WHERE gender = 'M' OR age < 20;
```

これはされます

サム
ジョン
ボブ

これらのキーワードをみわせると、よりなのみわせがになります。

```
SELECT name
FROM persons
WHERE (gender = 'M' AND age < 20)
   OR (gender = 'F' AND age > 20);
```

これはされます

サム
メアリー

オンラインでセレクトをむ https://riptutorial.com/ja/sql/topic/222/セレクト

33: データベースとテーブルの

Examples

オートショップデータベース

のでは、オートショップのデータベースに、、、およびののリストがあります。キーをして、さまざまなテーブルのをしています。

SQL fiddle

テーブルの

- には0のがいるかもしれません
- は0または1のマネージャーをつことができます
- は0のをしているがあります

イド	
1	HR
2	
3	テック

テーブルをするSQLステートメント

```
CREATE TABLE Departments (
    Id INT NOT NULL AUTO_INCREMENT,
    Name VARCHAR(25) NOT NULL,
    PRIMARY KEY(Id)
);

INSERT INTO Departments
    ([Id], [Name])
VALUES
    (1, 'HR'),
    (2, 'Sales'),
    (3, 'Tech')
;
```

イド	FName	LName		マネージャーID	DepartmentId		HireDate
1	ジエームス	スミス	1234567890	ヌル	1	1000	01-01-2002
2	ジョン	ジョンソン	2468101214	1	1	400	23-03-2005
3	マイケル	ウィリアムズ	1357911131	1	2	600	12-05-2009
4	ジョンナトン	スミス	1212121212	2	1	500	24-07-2016

テーブルをするSQLステートメント

```
CREATE TABLE Employees (
    Id INT NOT NULL AUTO_INCREMENT,
    FName VARCHAR(35) NOT NULL,
    LName VARCHAR(35) NOT NULL,
    PhoneNumber VARCHAR(11),
    ManagerId INT,
    DepartmentId INT NOT NULL,
    Salary INT NOT NULL,
    HireDate DATETIME NOT NULL,
    PRIMARY KEY(Id),
    FOREIGN KEY (ManagerId) REFERENCES Employees(Id),
    FOREIGN KEY (DepartmentId) REFERENCES Departments(Id)
);

INSERT INTO Employees
    ([Id], [FName], [LName], [PhoneNumber], [ManagerId], [DepartmentId], [Salary], [HireDate])
VALUES
    (1, 'James', 'Smith', 1234567890, NULL, 1, 1000, '01-01-2002'),
    (2, 'John', 'Johnson', 2468101214, '1', 1, 400, '23-03-2005'),
    (3, 'Michael', 'Williams', 1357911131, '1', 2, 600, '12-05-2009'),
    (4, 'Johnathon', 'Smith', 1212121212, '2', 1, 500, '24-07-2016')
;
```

イド	FName	LName	Eメール		PreferredContact
1	ウィリアム	ジョーンズ	william.jones@example.com	3347927472	
2	デビッド	ミラー	dmiller@example.net	2137921892	Eメール
3	リチャード	デイビス	richard0123@example.com	ヌル	Eメール

テーブルをするSQLステートメント

```
CREATE TABLE Customers (
    Id INT NOT NULL AUTO_INCREMENT,
    FName VARCHAR(35) NOT NULL,
    LName VARCHAR(35) NOT NULL,
    Email varchar(100) NOT NULL,
    PhoneNumber VARCHAR(11),
    PreferredContact VARCHAR(5) NOT NULL,
    PRIMARY KEY(Id)
);

INSERT INTO Customers
    ([Id], [FName], [LName], [Email], [PhoneNumber], [PreferredContact])
VALUES
    (1, 'William', 'Jones', 'william.jones@example.com', '3347927472', 'PHONE'),
    (2, 'David', 'Miller', 'dmiller@example.net', '2137921892', 'EMAIL'),
    (3, 'Richard', 'Davis', 'richard0123@example.com', NULL, 'EMAIL')
;
```

イド	ID	ID	モデル		
1	1	2	フォードF-150	READY	230
2	1	2	フォードF-150	READY	200
3	2	1	フォードマスタング	ち	100
4	3	3	トヨタプリウス	ワーキング	1254

テーブルをするSQLステートメント

```
CREATE TABLE Cars (
    Id INT NOT NULL AUTO_INCREMENT,
    CustomerId INT NOT NULL,
    EmployeeId INT NOT NULL,
    Model varchar(50) NOT NULL,
    Status varchar(25) NOT NULL,
    TotalCost INT NOT NULL,
    PRIMARY KEY(Id),
    FOREIGN KEY (CustomerId) REFERENCES Customers(Id),
    FOREIGN KEY (EmployeeId) REFERENCES Employees(Id)
);

INSERT INTO Cars
    ([Id], [CustomerId], [EmployeeId], [Model], [Status], [TotalCost])
VALUES
    ('1', '1', '2', 'Ford F-150', 'READY', '230'),
    ('2', '1', '2', 'Ford F-150', 'READY', '200'),
    ('3', '2', '1', 'Ford Mustang', 'WAITING', '100'),
    ('4', '3', '3', 'Toyota Prius', 'WORKING', '1254')
;
```

ライブラリデータベース

このライブラリのサンプルデータベースには、 *Authors* 、 *Books* 、および*BooksAuthors*テーブルがあります。

SQL fiddle

とは、リレーショナル·モデルののエンティティーのとデータをんでいるため、 とばれます。 *BooksAuthors*は、 *Books*テーブルと*Authors*テーブルのをするため、 *Relationship* テーブルとばれます。

テーブルの

- それぞれのは1のをつことができます
- は1のをつことができます

ビューテーブル

イド		
1	JDサリンジャー	
2	F.スコット。フィッツジェラルド	
3	ジェーン·オースティン	
4	スコットハンセルマン	
5	ジェイソン·N·ゲイロード	
6	プラナフ·ラストギ	インド
7	トッド·ミランダ	
8	クリスチャンウェンツ	

テーブルをするSQL

```
CREATE TABLE Authors (
    Id INT NOT NULL AUTO_INCREMENT,
    Name VARCHAR(70) NOT NULL,
    Country VARCHAR(100) NOT NULL,
    PRIMARY KEY(Id)
);

INSERT INTO Authors
```

```
    (Name, Country)
VALUES
    ('J.D. Salinger', 'USA'),
    ('F. Scott. Fitzgerald', 'USA'),
    ('Jane Austen', 'UK'),
    ('Scott Hanselman', 'USA'),
    ('Jason N. Gaylord', 'USA'),
    ('Pranav Rastogi', 'India'),
    ('Todd Miranda', 'USA'),
    ('Christian Wenz', 'USA')
;
```

ビューテーブル

イド	タイトル
1	ライでつかまえて
2	ナインストーリーズ
3	フラニーとズーイー
4	グレート·ギャツビー
5	テンダー·イット·ザ·ナイト
6	と
7	プロフェッショナルASP.NET 4.5でCとVB

テーブルをするSQL

```
CREATE TABLE Books (
    Id INT NOT NULL AUTO_INCREMENT,
    Title VARCHAR(50) NOT NULL,
    PRIMARY KEY(Id)
);

INSERT INTO Books
    (Id, Title)
VALUES
    (1, 'The Catcher in the Rye'),
    (2, 'Nine Stories'),
    (3, 'Franny and Zooey'),
    (4, 'The Great Gatsby'),
    (5, 'Tender id the Night'),
    (6, 'Pride and Prejudice'),
    (7, 'Professional ASP.NET 4.5 in C# and VB')
;
```

BooksAuthors

ビューテーブル

BookId	ID
1	1
2	1
3	1
4	2
5	2
6	3
7	4
7	5
7	6
7	7
7	8

テーブルをするSQL

```
CREATE TABLE BooksAuthors (
    AuthorId INT NOT NULL,
    BookId  INT NOT NULL,
    FOREIGN KEY (AuthorId) REFERENCES Authors(Id),
    FOREIGN KEY (BookId) REFERENCES Books(Id)
);

INSERT INTO BooksAuthors
    (BookId, AuthorId)
VALUES
    (1, 1),
    (2, 1),
    (3, 1),
    (4, 2),
    (5, 2),
    (6, 3),
    (7, 4),
    (7, 5),
    (7, 6),
    (7, 7),
    (7, 8)
;
```

すべてのを ライブのを

```
SELECT * FROM Authors;
```

すべてののタイトルをする を

```
SELECT * FROM Books;
```

すべてのとそのをします のを

```
SELECT
  ba.AuthorId,
  a.Name AuthorName,
  ba.BookId,
  b.Title BookTitle
FROM BooksAuthors ba
  INNER JOIN Authors a ON a.id = ba.authorid
  INNER JOIN Books b ON b.id = ba.bookid
;
```

カントリーテーブル

このでは、 **Countries**テーブルがあります。のためのテーブルは、にやレートをむアプリケーションでは、くのがあります。

SQL fiddle

BloombergやReutersのようなMarketデータソフトウェアアプリケーションでは、APIにコードとともに2または3のコードをするがあります。したがって、こののには、2の$_{ISO}$コードと3の$_{ISO3}$コードがあります。

ビューテーブル

イド	ISO	ISO3	ISONumeric			ContinentCode	コード
1	AU	AUS	36	オーストラリア	キャンベラ	OC	AUD
2	DE	DEU	276	ドイツ	ベルリン	EU	ユーロ
2	に	インド	356	インド	ニューデリー	として	INR
3	LA	LAO	418	ラオス	ビエンチャン	として	LAK
4			840	アメリカ	ワシントン	NA	ドル

イド	ISO	ISO3	ISONumeric			ContinentCode	コード
5	ZW	ZWE	716	ジンバブエ	ハラレ	AF	ZWL

テーブルをするSQL

```
CREATE TABLE Countries (
    Id INT NOT NULL AUTO_INCREMENT,
    ISO VARCHAR(2) NOT NULL,
    ISO3 VARCHAR(3) NOT NULL,
    ISONumeric INT NOT NULL,
    CountryName VARCHAR(64) NOT NULL,
    Capital VARCHAR(64) NOT NULL,
    ContinentCode VARCHAR(2) NOT NULL,
    CurrencyCode VARCHAR(3) NOT NULL,
    PRIMARY KEY(Id)
)
;

INSERT INTO Countries
    (ISO, ISO3, ISONumeric, CountryName, Capital, ContinentCode, CurrencyCode)
VALUES
    ('AU', 'AUS', 36, 'Australia', 'Canberra', 'OC', 'AUD'),
    ('DE', 'DEU', 276, 'Germany', 'Berlin', 'EU', 'EUR'),
    ('IN', 'IND', 356, 'India', 'New Delhi', 'AS', 'INR'),
    ('LA', 'LAO', 418, 'Laos', 'Vientiane', 'AS', 'LAK'),
    ('US', 'USA', 840, 'United States', 'Washington', 'NA', 'USD'),
    ('ZW', 'ZWE', 716, 'Zimbabwe', 'Harare', 'AF', 'ZWL')
;
```

オンラインでデータベースとテーブルのをむ https://riptutorial.com/ja/sql/topic/280/データベースとテーブルの

34: データ

Examples

DECIMALおよびNUMERIC

とり。 `DECIMAL`と`NUMERIC`はにです。

```
DECIMAL ( precision [ , scale] )
NUMERIC ( precision [ , scale] )
```

```
SELECT CAST(123 AS DECIMAL(5,2)) --returns 123.00
SELECT CAST(12345.12 AS NUMERIC(10,5)) --returns 12345.12000
```

FLOATとREAL

データでするデータ。

```
SELECT CAST( PI() AS FLOAT) --returns 3.14159265358979
SELECT CAST( PI() AS REAL) --returns 3.141593
```

データをするなのデータ。

データ·タイプ		ストレージ
ビッグトリント	-2 ^ 63-9,223,372,036,854,775,808から2 ^ 63-1 9,223,372,036,854,775,807	8バイト
int	-2 ^ 31-2,147,483,648から2 ^ 31-12,147,483,647	4バイト
さい	-2 ^ 15-32,7682 ^ 15-132,767	2バイト
tinyint	0255	1バイト

と

またはのをすデータ。

データ·タイプ		ストレージ
お	-922,337,203,685,477.5808922,337,203,685,477.5807	8バイト
スモールマネー	-214,748.3648214,748.3647	4バイト

バイナリと

またはのバイナリデータ。

```
BINARY [ ( n_bytes ) ]
VARBINARY [ ( n_bytes | max ) ]
```

`n_bytes`は、 `n_bytes`バイトののです。 `max`は、が2 ^ 31-1であることをします。

```
SELECT CAST(12345 AS BINARY(10)) -- 0x0000000000000003039
SELECT CAST(12345 AS VARBINARY(10)) -- 0x00003039
```

CHARおよびVARCHAR

またはのいずれかのデータ。

```
CHAR [ ( n_chars ) ]
VARCHAR [ ( n_chars ) ]
```

```
SELECT CAST('ABC' AS CHAR(10)) -- 'ABC       ' (padded with spaces on the right)
SELECT CAST('ABC' AS VARCHAR(10)) -- 'ABC' (no padding due to variable character)
SELECT CAST('ABCDEFGHIJKLMNOPQRSTUVWXYZ' AS CHAR(10))  -- 'ABCDEFGHIJ' (truncated to 10
characters)
```

NCHARとNVARCHAR

UNICODEまたはのデータ。

```
NCHAR [ ( n_chars ) ]
NVARCHAR [ ( n_chars | MAX ) ]
```

8000をえるにいには`MAX`をしてください。

ユニークアイデンティファイア

16バイトのGUID / UUID。

