

# 8-1-1 文字の扱いについて

- ・キーワードの大文字と小文字は区別しない
- ・文字や日付情報はシングルクォートで囲む

student/postgres@student

Query EditorQuery History

```
1 create table member (  
2  
3     id integer,  
4     名前 text,  
5     年齢 integer,  
6     部署 text  
7 );
```

※カラム名に、'は不要

student/postgres@student

Query EditorQuery History

```
1 insert into member  
2  
3 values(1, '鈴木', 25, '営業'),(2, '佐藤', 30, '開発'),(2, '田中', 25, '開発'),(4, '渡辺', 30, '総務'), (5, '後藤', 41, '営業'), (6, '渡辺', 25, '開発');  
4  
5  
6 select * from member;
```

Data Output

Explain

Messages

Notifications

	id integer	名前 text	年齢 integer	部署 text
1	1	鈴木	25	営業
2	2	佐藤	30	開発
3	2	田中	25	開発
4	4	渡辺	30	総務
5	5	後藤	41	営業
6	6	渡辺	25	開発

# 8-1-2 列（カラム）、テーブル、検索条件の指定

student/postgres@student

Query EditorQuery History

1 select \* from member where 部署 = '営業' and 年齢 >= 30;

Data Output

Explain

Messages

Notifications

	id integer	名前 text	年齢 integer	部署 text
1	5	後藤	41	営業

student/postgres@student

Query EditorQuery History

1 select now();

Data Output

Explain

Messages

Notifications

	now timestamp with time zone
1	2020-11-30 14:20:49.184765+09

### 8-1-3 ORDER BY

student/postgres@student

Query EditorQuery History

1 select 名前, 年齢 from member order by 年齢;

Data OutputExplainMessagesNotifications

	名前 text	年齢 integer
1	鈴木	25
2	田中	25
3	渡辺	25
4	佐藤	30
5	渡辺	30
6	後藤	41

student/postgres@student


Query EditorQuery History

1 select 名前, 年齢 from member where 部署 = '営業' order by 年齢 desc;

Data OutputExplainMessagesNotifications

	名前 text	年齢 integer
1	後藤	41
2	鈴木	25

order byは左から順に処理される。1つ目の並べ替えで同じ順のものがあれば、2つ目の並べ替えが適用される。

 student/postgres@student

[Query Editor](#)
[Query History](#)

---

1

```
select * from member order by 年齢 desc, id ASC;
```

---

[Data Output](#)
[Explain](#)
[Messages](#)
[Notifications](#)

---

	id integer	名前 text	年齢 integer	部署 text
1	5	後藤	41	営業
2	2	佐藤	30	開発
3	4	渡辺	30	総務
4	1	鈴木	25	営業
5	2	田中	25	開発
6	6	渡辺	25	開発

## 8-1-4 LIMIT と OFFSET

 student/postgres@student

[Query Editor](#)
[Query History](#)

---

1

```
select * from member order by id limit 1;
```

---

[Data Output](#)
[Explain](#)
[Messages](#)
[Notifications](#)

---

	id integer	名前 text	年齢 integer	部署 text
1	1	鈴木	25	営業

student/postgres@student

Query Editor Query History

1 select \* from member order by id limit 3 offset 1;

Data Output Explain Messages Notifications

	id integer	名前 text	年齢 integer	部署 text
1	2	佐藤	30	開発
2	2	田中	25	開発
3	4	渡辺	30	総務

抽出されたデータの1件目をスキップし、2件目以降から3個のデータを取り出している。

## 8-1-5 DISTINCT

student/postgres@student

Query Editor Query History

1 select \* from member;

Data Output Explain Messages Notifications

	id integer	名前 text	年齢 integer	部署 text
1	1	鈴木	25	営業
2	2	佐藤	30	開発
3	2	田中	25	開発
4	4	渡辺	30	総務
5	5	後藤	41	営業
6	6	渡辺	25	開発

student/postgres@student

Query EditorQuery History

1 select distinct 年齢 from member;

Data Output

Explain

Messages

Notifications

	年齢	
	integer	
1	30	
2	41	
3	25	

student/postgres@student

Query EditorQuery History

1 select distinct on (年齢) \* from member;

