SQL演習

23新卒データベース研修

目次

概要

- 本演習では座学と演習を通して、SQLについて学んでいきます
- SQLの使い方にも触れますが、データ分析に寄った内容になることが多いです
- 演習の時間は区切り毎に取りますが、解説中に演習を解いても構いません。やりやすい方法で取り組んで下さい
    - 解説で登場するテーブルと、演習で扱うテーブルは、別物です！

概要 - 実行環境について

- 演習ではBigQueryとSQLiteを用います
- 演習問題は次のURLにあります
  - https://github.com/mixigroup/2023BeginnerTrainingDataBasePublic
- SQLiteを使った演習は、Colab Notebookを使います
- BigQueryを使った演習は、以下のいずれかを使います
  - Colab Notebook
  - BigQueryコンソール
    - 問題はColab Notebookにあります

1. SQLiteを用いたデータの操作

基本的なテーブル・レコード操作について、SQLiteを用いて演習します。

- CREATE TABLE
- INSERT
- UPDATE
- DELETE
- DROP TABLE

データの操作 (テーブルの作成)

- CREATE TABLE文を使うと、テーブルを作成することができます。
- usersテーブルを作成する場合は以下のようになります。

```
CREATE TABLE <テーブル名> (
    <カラム名> <データ型>,
    ...
)
```

```
CREATE TABLE users (
    id bigint,
    name string,
    age int
)
```

id	name	age

データの操作 (レコードの追加)

- INSERT文を使うと、テーブルにレコードを追加することができます。
- usersテーブルにレコードを追加する場合は以下のように記述します。
  - カンマで区切ることで、複数レコードを同時に追加することも可能です。

```
INSERT INTO
    <テーブル名>
VALUES
    (v1, v2, ...),
    ...
```

```
INSERT INTO
    users
VALUES
    (100, '三串', 30),
    (101, '四串', 20)
```

id	name	age
100	三串	30
101	四串	20

データの操作 (レコードの更新)

- UPDATE文を使うと、検索条件にマッチしたレコードを更新できます。
- usersテーブルのid: 100のレコードのnameを「佐藤」に置き換える場合、以下のように記述します。

```
UPDATE
    <テーブル名>
SET
    <カラム> = <値>,
    ...
WHERE
    <検索条件>
```

```
UPDATE
    users
SET
    name = ‘佐藤’
WHERE
    id = 100
```

id	name	age
100	佐藤	30
101	四串	20

データの操作 (レコードの削除)

- DELETE文を使うと、検索条件にマッチしたレコードを削除できます。
- usersテーブルからid: 100のレコードを削除する場合は
  以下のように記述します。

```
DELETE FROM
    <テーブル名>
WHERE
    <検索条件>
```

```
DELETE FROM
    users
WHERE
    id = 100
```

id	name	age
101	四串	20

←id: 100のレコードが削除された

データの操作 (レコードの削除)

- 検索条件を指定しない場合は全レコードが削除されます。
- usersテーブルの全レコードを削除する場合は以下のように記述します。

```
DELETE FROM users
```

- なお、MySQL等のRDBMSの場合、テーブルの全レコードを削除するSQLは2種類存在します。
  - DELETE文
    - トランザクションを使っている場合はロールバックできる。
  - TRUNCATE文
    - テーブルを作り直す。
    - DELETE文より高速に動作し、AUTO INCREMENTを初期化できる。
    - RDBMSによってはロールバックできない。

データの操作 (テーブルの削除)

- DROP TABLE文を使うと、テーブルを削除できます。
  - usersテーブルを削除する場合は右の例のように記述します。

DROP TABLE <テーブル名>	DROP TABLE users

データの操作 (演習)

- ★ Notebookの「データの操作」をやってみましょう。

2. BigQueryを用いたデータの取得

SELECT文を用いたデータの取得について、BigQueryを用いて演習を行います。

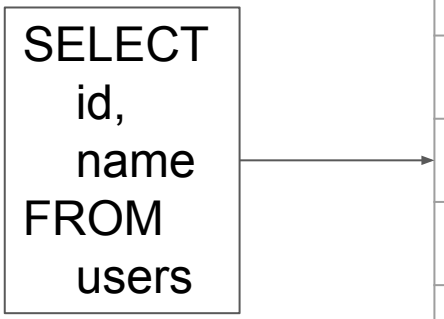
- SELECT
- LIMIT
- ORDER BY
- WHERE

データの取得 (SELECT文)