```
DECLARE @GUID UNIQUEIDENTIFIER = NEWID();
SELECT @GUID -- 'E28B3BD9-9174-41A9-8508-899A78A33540'
DECLARE @bad_GUID_string VARCHAR(100) = 'E28B3BD9-9174-41A9-8508-899A78A33540_foobarbaz'
SELECT
    @bad_GUID_string,   -- 'E28B3BD9-9174-41A9-8508-899A78A33540_foobarbaz'
    CONVERT(UNIQUEIDENTIFIER, @bad_GUID_string) -- 'E28B3BD9-9174-41A9-8508-899A78A33540'
```

オンラインでデータをむ https://riptutorial.com/ja/sql/topic/1166/データ

35: テーブルデザイン

オープン1999リレーショナルデータベースシステムブロック2リレーショナル、ミルトンケインズ、オープン。

Examples

よくされたテーブルのプロパティ。

のリレーショナル·データベースは、データをいくつかのにげみ、そのデータをりすためのSQLをくがあります。
テーブルがひどくされていると、クエリのがくなり、データベースがしたとおりにしなくなるがあります。

データベーステーブルはなるのテーブルとなすべきではありません。それはにリレーショナルとなされるためののルールにわなければなりません。には、それをするために「」とばれます。

リレーショナルの5つのルールはのとおりです。

1. はアトミックです。のフィールドのはのでなければなりません。
2. フィールドには、じデータのがまれています。
3. フィールドしにはのがあります。
4. のには、ののレコードでにするためのがなくとも1つです。
5. とのにはがありません。

5つのルールにうテーブル

イド		DOB	マネージャー
1	フレッド	1971112	3
2	フレッド	1971112	3
3	える	197587	2

- ルール1はアトミックです。 `Id` 、 `Name` 、 `DOB`および`Manager`にはのしかまれていません。
- ルール2 `Id`にはだけがまれ、 `Name`はテキストがまれています4のテキストであることをできます `DOB`にはなタイプのがまれており、 `Manager`はがまれています。
- ルール3 `Id` 、 `Name` 、 `DOB`および`Manager`は、ののしです。
- ルール4 `Id`フィールドをめると、レコードがテーブルののレコードとされます。

ひどくされたテーブル

イド		DOB	
1	フレッド	1971112	3
1	フレッド	1971112	3
3	える	1975718	2,1

- ルール12のフィールドには、2と1の2つのがまれます。
- ルール2DOBフィールドにはとテキストがまれます。
- ルール3「」という2つのフィールドがあります。
- ルール4と2のレコードはまったくじです。
- ルール5このルールはられていません。

オンラインでテーブルデザインをむ https://riptutorial.com/ja/sql/topic/2515/テーブルデザイン

36: トランザクション

トランザクションは、1つまたはのステップをむのです。ステップは、トランザクションがデータベースにコミットするためににするがあります。エラーがあると、すべてのデータがされ、データベースはトランザクションのににロールバックされます。

Examples

```
BEGIN TRANSACTION
    INSERT INTO DeletedEmployees(EmployeeID, DateDeleted, User)
        (SELECT 123, GetDate(), CURRENT_USER);
    DELETE FROM Employees WHERE EmployeeID = 123;
COMMIT TRANSACTION
```

ロールバック

トランザクションコードでかがし、にすは、トランザクションをロールバックすることができます。

```
BEGIN TRY
    BEGIN TRANSACTION
        INSERT INTO Users(ID, Name, Age)
        VALUES(1, 'Bob', 24)

        DELETE FROM Users WHERE Name = 'Todd'
   COMMIT TRANSACTION
END TRY
BEGIN CATCH
   ROLLBACK TRANSACTION
END CATCH
```

オンラインでトランザクションをむ https://riptutorial.com/ja/sql/topic/2424/トランザクション

37: トリガー

Examples

CREATE TRIGGER

このでは、トリガがされているテーブルMyTableにレコードがされた、2のテーブルMyAuditにレコードをするトリガをします。ここで、「された」テーブルは、INSERTステートメントおよびUPDATEステートメントのにをけたをするためにMicrosoft SQL Serverによってされるなテーブルです。 DELETEにしてもじをするな「」があります。

```
CREATE TRIGGER MyTrigger
    ON MyTable
    AFTER INSERT

AS

BEGIN
    -- insert audit record to MyAudit table
    INSERT INTO MyAudit(MyTableId, User)
    (SELECT MyTableId, CURRENT_USER FROM inserted)
END
```

トリガーをして、みアイテムの「ごみ」をする

```
CREATE TRIGGER BooksDeleteTrigger
    ON MyBooksDB.Books
    AFTER DELETE
AS
  INSERT INTO BooksRecycleBin
    SELECT *
    FROM deleted;
GO
```

オンラインでトリガーをむ https://riptutorial.com/ja/sql/topic/1432/トリガー

38: ヌル

き

SQLのNULLは、プログラミングとに、り「」をします。 SQLでは、「の」としてするのがです。

の''や0などののとすることがです。どちらもにはNULLではありません。

また、 NULLをでまないようにすることもです。これは、テキストをけれるではされていNULLではなく、エラーやなデータセットをきこすのある'NULL'などです。

Examples

クエリでNULLをフィルタリングする

WHEREブロックでNULLをフィルタリングするためのつまり、がないことは、ののフィルタリングとはしなります。

```
SELECT * FROM Employees WHERE ManagerId IS NULL ;
SELECT * FROM Employees WHERE ManagerId IS NOT NULL ;
```

NULLはにもしくないので、= NULLまたは<> NULL または!= NULL をすると、 WHEREによってされるUNKNOWNのがにられ!= NULL。

WHEREは、がFALSEまたはUKNOWNであるすべてのをフィルタリングし、がTRUEのみをしFALSE。

のNULLな

テーブルをするときには、カラムをヌルまたはヌルでないとすることができます。

```
CREATE TABLE MyTable
(
    MyCol1 INT NOT NULL, -- non-nullable
    MyCol2 INT NULL      -- nullable
) ;
```

デフォルトでは、 NOT NULLをにしないり、すべてのキーのをくはNULLです。

NULLをできないにNULLをしようとすると、エラーがします。

```
INSERT INTO MyTable (MyCol1, MyCol2) VALUES (1, NULL) ;  -- works fine

INSERT INTO MyTable (MyCol1, MyCol2) VALUES (NULL, 2) ;
        -- cannot insert
        -- the value NULL into column 'MyCol1', table 'MyTable';
        -- column does not allow nulls. INSERT fails.
```

フィールドをNULLにする

フィールドをNULLすると、のとまったくじようにしNULL。

```
UPDATE Employees
SET ManagerId = NULL
WHERE Id = 4
```

NULLフィールドをつの

たとえば、をたずマネージャをたないをEmployeesののにするとします。

```
INSERT INTO Employees
    (Id, FName, LName, PhoneNumber, ManagerId, DepartmentId, Salary, HireDate)
VALUES
    (5, 'Jane', 'Doe', NULL, NULL, 2, 800, '2016-07-22') ;
```

オンラインでヌルをむ https://riptutorial.com/ja/sql/topic/3421/ヌル

39: ビュー

Examples

シンプルなビュー

ビューは、からいくつかのをフィルターにけることも、そののみをすることもできます。

```
CREATE VIEW new_employees_details AS
SELECT E.id, Fname, Salary, Hire_date
FROM Employees E
WHERE hire_date > date '2015-01-01';
```

フォームをした

```
select * from new_employees_details
```

イド	FName		Hire_date
4	ジョンナトン	500	24-07-2016

なビュー

ビューはになクエリ、、サブクエリなどです。したすべてのをずしてください

```
Create VIEW dept_income AS
SELECT d.Name as DepartmentName, sum(e.salary) as TotalSalary
FROM Employees e
JOIN Departments d on e.DepartmentId = d.id
GROUP BY d.Name;
```

これで、どのテーブルからでもできます

```
SELECT *
FROM dept_income;
```

	TotalSalary
HR	1900
	600

オンラインでビューをむ https://riptutorial.com/ja/sql/topic/766/ビュー

40: マージ

き

MERGE「または」のためにUPSERTとばれることもいでは、しいをするか、がすでにするはのをすることができます。なは、をにしてデータのをつため、クライアント/サーバー・システムののSQLにするオーバーヘッドをぐことです。

Examples

ターゲットマッチソースをるためのマージ

```
MERGE INTO targetTable t
    USING sourceTable s
        ON t.PKID = s.PKID
    WHEN MATCHED AND NOT EXISTS (
            SELECT s.ColumnA, s.ColumnB, s.ColumnC
            INTERSECT
            SELECT t.ColumnA, t.ColumnB, s.ColumnC
            )
        THEN UPDATE SET
            t.ColumnA = s.ColumnA
            ,t.ColumnB = s.ColumnB
            ,t.ColumnC = s.ColumnC
    WHEN NOT MATCHED BY TARGET
        THEN INSERT (PKID, ColumnA, ColumnB, ColumnC)
        VALUES (s.PKID, s.ColumnA, s.ColumnB, s.ColumnC)
    WHEN NOT MATCHED BY SOURCE
        THEN DELETE
    ;
```

`AND NOT EXISTS`は、されていないレコードのをします。 `INTERSECT`をすると、なをわずにNULLのをすることができます。

MySQLユーザーをでえる

じのユーザーをりたいとします。のようにテーブル`users`をしましょう。

```
create table users(
    id int primary key auto_increment,
    name varchar(8),
    count int,
    unique key name(name)
);
```

さて、ジョーというのしいユーザーをつけたので、をにれたいとします。これをするには、をつのがあるかどうかをするがあります。するはカウントにします。、のがないは、するがあります。

MySQLはのをします insert ... on duplicate key update ...この

```
insert into users(name, count)
       values ('Joe', 1)
       on duplicate key update count=count+1;
```

PostgreSQLユーザーをでえる

じのユーザーをりたいとします。のようにテーブル`users`をしましょう。

```
create table users(
    id serial,
    name varchar(8) unique,
    count int
);
```

さて、ジョーというのしいユーザーをつけたので、をにれたいといます。これをするには、をつのがあるかどうかをするがあります。するはカウントにします。、のがないは、するがあります。

PostgreSQLはのをします insert ... on conflict ... do update ...。この

```
insert into users(name, count)
    values('Joe', 1)
    on conflict (name) do update set count = users.count + 1;
```

オンラインでマージをむ https://riptutorial.com/ja/sql/topic/1470/マージ

41: マテリアライズド･ビュー

き

マテリアライズド･ビューは、がにされ、のにするためににリフレッシュするがあるビューです。したがって、リアルタイムがなに、でのクエリのをするのにです。マテリアライズド･ビューは、OracleおよびPostgreSQLでできます。のデータベースシステムは、SQL ServerのインデックスきビューやDB2のマテリアライズクエリテーブルなど、のをします。

Examples

PostgreSQLの

```
CREATE TABLE mytable (number INT);
INSERT INTO mytable VALUES (1);

CREATE MATERIALIZED VIEW myview AS SELECT * FROM mytable;

SELECT * FROM myview;
 number
--------
      1
(1 row)

INSERT INTO mytable VALUES(2);

SELECT * FROM myview;
 number
--------
      1
(1 row)

REFRESH MATERIALIZED VIEW myview;

SELECT * FROM myview;
 number
--------
      1
      2
(2 rows)
```

オンラインでマテリアライズド･ビューをむ https://riptutorial.com/ja/sql/topic/8367/マテリアライズド-ビュー

42: キー

- MySQLCREATE TABLEId int NOT NULL、PRIMARY KEYID、...;
- そのCREATE TABLEId int NOT NULL PRIMARY KEY、...;

Examples

キーの

```
CREATE TABLE Employees (
    Id int NOT NULL,
    PRIMARY KEY (Id),
    ...
);
```

これは、キーとして 'Id'をつEmployeesテーブルをします。キーは、のをにするためにできます。1つのテーブルにつき1つのキーのみがされます。

キーは、キーとばれるのの1つのフィールドですることもできます。

```
CREATE TABLE EMPLOYEE (
    e1_id INT,
    e2_id INT,
    PRIMARY KEY (e1_id, e2_id)
)
```

インクリメントをする

くのデータベースでは、しいキーがされるとプライマリキーがにインクリメントされます。これにより、すべてのキーがなることがされます。

MySQL

```
CREATE TABLE Employees (
    Id int NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (Id)
);
```

PostgreSQL

```
CREATE TABLE Employees (
    Id SERIAL PRIMARY KEY
);
```

SQLサーバー

```
CREATE TABLE Employees (
    Id int NOT NULL IDENTITY,
    PRIMARY KEY (Id)
);
```

SQLite

```
CREATE TABLE Employees (
    Id INTEGER PRIMARY KEY
);
```

オンラインでキーをむ https://riptutorial.com/ja/sql/topic/505/キー

43: の

き

SQLのALTERコマンドは、の/をするためにされます。

- ALTER TABLE [テーブル] ADD [カラム] [データ]

Examples

を

```
ALTER TABLE Employees
ADD StartingDate date NOT NULL DEFAULT GetDate(),
    DateOfBirth date NULL
```

のステートメントは、のとしてデフォルトでNULLにできない`StartingDate`というのをし、Employeesテーブルで NULLになる`DateOfBirth`をします。

をとす

```
ALTER TABLE Employees
DROP COLUMN salary;
```

これにより、そののがされるだけでなく、テーブルのからのがされますはしなくなります。

ドロップ

```
ALTER TABLE Employees
DROP CONSTRAINT DefaultSalary
```

これは、EmployeeテーブルからDefaultSalaryというをします。

-をするに、のがされていることをしてください。

をする

```
ALTER TABLE Employees
ADD CONSTRAINT DefaultSalary DEFAULT ((100)) FOR [Salary]
```

これにより、Salaryのデフォルト100をするDefaultSalaryというがされます。

レベルでをできます。

の

- キー - テーブルのレコードがしないようにする
- キー - のテーブルからのキーへのポインタ
- Not Null - NULLがにされないようにする
- ユニーク - テーブルのレコードをに
- デフォルト - デフォルトをします。
- チェック - にできるのをします。

のは、 Oracleのマニュアルをしてください。

をする

```
ALTER TABLE Employees
ALTER COLUMN StartingDate DATETIME NOT NULL DEFAULT (GETDATE())
```

このせは、 `StartingDate`のデータをし、な`date`から`datetime`し、デフォルトをのにします。

キーを

```
ALTER TABLE EMPLOYEES ADD pk_EmployeeID PRIMARY KEY (ID)
```

これにより、フィールド`ID`テーブルEmployeesにプライマリキーがされ`ID`。 IDとにカッコにのをめると、キーがされます。のをするは、をコンマでるがあります。

