4-4-5.SELECT文①

はじめに

SELECT文 は **SQL(データベース言語)** において、とっても重要な文です。
SELECT文は名前のとおり、データベースにある情報を抽出する文です。
単純に **データの一覧を抽出** したり、 **条件を指定して必要な情報を抽出** することができます。
この章では、SELECT文の基本的な部分を学んで行きましょう！

Step1 : SELECT文の使い方

SELECT文はDBにあるデータを取得する命令文です。

```
SELECT 取得したい列名 FROM テーブル名
```

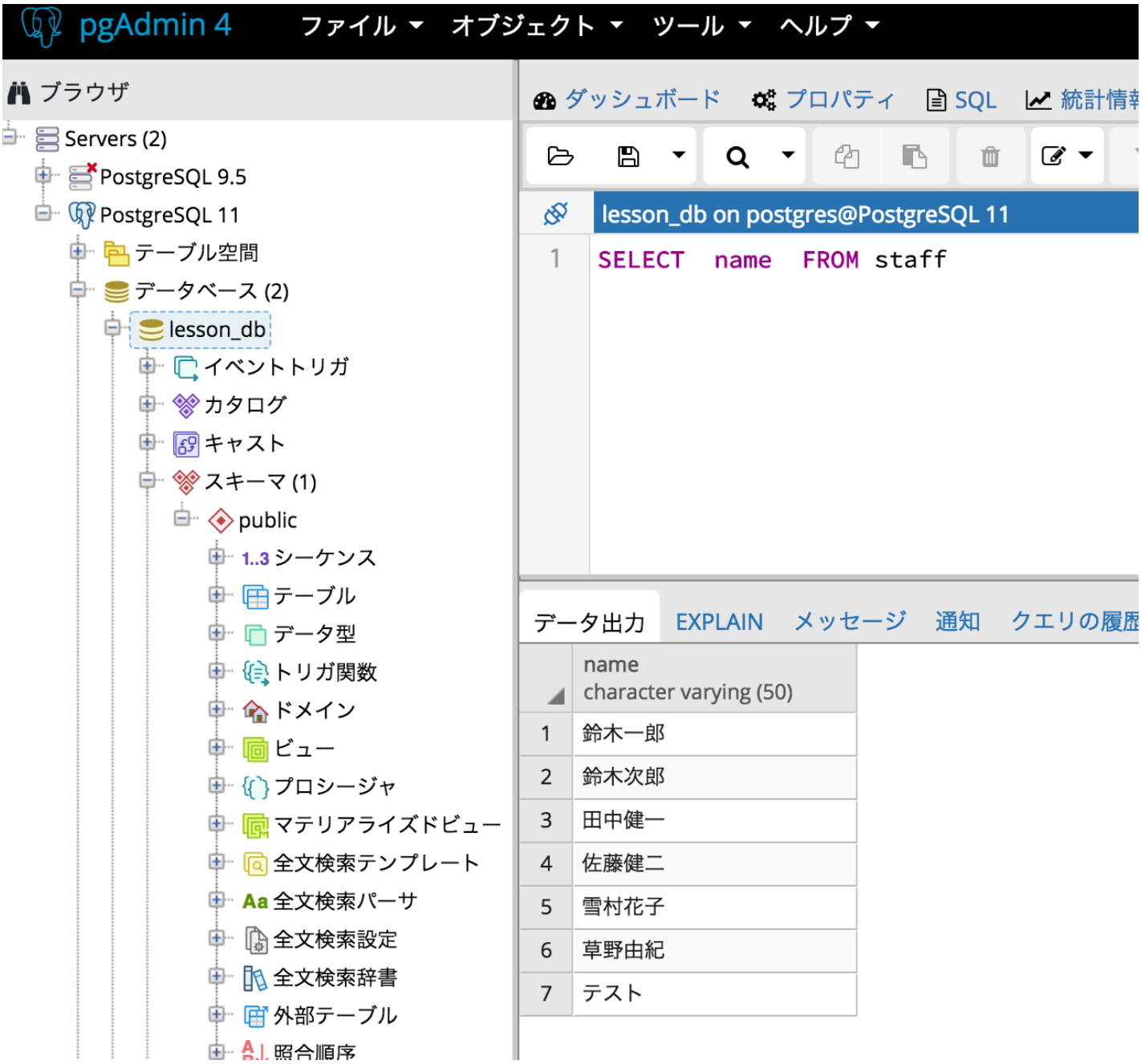
SQL文では大文字、小文字の区別がありません。
ここでは見やすいように列名、テーブル名は小文字で記述します。

※SQLPlusやMySQLコマンドラインクライアントなどで実行する時は、
最後に ; をつけて、文の終了であることを示します。

図のテーブル名は staff です。
ここから社員全員の氏名を抽出したい場合は次のように記述します。

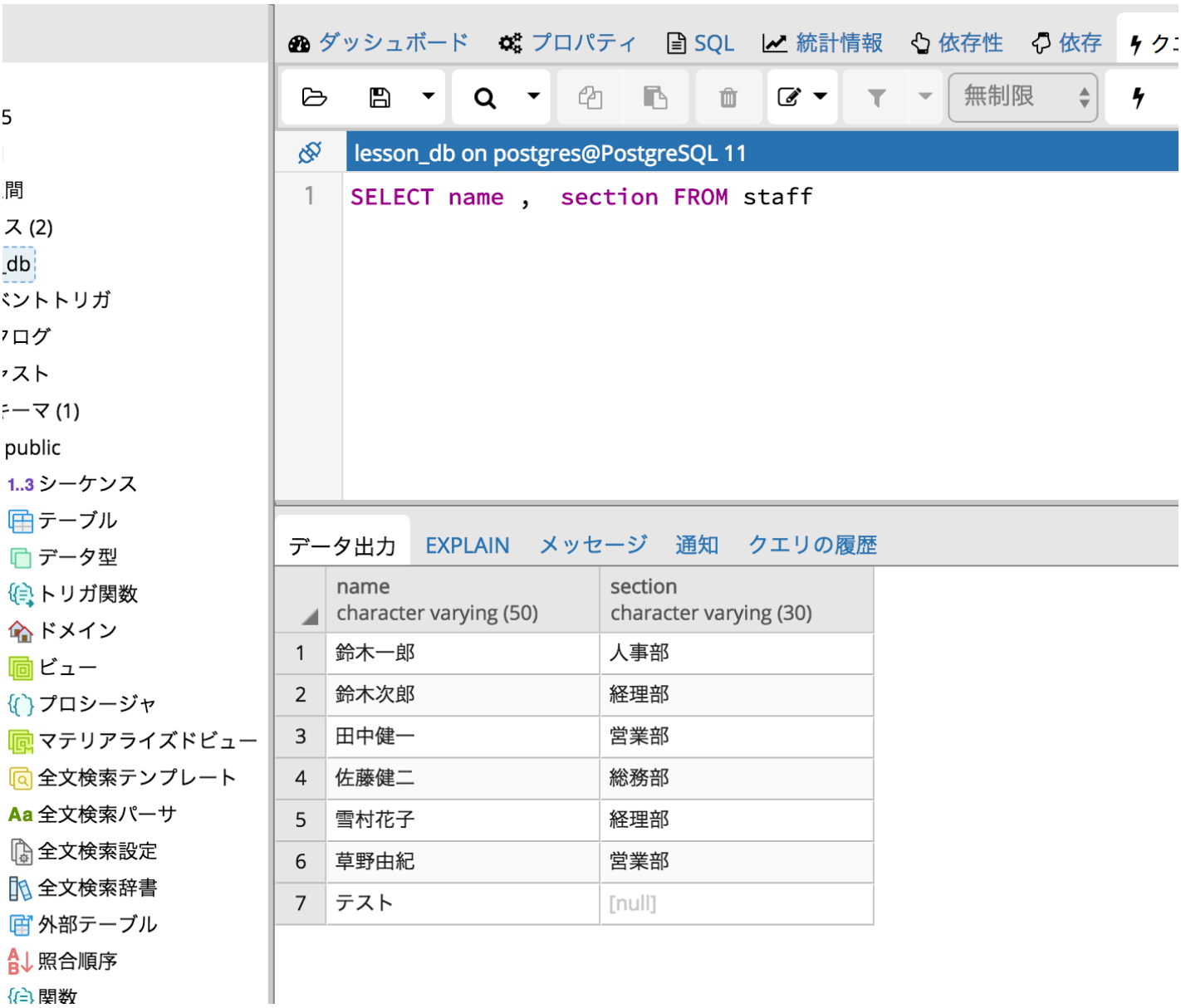
```
SELECT  name  FROM staff
```

?



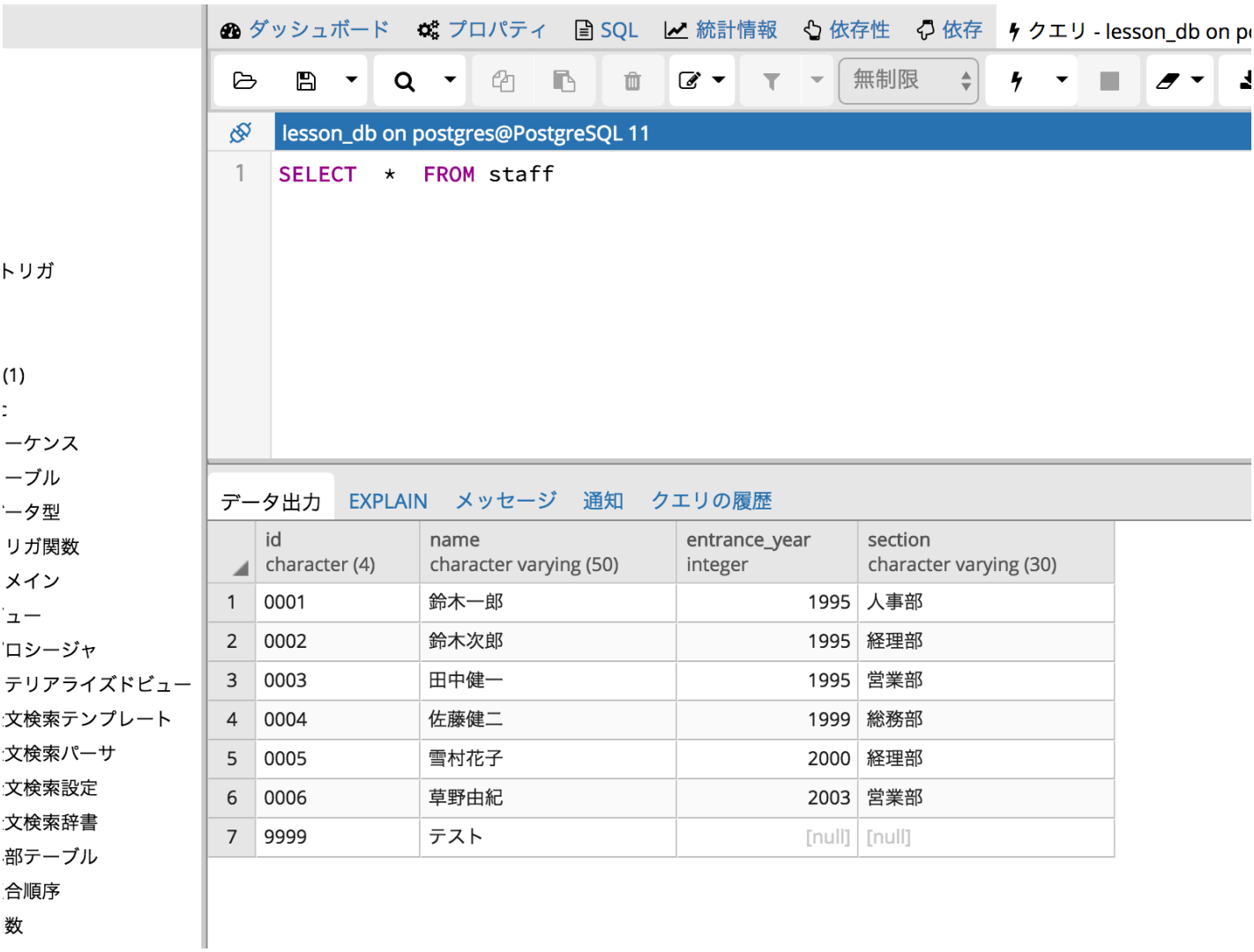
全社員の氏名と所属部署を抽出したい場合は次のように列名をカンマで区切ります。

```
SELECT name ,  section FROM staff
```