- DBからデータを取得する際にはSELECT文を使用します。
- usersテーブルから、ユーザーID (id) とユーザー名 (name) を取得する場合は右の例のように記述します。

```
SELECT
    <カラム>
FROM
    <テーブル名>
```

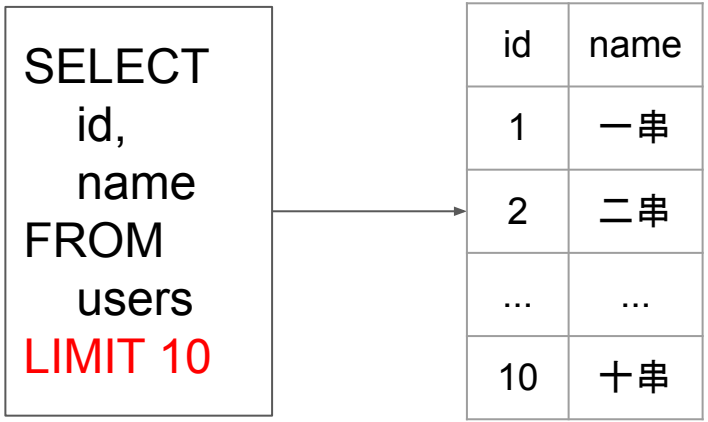
```
SELECT
    id,
    name
FROM
    users
```



id	name
1	一串
2	二串
...	...
99	九九串

データの取得 (LIMIT句)

- SELECT文の末尾にLIMIT句をつけると、取得するレコード数を制限できます。
- usersテーブルから10件取得したい場合は以下のように記述します。
  - 大量のデータを制限なしに取得すると、データベースのクライアントに負荷がかかるため、 SELECT文を使ってデータを見るときは LIMIT句で取得するレコード数を制限する習慣を付けると良いです。



データの取得 (ORDER BY句)

- 取得するレコードの順番を並び替える際はORDER BY句を使います。
- ソートに使うカラムの後ろに昇順で並べる場合は「ASC」を、降順で並べる場合は「DESC」を指定します。何も指定しない場合は昇順になります。
- idでソートする場合は以下のように記述します。

```
SELECT
    id,
    name
FROM
    users
ORDER BY
    id ASC
LIMIT 10
```

id	name
1	一串
2	二串
...	...
10	十串

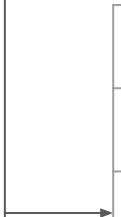
```
SELECT
    id,
    name
FROM
    users
ORDER BY
    id DESC
LIMIT 10
```

id	name
99	九九串
98	九八串
...	...
90	九〇串

データの取得 (WHERE句)

- 検索条件を指定するときはWHERE句を使います。
- ユーザーの年齢が入っている「age」から、18歳以上のユーザーの情報を取得する場合は以下のように記述します。

```
SELECT
    id,
    name,
    age
FROM
    users
WHERE
    18 <= age
LIMIT 10
```



id	name	age
1	一串	32
2	二串	61
...	...	...
10	十串	20

SELECT文の構成

SELECT文は句の順番が決まっています。順番がわからなくなった時は、下記を確認してください。

- WITH
- SELECT
- FROM
- JOIN
- WHERE
- GROUP BY
- HAVING
- ORDER BY
- LIMIT

データの取得 (演習)

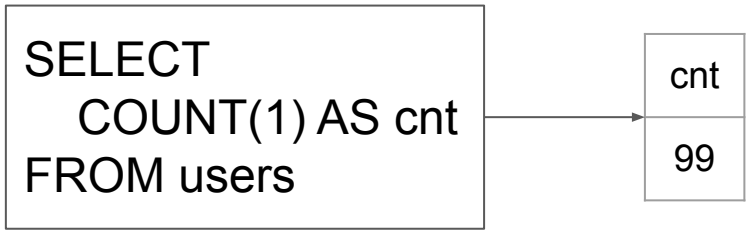
- ★ Notebookの「データの取得」をやってみましょう。

3. BigQueryを用いた集計

- COUNT
- GROUP BY

集計 (COUNT関数)