```
ALTER TABLE EMPLOYEES ADD pk_EmployeeID PRIMARY KEY (ID, FName)
```

オンラインでのをむ https://riptutorial.com/ja/sql/topic/356/の

44:

EXCEPTは、EXCEPTのにあるデータセットから、しいデータセットからされないのをします。

Examples

こののデータセットにがあるをいて、データセットをしてください

```
--dataset schemas must be identical
SELECT 'Data1' as 'Column' UNION ALL
SELECT 'Data2' as 'Column' UNION ALL
SELECT 'Data3' as 'Column' UNION ALL
SELECT 'Data4' as 'Column' UNION ALL
SELECT 'Data5' as 'Column'
EXCEPT
SELECT 'Data3' as 'Column'
--Returns Data1, Data2, Data4, and Data5
```

オンラインでをむ https://riptutorial.com/ja/sql/topic/4082/

45: テーブル

- WITH QueryName [ColumnName、...] AS
  SELECT ...

  SELECT ... FROM QueryName ...;

- WITH RECURSIVE QueryName [ColumnName、...] AS
  SELECT ...
  UNION [すべて]
  SELECT ... FROM QueryName ...

  SELECT ... FROM QueryName ...;

ドキュメント WITH

はなセットであり、なせのであるがあります。 WITHをしてします。 CTEはをさせ、Temp TableとTableがされるTempDBデータベースではなくメモリにされます。

Common Table Expressionsのな

- なクエリ、になやサブクエリをするためにできます。
- クエリをカプセルするです。
- のクエリがされるまでします。
- しくすると、コードの/とのがします。
- じステートメントでのをすることができますSQLのをします。
- ビューのながない、ビューのわりになることができます。つまり、をメタデータにするはありません。
- されているときではなく、びされたときにされます。 CTEがクエリでされている、CTEはされますがなるがあります。

Examples

なクエリ

これらはネストされたサブクエリとじようにしますが、はなります。

```
WITH ReadyCars AS (
  SELECT *
  FROM Cars
  WHERE Status = 'READY'
)
SELECT ID, Model, TotalCost
FROM ReadyCars
ORDER BY TotalCost;
```

ID	モデル	
1	フォードF-150	200
2	フォードF-150	230

のサブクエリ

```
SELECT ID, Model, TotalCost
FROM (
  SELECT *
  FROM Cars
  WHERE Status = 'READY'
) AS ReadyCars
ORDER BY TotalCost
```

ににる

```
WITH RECURSIVE ManagersOfJonathon AS (
    -- start with this row
    SELECT *
    FROM Employees
    WHERE ID = 4

    UNION ALL

    -- get manager(s) of all previously selected rows
    SELECT Employees.*
    FROM Employees
    JOIN ManagersOfJonathon
        ON Employees.ID = ManagersOfJonathon.ManagerID
)
SELECT * FROM ManagersOfJonathon;
```

イド	FName	LName		マネージャーID	DepartmentId
4	ジョンナトン	スミス	1212121212	**2**	1
2	ジョン	ジョンソン	2468101214	**1**	1
1	ジエームス	スミス	1234567890	ヌル	1

をする

ほとんどのデータベースには、のためののをするのはありません。ただし、テーブルをでして、そのタイプのをエミュレートすることができます。

のでは、15のをつ`i`をつ`Numbers`というテーブルをします。

```
--Give a table name `Numbers" and a column `i` to hold the numbers
WITH Numbers(i) AS (
```

```
    --Starting number/index
    SELECT 1
    --Top-level UNION ALL operator required for recursion
    UNION ALL
    --Iteration expression:
    SELECT i + 1
    --Table expression we first declared used as source for recursion
    FROM Numbers
    --Clause to define the end of the recursion
    WHERE i < 5
)
--Use the generated table expression like a regular table
SELECT i FROM Numbers;
```

1
2
3
4
5

このは、のおよびのタイプのデータとともにできます。

にする

```
WITH RECURSIVE ManagedByJames(Level, ID, FName, LName) AS (
    -- start with this row
    SELECT 1, ID, FName, LName
    FROM Employees
    WHERE ID = 1

    UNION ALL

    -- get employees that have any of the previously selected rows as manager
    SELECT ManagedByJames.Level + 1,
           Employees.ID,
           Employees.FName,
           Employees.LName
    FROM Employees
    JOIN ManagedByJames
        ON Employees.ManagerID = ManagedByJames.ID

    ORDER BY 1 DESC   -- depth-first search
)
SELECT * FROM ManagedByJames;
```

レベル	ID	FName	LName
1	1	ジェームス	スミス

レベル	ID	FName	LName
2	2	ジョン	ジョンソン
3	4	ジョンナトン	スミス
2	3	マイケル	ウィリアムズ

CTEをしたOracle CONNECT BY

OracleのCONNECT BYは、SQLのCTEをするにみみのない、くのでなをします。このでは、SQL Serverのをして、これらのをしていますのためにいくつかしています。 Oracleのは、のデータベースのなせにけているくのをつけるのにもちますが、になせでができるかをすこともできます。

```
  WITH tbl AS (
       SELECT id, name, parent_id
         FROM mytable)
     , tbl_hierarchy AS (
       /* Anchor */
       SELECT 1 AS "LEVEL"
            --, 1 AS CONNECT_BY_ISROOT
            --, 0 AS CONNECT_BY_ISBRANCH
            , CASE WHEN t.id IN (SELECT parent_id FROM tbl) THEN 0 ELSE 1 END AS
CONNECT_BY_ISLEAF
            , 0 AS CONNECT_BY_ISCYCLE
            , '/' + CAST(t.id   AS VARCHAR(MAX)) + '/' AS SYS_CONNECT_BY_PATH_id
            , '/' + CAST(t.name AS VARCHAR(MAX)) + '/' AS SYS_CONNECT_BY_PATH_name
            , t.id AS root_id
            , t.*
         FROM tbl t
        WHERE t.parent_id IS NULL                           -- START WITH parent_id IS NULL
       UNION ALL
       /* Recursive */
       SELECT th."LEVEL" + 1 AS "LEVEL"
            --, 0 AS CONNECT_BY_ISROOT
            --, CASE WHEN t.id IN (SELECT parent_id FROM tbl) THEN 1 ELSE 0 END AS
CONNECT_BY_ISBRANCH
            , CASE WHEN t.id IN (SELECT parent_id FROM tbl) THEN 0 ELSE 1 END AS
CONNECT_BY_ISLEAF
            , CASE WHEN th.SYS_CONNECT_BY_PATH_id LIKE '%/' + CAST(t.id AS VARCHAR(MAX)) +
'/%' THEN 1 ELSE 0 END AS CONNECT_BY_ISCYCLE
            , th.SYS_CONNECT_BY_PATH_id   + CAST(t.id   AS VARCHAR(MAX)) + '/' AS
SYS_CONNECT_BY_PATH_id
            , th.SYS_CONNECT_BY_PATH_name + CAST(t.name AS VARCHAR(MAX)) + '/' AS
SYS_CONNECT_BY_PATH_name
            , th.root_id
            , t.*
         FROM tbl t
              JOIN tbl_hierarchy th ON (th.id = t.parent_id) -- CONNECT BY PRIOR id =
parent_id
        WHERE th.CONNECT_BY_ISCYCLE = 0)                    -- NOCYCLE
SELECT th.*
     --, REPLICATE(' ', (th."LEVEL" - 1) * 3) + th.name AS tbl_hierarchy
  FROM tbl_hierarchy th
       JOIN tbl CONNECT_BY_ROOT ON (CONNECT_BY_ROOT.id = th.root_id)
 ORDER BY th.SYS_CONNECT_BY_PATH_name;                      -- ORDER SIBLINGS BY name
```

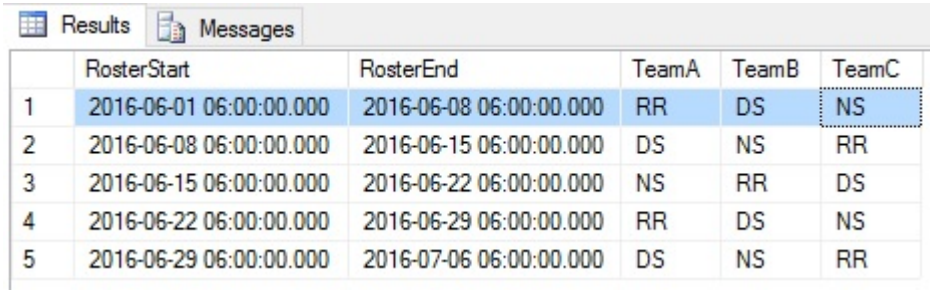
CONNECT BYのとのと

- ◦ CONNECT BYをするをします。
  ◦ START WITHルートノードをします。
  ◦ のはです。
- パラメータ
  ◦ NOCYCLEループがされたときにのをします。なはDirected Acyclic Graphsであり、はこのにします。
- ◦ PRIORノードのからデータをします。
  ◦ CONNECT_BY_ROOTノードのルートからデータをします。
- ◦ LEVELルートからのノードのをします。
  ◦ CONNECT_BY_ISLEAFがないノードをします。
  ◦ CONNECT_BY_ISCYCLEをつノードをします。
- ◦ SYS_CONNECT_BY_PATHルートからノードへのパスのフラット/をします。

にをし、としてチームのロストをむように

```
DECLARE @DateFrom DATETIME = '2016-06-01 06:00'
DECLARE @DateTo DATETIME = '2016-07-01 06:00'
DECLARE @IntervalDays INT = 7

-- Transition Sequence = Rest & Relax into Day Shift into Night Shift
-- RR (Rest & Relax) = 1
-- DS (Day Shift) = 2
-- NS (Night Shift) = 3

;WITH roster AS
(
   SELECT @DateFrom AS RosterStart, 1 AS TeamA, 2 AS TeamB, 3 AS TeamC
   UNION ALL
   SELECT DATEADD(d, @IntervalDays, RosterStart),
          CASE TeamA WHEN 1 THEN 2 WHEN 2 THEN 3 WHEN 3 THEN 1 END AS TeamA,
          CASE TeamB WHEN 1 THEN 2 WHEN 2 THEN 3 WHEN 3 THEN 1 END AS TeamB,
          CASE TeamC WHEN 1 THEN 2 WHEN 2 THEN 3 WHEN 3 THEN 1 END AS TeamC
   FROM roster WHERE RosterStart < DATEADD(d, -@IntervalDays, @DateTo)
)

SELECT RosterStart,
       ISNULL(LEAD(RosterStart) OVER (ORDER BY RosterStart), RosterStart + @IntervalDays) AS
RosterEnd,
       CASE TeamA WHEN 1 THEN 'RR' WHEN 2 THEN 'DS' WHEN 3 THEN 'NS' END AS TeamA,
       CASE TeamB WHEN 1 THEN 'RR' WHEN 2 THEN 'DS' WHEN 3 THEN 'NS' END AS TeamB,
       CASE TeamC WHEN 1 THEN 'RR' WHEN 2 THEN 'DS' WHEN 3 THEN 'NS' END AS TeamC
FROM roster
```

つまり、1はTeamAがRR、TeamBがDay Shift、TeamCがNight Shiftです。



Results Messages

	RosterStart	RosterEnd	TeamA	TeamB	TeamC
1	2016-06-01 06:00:00.000	2016-06-08 06:00:00.000	RR	DS	NS
2	2016-06-08 06:00:00.000	2016-06-15 06:00:00.000	DS	NS	RR
3	2016-06-15 06:00:00.000	2016-06-22 06:00:00.000	NS	RR	DS
4	2016-06-22 06:00:00.000	2016-06-29 06:00:00.000	RR	DS	NS
5	2016-06-29 06:00:00.000	2016-07-06 06:00:00.000	DS	NS	RR

テーブルをするクエリのリファクタリング

が20をえるすべてのカテゴリをするとします。

テーブルがないクエリをにします。

```
SELECT category.description, sum(product.price) as total_sales
FROM sale
LEFT JOIN product on sale.product_id = product.id
LEFT JOIN category on product.category_id = category.id
GROUP BY category.id, category.description
HAVING sum(product.price) > 20
```

Common Table Expressionsをしたのクエリ

```
WITH all_sales AS (
  SELECT product.price, category.id as category_id, category.description as
category_description
  FROM sale
  LEFT JOIN product on sale.product_id = product.id
  LEFT JOIN category on product.category_id = category.id
)
, sales_by_category AS (
  SELECT category_description, sum(price) as total_sales
  FROM all_sales
  GROUP BY category_id, category_description
)
SELECT * from sales_by_category WHERE total_sales > 20
```

をつSQLの

「のカテゴリ」から「もい」をするとします。

Common Table Expressionsをしたクエリのをにします

```
-- all_sales: just a simple SELECT with all the needed JOINS
WITH all_sales AS (
  SELECT
  product.price as product_price,
  category.id as category_id,
  category.description as category_description
  FROM sale
  LEFT JOIN product on sale.product_id = product.id
  LEFT JOIN category on product.category_id = category.id
)
```

```
-- Group by category
, sales_by_category AS (
  SELECT category_id, category_description,
  sum(product_price) as total_sales
  FROM all_sales
  GROUP BY category_id, category_description
)
-- Filtering total_sales > 20
, top_categories AS (
  SELECT * from sales_by_category WHERE total_sales > 20
)
-- all_products: just a simple SELECT with all the needed JOINS
, all_products AS (
  SELECT
  product.id as product_id,
  product.description as product_description,
  product.price as product_price,
  category.id as category_id,
  category.description as category_description
  FROM product
  LEFT JOIN category on product.category_id = category.id
)
-- Order by product price
, cheapest_products AS (
  SELECT * from all_products
  ORDER by product_price ASC
)
-- Simple inner join
, cheapest_products_from_top_categories AS (
  SELECT product_description, product_price
  FROM cheapest_products
  INNER JOIN top_categories ON cheapest_products.category_id = top_categories.category_id
)
--The main SELECT
SELECT * from cheapest_products_from_top_categories
```

オンラインでテーブルをむ https://riptutorial.com/ja/sql/topic/747/テーブル

46:

き

DELETEステートメントは、テーブルからレコードをするためにされます。

1. DELETE FROM *TableName* [WHERE] [LIMIT カウント]

Examples

のをWHEREでする

これにより、 `WHERE`にするすべてのがされます。