また、列名に＊（アスタリスク） を使用することで全列指定することもできます。

```
SELECT  *  FROM staff
```

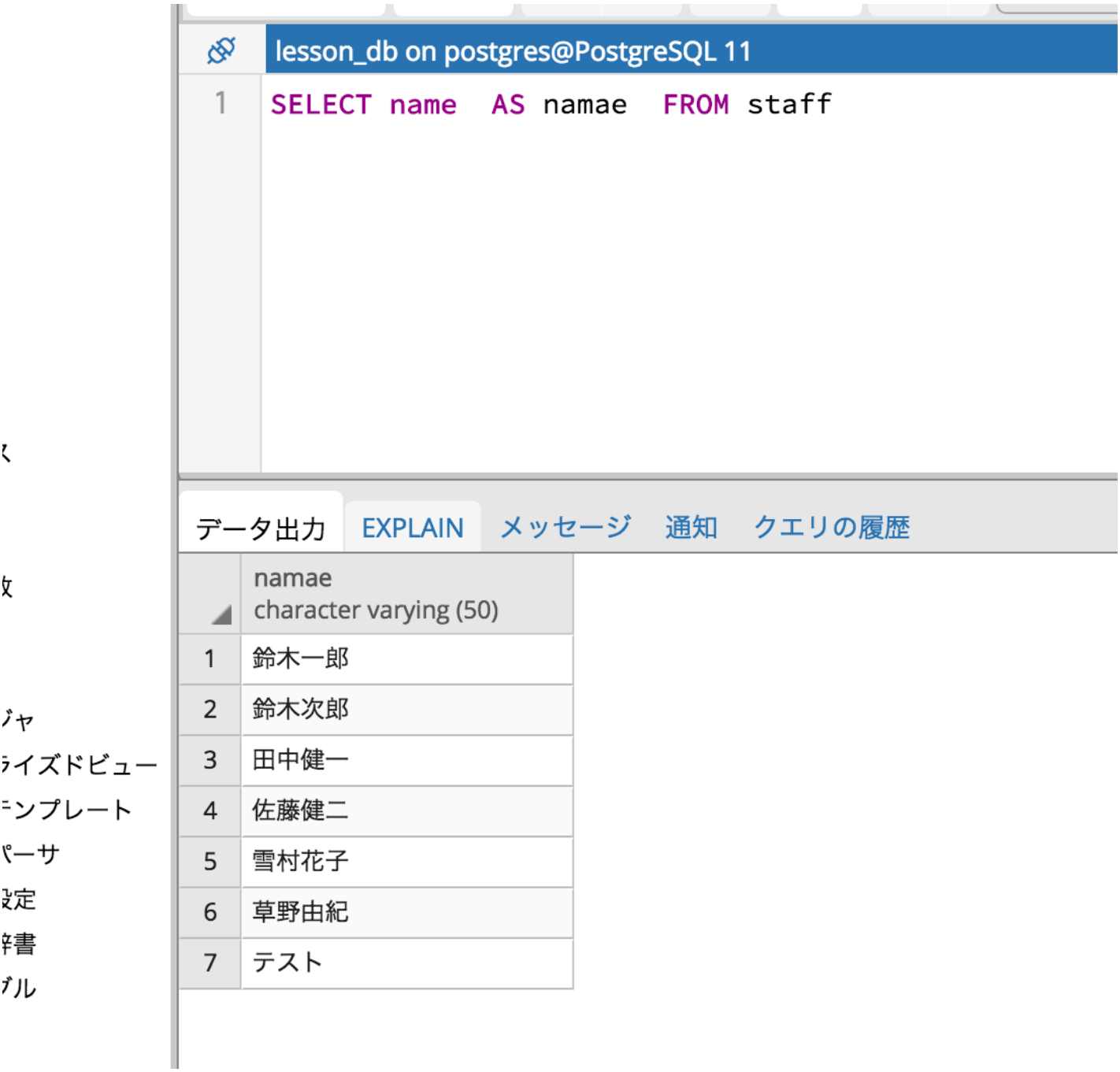


※しかしアスタリスクを使用すると、処理速度が列名を指定する場合より下がり、
テーブルレイアウト変更による障害の原因にもなるため実際の開発では通常使用しません。

Step2 : 別名

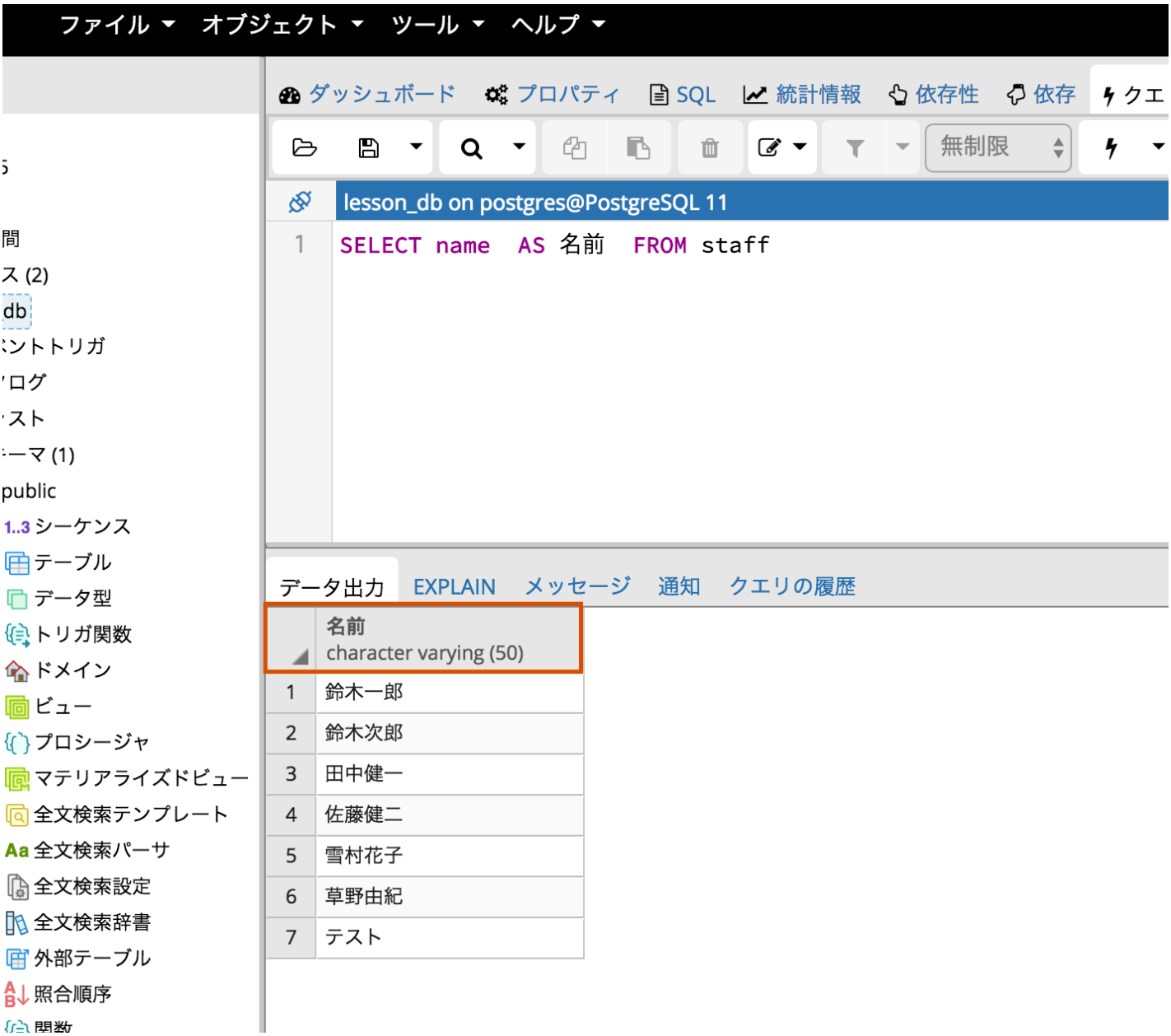
列名に別名をつけることもできます。

```
SELECT name AS namae FROM staff
```



日本語を表示できる環境、設定ならば別名には日本語も使えます。

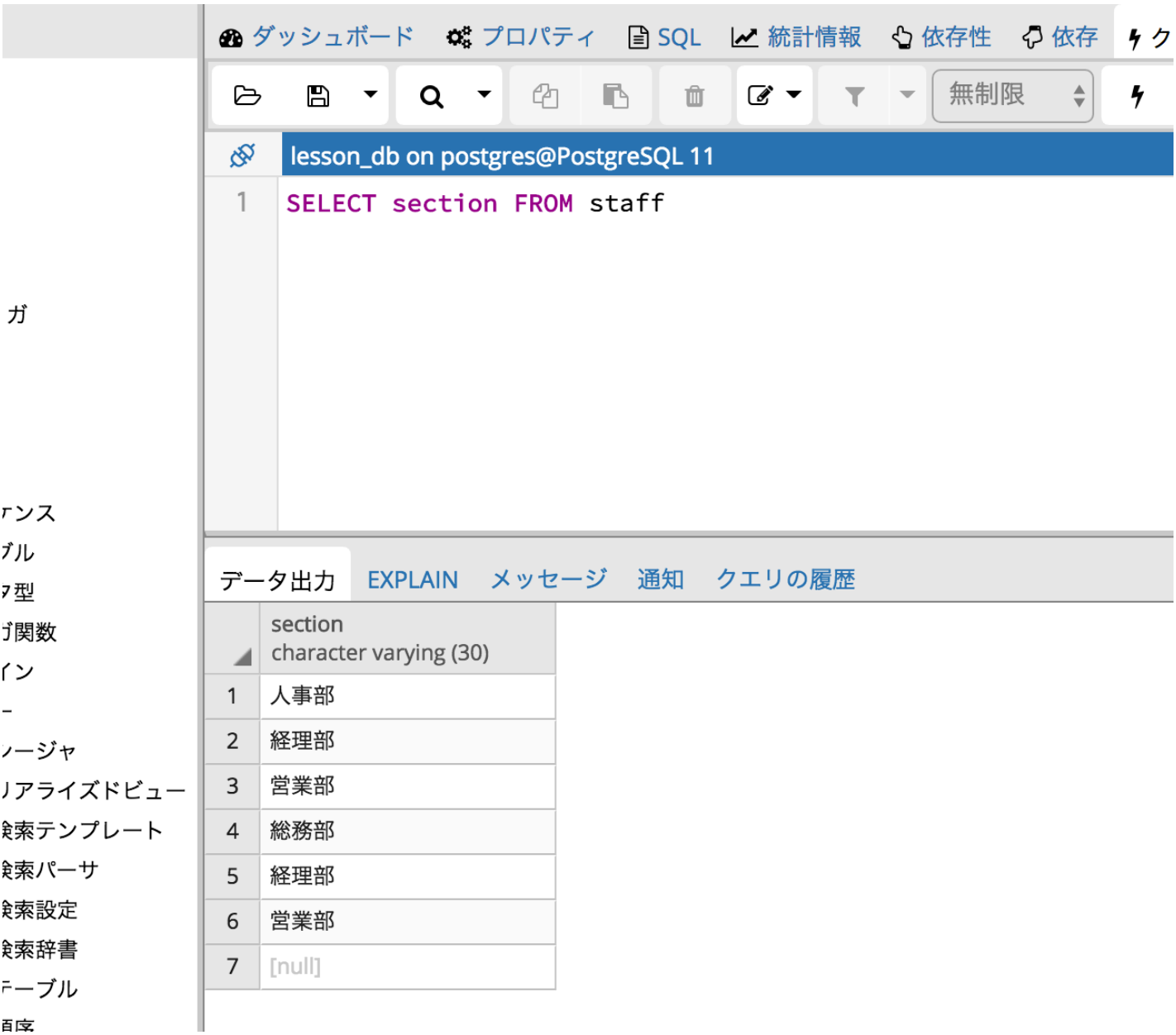
```
SELECT name AS 名前 FROM staff
（DBMSによりシングルクォートやダブルクォートが必要な場合があります）
```