- レコード数を数える際はCOUNT関数を使います。
  - COUNT関数は評価する式がNULLではないレコードの数を返します。
- usersテーブルのレコード数を数える場合は以下のように記述します。
- 集計関数にはCOUNT関数以外に、合計を出すSUM関数や平均を出すAVG関数などもあります。



集計 (GROUP BY句)

- カラムでグループ分けした結果に対して集計を行う場合、GROUP BY句を使います。また、HAVING句で集計後の値を使った条件指定を行えます。
- 20歳以上のユーザーを対象に、2人以上該当者がいる年齢とその人数を集計する場合、右の例のように記述します。

```
SELECT
    <カラム>
FROM
    <テーブル>
WHERE
    <条件>

GROUP BY
    <集約するカラム名 or カラムの番号>
HAVING
    <集約後の結果に対する条件>
```

```
SELECT
    age,
    COUNT(1) AS cnt
FROM
    users
WHERE
    20 <= age
GROUP BY
    age
HAVING
    2 <= cnt
```

age	cnt
20	2
22	5
...	...

集計 (演習)

- ★ Notebookの「集計」をやってみましょう。

4. BigQueryを用いた条件分岐

- CASE式
- 真偽値を返す構文
  - IN句
  - LIKE
  - BETWEEN

条件式 (CASE式)

- 条件分岐に該当するもので、2通りの書き方があります。
- 単純CASE式

```
CASE <カラム>
    WHEN <値> THEN <カラム = 値のときの返り値>
    …
    ELSE <上記を満たさないときの返り値>
END
```

- 検索CASE式

```
CASE
    WHEN <条件> THEN <条件を満たすときの返り値>
    …
    ELSE <上記を満たさないときの返り値>
END
```

- ELSEを省略するとNULLを返します

真偽値を返す構文

便利な構文がいくつかあります。

- IN句
  - item IN (1,2,3)
  - item = 1 OR item = 2 OR item = 3 と同義
- BETWEEN句
  - id BETWEEN 10 AND 20
  - 10 <= id AND id <= 20 と同義。両端を含みます。
- LIKE句
  - name LIKE "Wata%be"
  - % は0文字以上の任意の文字列にマッチ。上の例は「 Watabe」「Watanabe」両方にマッチします。
  - %以外にも _（アンダースコア）は任意の1文字にマッチ

条件式 (演習)

- ★ Notebookの「条件式」をやってみましょう。

5. BigQueryとSQLiteを用いた集合操作

BigQueryで行う演習

- サブクエリ
- WITH
- UNION

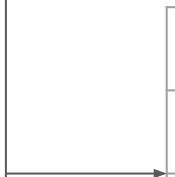
SQLiteで行う演習

- JOIN
- LEFT OUTER JOIN
- CROSS JOIN

集合操作 (サブクエリ)

- SELECTの結果を別のクエリ内で使用することをサブクエリと言います。
- ユーザーテーブルにおいて、2人以上該当者がいる年齢とその人数を数える場合、以下のように記述します。

```
SELECT
    age,
    cnt
FROM
(
    SELECT
        age,
        COUNT(1) AS cnt
    FROM
        users
    GROUP BY
        age
)
WHERE
    2 <= cnt
```



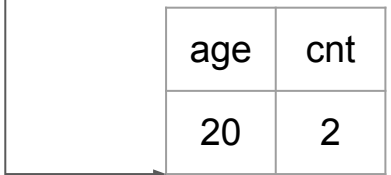
age	cnt
20	2
22	5
...	...

集合操作 (WITH句)

- サブクエリに名前を付けてクエリの外側へ持って行く場合はWITH句を使います。
- ネストを下げたり、サブクエリを複数箇所で使用したりすることができます。

```
WITH ages AS (
    SELECT
        age,
        COUNT(1) AS cnt
    FROM
        users
    GROUP BY
        age
)

SELECT
    age,
    cnt
FROM
    ages
WHERE
    2 <= cnt
```



age	cnt
20	2
22	5
...	...