```
DELETE FROM Employees
WHERE FName = 'John'
```

すべてのをする

`WHERE`をすると、テーブルからすべてのがされます。

```
DELETE FROM Employees
```

TRUNCATEのパフォーマンスは、データをするだけのトリガーやインデックス、ログをするため、TRUNCATEのパフォーマンスがするみのについては、 TRUNCATEのマニュアルをしてください。

TRUNCATE

これをして、テーブルをしたにリセットします。これにより、すべてのがされ、インクリメントなどのがリセットされます。々ののもログにされません。

```
TRUNCATE TABLE Employees
```

のテーブルとのにづいてのをする

のテーブルののデータとするまたはしない、テーブルからデータを`DELETE`することはです。

いったんターゲットにロードされたソースからデータを`DELETE`たいとしましょう。

```
DELETE FROM Source
WHERE  EXISTS ( SELECT 1 -- specific value in SELECT doesn't matter
                FROM Target
                Where Source.ID = Target.ID )
```

もなRDBMSMySQL、Oracle、PostgresSQL、Teradataなどでは、 `DELETE`にテーブルをすることができ、コンパクトなでよりながです。

のシナリオにさをえると、Aggregateは1に1Targetからされ、じIDをまずじをむとします。がそのにされたにのみ 、Sourceからデータをしたいとします。

MySQL、Oracle、Teradataでは、をしてこれをできます。

```
DELETE FROM Source
WHERE  Source.ID = TargetSchema.Target.ID
       AND TargetSchema.Target.Date = AggregateSchema.Aggregate.Date
```

PostgreSQLでの

```
DELETE FROM Source
USING  TargetSchema.Target, AggregateSchema.Aggregate
WHERE  Source.ID = TargetSchema.Target.ID
       AND TargetSchema.Target.DataDate = AggregateSchema.Aggregate.AggDate
```

これはにソース、ターゲット、ののINNER JOINをもたらします。これらのIDのTargetにするTargetとDateにじIDがするは、Sourceにもがされます。

のように、じクエリをきむこともできますMySQL、Oracle、Teradata

```
DELETE Source
FROM   Source, TargetSchema.Target, AggregateSchema.Aggregate
WHERE  Source.ID = TargetSchema.Target.ID
       AND TargetSchema.Target.DataDate = AggregateSchema.Aggregate.AggDate
```

なは、のRDBMSOracle、MySQLなどでは`Delete`でできますが、すべてのプラットフォームではサポートされていませんTeradataではサポートされていません

は、シナリオをすべてのスタイルとさせるわりにチェックするようにできますにし`NOT EXISTS`ことをしてください

```
DELETE FROM Source
WHERE NOT EXISTS ( SELECT 1 -- specific value in SELECT doesn't matter
               FROM Target
               Where Source.ID = Target.ID )
```

オンラインでをむ https://riptutorial.com/ja/sql/topic/1105/

47: と

- GRANT [1] [、[2] ...] ON [テーブル] TO [1] [、[2] ...] [された]
- REVOKE [1] [、[2] ...] ON [テーブル] FROM [1] [、[2] ...]

ユーザーにアクセスをえます。 `WITH GRANT OPTION`がされている、はされたをするまたはにされたをりすをします。

Examples

のりし

```
GRANT SELECT, UPDATE
ON Employees
TO User1, User2;
```

`Employees`テーブルにして`SELECT`と`UPDATE`をするを`User1`と`User2`えます。

```
REVOKE SELECT, UPDATE
ON Employees
FROM User1, User2;
```

`User1`および`User2`から、Employeesテーブルにして`SELECT`および`UPDATE`をするをりします。

オンラインでとをむ https://riptutorial.com/ja/sql/topic/5574/と

48:

き

CASEは、if-thenロジックをするためにされます。

- CASE input_expression
  WHEN compare1 THEN result1
  [WHEN compare2 THEN result2] ...
  [ELSE resultX]
  わり
- WHEN1 THEN1
  [WHEN condition2 THEN result2] ...
  [ELSE resultX]
  わり

*CASE*は、そののをし`compareX`にしい`input_expression`。

された*CASE*は、 `conditionX`がであるのをします。

Examples

SELECTでCASEをするブールに

ブールがTRUEの、 された CASEはをします。

これは、とののみをチェックできるなケースとはなります。

```
SELECT Id, ItemId, Price,
  CASE WHEN Price < 10 THEN 'CHEAP'
       WHEN Price < 20 THEN 'AFFORDABLE'
       ELSE 'EXPENSIVE'
  END AS PriceRating
FROM ItemSales
```

イド	ItemId		PriceRating
1	100	34.5	な
2	145	2.3	いです
3	100	34.5	な
4	100	34.5	な
5	145	10	

にするのを**CASE**から**COUNT**までします。

`CASE`は、 `SUM`とみわせてして、されたにするのみのカウントをすことができます。 Excelの`COUNTIF`にています。

このトリックは、をすバイナリをすことで、するエントリにしてされる "1"は、ののにしてされることができます。

この`ItemSales`えられた、「」としてされたアイテムのをりたいとしましょう。

イド	ItemId		PriceRating
1	100	34.5	な
2	145	2.3	いです
3	100	34.5	な
4	100	34.5	な
5	145	10	

クエリ

```
SELECT
    COUNT(Id) AS ItemsCount,
    SUM ( CASE
            WHEN PriceRating = 'Expensive' THEN 1
            ELSE 0
          END
        ) AS ExpensiveItemsCount
FROM ItemSales
```

ItemsCount	ExpensiveItemsCount
5	3

```
SELECT
    COUNT(Id) as ItemsCount,
    SUM (
        CASE PriceRating
            WHEN 'Expensive' THEN 1
            ELSE 0
        END
       ) AS ExpensiveItemsCount
FROM ItemSales
```

SELECTの

CASEのは、のにしてはをします。このはしく、されたをももりしてします。ただし、 ELSEはききできます。

```
SELECT Id, ItemId, Price,
  CASE Price WHEN 5  THEN 'CHEAP'
             WHEN 15 THEN 'AFFORDABLE'
             ELSE         'EXPENSIVE'
  END as PriceRating
FROM ItemSales
```

の。ショートバリアントをする、 WHENごとにがされることをすることがです。したがって、のステートメント

```
SELECT
    CASE ABS(CHECKSUM(NEWID())) % 4
        WHEN 0 THEN 'Dr'
        WHEN 1 THEN 'Master'
        WHEN 2 THEN 'Mr'
        WHEN 3 THEN 'Mrs'
    END
```

NULLをするがありNULL。なぜなら、 NEWID()がしいでびびされているWHENです。に

```
SELECT
    CASE
        WHEN ABS(CHECKSUM(NEWID())) % 4 = 0 THEN 'Dr'
        WHEN ABS(CHECKSUM(NEWID())) % 4 = 1 THEN 'Master'
        WHEN ABS(CHECKSUM(NEWID())) % 4 = 2 THEN 'Mr'
        WHEN ABS(CHECKSUM(NEWID())) % 4 = 3 THEN 'Mrs'
    END
```

したがって、 WHENすべてのケースがWHEN 、がNULL。

ORDER BYのCASE

1,2,3 ..をってのタイプをめることができます

```
SELECT * FROM DEPT
ORDER BY
CASE DEPARTMENT
      WHEN 'MARKETING' THEN  1
      WHEN 'SALES' THEN 2
      WHEN 'RESEARCH' THEN 3
      WHEN 'INNOVATION' THEN 4
      ELSE        5
      END,
      CITY
```

ID		シティ		EMPLOYEES_NUMBER
12	ニューイングランド	ボストン	マーケティング	9

ID		シティ		EMPLOYEES_NUMBER
15		サンフランシスコ	マーケティング	12
9		シカゴ		8
14		ニューヨーク		12
5		ロサンゼルス		11
10		フィラデルフィア		13
4		シカゴ		11
2		デトロイト		9

UPDATEでのCASEの

のサンプル

```
UPDATE ItemPrice
SET Price = Price *
  CASE ItemId
    WHEN 1 THEN 1.05
    WHEN 2 THEN 1.10
    WHEN 3 THEN 1.15
    ELSE 1.00
  END
```

にけられたNULLのCASE

このようにして、のをす '0'がにランクけされ、NULLをす '1'がにソートされます。

```
SELECT ID
      ,REGION
      ,CITY
      ,DEPARTMENT
      ,EMPLOYEES_NUMBER
  FROM DEPT
  ORDER BY
  CASE WHEN REGION IS NULL THEN 1
  ELSE 0
  END,
  REGION
```

ID		シティ		EMPLOYEES_NUMBER
10		フィラデルフィア		13
14		ニューヨーク		12

ID		シティ		EMPLOYEES_NUMBER
9		シカゴ		8
12	ニューイングランド	ボストン	マーケティング	9
5		ロサンゼルス		11
15	ヌル	サンフランシスコ	マーケティング	12
4	ヌル	シカゴ		11
2	ヌル	デトロイト		9

ORDER BYのCASEをして、2のでレコードをソートします。

2つののいずれかのでソート·レコードがであるとします。のデータベースでは、されていない`MIN()`または`LEAST()`をこの `... ORDER BY MIN(Date1, Date2)` でできますが、SQLでは`CASE`をするがあります。

のクエリの`CASE`は、 `Date1`と`Date2`をべ、どののがさいかをべ、このにじてレコードをソートします。

サンプルデータ

イド	Date1	Date2
1	2017-01-01	2017-01-31
2	2017-01-31	2017-01-03
3	2017-01-31	2017-01-02
4	2017-01-06	2017-01-31
5	2017-01-31	2017-01-05
6	2017-01-04	2017-01-31

クエリ

```
SELECT Id, Date1, Date2
FROM YourTable
ORDER BY CASE
           WHEN COALESCE(Date1, '1753-01-01') < COALESCE(Date2, '1753-01-01') THEN Date1
```

```
          ELSE Date2
        END
```

イド	Date1	Date2
1	**2017-01-01**	2017-01-31
3	2017-01-31	**2017-01-02**
2	2017-01-31	**2017-01-03**
6	**2017-01-04**	2017-01-31
5	2017-01-31	**2017-01-05**
4	**2017-01-06**	2017-01-31

`2017-01-01` `Id = 1`があることがわかります。これは、 `Date1`がテーブル`2017-01-01`からレコードをつため、 `2017-01-01` `Id = 3`は、 `Date2`がテーブルの2にさい`2017-01-02`々。

したがって、 `2017-01-01`から`2017-01-01`までのレコードを`2017-01-01`にべえ、 `Date1`または`Date2`どちらのがこれらのであるかをにする`2017-01-06`ありません。

オンラインでをむ https://riptutorial.com/ja/sql/topic/456/

49: キー

Examples

キーをつテーブルの

このでは、のテーブル`SuperHeros`ます。

このテーブルにはキー`ID`がまれてい`ID`。

スーパーヒーローのをするためにしいテーブルをします

```
CREATE TABLE HeroPowers
(
    ID int NOT NULL PRIMARY KEY,
    Name nvarchar(MAX) NOT NULL,
    HeroId int REFERENCES SuperHeros(ID)
)
```

`HeroId`カラムは、テーブル`SuperHeros` キーです。

キーの

キーは、あるテーブルのがのテーブルのとしなければならないことをすることによって、データのをします。

キーがなのはのとおりです。では、コースはにしているがあります。このシナリオのコードはのとおりです。

```
CREATE TABLE Department (
    Dept_Code        CHAR (5)     PRIMARY KEY,
    Dept_Name        VARCHAR (20) UNIQUE
);
```

のをしてをします。

```
INSERT INTO Department VALUES ('CS205', 'Computer Science');
```

のには、コンピュータサイエンスがするのがまれています。

```
CREATE TABLE Programming_Courses (
    Dept_Code       CHAR(5),
    Prg_Code        CHAR(9) PRIMARY KEY,
    Prg_Name        VARCHAR (50) UNIQUE,
    FOREIGN KEY (Dept_Code) References Department(Dept_Code)
);
```

キーのデータは、されるキーのデータとするがあります。

`Dept_Code`のキーは、されている`Department`にすでにするにのみをします。つまり、のをしようとすると、

```
INSERT INTO Programming_Courses Values ('CS300', 'FDB-DB001', 'Database Systems');
```

`CS300`が`Department`にしないため、データベースはキーエラーをさせます。しかし、するキーをしてみると

```
INSERT INTO Programming_Courses VALUES ('CS205', 'FDB-DB001', 'Database Systems');
INSERT INTO Programming_Courses VALUES ('CS205', 'DB2-DB002', 'Database Systems II');
```

データベースはこれらのをします。

キーをするためのヒント

- キーは、テーブルのUNIQUEまたはPRIMARYキーをするがあります。
- キーにNULLをしても、エラーはしません。
- キーは、じデータベースのテーブルをできます。
- キーは、じテーブルののをすることができます。

オンラインでキーをむ https://riptutorial.com/ja/sql/topic/1533/キー

50:

Examples

テーブル

イド	ファーストネーム	
1	Ozgur	Ozturk
2	ユセフ	メディ
3	ヘンリー	タイ

イド	ID	
1	2	123.50
2	3	14.80

なくとも1つのですべてのをする

```
SELECT * FROM Customer WHERE EXISTS (
    SELECT * FROM Order WHERE Order.CustomerId=Customer.Id
)
```

イド	ファーストネーム	
2	ユセフ	メディ
3	ヘンリー	タイ

なしですべてのをる

```
SELECT * FROM Customer WHERE NOT EXISTS (
    SELECT * FROM Order WHERE Order.CustomerId = Customer.Id
)
```

イド	ファーストネーム	
1	Ozgur	Ozturk

`EXISTS`、 `IN`、 `JOIN`はじにされることがありますが、しくはありません。

- `EXISTS`をして、のテ―ブルにがするかどうかをするがあります
- リストに`IN`をするがあります
- `JOIN`をして、のテ―ブルからデ―タをするがあります。

オンラインでをむ https://riptutorial.com/ja/sql/topic/7933/

51: ブロック

Examples

BEGINをして... END