Step3 : Distinct

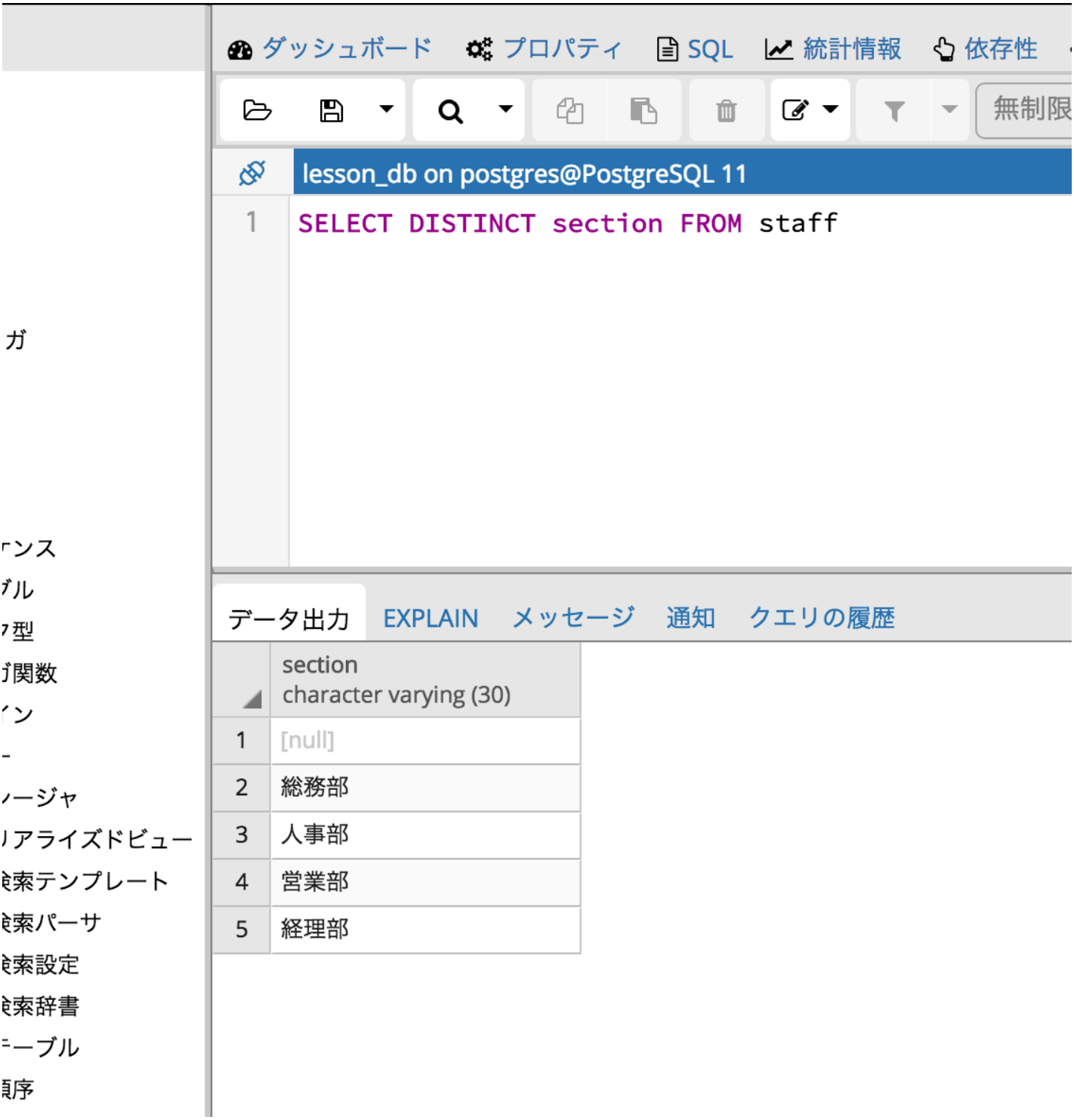
社内にはどれだけの部署があるのか抽出してみましょう。

```
SELECT section FROM staff
```



重複して存在する名前がいくつも出てきてしまいます。
DISTINCTで列名を指定すると重複分は抽出されません。

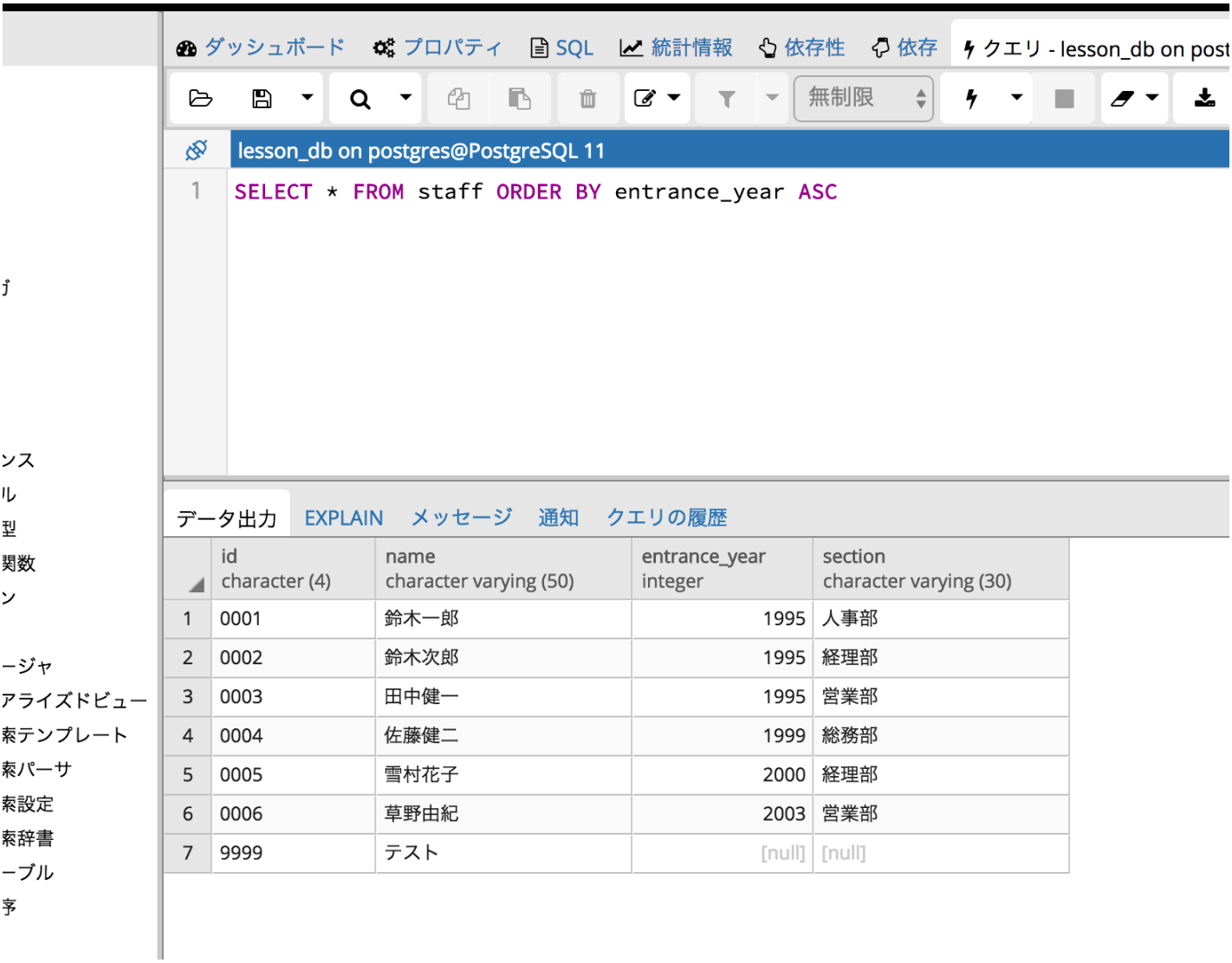
```
SELECT DISTINCT section FROM staff
```



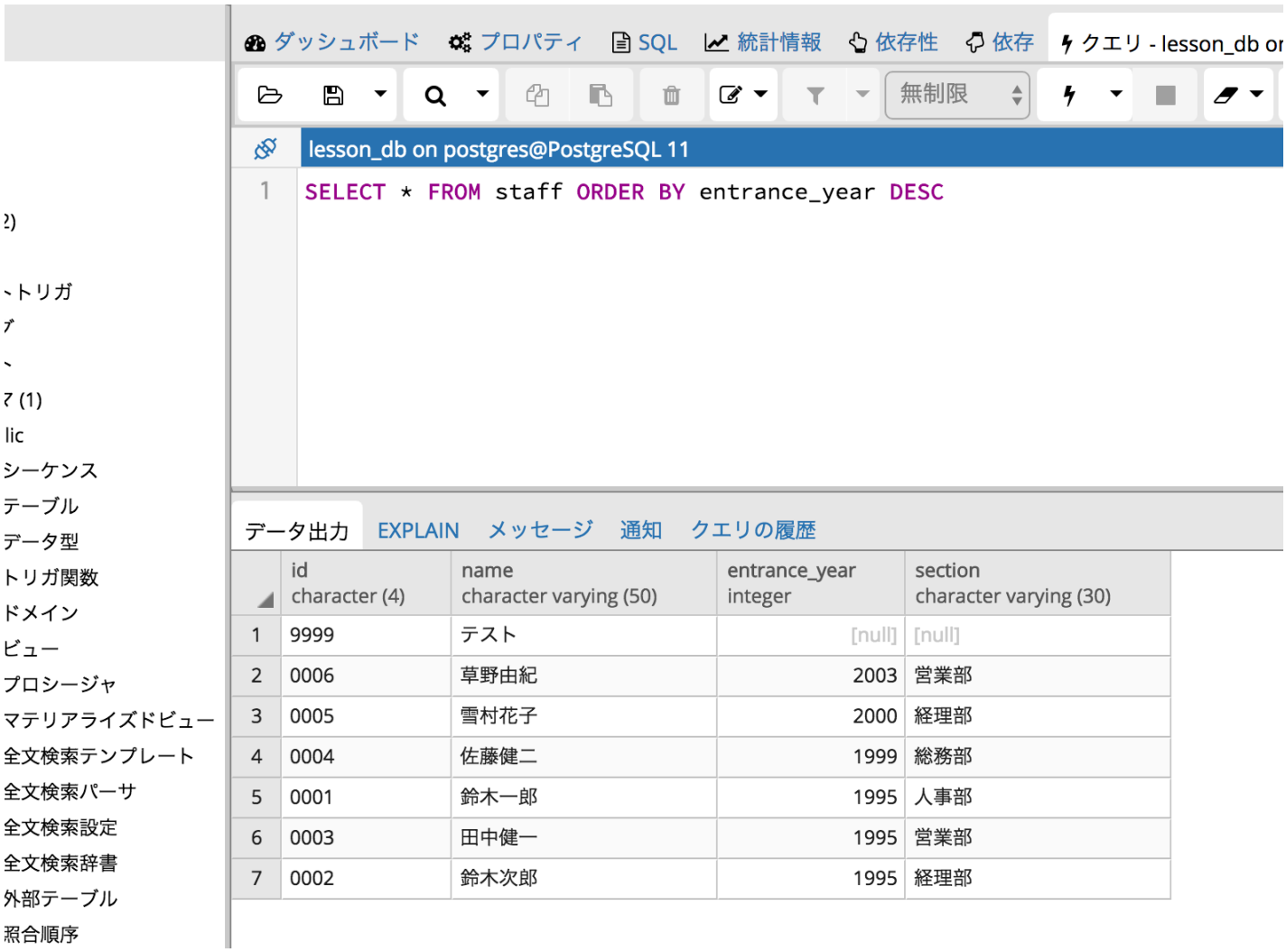
Step4 : ORDER BY　（ソート）

データをソートすることも可能です。 `ORDER BY句` を使用します。
昇順、降順をそれぞれ ASC、DESC で指定します。 **ASCは ascending** 、 **DESCは descending** の略です。
では、入社年度をそれぞれ昇順と降順で抽出してみましょう。

```
SELECT * FROM staff ORDER BY entrance_year ASC （昇順）
SELECT * FROM staff ORDER BY entrance_year DESC （降順）
```