集合操作 (UNION句)

- クエリ結果を縦方向に結合させる場合はUNION句を使います。
- UNION句は2種類あります。

- UNION
  - 重複するレコードの排除が行われるが、その分速度が落ちる。
- UNION ALL
  - 重複するレコードの排除を行わないため高速。

```
SELECT
   'table1' AS label,
   col1
FROM
   table1
UNION
SELECT
   'table2' AS label,
   col1
FROM
   table2
```

label	col1
table1	A
...	...
table2	B
...	...

```
SELECT
   'table1' AS label,
   col1
FROM
   table1
UNION ALL
SELECT
   'table2' AS label,
   col1
FROM
   table2
```

集合操作 (演習)

- ★ Notebookの「集合操作（サブクエリ・UNION）」をやってみましょう。

集合操作 (JOIN)

- 複数のテーブルやサブクエリを特定の条件で横方向に結合することができます。
- JOINは3種類あります。

- 内部結合 (INNER JOIN)
  - 条件に一致するレコードのみを結合する
- 外部結合 (OUTER JOIN)
  - 条件に一致するレコードがない場合はNULLとして結合する
- クロス結合 (CROSS JOIN)
  - 直積
  - 配列データをバラすときなどに使用

集合操作 (INNER JOIN)

- ユーザーテーブル users とログイン履歴 logins を内部結合 (INNER JOIN) する場合を考えてみます。
- users.idとlogins.user_idで結合できるので、以下のようなクエリになります。

```
SELECT
    *
FROM
    users AS u
JOIN
    logins AS l
ON
    u.id = l.user_id
```

users由来			logins由来			
id	name	age	id	user_id	time	kind
2	二串	61	1	2	2021-04-01 12:00:01	1
5	五串	45	3	5	2021-04-01 12:03:35	0
...	...	...	...	...	...	...

集合操作 (OUTER JOIN)

- ログインしていないユーザー (JOINの左のusersにはレコードがあるが、右のloginsにはない) のレコードも出す場合はLEFT OUTER JOINにします。
- ログインしていないユーザーのlogins由来のカラムにはNULLが入ります。
  - なお、実際に使うことはほぼありませんが、RIGHT OUTER JOINもあります。

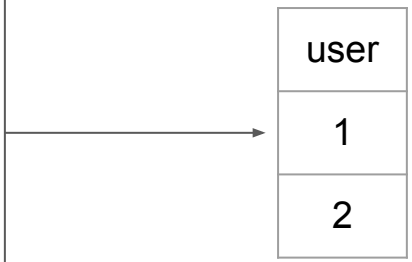
```
SELECT
   *
FROM
   users AS u LEFT OUTER JOIN logins AS l
ON
   u.id = l.user_id
```

users由来			logins由来			
id	name	age	id	user_id	time	kind
1	一串	32	NULL	NULL	NULL	NULL
2	二串	61	1	2	2021-04-01 12:00:01	1
...	...	...	...	...	...	...

集合操作 (CROSS JOIN)

- テーブルの直積をとる場合はCROSS JOINを使います。
- 配列データをバラすときなどに使用します。

```
SELECT
    t.id,
    u.value AS user
FROM user_list AS t
CROSS JOIN
    JSON_EACH(t.users) AS u
```



user
1
2

集合操作 (演習)

- ★ Notebookの「集合操作（JOIN)_」をやってみましょう。

インデックス

- データ量が多いテーブルにおいて、検索条件としてよく使われるカラムにインデックスを付与すると、データ取得を高速化できます。
- ただし、インデックスを付与するとINSERTが遅くなるので、その辺りも考慮する必要があります。
- ★ インデックスの効果は実際にテーブルに触れてみると分かりやすいので、Notebookの「インデックス」をやってみましょう。

応用編のご紹介

- 演習では応用編としてデータ分析や機械学習を行う際に便利な分析関数 (ウィンドウ関数) についても扱っています。興味がある人は見てみてください。

最後に

- ★ Notebookの「片付け」でGoogle Cloudのプロジェクトを削除してください。
- 復習を行いたい場合はもう一度「設定」を行うと新たなGoogle Cloudプロジェクトを作成できます。