```
BEGIN
  UPDATE Employees SET PhoneNumber = '5551234567' WHERE Id = 1;
  UPDATE Employees SET Salary = 650 WHERE Id = 3;
END
```

オンラインでブロックをむ https://riptutorial.com/ja/sql/topic/1632/ブロック

52:

Examples

SQLでのクエリの

```
/*(8)*/  SELECT /*9*/ DISTINCT /*11*/ TOP
/*(1)*/  FROM
/*(3)*/       JOIN
/*(2)*/      ON
/*(4)*/  WHERE
/*(5)*/  GROUP BY
/*(6)*/  WITH {CUBE | ROLLUP}
/*(7)*/  HAVING
/*(10)*/ ORDER BY
/*(11)*/ LIMIT
```

クエリがされるとセクションの。

VTは「テーブル」ので、クエリのにさまざまなデータがどのようにされるかをします

1. FROMFROMのの2つのテーブルのにデカルトクロスがされ、としてテーブルVT1がされます。
2. ONオンフィルタがVT1にされます。 TRUEののみがVT2にされます。
3. OUTEROUTER JOINがされているCROSS JOINまたはINNER JOINとはに、していないテーブルのがVT2のにとしてされ、 VT3。 3つのテーブルがFROMにまれる、すべてのテーブルがされるまで、ののとFROMののテーブルのに、ステップ13がりしされます。
4. WHEREWHEREフィルタはVT3にされます。 TRUEののみがVT4にされます。
5. GROUP BYVT4からのは、GROUP BYでされたリストにづいてグループされます。 VT5がされる。
6. キューブ| ROLLUPスーパーグループグループのグループがVT5からにされ、VT6がされます。
7. HAVINGHAVINGフィルタがVT6にされます。 TRUEのグループのみがVT7にされます。
8. SELECTSELECTリストがされ、VT8がされます。
9. DISTINCTしたがVT8からされます。 VT9がされます。
10. ORDER BYVT9からのは、ORDER BYでされたリストにってソートされます。カーソルがされますVC10。
11. TOPVC10のからをします。テーブルVT11がされ、びしにされます。 LIMITは、Postgresや

NetezzaなどののSQLでは、TOPとじをちます。

オンラインでをむ https://riptutorial.com/ja/sql/topic/3671/

53: スキーマ

Examples

スキーマ

なRDBMSのエンドユーザにとってもなクエリの1つは、スキーマのです。

このようなクエリによって、ユーザは、どのテーブルにキーまたはのなカラムをターゲットテーブルとにむかについてののなしに、2つのテーブルから3のテーブルににデータをけるときなど、のあるカラムをむデータベーステーブルをにつけることができる。

このでT-SQLをすると、データベースのスキーマをのようにできます。

```
SELECT *
FROM INFORMATION_SCHEMA.COLUMNS
WHERE COLUMN_NAME LIKE '%Institution%'
```

には、するのリスト、テーブルの、およびそののながまれています。

オンラインでスキーマをむ https://riptutorial.com/ja/sql/topic/3151/スキーマ

54:

き

はにしてをし、またはをします。

をすると、データをしたり、をしたり、をしたり、をすべてまたはにすることができます。

- CONCATstring_value1、string_value2 [、string_valueN]
- LTRIMcharacter_expression
- RTRIMcharacter_expression
- SUBSTRING、、さ
- ASCIIcharacter_expression
- REPLICATEstring_expression、integer_expression
- リバースstring_expression
- UPPERcharacter_expression
- TRIM[FROM]
- STRING_SPLIT、り
- STUFFcharacter_expression、start、length、replaceWith_expression
- REPLACEstring_expression、string_pattern、string_replacement

Transact-SQL / Microsoftのリファレンス

MySQLのリファレンス

PostgreSQLのリファレンス

Examples

をえる

トリムは、のめまたはわりにきみをするためにされます

MSSQLでは、の`TRIM()`

```
SELECT LTRIM('  Hello  ') --returns 'Hello  '
SELECT RTRIM('  Hello  ') --returns '  Hello'
SELECT LTRIM(RTRIM('  Hello  ')) --returns 'Hello'
```

MySqlとOracle

```
SELECT TRIM('  Hello  ') --returns 'Hello'
```

する

ANSI / ISOSQLでは、のは`||`。このは、SQL Serverのすべてのデータベースでサポートされています。

```
SELECT 'Hello' || 'World' || '!'; --returns HelloWorld!
```

くのデータベースでは、をするための`CONCAT`がサポートされて`CONCAT`ます。

```
SELECT CONCAT('Hello', 'World'); --returns 'HelloWorld'
```

`CONCAT`をして2つのをすることをサポートするデータベースもありますOracleではサポートしていません。

```
SELECT CONCAT('Hello', 'World', '!'); --returns 'HelloWorld!'
```

のデータベースでは、をキャストまたはするがあります。

```
SELECT CONCAT('Foo', CAST(42 AS VARCHAR(5)), 'Bar'); --returns 'Foo42Bar'
```

のをするデータベースOracleなどもあります。えば、`CONCAT`の`CLOB`および`NCLOBNCLOB`。と`varchar2` `CONCAT`は`varchar2`などになります。

```
SELECT CONCAT(CONCAT('Foo', 42), 'Bar') FROM dual; --returns Foo42Bar
```

のデータベースでは、の`+`をできますただし、`+`はでのみします。

```
SELECT 'Foo' + CAST(42 AS VARCHAR(5)) + 'Bar';
```

`CONCAT`がサポートされていないSQL Server 2012では、`+`がをするのです。

と

```
SELECT UPPER('HelloWorld') --returns 'HELLOWORLD'
SELECT LOWER('HelloWorld') --returns 'helloworld'
```

はのとおりです`SUBSTRING ( string_expression, start, length )`。SQLは1インデックスであることにしてください。

```
SELECT SUBSTRING('Hello', 1, 2) --returns 'He'
SELECT SUBSTRING('Hello', 3, 3) --returns 'llo'
```

これは、`LEN()`とみわせてされ、なさののの`n`をすることがよくあります。

```
DECLARE @str1 VARCHAR(10) = 'Hello', @str2 VARCHAR(10) = 'FooBarBaz';
SELECT SUBSTRING(@str1, LEN(@str1) - 2, 3) --returns 'llo'
SELECT SUBSTRING(@str2, LEN(@str2) - 2, 3) --returns 'Baz'
```

スプリット

りをしてをします。 `STRING_SPLIT()`はテーブルです。

```
SELECT value FROM STRING_SPLIT('Lorem ipsum dolor sit amet.', ' ');
```

```
value
-----
Lorem
ipsum
dolor
sit
amet.
```

もの

をのにれ、ので0のをきえます。

`start`は1でインデックスけされていますインデックスは0ではなく1でされます。

```
STUFF ( character_expression , start , length , replaceWith_expression )
```

```
SELECT STUFF('FooBarBaz', 4, 3, 'Hello') --returns 'FooHelloBaz'
```

さ

*SQL*サーバー

LENはのスペースをカウントしません。

```
SELECT LEN('Hello') -- returns 5

SELECT LEN('Hello '); -- returns 5
```

DATALENGTHはのスペースをカウントします。

```
SELECT DATALENGTH('Hello') -- returns 5

SELECT DATALENGTH('Hello '); -- returns 6
```

ただし、DATALENGTHは、をするためにされるcharsetにする、のとなるバイトのさをすことにしてください。

```
DECLARE @str varchar(100) = 'Hello ' --varchar is usually an ASCII string, occupying 1 byte
per char
SELECT DATALENGTH(@str) -- returns 6
```

```
DECLARE @nstr nvarchar(100) = 'Hello ' --nvarchar is a unicode string, occupying 2 bytes per
char
SELECT DATALENGTH(@nstr) -- returns 12
```

オラクル

Lengthchar

```
SELECT Length('Bible') FROM dual; --Returns 5
SELECT Length('righteousness') FROM dual; --Returns 13
SELECT Length(NULL) FROM dual; --Returns NULL
```

LengthB、LengthC、Length2、Length4

REPLACE(する,をおよびするため,ののにするを)

```
SELECT REPLACE( 'Peter Steve Tom', 'Steve', 'Billy' ) --Return Values: Peter Billy Tom
```

はのとおりです。
LEFT、
RIGHT、

```
SELECT LEFT('Hello',2)  --return He
SELECT RIGHT('Hello',2) --return lo
```

Oracle SQLには、LEFTおよびRIGHTはありません。 SUBSTRとLENGTHでエミュレートできます。
SUBSTR、1、
SUBSTR、さ-integer + 1、

```
SELECT SUBSTR('Hello',1,2)  --return He
SELECT SUBSTR('Hello',LENGTH('Hello')-2+1,2) --return lo
```

はのとおりです。REVERSE

```
SELECT REVERSE('Hello') --returns olleH
```

レプリケート

REPLICATEは、されただけをします。

はのとおりです。REPLICATEstring-expression、integer

```
SELECT REPLICATE ('Hello',4) --returns 'HelloHelloHelloHello'
```

REGEXP

がのでとするかどうかをチェックします。

```
SELECT 'bedded' REGEXP '[a-f]' -- returns True

SELECT 'beam' REGEXP '[a-f]' -- returns False
```

sqlのとクエリのと

SQLのReplaceは、のをするためにされます。 MySQL、Oracle、およびSQL Serverの、びしはREPLACEです。

Replaceのはのとおりです。

```
REPLACE (str, find, repl)
```

のでは、 `South`をEmployeesテーブルの`Southern`にきえます。

ファーストネーム	
ジェームス	ニューヨーク
ジョン	サウスボストン
マイケル	サウスサンディエゴ

Selectステートメント

のReplaceをすると

```
SELECT
    FirstName,
    REPLACE (Address, 'South', 'Southern') Address
FROM Employees
ORDER BY FirstName
```

ファーストネーム	
ジェームス	ニューヨーク
ジョン	ボストン
マイケル	サンディエゴ

をして、のアプローチでテーブルのなをうことができます。

```
Update Employees
```

```
Set city = (Address, 'South', 'Southern');
```

よりなは、WHEREとみわせてこれをすることです。

```
Update Employees
Set Address = (Address, 'South', 'Southern')
Where Address LIKE 'South%';
```

パーセナム

DATABASESQL Serverの

PARSENAMEは、されたののオブジェクトをします。オブジェクトには、オブジェクト、、データベース、サーバーなどのをめることができます。

MSDN PARSENAME

```
PARSENAME('NameOfStringToParse',PartIndex)
```

オブジェクトをするには、パート・インデックス1

```
SELECT PARSENAME('ServerName.DatabaseName.SchemaName.ObjectName',1)  // returns `ObjectName`
SELECT PARSENAME('[1012-1111].SchoolDatabase.school.Student',1)     // returns `Student`
```

スキーマをするには、パート・インデックス2します。

```
SELECT PARSENAME('ServerName.DatabaseName.SchemaName.ObjectName',2)  // returns `SchemaName`
SELECT PARSENAME('[1012-1111].SchoolDatabase.school.Student',2)     // returns `school`
```

データベースをするには、パート・インデックス3します。

```
SELECT PARSENAME('ServerName.DatabaseName.SchemaName.ObjectName',3) // returns `DatabaseName`
SELECT PARSENAME('[1012-1111].SchoolDatabase.school.Student',3)    // returns `SchoolDatabase`
```

サーバーをするには、パート・インデックス4します。

```
SELECT PARSENAME('ServerName.DatabaseName.SchemaName.ObjectName',4)  // returns `ServerName`
SELECT PARSENAME('[1012-1111].SchoolDatabase.school.Student',4)     // returns `[1012-1111]`
```

PARSENAMEはりをすnullはされたオブジェクトのにしない

INSTR

ののののインデックスをしますつからないはゼロ。

INSTR、

```
SELECT INSTR('FooBarBar', 'Bar') -- return 4
SELECT INSTR('FooBarBar', 'Xar') -- return 0
```

オンラインでをむ https://riptutorial.com/ja/sql/topic/1120/

55:

- UPDATE テーブル
  SET *column_name = value* 、 *column_name2 = value_2* 、...、 *column_name_n = value_n*
  WHERE　condition_n

Examples

すべてのの

このでは、サンプルデータベースのCars Tableをしています。

```
UPDATE Cars
SET Status = 'READY'
```

このステートメントは、 'Cars'のすべてのの 'status'に、セットをフィルタリングするWHEREがないため、「READY」にされます。

されたの

このでは、サンプルデータベースのCars Tableをしています。

```
UPDATE
    Cars
SET
    Status = 'READY'
WHERE
    Id = 4
```

このステートメントは、ID 4の「Cars」ののステータスを「READY」にします。

WHEREには、にしてされるがまれます。がをたす、そのがされます。それのは、はされません。

ののの

このでは、サンプルデータベースのCars Tableをしています。

```
UPDATE Cars
SET TotalCost = TotalCost + 100
WHERE Id = 3 or Id = 4
```

には、されたののをめることができます。このなでは、 TotalCostは2で100ずつえます。

- 3のトータルコストが100から200にしました
- 4のトータルコストは1254から1354にしました

のしいは、のから、またはじまたはされたののののからすることがあります。

のテーブルからのデータでUPDATEする

のは、 `Customer`でもあり、 `Employees`にがされていないの`PhoneNumber`をしています。

これらのでは、サンプルデータベースのEmployeesテーブルとCustomersテーブルをしています。

SQL

サブクエリをした

```
UPDATE
    Employees
SET PhoneNumber =
    (SELECT
         c.PhoneNumber
     FROM
         Customers c
     WHERE
         c.FName = Employees.FName
         AND c.LName = Employees.LName)
WHERE Employees.PhoneNumber IS NULL
```

SQL2003

`MERGE`をした

```
MERGE INTO
    Employees e
USING
    Customers c
ON
    e.FName = c.Fname
    AND e.LName = c.LName
    AND e.PhoneNumber IS NULL
WHEN MATCHED THEN
   UPDATE
      SET PhoneNumber = c.PhoneNumber
```

SQLサーバー

`INNER JOIN`をした

```
UPDATE
    Employees
```

```
SET
    PhoneNumber = c.PhoneNumber
FROM
    Employees e
INNER JOIN Customers c
        ON e.FName = c.FName
        AND e.LName = c.LName
WHERE
    PhoneNumber IS NULL
```

されたレコードのキャプチャ

によっては、されたばかりのレコードをキャプチャしたいもあります。