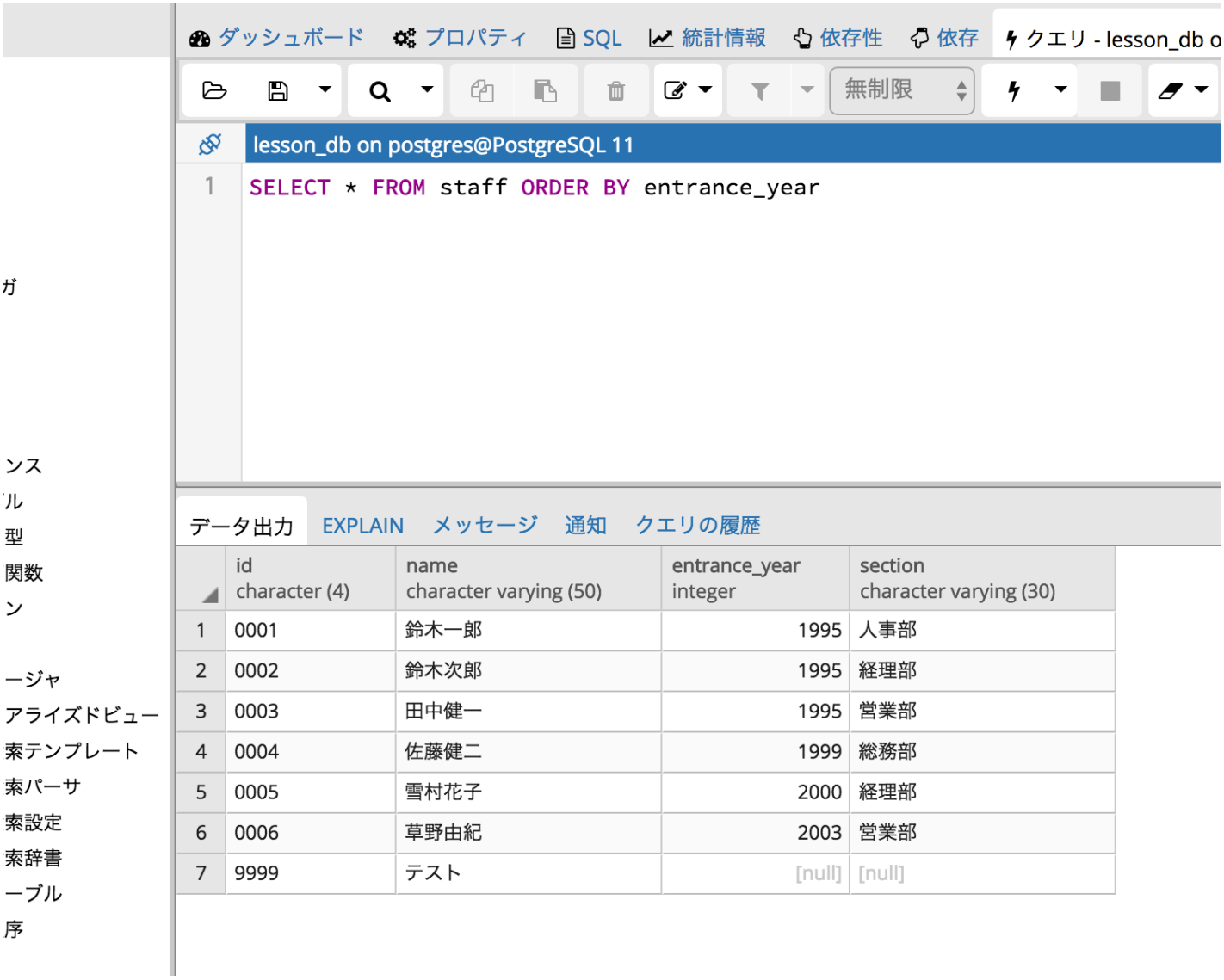


	id character (4)	name character varying (50)	entrance_year integer	section character varying (30)
1	0001	鈴木一郎	1995	人事部
2	0002	鈴木次郎	1995	経理部
3	0003	田中健一	1995	営業部
4	0004	佐藤健二	1999	総務部
5	0005	雪村花子	2000	経理部
6	0006	草野由紀	2003	営業部
7	9999	テスト	[null]	[null]



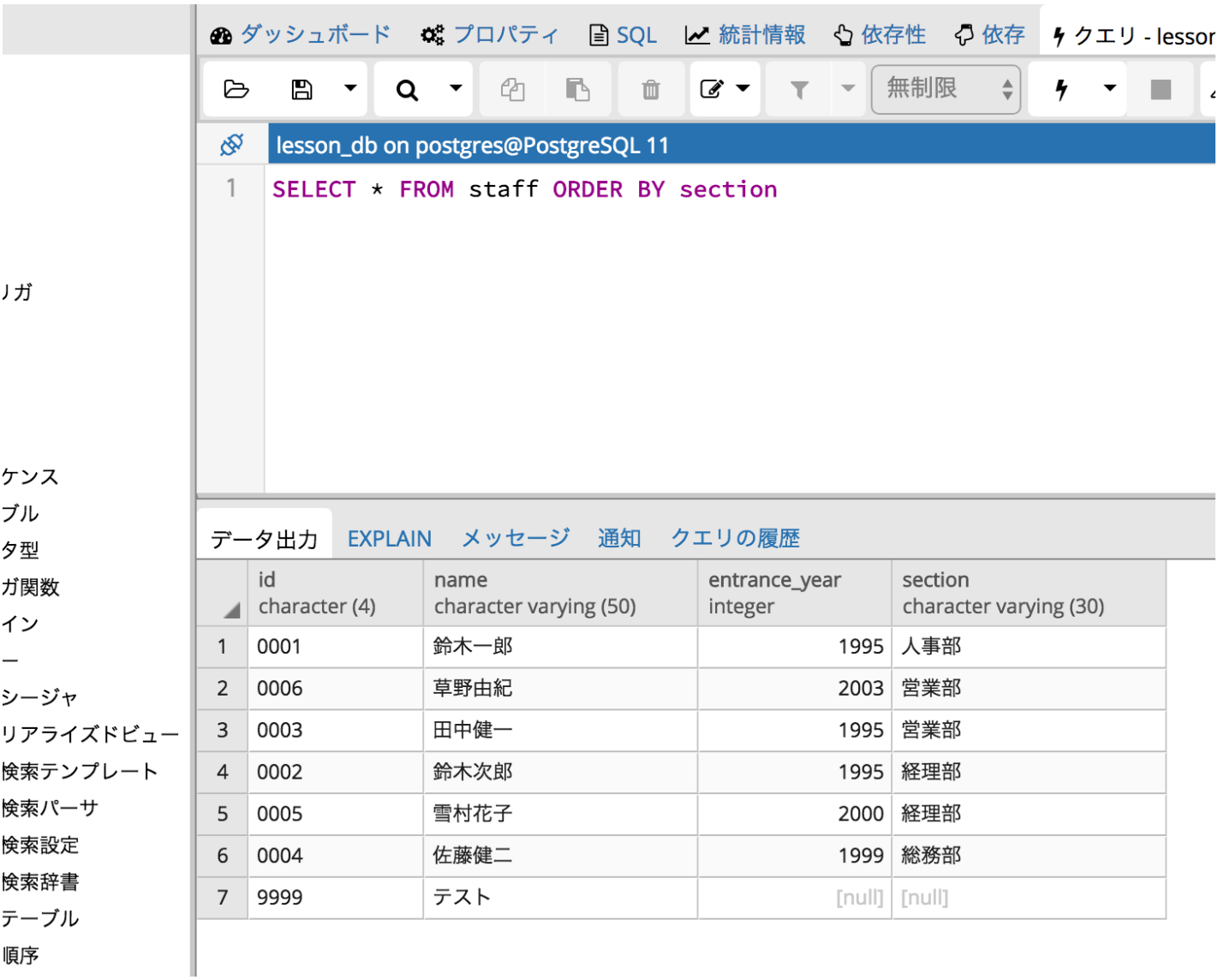
ASC、DESCを指定しない場合、ASC（昇順）と判断されます。

```
SELECT * FROM staff ORDER BY entrance_year
```



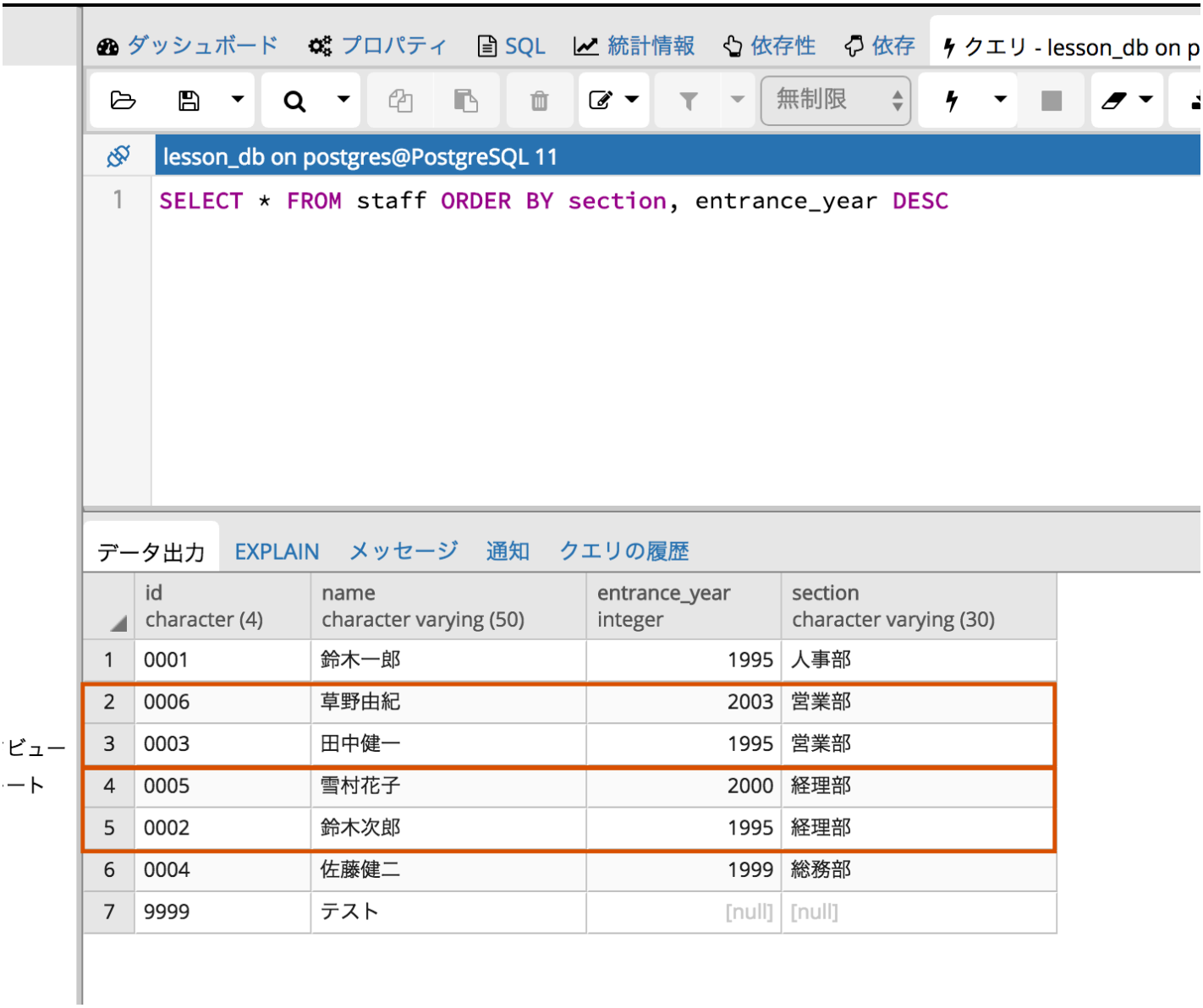
ソート対象が文字列型の場合、文字コード順に並びます。

```
SELECT * FROM staff ORDER BY section
```



部署でソートされました。しかし、その中でも更に入社年度の新しい人から出力したい・・・。
そのような場合は、ソート対象の列名をカンマで区切って複数指定することで、
複数のソートキーによるソートを行うことができます。

```
SELECT * FROM staff ORDER BY section, entrance_year DESC
```



	id character (4)	name character varying (50)	entrance_year integer	section character varying (30)
1	0001	鈴木一郎	1995	人事部
2	0006	草野由紀	2003	営業部
3	0003	田中健一	1995	営業部
4	0005	雪村花子	2000	経理部
5	0002	鈴木次郎	1995	経理部
6	0004	佐藤健二	1999	総務部
7	9999	テスト	[null]	[null]

ビュー
ート

これで部署ごとに入社年度の新しい人から並びました。

条件指定、各種演算子

SELECT文で、テーブルの情報を抽出できるようになりました。
ただいつも全レコードが出てきてしまいます。
SQL文でも、条件を指定することで、抽出したものを選定できます。
この章では、 **必要な情報** を抜き出す方法を学んで行きましょう！

Step1: WHERE（条件）

条件を指定するには、`WHERE句` を記述します。

```
SELECT xxxxx FROM xxxxx WHERE 列名 = 値
```

さて、ここで重要な事柄があります。
他のプログラミング言語と同様にDBにも データの型 があります。
各DBに特有の型もありますが、おおまかに以下が存在します。

種類	データ型名
文字列型	CHAR、VARCHAR、VARCHAR2、・・・

種類	データ型名
数値型	NUMBER、NUMERIC、INT、DECIMAL、・・・
日付型	DATE、TIMESTAMP、・・・

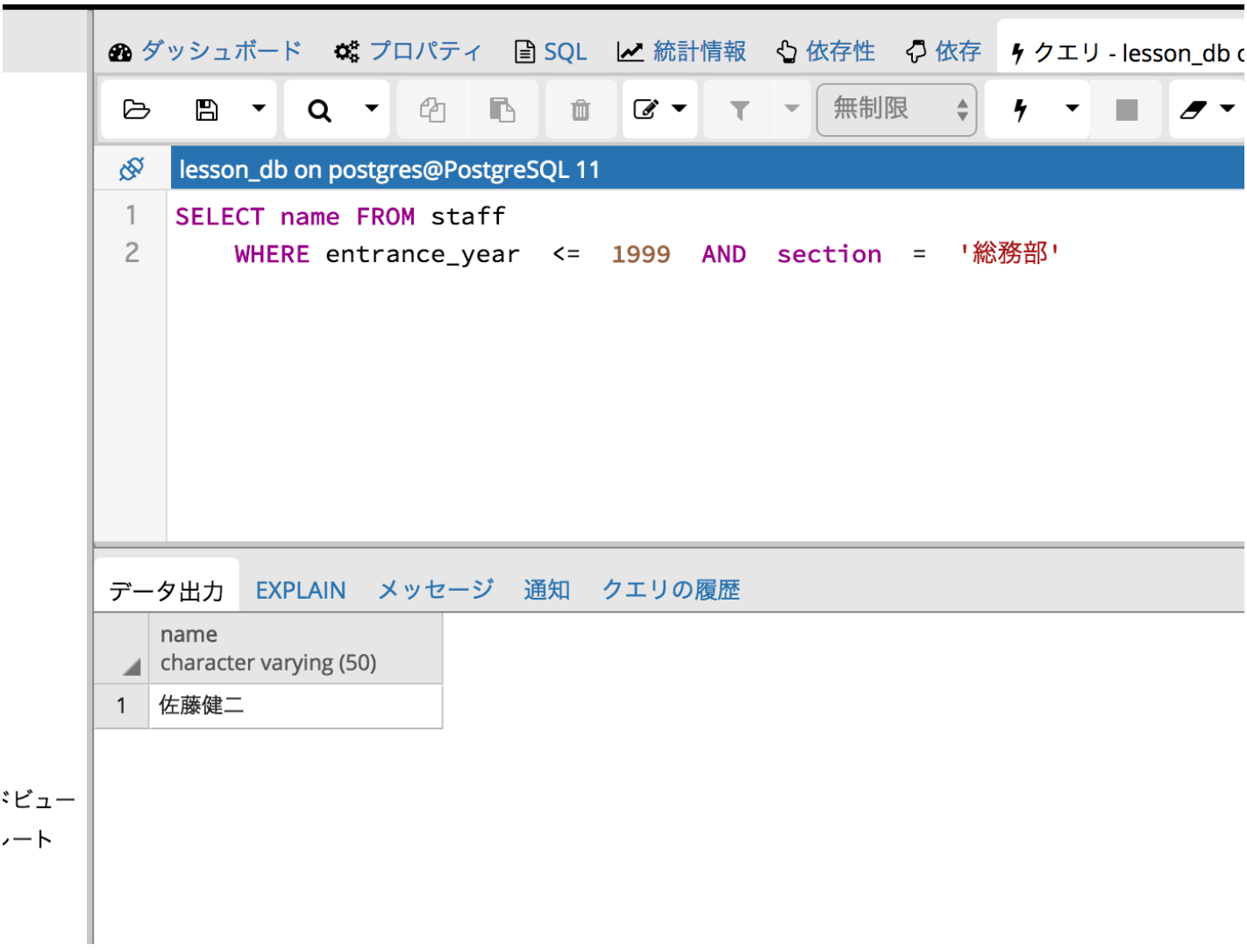
DBによりデータ型名に相違があるため、ここでの詳細な記述はしません。
実際に使用するデータベースについて各自調べてみて下さい。

SQL文に数値型を記述する際は、そのまま数値を記述します。
文字列型の場合は ' シングルクォート で囲みます。
文字列型の数字などは、シングルクォートで囲まなくてもDB側で自動的に解釈して結果を返しますが、
データ型指定をきちんと行わないと後で不具合の原因になります。
テーブルが、どのようなデータ型で構成されているかをいつも意識するとよいでしょう。

Step2 : 演算子

条件はANDでつなげることで複数指定できます。
次のSELECT文では入社年度が1999年以前の総務部に在籍している社員名を抽出します。

```
SELECT name FROM staff
    WHERE entrance_year <= 1999 AND section = '総務部'
```



上記の例では3つの演算子が現れました。SQLでは以下の演算子が使えます。

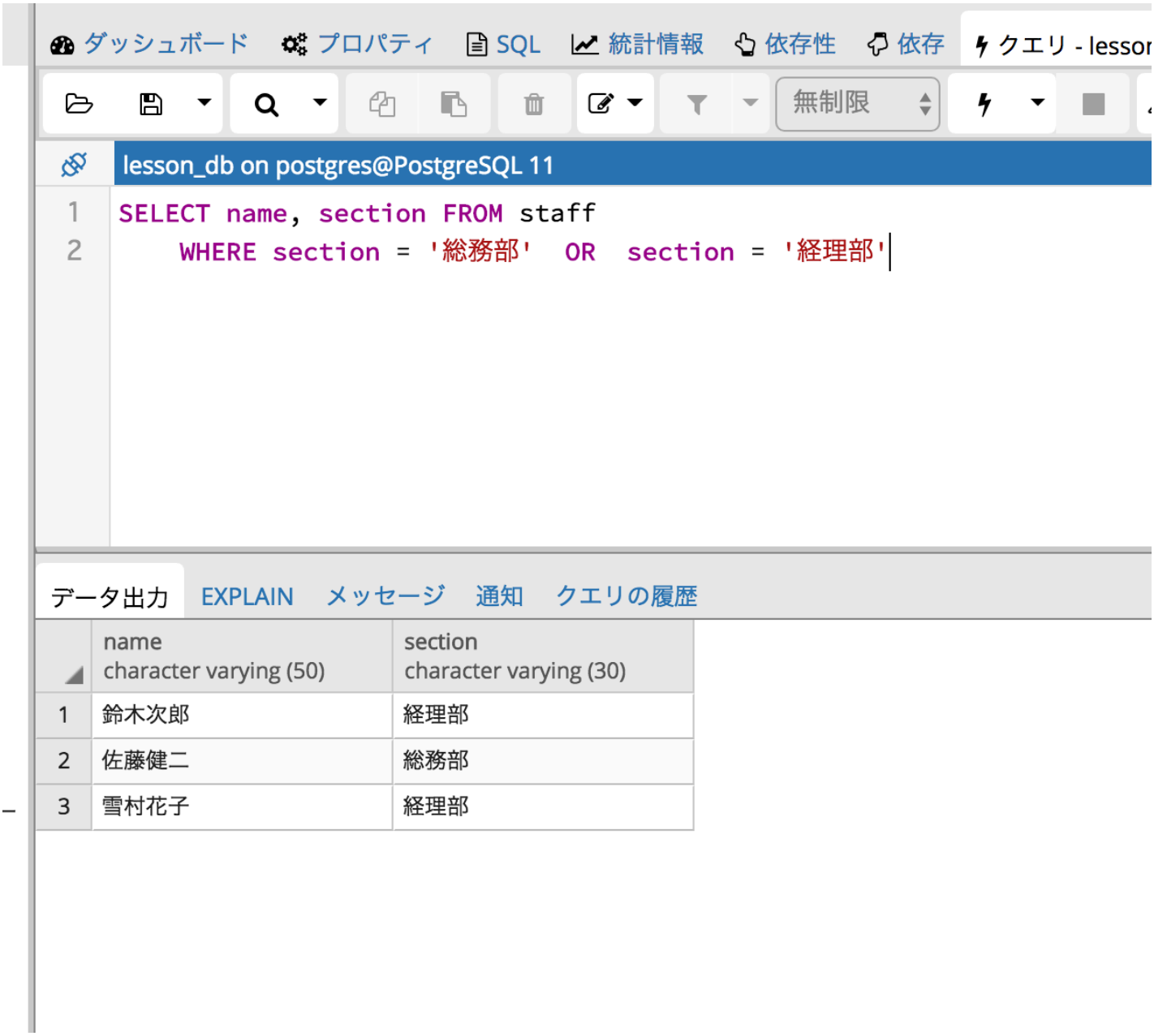
比較演算子	内容
=	等しい
>	大きい
<	小さい
>=	以上
<=	以下
<> または !=	等しくない

論理演算子	内容
AND	かつ
OR	または
NOT	ではない

Step3：IN演算子

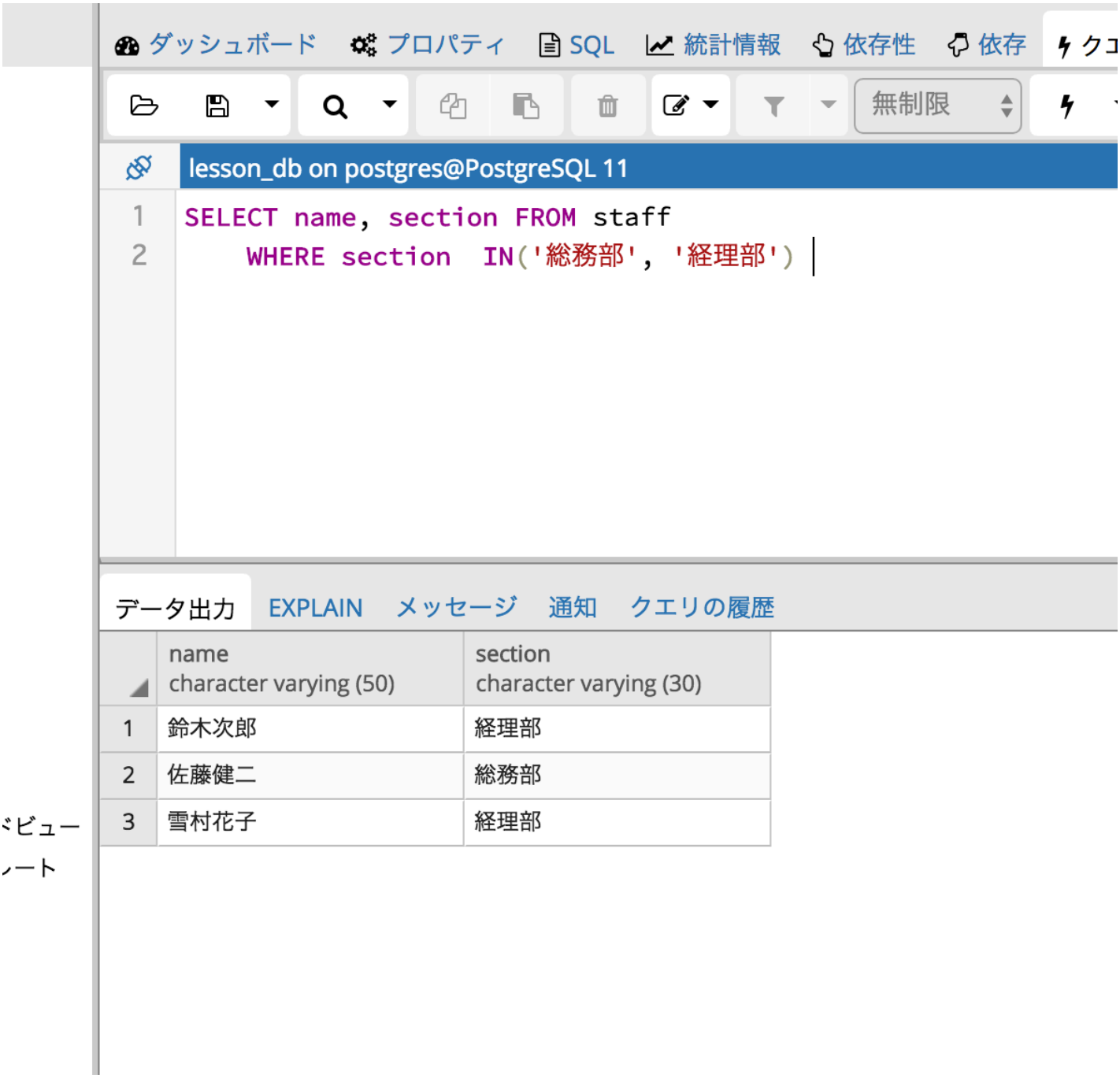
それでは総務部か経理部の社員の名前と部署を抽出してみましょう。

```
SELECT name, section FROM staff
    WHERE section = '総務部' OR section = '経理部'
```