```
CREATE TABLE #TempUpdated(ID INT)

Update TableName SET Col1 = 42
    OUTPUT inserted.ID INTO #TempUpdated
    WHERE Id > 50
```

オンラインでをむ https://riptutorial.com/ja/sql/topic/321/

56:

- ROW_NUMBER
- OVER[PARTITION BY value_expression、... [n]] order_by_clause

Examples

パーティションのない

されたにってをめます。

```
SELECT
  ROW_NUMBER() OVER(ORDER BY Fname ASC) AS RowNumber,
  Fname,
  LName
FROM Employees
```

パーティションをつ

パーティションをしてをグループします。

```
SELECT
  ROW_NUMBER() OVER(PARTITION BY DepartmentId ORDER BY DepartmentId ASC) AS RowNumber,
  DepartmentId, Fname, LName
FROM Employees
```

のレコードをくすべてのレコードをする1のテーブル

```
WITH cte AS (
  SELECT ProjectID,
         ROW_NUMBER() OVER (PARTITION BY ProjectID ORDER BY InsertDate DESC) AS rn
  FROM ProjectNotes
)
DELETE FROM cte WHERE rn > 1;
```

オンラインでをむ https://riptutorial.com/ja/sql/topic/1977/

57: があるサブセットでをつける

- がないをするには、WHEREを "RowCnt = 1"にします。
- セットから1をするには、SumのわりにRankをし、のWHEREをしてRank= 1のをします

Examples

じとの

```
WITH CTE (StudentId, Fname, LName, DOB, RowCnt)
as (
SELECT StudentId, FirstName, LastName, DateOfBirth as DOB, SUM(1) OVER (Partition By
FirstName, LastName, DateOfBirth) as RowCnt
FROM tblStudent
)
SELECT * from CTE where RowCnt > 1
ORDER BY DOB, LName
```

このでは、テーブルとウィンドウをして、するすべてのをのサブセットにべてします。

オンラインでがあるサブセットでをつけるをむ https://riptutorial.com/ja/sql/topic/1585/があるサブセットでをつける

58: と

Examples

DESCRIBE tablename;

DESCRIBEとEXPLAINはです。 tablenameのDESCRIBEはのをします。

```
DESCRIBE tablename;
```

```
COLUMN_NAME     COLUMN_TYPE     IS_NULLABLE     COLUMN_KEY     COLUMN_DEFAULT     EXTRA
id              int(11)         NO              PRI            0
auto_increment
test            varchar(255)    YES                            (null)
```

ここではのとそれにくのがされます。にnullがされているかどうか、またそのがインデックスをしているかどうかがされます。デフォルトもされ、にauto_incrementようなながまれているにもされます。

EXPLAIN クエリの

Explainのインフロントselectクエリは、クエリがされるをします。これにより、クエリでインデックスがされているかどうか、またはインデックスをしてクエリをできるかどうかをできます。

クエリ

```
explain select * from user join data on user.test = data.fk_user;
```

の

```
id  select_type  table   type    possible_keys  key     key_len ref       rows  Extra
1   SIMPLE       user    index   test           test    5       (null)    1     Using where;
Using index
1   SIMPLE       data    ref     fk_user        fk_user 5       user.test 1     (null)
```

インデックスがされたにされるtypeです。 possible_keysには、がしないになるインデックスからできるかどうかがされます。 keyは、されるacutalインデックスをします。 key_lenは1つのインデックスのサイズをバイトでします。このがさいほど、インデックスがじメモリサイズにまるほどがくなります。 rowsは、クエリがスキャンするがあるのがされます。

オンラインでとをむ https://riptutorial.com/ja/sql/topic/2928/と

59:

き

このトピックでは、、つまりテーブル、カラム、およびそののデータベースオブジェクトののについてします。

にじて、さまざまなSQLでされるバリエーションをカバーするか、こののSQLをするがあります。

Examples

でまれていない

でまれていないは、することができ-`a z`、 `0 - 9`、および `_`、およびでしなければなりません。

SQLのやデータベースのによっては、のをすることもできます。たとえば、のとしてもできます。

- MS SQL `@`、 `$`、 `#`、そののUnicode ソース
- MySQL `$` ソース
- Oracleデータベースキャラクタセットからの`$`、 `#`、およびその ソース
- PostgreSQL `$`、そののUnicode ソース

でまれていないはとをしません。これがどのようにされるかは、SQLにきくします。

- MS SQLとをし、データベースのセットによってされるとのがあるため、とがされます。
- MySQLとのは、データベースのとファイルシステムのにします。
- Oracleにし、きののようにいます。
- PostgreSQLにし、きののようにいます。
- SQLiteをする。のがないのはASCIIのみです。

オンラインでをむ https://riptutorial.com/ja/sql/topic/9677/

60:

Examples

リレーショナルはなSQLではなく、リレーショナルのをるです。したがって、テーブル、レコード、フィールドなどのなエンティティへのはわないでください。リレーション、タプル、アトリビュートなどのをするがあります。それをって、はこののをわず、よりくられているテーブル、レコード、フィールドというにいます。

するの2つのルールがまるに

- リレーショナルでされるは、々のレコードではなくテーブルをします。
- リレーショナルのはにテーブルになりますこれはクロージャプロパティとばれます

このでは、の2つのをします。

Departments

ID	Dept
1	Production
2	Quality Control

People

ID	PersonName	StartYear	ManagerID	DepartmentID
1	Darren	2005		1
2	David	2006	1	1
3	Burt	2006	1	1
4	Sarah	2004		2
5	Fred	2008	4	2
6	Joanne	2005	4	2

セレクト

selectはメインテーブルのサブセットをします。
*<table>*をします。 *<condition>*

たとえば、のをべます。

々をする のDepartmentID = 2

これはのようにくことができます

$$\sigma_{\text{DepartmentID} = 2}(\text{People})$$

これにより、 *DepartmentID*が2の*People*テーブルのすべてのレコードからなるレコードがまれるテーブルがされます。

ID	PersonName	StartYear	ManagerID	DepartmentID
4	Sarah	2004		2
5	Fred	2008	4	2
6	Joanne	2005	4	2

をして、をさらにすることもできます。

々をする StartYear> 2005とのDepartmentID = 2

のがされます。

ID	PersonName	StartYear	ManagerID	DepartmentID
5	Fred	2008	4	2

プロジェクト

プロジェクトは、テーブルからのフィールドをします。
プロジェクト *<table>* ***over*** *<field list>*

たとえば、のをべます。
StartYearをえる々 を プロジェクト する

これはのようにくことができます

$$\pi_{\text{StartYear}}(\text{People})$$

これにより、 *People*テーブルの*StartYear*フィールドにされているのでされるがされます。

StartYear
2005
2006
2004
2008

リレーショナルをするクロージャーのため、のからしたがされます。リレーショナルのすべてのレコードはするがあります。

フィールドリストにのフィールドがまれている、のはこれらのフィールドのなるバージョンです。
プロジェクトをぎた、DepartmentIDはします

StartYear	DepartmentID
2005	1
2006	1
2004	2
2008	2
2005	2

2006の*StartYear*と1つの*DepartmentID*がして1つのレコードがされます。

る

は、々のに**giving**キーワードをしてするか、ののにあるをめむことによって、させることができます。

<> える <エイリアス>

たとえば、のをえます。
Aをえるの DepartmentID = 2 をし々
B をえる PersonName にする プロジェクト A

これにより、Aがののであるの B がられます。

A

ID	PersonName	StartYear	ManagerID	DepartmentID
4	Sarah	2004		2
5	Fred	2008	4	2
6	Joanne	2005	4	2

B

PersonName
Sarah
Fred
Joanne

のがされ、のにAがえられます。このは2ののでされ、にBがけられます。

このをきむもう1つのは、2ののテーブルエイリアスを、ののテキストをでんできえることです。
Bをえる PersonNameのオーバー プロジェクト どこのDepartmentID = 2を

これはれとばれます 。

NATURAL JOIN

ナルジョインは、テーブルでされるフィールドをして2つのテーブルをします。

<table 1> と *<table 2>*をします。 *<field 1>* = *<field 2>*
<field 1>が<table 1>にあり<field 2>が<table 2>にあるとします。

たとえば、のは、それぞれののDepartmentIDとIDについてPeopleおよびDepartmentsをします。
DepartmentID = IDのとに する

ID	PersonName	StartYear	ManagerID	DepartmentID	Dept
1	Darren	2005		1	Production
2	David	2006	1	1	Production
3	Burt	2006	1	1	Production
4	Sarah	2004		2	Quality Control
5	Fred	2008	4	2	Quality Control
6	Joanne	2005	4	2	Quality Control

*People*テーブルの*DepartmentID*のみがされ、 *Department*テーブルの*ID*はされません。されるフィールドの1つのみがされるがあります。これは、、ののテーブルのフィールドです。

このにはされていませんが、テーブルをすると、じしをつ2つのフィールドがされるがあります。たとえば、 *Name*というしをして*PersonName*フィールドと*Dept*フィールドをしたつまり、Person NameとDepartment Nameをするこのがすると、をして、ドットをしてフィールドをします。People.Nameおよび*Departments.Name*

とみわせし、 プロジェクトがをきすためににすることができます。

DepartmentID = ID を するとに する
StartYear = 2005 および *DEPT*は= ''は*B*をえて *A*を
C をえる *PersonName* にする プロジェクト *B*

またはみわされたとして

プロジェクト *PersonName*のオーバー *C*をえる *StartYear = 2005* および *DEPT*は= ''はどこ の *DepartmentID = ID*々 やに

これはこのになります

PersonName
Darren

エイリアス

UPDATE=

オンラインでをむ https://riptutorial.com/ja/sql/topic/7311/

61: アナリティック

き

をして、のグループにづいてをします。たとえば、このタイプのをして、の、パーセンテージ、またはグループののをできます。

1. FIRST_VALUEscalar_expressionOVER[partition_by_clause] order_by_clause []
2. LAST_VALUEscalar_expressionOVER[partition_by_clause] order_by_clause []
3. LAGscalar_expression [、offset] [、default]OVER[partition_by_clause] order_by_clause
4. LEADscalar_expression [、offset]、[デフォルト]OVER[partition_by_clause] order_by_clause
5. PERCENT_RANKOVER[partition_by_clause] order_by_clause
6. CUME_DISTOVER[partition_by_clause] order_by_clause
7. PERCENTILE_DISCnumeric_literalWITHIN GROUPORDER BY order_by_expression [ASC | DESC]オーバー[<partition_by_clause>]
8. PERCENTILE_CONTnumeric_literalWITHIN GROUPORDER BY order_by_expression [ASC | DESC]OVER[<partition_by_clause>]

Examples

FIRST_VALUE

`FIRST_VALUE`をして、スカラーをしてするきセットののをします。

```
SELECT StateProvinceID, Name, TaxRate,
       FIRST_VALUE(StateProvinceID)
        OVER(ORDER BY TaxRate ASC) AS FirstValue
FROM SalesTaxRate;
```

このでは、 `FIRST_VALUE`をして、のまたはの`ID`をし`ID`。 `OVER`はをるためにをするためにされます。

StateProvinceID			FirstValue
74	ユタの	5.00	74
36	ミネソタ	6.75	74
30	マサチューセッツ	7.00	74
1	カナダのGST	7.00	74
57	カナダのGST	7.00	74
63	カナダのGST	7.00	74

LAST_VALUE

LAST_VALUEファンクションは、スカラーをしてしたきセットののをします。

```
SELECT TerritoryID, StartDate, BusinessentityID,
       LAST_VALUE(BusinessentityID)
        OVER(ORDER BY TerritoryID) AS LastValue
FROM SalesTerritoryHistory;
```

このでは、 LAST_VALUEをして、 LAST_VALUEれたのセットののをします。

テリトリーID		ビジネスID	LastValue
1	2005-07-01 00.00.00.000	280	283
1	2006-11-01 00.00.00.000	284	283
1	2005-07-01 00.00.00.000	283	283
2	2007-01-01 00.00.00.000	277	275
2	2005-07-01 00.00.00.000	275	275
3	2007-01-01 00.00.00.000	275	277

LAGとLEAD

LAGは、じセットののののにあるのデータをします。たとえば、 SELECTでは、ののとののをできます。

スカラーをして、するをします。 offsetパラメータは、にされるのののののです。をしないは、1のデフォルトがされます。

デフォルトのパラメータは、offsetのがNULLをつにされるをします。をしないは、 NULLがされます。

LEADは、セットののののにあるのデータをします。たとえば、 SELECTでは、ののとののをできます。

スカラーをしてするをします。 offsetパラメータは、にされるのののののです。

デフォルトのパラメータをして、offsetのにNULLがあるにNULLをします。これらのパラメーターをしないと、1のデフォルトがされ、 NULLがされます。

```
SELECT BusinessEntityID, SalesYTD,
       LEAD(SalesYTD, 1, 0) OVER(ORDER BY BusinessEntityID) AS "Lead value",
       LAG(SalesYTD, 1, 0) OVER(ORDER BY BusinessEntityID) AS "Lag value"
```

```
FROM SalesPerson;
```

このでは、LEADとLAGをして、ののととののとし、BusinessEntityIDについてレコードをべえます。

BusinessEntityID	SalesYTD	リード	ラグ
274	559697.5639	3763178.1787	0.0000
275	3763178.1787	4251368.5497	559697.5639
276	4251368.5497	3189418.3662	3763178.1787
277	3189418.3662	1453719.4653	4251368.5497
278	1453719.4653	2315185.6110	3189418.3662
279	2315185.6110	1352577.1325	1453719.4653

PERCENT_RANKとCUME_DIST

PERCENT_RANKファンクションは、セットにするのランキングをします。は、のよりもいをつグループののについています。

セットののはにパーセントランクをちます。セットのまたはののはに1です。

CUME_DISTは、のグループのされたのを、そのののをすることによってします。これはとばれます。