また、IN演算子を使って以下のように記述することもできます。

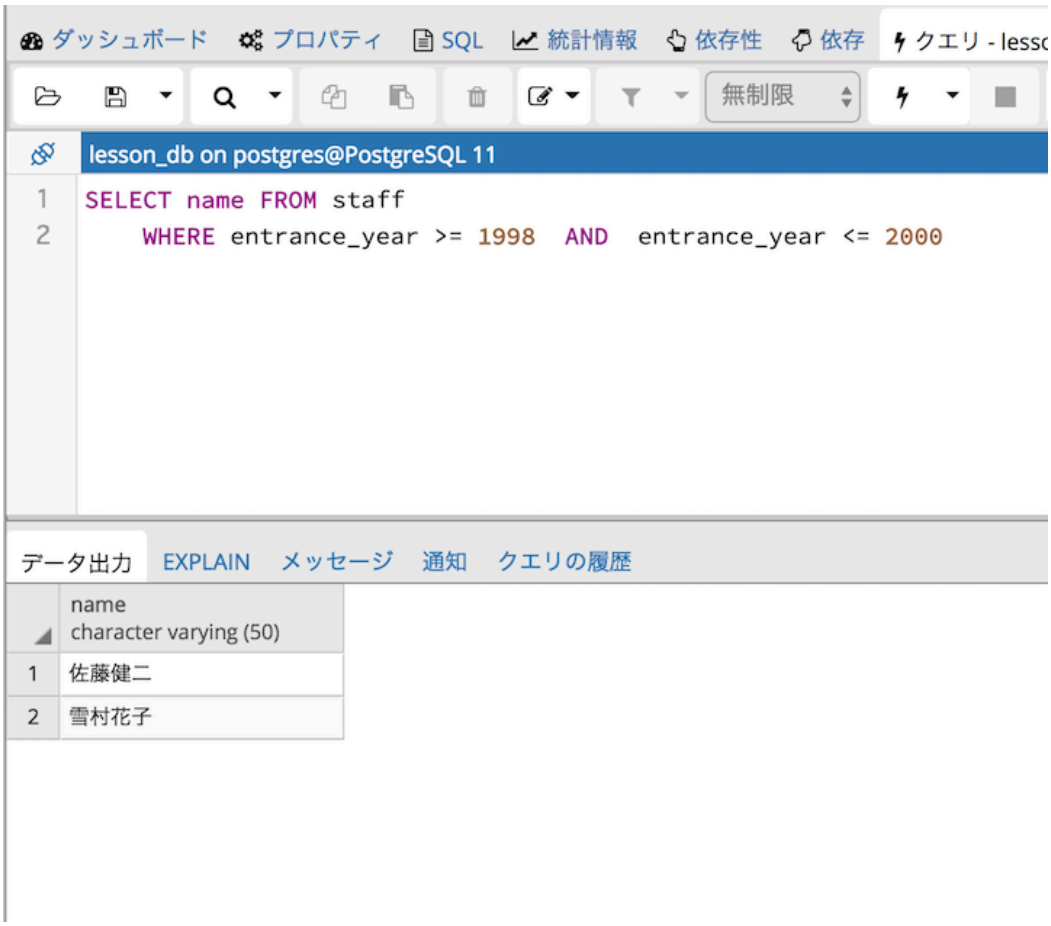
```
SELECT name, section FROM staff
    WHERE section IN('総務部', '経理部')
```



BETWEEN演算子

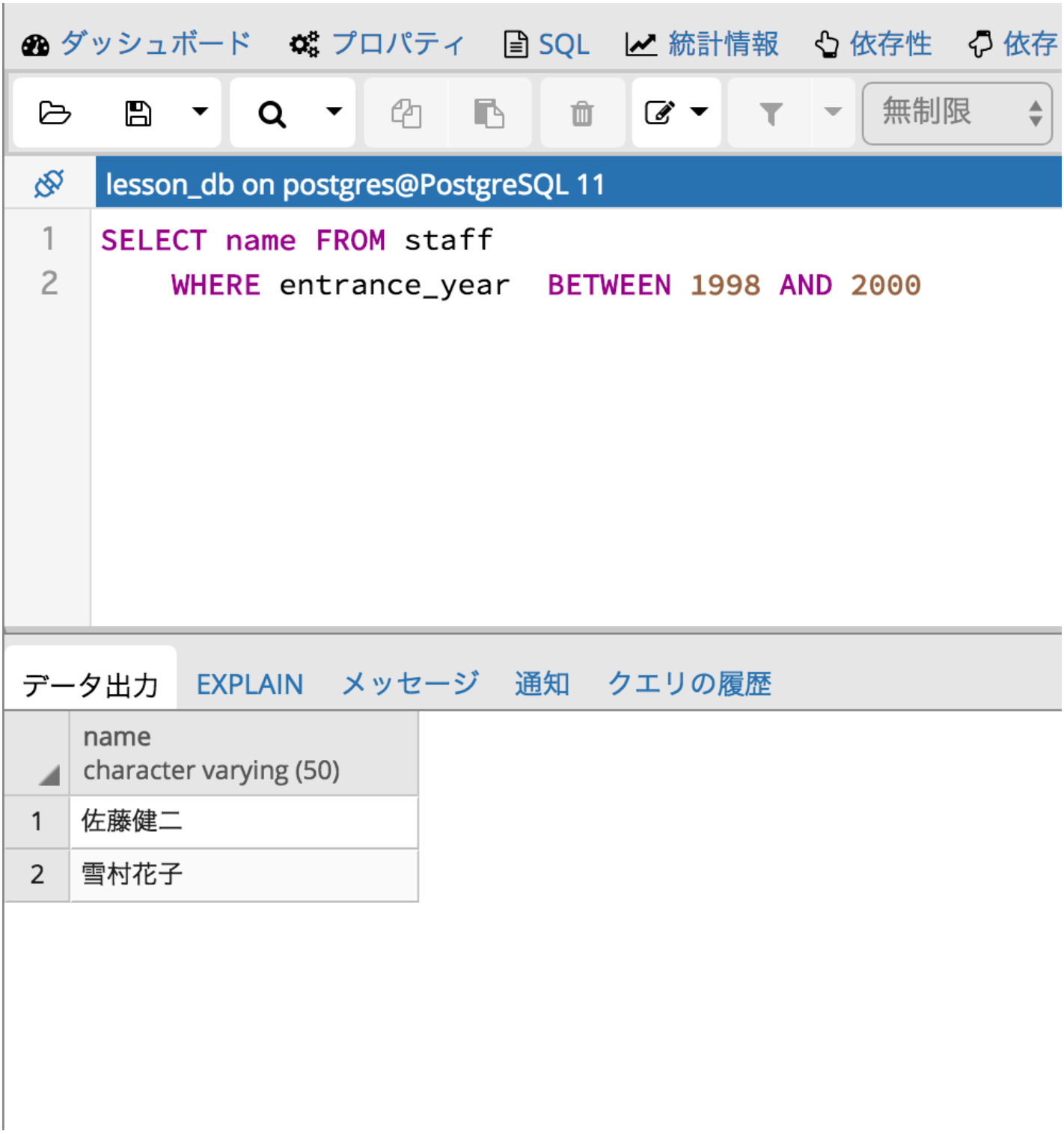
次は入社年度が1998年から2000年までの社員名を抽出してみましょう。

```
SELECT name FROM staff
    WHERE entrance_year >= 1998 AND entrance_year <= 2000
```



この場合は範囲を指定するBETWEEN演算子で書き換えることができます。

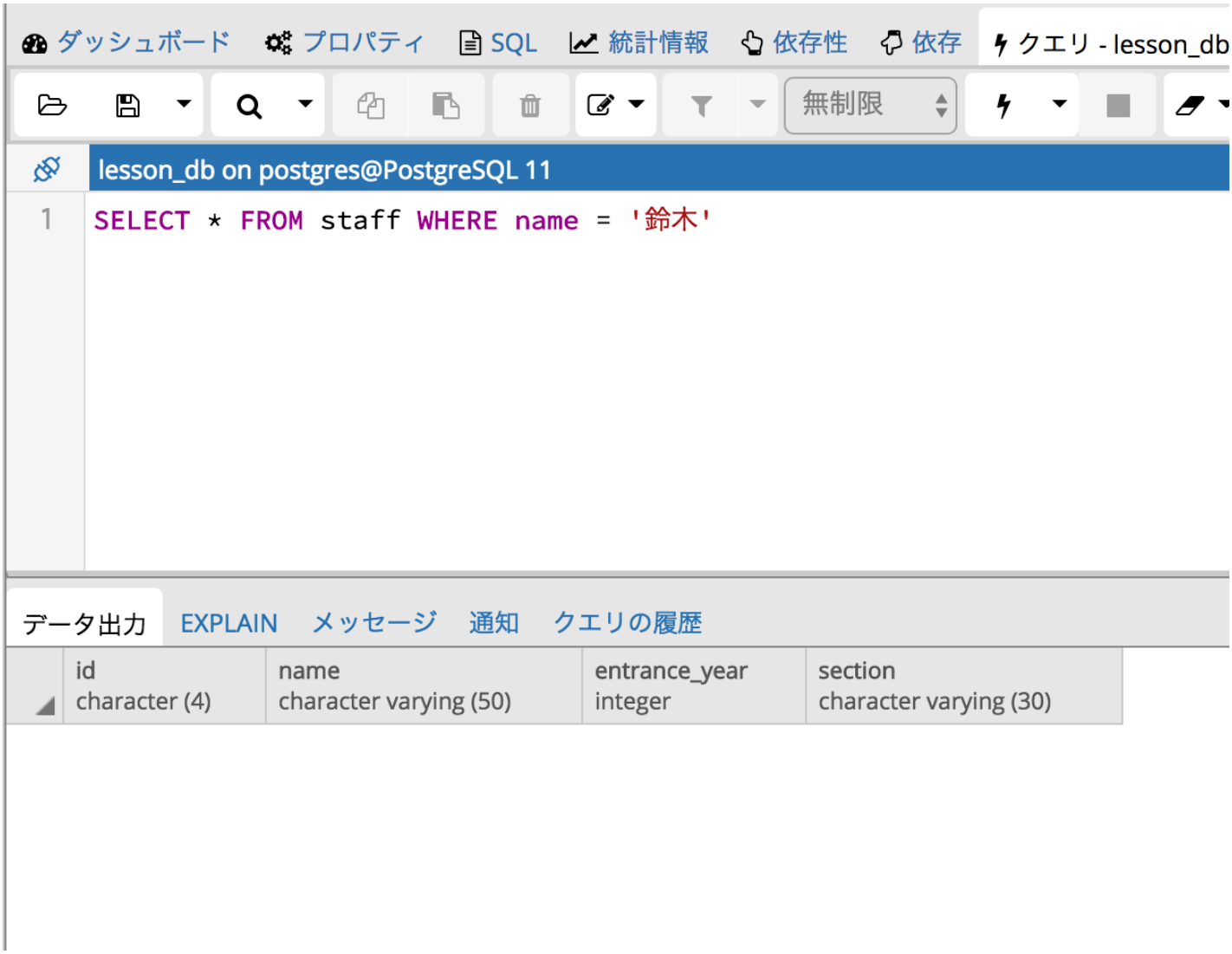
```
SELECT name FROM staff
    WHERE entrance_year BETWEEN 1998 AND 2000
```



LIKE演算子

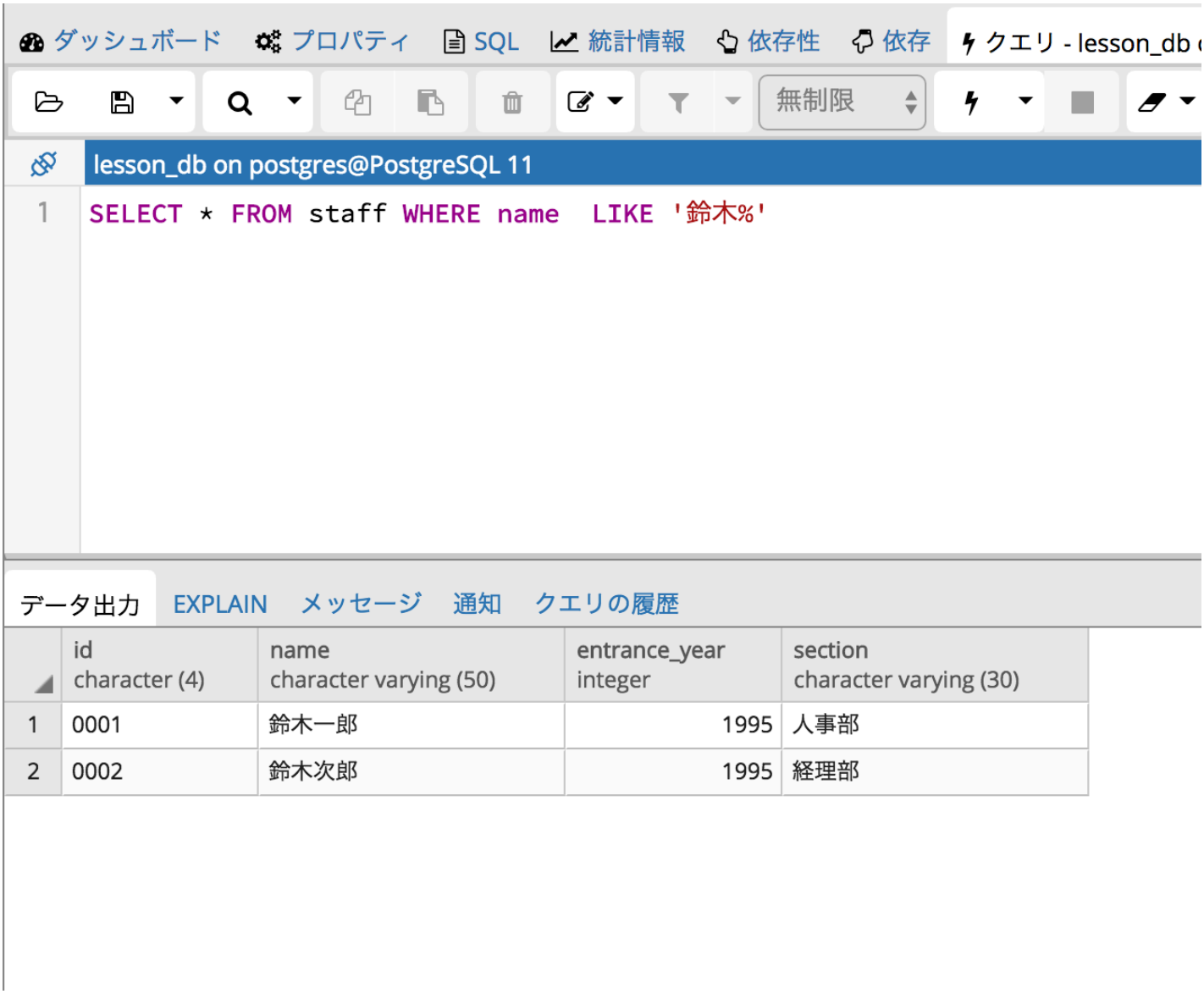
「鈴木」という苗字の全社員情報が知りたいです。どうしましょう。

```
SELECT * FROM staff WHERE name = '鈴木'
```