```
SELECT BusinessEntityID, JobTitle, SickLeaveHours,
PERCENT_RANK() OVER(PARTITION BY JobTitle ORDER BY SickLeaveHours DESC)
       AS "Percent Rank",
CUME_DIST() OVER(PARTITION BY JobTitle ORDER BY SickLeaveHours DESC)
       AS "Cumulative Distribution"
FROM Employee;
```

このでは、 ORDERをして、のについてSELECTでりされたをパーティションまたはグループし、グループのをがしたのについてソートします。

BusinessEntityID		SickLeaveHours	パーセントランク	
267	アプリケーションスペシャ	57	0	0.25

BusinessEntityID		SickLeaveHours	パーセントランク	
	リスト			
268	アプリケーションスペシャリスト	56	0.333333333333333	0.75
269	アプリケーションスペシャリスト	56	0.333333333333333	0.75
272	アプリケーションスペシャリスト	55	1	1
262	チーフ・ファイナンシャル・オフィサーのアシスタント	48	0	1

BusinessEntityID		SickLeaveHours	パーセントランク	
239	スペシャリスト	45	0	1
252	い	50	0	0.111111111111111
251	い	49	0.125	0.333333333333333
256	い	49	0.125	0.333333333333333
253	い	48	0.375	0.555555555555555
254	い	48	0.375	0.555555555555555

PERCENT_RANKは、グループのエントリをランクけします。エントリについて、じグループののいエントリのをします。

CUME_DISTは、ののののをすをいて、です。

PERCENTILE_DISCとPERCENTILE_CONT

PERCENTILE_DISCは、がnumeric_literalパラメーターをしてnumeric_literalたよりもいののをリストします。

は、 WITHIN GROUPでされたセットまたはパーティションによってグループされます。

PERCENTILE_CONTはであるPERCENTILE_DISCが、のするエントリとのエントリののをします。

```
SELECT BusinessEntityID, JobTitle, SickLeaveHours,
       CUME_DIST() OVER(PARTITION BY JobTitle ORDER BY SickLeaveHours ASC)
       AS "Cumulative Distribution",
       PERCENTILE_DISC(0.5) WITHIN GROUP(ORDER BY SickLeaveHours)
          OVER(PARTITION BY JobTitle) AS "Percentile Discreet"
FROM Employee;
```

0.5パーセンタイルとする、またはそれをえるからのなをつけるには、 PERCENTILE_DISCのリテラルとしてします。セットのPercentile Discreetには、がされたよりもいのがリストされます。

BusinessEntityID		SickLeaveHours		Percentile Discreet
272	アプリケーションスペシャリスト	55	0.25	**56**
268	アプリケーションスペシ	56	0.75	**56**

BusinessEntityID		SickLeaveHours		Percentile Discreet
	ャリスト			
269	アプリケーションスペシャリスト	56	0.75	**56**
267	アプリケーションスペシャリスト	57	1	**56**

をのについてうには、 `PERCENTILE_CONT`をします。の「Percentile Continuous」には、とにするののがされます。

```
SELECT BusinessEntityID, JobTitle, SickLeaveHours,
       CUME_DIST() OVER(PARTITION BY JobTitle ORDER BY SickLeaveHours ASC)
       AS "Cumulative Distribution",
       PERCENTILE_DISC(0.5) WITHIN GROUP(ORDER BY SickLeaveHours)
          OVER(PARTITION BY JobTitle) AS "Percentile Discreet",
       PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY SickLeaveHours)
          OVER(PARTITION BY JobTitle) AS "Percentile Continuous"
FROM Employee;
```

BusinessEntityID		SickLeaveHours		Percentile Discreet	パーセンティル
272	アプリケーションスペシャリスト	55	0.25	56	**56**
268	アプリケーションスペシャリスト	56	0.75	56	**56**
269	アプリケーションスペシャリスト	56	0.75	56	**56**
267	アプリケーションスペシャリスト	57	1	56	**56**

オンラインでアナリティックをむ https://riptutorial.com/ja/sql/topic/8811/-アナリティック-

62: スカラー/

き

SQLには、いくつかのみみスカラがされています。スカラーは、1つのをとしてり、セットののとして1つのをします。

スカラーは、T-SQLステートメントでがされているすべてのでします。

- CASTAS data_type [length]
- CONVERTデータ[さ]、[、スタイル]
- PARSEstring_value ASデータ[USINGカルチャ]
- DATENAMEdatepart、date
- GETDATE
- DATEDIFFdatepart、startdate、enddate
- DATEADDdatepart、number、date
- インデックス、val_1、val_2 [、val_n]
- IIFboolean_expression、true_value、false_value
- SIGN
- POWERfloat_expression、y

セットでするではなく、セットののデータをするために、スカラまたはがされます。

スカラーには10あります。

1. は、のSQLインスタンスのにするをします。
2. は、されたのデータをしいデータにします。たとえば、これらのタイプのは、をまたはにして2つのなるをできるようにをフォーマットすることができます。
3. とのは、とのをむフィールドをします。、、またはのをすことができます。たとえば、をして、のまたはをしたり、からそののみをしたりすることができます。

とのによってされるは、SQLインスタンスをしているコンピュータのオペレーティングシステムにされたとによってなります。

4. をしてをする。それはのをし、のをします。
5. は、にしてやをいます。このタイプのは、のをします。
6. メタデータは、されたデータベースやデータベースオブジェクトなどにするをします。
7. セキュリティは、データベースユーザーおよびロールにするなど、データベースのセキュリティをするためにできるをします。
8. はにしてをし、またはをします。

をすると、データをしたり、をしたり、をしたり、をすべてまたはにすることができます。

9. システムはをし、のSQLインスタンスの、オブジェクト、およびにするをします

10. システムは、のSQLインスタンスにするさまざまなをします。たとえば、システムののパフォーマンスレベルをできます。

Examples

の

をするには、をまたはにしたり、をフォーマットにしたり、をうなどがあります。

`lower(char)`は、されたパラメータをにします。

```
SELECT customer_id, lower(customer_last_name) FROM customer;
```

おのが「SMITH」から「smith」にされたものをします。

SQLでは、とのデータをしてカレンダーをします。これらのデータには、、、smalldatetime、datetime、datetime2、およびdatetimeoffsetがまれます。データにはのがあります。

データ.タイプ	フォーマット
	hhmmss [.nnnnnnn]
	YYYY-MM-DD
smalldatetime	YYYY-MM-DDhmmss
	YYYY-MM-DD hhmmss [.nnn]
2	YYYY-MM-DD hhmmss [.nnnnnnn]
datetimeオフセット	YYYY-MM-DD hhmmss [.nnnnnnn] [+/-] hhmm

`DATENAME`は、ののののまたはをします。

```
SELECT DATENAME (weekday,'2017-01-14') as Datename
```

データ

`GETDATE`をして、のSQLインスタンスをしているコンピュータののとをします。このには、タイムゾーンのいはまれません。

```
SELECT GETDATE() as Systemdate
```

システム
2017-01-14 111147.7230728

DATEDIFFは、2つののをします。

では、datepartは、のにするのをするパラメータです。 datepartは、、、、、、、、またはミリです。 startdateパラメーターでをし、enddateパラメーターでをします。

```
SELECT SalesOrderID, DATEDIFF(day, OrderDate, ShipDate)
AS 'Processing time'
FROM Sales.SalesOrderHeader
```

SalesOrderID	
43659	7
43660	7
43661	7
43662	7

DATEADDをすると、ののにをできます。

```
SELECT DATEADD (day, 20, '2017-01-14') AS Added20MoreDays
```

20MoreDaysをしました
2017-02-03 000000.000

および

SQLののは、 @@SERVERNAMEです。このは、SQLをしているロカル・サバのをします。

```
SELECT @@SERVERNAME AS 'Server'
```

サバ
SQL064

SQLでは、ほとんどのデタはユザのなしににわれます。

にできないをするには、 CASTまたはCONVERTをします。

CAST

のは、 CONVERTのよりですが、できることにはりがあります。

ここでは、 CASTとCONVERTのをして、datetimeデータをvarcharデータにします。

CASTは、にデフォルトのスタイルをします。たとえば、YYYY-MM-DDのでとをします。

CONVERTは、したとのスタイルをします。この、3はdd / mm / yyをします。

```
USE AdventureWorks2012
GO
SELECT FirstName + ' ' + LastName + ' was hired on ' +
       CAST(HireDate AS varchar(20)) AS 'Cast',
       FirstName + ' ' + LastName + ' was hired on ' +
       CONVERT(varchar, HireDate, 3) AS 'Convert'
FROM Person.Person AS p
JOIN HumanResources.Employee AS e
ON p.BusinessEntityID = e.BusinessEntityID
GO
```

キャスト	
David Hamiltonは200324にわれました	David Hamiltonが04/02/03にわれました

ののは、 PARSEです。このは、をされたデータにします。

のでは、するのある、 ASキーワード、およびなデータをします。にじて、をするカルチャをすることもできます。これをしないと、セッションのがされます。

のを、、またはにできない、エラーになります。にはCASTまたはCONVERTをするがあります。

```
SELECT PARSE('Monday, 13 August 2012' AS datetime2 USING 'en-US') AS 'Date in English'
```

での
2012-08-13 000000.0000000

と

SQLには、2つの CHOOSE と IIF ます。

CHOOSEは、リストのにづいて、のリストからをします。このはインデックスによってされます。

では、indexパラメーターはをし、またはです。 val_1 ... val_nパラメータは、のリストをします。

```
SELECT CHOOSE(2, 'Human Resources', 'Sales', 'Admin', 'Marketing' ) AS Result;
```

このでは、 CHOOSEをして、リストの2のをします。

IIFは、のについて2つののいずれかをします。がの、のをします。さもなければ、それはのをします。

では、boolean_expressionパラメータにブールをします。 true_valueパラメーターは、boolean_expressionがtrueとされたにされるをし、false_valueパラメーターは、boolean_expressionがfalseにされたにされるをします。

```
SELECT BusinessEntityID, SalesYTD,
       IIF(SalesYTD > 200000, 'Bonus', 'No Bonus') AS 'Bonus?'
FROM Sales.SalesPerson
GO
```

BusinessEntityID	SalesYTD	ボーナス
274	559697.5639	ボーナス
275	3763178.1787	ボーナス
285	172524.4512	ボーナスなし

このでは、IIFをして2つののいずれかをします。のが200,000をえる、このはボーナスのとなります。 200,000のは、がのにならないことをします。

SQLには、をしをするためにできるいくつかのがまれています。

1つのは、のをするをすSIGNです。 -1のはのをし、+1のはのをし、0はゼロをします。

```
SELECT SIGN(-20) AS 'Sign'
```

-1

このでは、はのなので、Resultsペインには-1がされます。

もうつのはPOWERです。このは、されたにきげられたのをします。

では、float_expressionパラメーターはをし、yパラメーターはをきげるをします。

```
SELECT POWER(50, 3) AS Result
```

```
125000
```

オンラインでスカラー/をむ https://riptutorial.com/ja/sql/topic/6898/-スカラー--

63:

- [*DISTINCT*]-DISTINCTはオプションのパラメータです
- AVG[ALL | DISTINCT]
- COUNT{[ALL | DISTINCT]] | *}
- GROUPING<column_expression>
- MAX[ALL | DISTINCT]
- MIN[ALL | DISTINCT]
- SUM[すべて| DISTINCT]
- VAR[ALL | DISTINCT]
  OVER[partition_by_clause] order_by_clause
- VARP[ALL | DISTINCT]
  OVER[partition_by_clause] order_by_clause
- STDEV[ALL | DISTINCT]
  OVER[partition_by_clause] order_by_clause
- STDEVP[ALL | DISTINCT]
  OVER[partition_by_clause] order_by_clause

データベースでは、は、ののがのでとしてグループされ、、またはリストなどのよりなまたはののをするです。

```
MIN         returns the smallest value in a given column
MAX         returns the largest value in a given column
SUM         returns the sum of the numeric values in a given column
AVG         returns the average value of a given column
COUNT       returns the total number of values in a given column
COUNT(*)    returns the number of rows in a table
GROUPING    Is a column or an expression that contains a column in a GROUP BY clause.
STDEV       returns the statistical standard deviation of all values in the specified
expression.
STDEVP      returns the statistical standard deviation for the population for all values in the
specified expression.
VAR         returns the statistical variance of all values in the specified expression. may be
followed by the OVER clause.
VARP        returns the statistical variance for the population for all values in the specified
expression.
```

> は、 SELECTステートメントの「されたデータの」をするためにされます。には、したデータののののをします。 - SQLCourse2.com

すべてのはNULLをします。

Examples

Sumは、グループのすべてののをします。 group byをすると、すべてのがされます。

```
select sum(salary) TotalSalary
from employees;
```

TotalSalary
2500

```
select DepartmentId, sum(salary) TotalSalary
from employees
group by DepartmentId;
```

DepartmentId	TotalSalary
1	2000
2	500

き

いテーブル

	いタイプ	
ピーター	クレジット	100
ピーター	クレジット	300
ジョン	クレジット	1000
ジョン	デビット	500

```
select customer,
       sum(case when payment_type = 'credit' then amount else 0 end) as credit,
       sum(case when payment_type = 'debit' then amount else 0 end) as debit
from payments
group by customer
```

	クレジット	デビット
ピーター	400	0
ジョン	1000	500

```
select customer,
       sum(case when payment_type = 'credit' then 1 else 0 end) as credit_transaction_count,
       sum(case when payment_type = 'debit' then 1 else 0 end) as debit_transaction_count
from payments
group by customer
```

	credit_transaction_count	debit_transaction_count
ピーター	2	0
ジョン	1	1

AVG

AVGは、されたのをします。、のです。のにおけるのをむがあるとします。ニュ―ヨ―クのは、のようになります。

テ―ブルの

の		
ニュ―ヨ―ク	8,550,405	2015
ニュ―ヨ―ク	...	...
ニュ―ヨ―ク	8,000,906	2005

10の、、およびをむから、ニュ―ヨ―クのをするには

QUERY

```
select city_name, AVG(population) avg_population
from city_population
where city_name = 'NEW YORK CITY';
```

はのとともにされているので、がクエリからどのようにけているかにしてください。

の	
ニュ―ヨ―ク	8,250,754

AVGはをにします。これは、をうにすることがにです。

リストの

この SOえのな。

List Concatenationは、グル―プのをのにみわせることによって、またはをします。またはのカンマをるで、ののをできます。 SQLのではありませんが、すべてのなリレ―ショナルデ―タベ―スベンダ―は、のでサポ―トしています。

MySQL