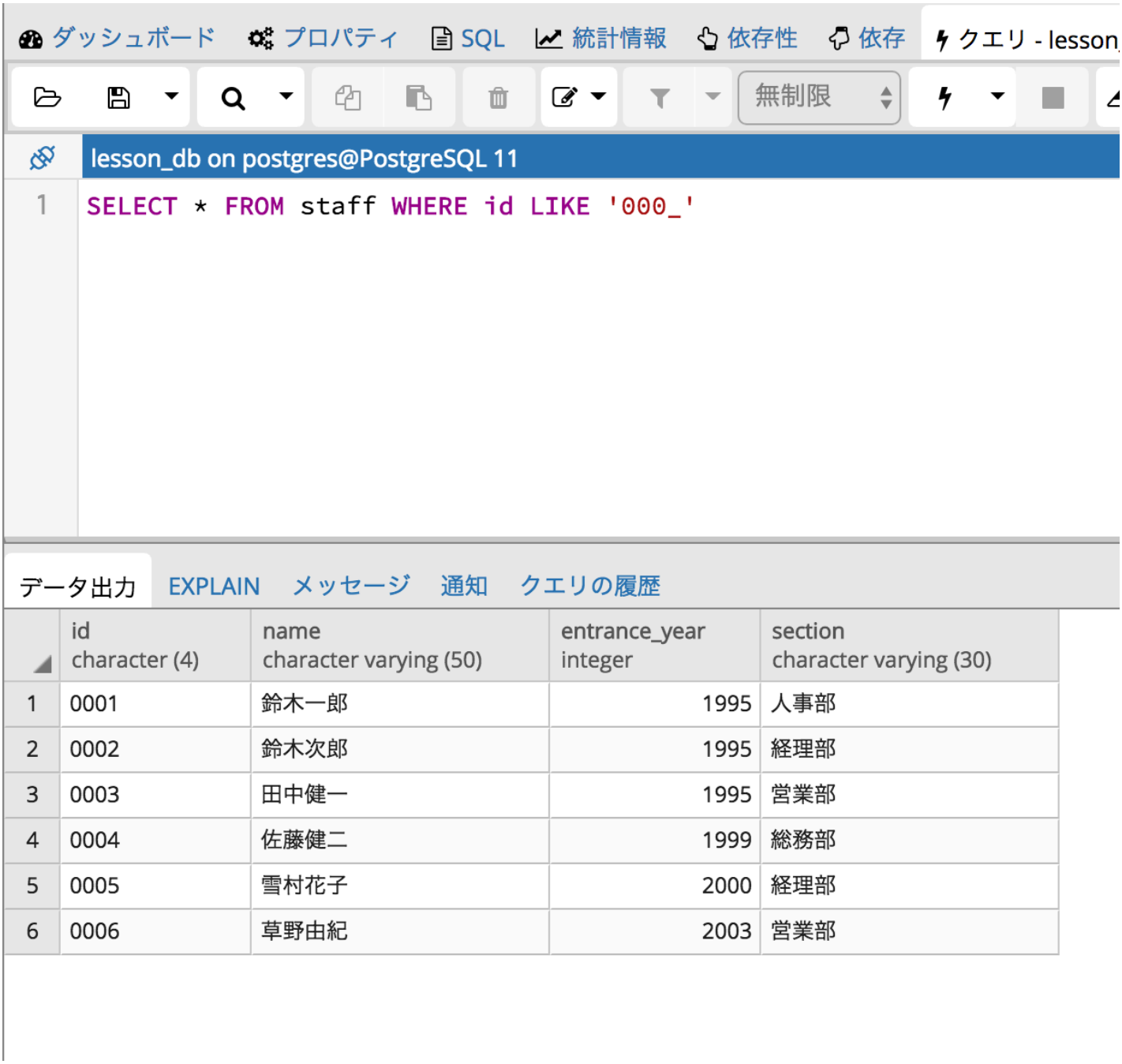
と記述した結果、一件もヒットしませんでした。「'鈴木一郎' = '鈴木'」ではないからです。
このような場合はLIKE演算子を使います。

```
SELECT * FROM staff WHERE name  LIKE '鈴木%'
```



今度は表示されましたね。
「％」はワイルドカードと言って任意の文字列を表します。
任意の一文字を表すワイルドカードは「_（アンダースコア）」です。

```
SELECT * FROM staff WHERE id LIKE '000_'
```



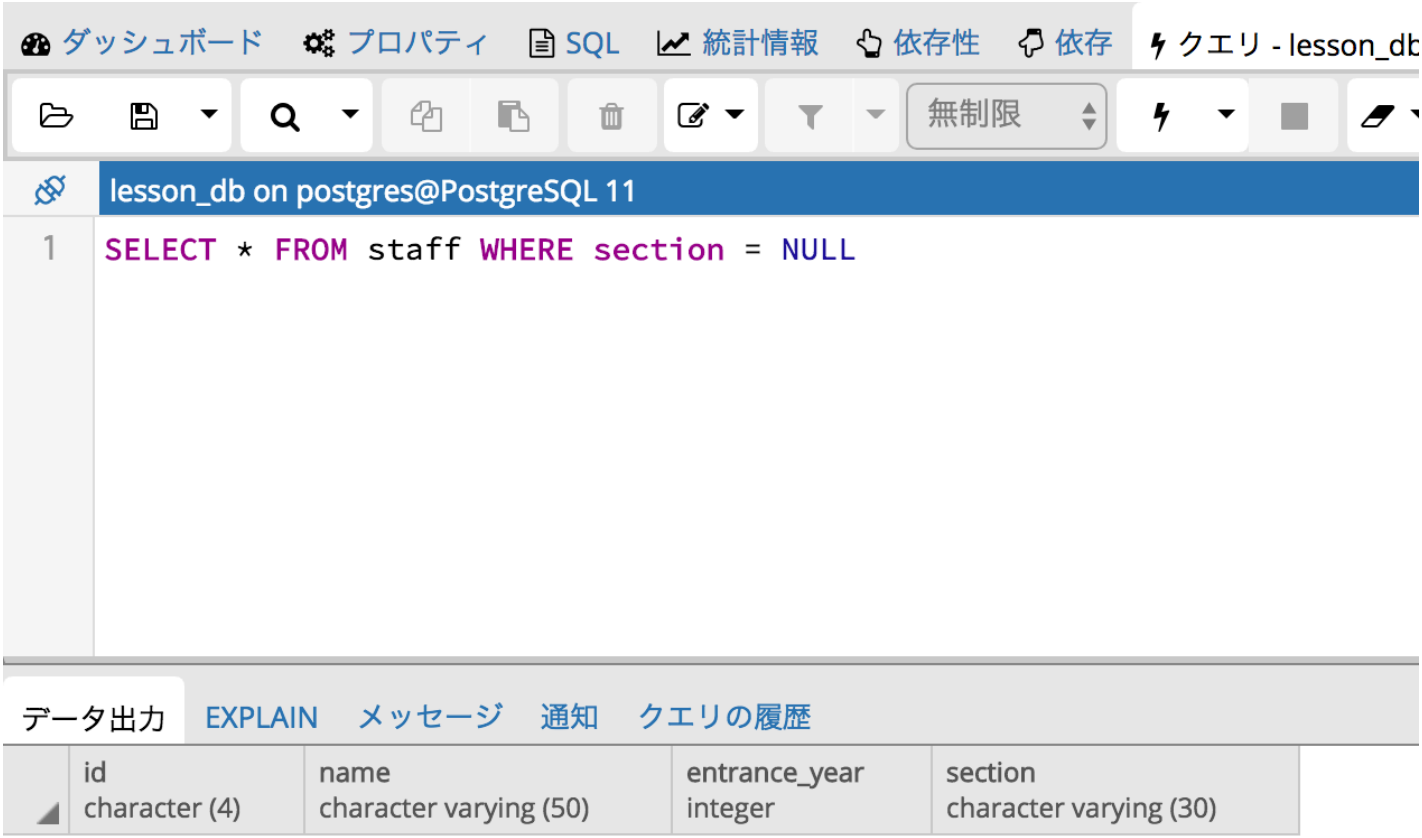
	id character (4)	name character varying (50)	entrance_year integer	section character varying (30)
1	0001	鈴木一郎	1995	人事部
2	0002	鈴木次郎	1995	経理部
3	0003	田中健一	1995	営業部
4	0004	佐藤健二	1999	総務部
5	0005	雪村花子	2000	経理部
6	0006	草野由紀	2003	営業部

ワイルドカードを使用した指定方法をパターンマッチ（ング）と言います。

IS NULL演算子

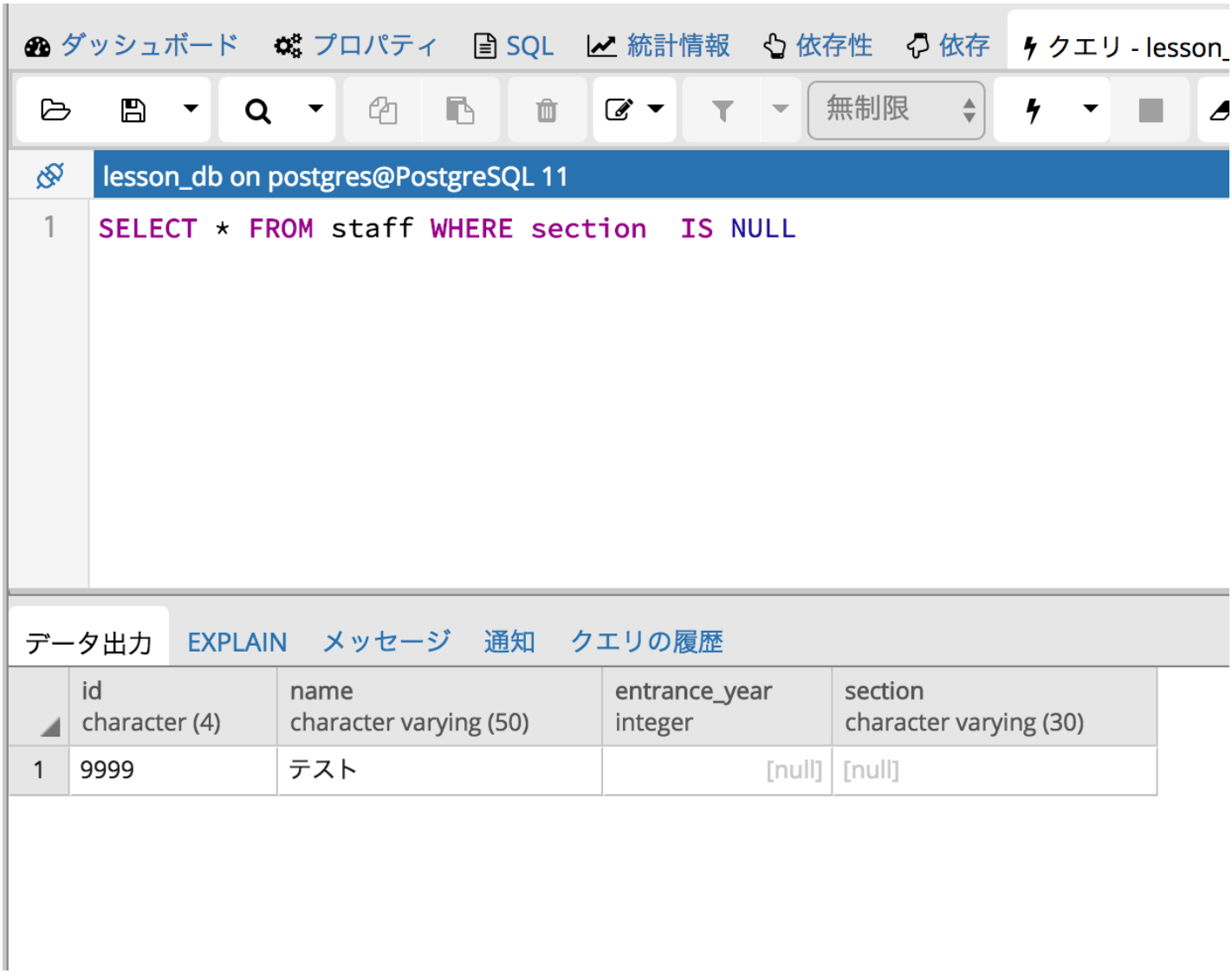
最後の演算子はNULLを判定する演算子です。 DBにもNULL値が存在します。
概念的には「値がない」か「未知な値」を意味します。 NULLに対して比較演算子は使えません
（比較を行なった場合、結果は全て偽になります）。

```
SELECT * FROM staff WHERE section = NULL
```



これではNULLを判定することはできません。IS NULL演算子を使用します。

```
SELECT * FROM staff WHERE section  IS NULL
```



NOT（否定）

ここまでさまざまなパターンのWHERE文を紹介してきましたが、どれも NOT を付け加えると逆の結果になります。 実際に実行して、先の結果と比較してみましょう。

```
SELECT name FROM staff
    WHERE NOT(section = '総務部' OR section = '経理部')

SELECT name FROM staff
    WHERE section NOT IN('総務部', '経理部')

SELECT name FROM staff
    WHERE NOT(entrance_year >= 1998 AND entrance_year <= 2000)

SELECT name FROM staff
    WHERE entrance_year NOT BETWEEN 1998 AND 2000

SELECT * FROM staff WHERE name NOT LIKE '鈴木%'

SELECT * FROM staff WHERE id NOT LIKE '000_'

SELECT * FROM staff WHERE section IS NOT NULL

  （ここだけNOT IS NULLでないことに気をつけてください。英語の文法通りです。）
```

テーブルの結合

今度は、 **合わせ技** です。
ここでは、よく使うテーブルの結合について紹介します。
DBからデータを取り出す際、「○○からXXを取り出す」といった選択を行っているかと思います。

そしてこのデータ取り出しの際、複数テーブルからデータを検索して取得するといったケースも多々あるかと思います。
ですが、例えばAとBというテーブルに対して、「Aを調べる⇒Bを調べる⇒２つの検索結果をがっちゃんこ」
なんてしてると時間がかかりますよね？

なので通常の検索の場合、「AとBのテーブルをまとめる⇒そこから検索」という方法を取ります。
この「 **まとめる** 」ことを「 **結合** 」と呼び、その方法として、今から取り扱う
内部結合や外部結合などといったものが存在しています。
テーブル結合について学んでいきましょう！

Step1：テーブルの結合

今回は、駅の名前と路線名を扱う２つのテーブル(stationテーブルと、routeテーブル)を扱いながら、
その結合方法に関して追っていきます。

まずはテーブルを準備しますので、下記のSQL文を実行してください。

```
CREATE TABLE station(
id SERIAL,
station_name varchar(50),
route_id integer);

CREATE TABLE route(
route_id integer not null,
route_name varchar(50));
```

これで *station*テーブルと、*route*テーブル ができました。
※上記のテーブルは、このページでのみ使用します。

続いて、データを挿入していきます。
下記のSQL文を実行してください。

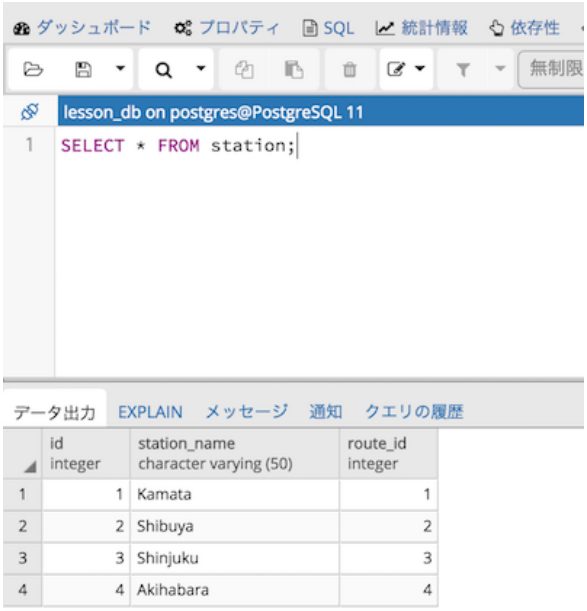
```
INSERT INTO station (station_name, route_id) VALUES ('Kamata', 1);
INSERT INTO station (station_name, route_id) VALUES ('Shibuya', 2);
INSERT INTO station (station_name, route_id) VALUES ('Shinjuku', 3);
INSERT INTO station (station_name, route_id) VALUES ('Akihabara', 4);

INSERT INTO route (route_id, route_name) VALUES (1, 'KehinTohokuSen');
INSERT INTO route (route_id, route_name) VALUES (2, 'YamanoteSen');
INSERT INTO route (route_id, route_name) VALUES (5, 'HibiyaSen');
```

これで、2つのテーブルにデータが挿入されました。

データが入っているか確認していきましょう。

【stationテーブル】

```
SELECT * FROM station;
```

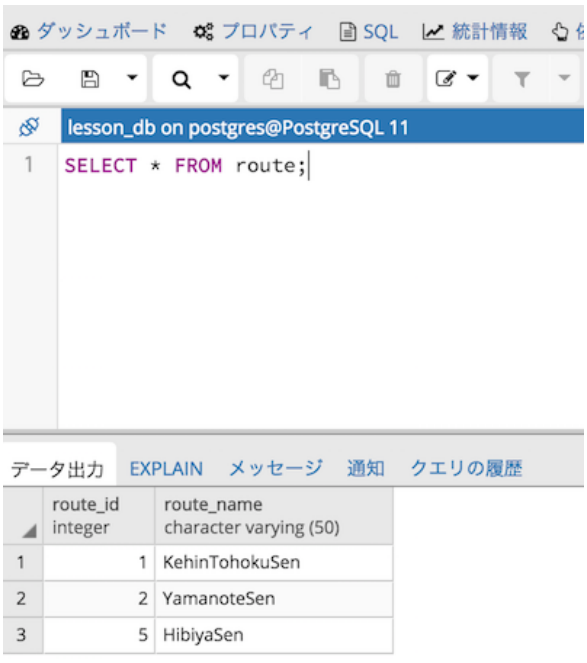


	id integer	station_name character varying (50)	route_id integer
1	1	Kamata	1
2	2	Shibuya	2
3	3	Shinjuku	3
4	4	Akihabara	4

続いて、

【routeテーブル】

```
SELECT * FROM route;
```



	route_id integer	route_name character varying (50)
1	1	KehinTohokuSen
2	2	YamanoteSen
3	5	HibiyaSen

データが入っていることも確認できたので、結合について学んでいきます！

内部結合

上でも言ったように、結合は、2つ以上のテーブルをくっつけることです。

SELECT文は知ってますね？
SELECT文で取得したデータの各レコード（行）の後ろにくっつけたいテーブルのデータを
くっつけます。

ただ、くっつけるにも条件が必要なので、この条件でくっつけてね。っていう書き方をします。

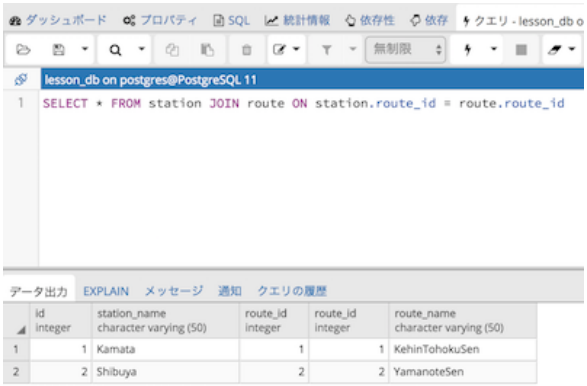
【基本構文】

```
SELECT * FROM 《メインのテーブル》
JOIN 《くっつけたいテーブル》 ON 《メインテーブルのカラム》 = 《くっつけたいテーブルのカラム》;
```

ちなみに、ONの後ろが条件です。

それでは実際のデータを使ってみてみましょう。

```
SELECT * FROM station JOIN route ON station.route_id = route.route_id
```



なんとなく使い方はわかりましたか？
JOINは内部結合といって、条件にあるstationテーブルのstation_idとrouteテーブルのroute_id
どちらもあるデータが取得されます。

なので、画像のようにどちらのテーブルにもないデータは取得されません。

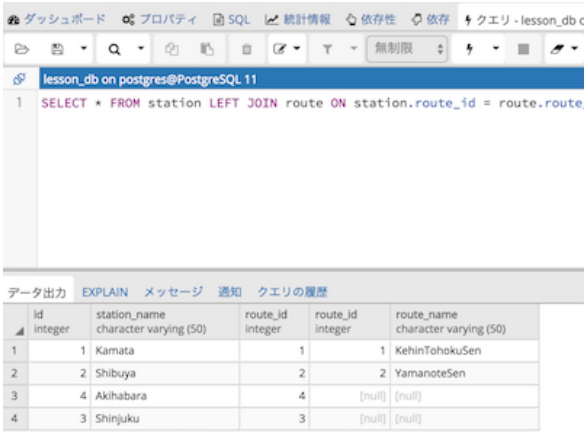
左外部結合

結合の基本はなんとなくわかったと思います。
左外部結合では *LEFT JOIN* を使います。

特徴は、メインのテーブルのデータは全部表示して、条件に対応した
くっつけたいテーブルのデータは空白で表示するということです。

実際に見ていきましょう。

```
SELECT * FROM station
LEFT JOIN route ON station.route_id = route.route_id
```



画像を見たら意味がわかるのではないでしょうか。
LEFT JOIN から見て左のテーブルは全部表示しますよ。という結合です。

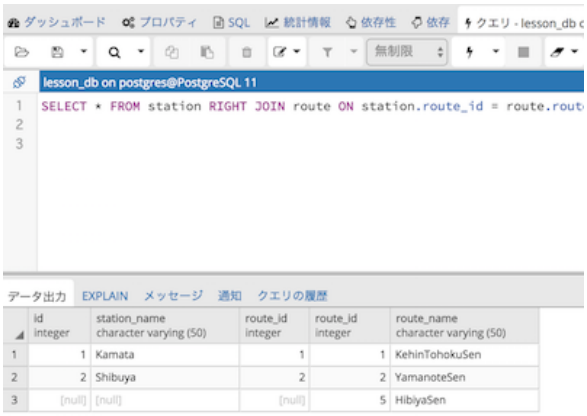
右外部結合

最後に、右外部結合について説明します。
右外部結合では *RIGHT JOIN* を使います。

これは、左外部結合の逆で、
RIGHT JOIN から見て右のテーブルは全部表示しますよ。という結合です。

実際に使っていきましょう。

```
SELECT * FROM station RIGHT JOIN route ON station.route_id = route.route_id
```



画像のように表示されましたでしょうか。

- 内部結合
- 左外部結合
- 右外部結合

しっかり理解していきましょう。

> **補足**

実は各JOINの名称は省略されており、正式名称ではありません。
今後、現場に出た際に正式名称に出くわす場面もあるかもしれないので、補足として付け加えておきます。

- JOIN
  - INNER JOIN
- OUTER JOIN
  - LEFT JOIN = LEFT OUTER JOIN
  - RIGHT JOIN = LEFT OUTER JOIN

`JOIN` の 正式な書き方は `INNER JOIN` となります。
実際のSQLを記述する際には、 `JOIN（or INNER JOIN）` どちらでも構いません。
「 **この結合は内部結合ですよ** 」という意味合いを含ませたい場合は、
`INNER JOIN` としておけば可読性という面で優しい作りかなと思います。

また、外部結合の `LEFT／RIGHT JOIN` も、
大きなくくりとして `OUTER JOIN` となっています。

こちらも同様に `LEFT／RIGHT JOIN（or LEFT／RIGHT OUTER JOIN）` のように記述可能です。
（冗長な書き方になるので、あまり見ない書き方かなと思います。

出くわした際には、混乱することが無いよう覚えておくとよいでしょう！

課題

提出課題はありませんので、一通り学習が終わったら次の章に進んで下さい。

最終更新日時: 2022年 09月 10日(土曜日) 07:31