```
SELECT ColumnA
     , GROUP_CONCAT(ColumnB ORDER BY ColumnB SEPARATOR ',') AS ColumnBs
  FROM TableName
 GROUP BY ColumnA
 ORDER BY ColumnA;
```

OracleDB2

```
SELECT ColumnA
     , LISTAGG(ColumnB, ',') WITHIN GROUP (ORDER BY ColumnB) AS ColumnBs
  FROM TableName
 GROUP BY ColumnA
 ORDER BY ColumnA;
```

PostgreSQL

```
SELECT ColumnA
     , STRING_AGG(ColumnB, ',' ORDER BY ColumnB) AS ColumnBs
  FROM TableName
 GROUP BY ColumnA
 ORDER BY ColumnA;
```

SQLサーバー

SQL Server 2016およびそれ

DRYをするためにCTEがまれています

```
  WITH CTE_TableName AS (
       SELECT ColumnA, ColumnB
         FROM TableName)
SELECT t0.ColumnA
     , STUFF((
       SELECT ',' + t1.ColumnB
         FROM CTE_TableName t1
        WHERE t1.ColumnA = t0.ColumnA
        ORDER BY t1.ColumnB
          FOR XML PATH('')), 1, 1, '') AS ColumnBs
  FROM CTE_TableName t0
 GROUP BY t0.ColumnA
 ORDER BY ColumnA;
```

SQL Server 2017およびSQL Azure

```
SELECT ColumnA
     , STRING_AGG(ColumnB, ',') WITHIN GROUP (ORDER BY ColumnB) AS ColumnBs
  FROM TableName
 GROUP BY ColumnA
 ORDER BY ColumnA;
```

SQLite

なしで

```
SELECT ColumnA
     , GROUP_CONCAT(ColumnB, ',') AS ColumnBs
  FROM TableName
 GROUP BY ColumnA
 ORDER BY ColumnA;
```

けにはサブクエリまたはCTEがです。

```
  WITH CTE_TableName AS (
       SELECT ColumnA, ColumnB
         FROM TableName
        ORDER BY ColumnA, ColumnB)
SELECT ColumnA
     , GROUP_CONCAT(ColumnB, ',') AS ColumnBs
  FROM CTE_TableName
 GROUP BY ColumnA
 ORDER BY ColumnA;
```

カウント

あなたはのをえることができます

```
SELECT count(*) TotalRows
FROM employees;
```

TotalRows
4

またはごとのをえます

```
SELECT DepartmentId, count(*) NumEmployees
FROM employees
GROUP BY DepartmentId;
```

DepartmentId	NumEmployees
1	3
2	1

`NULL`をえないエフェクトをして、*/*をえることができます。

```
SELECT count(ManagerId) mgr
FROM EMPLOYEES;
```

mgr
3

nullのmanagerIDが1つあります

また、 **COUNT**などののでで**DISTINCT**をすると、そのセットの**DISTINCT**メンバーだけをしてをできます。

えば

```
SELECT COUNT(ContinentCode) AllCount
,      COUNT(DISTINCT ContinentCode) SingleCount
FROM Countries;
```

なるがされます。 *SingleCount*は々のをカウントしますが、 *AllCount*はをみます。

ContinentCode
OC
EU
として
NA
NA
AF
AF

AllCount7 SingleCount5

のをめる

```
select max(age) from employee;
```

のでは、 `employee`テーブルののの`age`にもきながされます。

```
SELECT MAX(column_name) FROM table_name;
```

のをめる

```
 select min(age) from employee;
```

ののののためのをします`age`の`employee`テーブル。

```
 SELECT MIN(column_name) FROM table_name;
```

オンラインでをむ https://riptutorial.com/ja/sql/topic/1002/--

64:

Examples

の

```
CREATE SYNONYM EmployeeData
FOR MyDatabase.dbo.Employees
```

オンラインでをむ https://riptutorial.com/ja/sql/topic/2518/

クレジット

S. No		Contributors
1	SQL	Arjan Einbu, brichins, Burkhard, cale_b, CL., Community, Devmati Wadikar, Epodax, geeksal, H. Pauwelyn, Hari, Joey, JohnLBevan, Jon Ericson, Lankymart, Laurel, Mureinik, Nathan, omini data, PeterRing, Phrancis, Prateek, RamenChef, Ray, Simone Carletti, SZenC, t1gor, ypercube
2	ANDとOR	guiguiblitz
3	CREATE FUNCTION	John Odom, Ricardo Pontual
4	CREATE TABLE	Aidan, alex9311, Almir Vuk, Ares, CL., drunken_monkey, Dylan Vander Berg, Franck Dernoncourt, H. Pauwelyn, Jojodmo, KIRAN KUMAR MATAM, Matas Vaitkevicius, Prateek
5	CREATEデータベース	Emil Rowland
6	DROPテーブル	CL., Joel, KIRAN KUMAR MATAM, Stu
7	DROPまたはDELETEデータベース	Abhilash R Vankayala, John Odom
8	GROUP BY	3N1GM4, Abe Miessler, Bostjan, Devmati Wadikar, Filipe Manuel, Frank, Gidil, Jaydles, juergen d, Nathaniel Ford, Peter Gordon, Simone - Ali One, WesleyJohnson, Zahiro Mor, Zoyd
9	IN	CL., juergen d, walid, Zaga
10	LIKE	Abhilash R Vankayala, Aidan, ashja99, Bart Schuijt, CL., Cristian Abelleira, guiguiblitz, Harish Gyanani, hellyale, Jenism, Lohitha Palagiri, Mark Perera, Mr. Developer, Ojen, Phrancis, RamenChef, Redithion, Stefan Steiger, Tot Zam, Vikrant, vmaroli
11	ORDER BY	Andi Mohr, CL., Cristian Abelleira, Jaydles, mithra chintha, nazark, Özgür Öztürk, Parado, Phrancis, Wolfgang
12	SQL CURSOR	Stefan Steiger
13	SQL Group by vs	carlosb

	Distinct	
14	SQLインジェクション	120196, CL., Clomp, Community, Epodax, Knickerless-Noggins, Stefan Steiger
15	SQLのクリーンコード	CL., Stivan
16	TRUNCATE	Abhilash R Vankayala, CL., Cristian Abelleira, DaImTo, Hynek Bernard, inquisitive_mind, KIRAN KUMAR MATAM, Paul Bambury, ss005
17	TRY / CATCH	Uberzen1
18	UNION / UNION ALL	Andrea, Athafoud, Daniel Langemann, Jason W, Jim, Joe Taras , KIRAN KUMAR MATAM, Lankymart, Mihai-Daniel Virna, sunkuet02
19	WHEREおよびHAVINGをしてをフィルタリングする	Arulkumar, Bostjan, CL., Community, Franck Dernoncourt, H. Pauwelyn, Jon Chan, Jon Ericson, juergen d, Matas Vaitkevicius, Mureinik, Phrancis, Tot Zam
20	XML	Steven
21	インサート	Ameya Deshpande, CL., Daniel Langemann, Dipesh Poudel, inquisitive_mind, KIRAN KUMAR MATAM, rajarshig, Tot Zam, zplizzi
22	インデックス	a1ex07, Almir Vuk, carlosb, CL., David Manheim, FlyingPiMonster, forsvarir, Franck Dernoncourt, Horaciux, Jenism, KIRAN KUMAR MATAM, mauris, Parado, Paulo Freitas , Ryan
23	ウィンドウ	Arkh, beercohol, bhs, Gidil, Jerry Jeremiah, Mureinik, mustaccio
24	カスケード	Stefan Steiger
25	クロス、	Karthikeyan, RamenChef
26	コメント	CL., Phrancis
27	サブクエリ	CL., dasblinkenlight, KIRAN KUMAR MATAM, Nunie123, Phrancis, RamenChef, tinlyx
28	シーケンス	John Smith
29	ジョイン	A_Arnold, Akshay Anand, Andy G, bignose, Branko Dimitrijevic, Casper Spruit, CL., Daniel Langemann, Darren Bartrup-Cook, Dipesh Poudel, enrico.bacis, Florin Ghita, forsvarir, Franck

		Dernoncourt, hairboat, Hari K M, HK1, HLGEM, inquisitive_mind, John C, John Odom, John Slegers, Mark Iannucci, Marvin, Mureinik, Phrancis, raholling, Raidri, Saroj Sasmal, Stefan Steiger, sunkuet02, Tot Zam, xenodevil, ypercube, Рахул Маквана
30	スキップページネーション	CL., Karl Blacquiere, Matas Vaitkevicius, RamenChef
31	ストアドプロシージャ	brichins, John Odom, Lamak, Ryan
32	セレクト	Abhilash R Vankayala, aholmes, Alok Singh, Amnon, Andrii Abramov, apomene, Arpit Solanki, Arulkumar, AstraSerg, Brent Oliver, Charlie West, Chris, Christian Sagmüller, Christos, CL., controller, dariru, Daryl, David Pine, David Spillett, day_dreamer, Dean Parker, DeepSpace, Dipesh Poudel, Dror, Durgpal Singh, Epodax, Eric VB, FH-Inway, Florin Ghita, FlyingPiMonster, Franck Dernoncourt, geeksal, George Bailey, Hari K M, HoangHieu, iliketocode, Imran Ali Khan, Inca, Jared Hooper, Jaydles, John Odom, John Slegers, Jojodmo, JonH, Kapep, KartikKannapur, Lankymart, Mark Iannucci, Mark Perera, Mark Wojciechowicz, Matas Vaitkevicius, Matt, Matt S, Mattew Whitt, Matthew Moisen, MegaTom, Mihai-Daniel Virna, Mureinik, mustaccio, mxmissile, Oded, Ojen, onedaywhen, Paul Bambury, penderi, Peter Gordon, Prateek, Praveen Tiwari, Přemysl Šťastný, Preuk, Racil Hilan, Robert Columbia, Ronnie Wang, Ryan, Saroj Sasmal, Shiva, SommerEngineering, sqluser, stark, sunkuet02, ThisIsImpossible, Timothy, user1336087, user1605665, waqasahmed, wintersolider, WMios, xQbert, Yury Fedorov, Zahiro Mor, zedfoxus
33	データベースとテーブルの	Abhilash R Vankayala, Arulkumar, Athafoud, bignose, Bostjan, Brad Larson, Christian, CL., Dariusz, Dr. J. Testington, enrico.bacis, Florin Ghita, FlyingPiMonster, forsvarir, Franck Dernoncourt, hairboat, JavaHopper, Jaydles, Jon Ericson, Magisch, Matt, Mureinik, Mzzzzzz, Prateek, rdans, Shiva, tinlyx, Tot Zam, WesleyJohnson
34	データ	bluefeet, Jared Hooper, John Odom, Jon Chan, JonMark Perry, Phrancis
35	テーブルデザイン	Darren Bartrup-Cook
36	トランザクション	Amir Pourmand, CL., Daryl, John Odom
37	トリガー	Daryl, IncrediApp

38	ヌル	Bart Schuijt, CL., dd4711, Devmati Wadikar, Phrancis, Saroj Sasmal, StanislavL, walid, ypercube
39	ビュー	Amir978, CL., Florin Ghita
40	マージ	Abhilash R Vankayala, CL., Kyle Hale, SQLFox, Zoyd
41	マテリアライズド·ビュー	dmfay
42	キー	Andrea Montanari, CL., FlyingPiMonster, KjetilNordin
43	の	Aidan, blackbishop, bluefeet, CL., Florin Ghita, Francis Lord, guiguiblitz, Joe W, KIRAN KUMAR MATAM, Lexi, mithra chintha , Ozair Kafray, Simon Foster, Siva Rama Krishna
44		LCIII
45	テーブル	CL., Daniel, dd4711, fuzzy_logic, Gidil, Luis Lema, ninesided, Peter K, Phrancis, Sibeesh Venu
46		Batsu, Chip, CL., Dylan Vander Berg, fredden, Joel, KIRAN KUMAR MATAM, Phrancis, Umesh, xenodevil, Zoyd
47	と	RamenChef, user2314737
48		ɐlǝx, Christos, CL., Dariusz, Fenton, Infinity, Jaydles, Matt, MotKohn, Mureinik, Peter Lang, Stanislovas Kalašnikovas
49	キー	CL., Harjot, Yehuda Shapira
50		Blag, Özgür Öztürk
51	ブロック	Phrancis
52		a1ex07, Gallus, Ryan Rockey, ypercube
53	スキーマ	Hack-R
54		ɐlǝx, Allan S. Hansen, Arthur D, Arulkumar, Batsu, Chris, CL., Damon Smithies, Franck Dernoncourt, Golden Gate, hatchet, Imran Ali Khan, IncrediApp, Jaydip Jadhav, Jones Joseph, Kewin Björk Nielsen, Leigh Riffel, Matas Vaitkevicius, Mateusz Piotrowski, Neria Nachum, Phrancis, RamenChef, Robert Columbia, vmaroli, ypercube
55		Akshay Anand, CL., Daniel Vérité, Dariusz, Dipesh Poudel, FlyingPiMonster, Gidil, H. Pauwelyn, Jon Chan, KIRAN KUMAR MATAM, Matas Vaitkevicius, Matt, Phrancis, Sanjay Bharwani,

		sunkuet02, Tot Zam, TriskalJM, vmaroli, WesleyJohnson
56		CL., Phrancis, user1221533
57	があるサブセットで をつける	Darrel Lee, mnoronha
58	と	Simulant
59		Andreas, CL.
60		CL., Darren Bartrup-Cook, Martin Smith
61	アナリティック	CL., omini data
62	スカラ—/	CL., Kewin Björk Nielsen, Mark Stewart
63		ashja99, CL., Florin Ghita, Ian Kenney, Imran Ali Khan, Jon Chan, juergen d, KIRAN KUMAR MATAM, Mark Stewart, Maverick, Nathan, omini data, Peter K, Reboot, Tot Zam, William Ledbetter, winseybash, Алексей Неудачин
64		Daryl