Data Output

Explain

Messages

Notifications

	id	名前	年齢	部署
	integer	text	integer	text
1	1	鈴木	25	営業
2	2	佐藤	30	開発
3	5	後藤	41	営業

on (列名) を省略すれば、重複除去および抽出対象が年齢となり  
on (年齢) \* と指定すれば、年齢に対してのみ重複除去して、全列を抽出する。

つまり、on (列名) を省略すると、重複除去・抽出の両方の対象となる。

また、DISTINCT ONでの実行結果はソートされる。  
ORDER BYを併用した場合は、ORDER BYが優先となる。

## 8-1-6 GROUP BY と Having

student/postgres@student

Query EditorQuery History

1 select 部署 from member group by 部署 ;|

Data OutputExplainMessagesNotifications

	部署 text
1	総務
2	開発
3	営業

集約関数を使用すると、列名を複数指定できる。

student/postgres@student

Query EditorQuery History

1 select 部署, sum(年齢) as total\_age from member where 年齢 >= 30 group by 部署 order by total\_age;

Data OutputExplainMessagesNotifications

	部署 text	total_age bigint
1	総務	30
2	開発	30
3	営業	41

処理の順番 : Where → GROUP BY → HAVING

student/postgres@student

Query EditorQuery History

1 select 部署, sum(年齢) as total\_age from member group by 部署 having max(年齢) <= 30;

Data Output

Explain

Messages

Notifications

	部署 text	total_age bigint	
1	総務	30	
2	開発	80	

部署でグループ化してから、最高年齢が30歳以下の部署に絞り、部署とsum(年齢)を取り出している。

## 8-1-7 副問い合わせ

SELECT（問い合わせ）→ サブセレクトと呼ばれる。



```
1 create table table_1(keyA integer, keyB text);
2 insert into table_1 values(1, 'X'),(2, 'Y');
3
4 create table table_2(keyA integer, keyB text);
5 insert into table_2 values(1, 'X'),(2, 'Y');
6
7 create table tableA(id integer, name text);
8 insert into tableA values(1, 'A'), (2, 'B');
9
10 create table tableB(id integer, name text);
11 insert into tableB values(1, 'A'), (2, 'B');
12
13 select * from table_1;
```

	keya integer	keyb text
1	1	X
2	2	Y

student/postgres@student

Query EditorQuery History

1 select st.keyA from (select keyA, keyB from table\_2) as st where keyB = 'X';

Data OutputExplainMessagesNotifications

	keyA integer	
1	1	

※副問い合わせの結果が2件以上になると、エラーとなるので注意すること。

student/postgres@student

Query EditorQuery History

1

select keyA from table\_1

2

where keyB = (select keyB from table\_2 where keyB = 'X');|

Data OutputExplainMessagesNotifications

keya  
integer

1

1

問い合わせ対象のテーブルが2つあるため、抽出結果も2つ分となる。

student/postgres@student

Query EditorQuery History

1

select a.id

2

from (select id from tableA) as a, (select id from tableB) as b;

Data OutputExplainMessagesNotifications

	id	
	integer	
1	1	
2	1	
3	2	
4	2	

副問い合わせが2個以上ある場合には、エイリアス（as でつける別名）が必要となる。

## 8-1-7 IN と NOT IN

student/postgres@student

Query EditorQuery History

1 select \* from member where 部署 in ('営業');

Data Output	Explain	Messages	Notifications
<div><div></div><div>idinteger</div><div>名前text</div><div>年齢integer</div><div>部署text</div></div>			
1	1 鈴木	25 営業	
2	5 後藤	41 営業	

student/postgres@student

Query EditorQuery History

1 select \* from member where 部署 not in ('開発', '営業');

Data Output	Explain	Messages	Notifications
<div><div></div><div>idinteger</div><div>名前text</div><div>年齢integer</div><div>部署text</div></div>			
1	4 渡辺	30 総務	

## 8-1-9 ANY

IN と同じ働きだが、副問い合わせの結果が必ず1列となる。

student/postgres@student

Query Editor   Query History

1   `select * from member where 部署 = any (select 部署 from member where id = 2);`

Data Output   Explain   Messages   Notifications

	id integer	名前 text	年齢 integer	部署 text
1	2	佐藤	30	開発
2	2	田中	25	開発
3	6	渡辺	25	開発

## 8-1-10   BETWEEN

student/postgres@student

Query Editor   Query History

1   `select * from member where id between 1 and 3;`

Data Output   Explain   Messages   Notifications

	id integer	名前 text	年齢 integer	部署 text
1	1	鈴木	25	営業
2	2	佐藤	30	開発
3	2	田中	25	開発

## 8-1-11   結合

student/postgres@student

Query EditorQuery History

1 create table users (id integer, 名前 text, 部署id integer);  
2 insert into users values(1, '鈴木', 2), (2, '佐藤', 3), (3, '田中', 4), (4, '渡辺', 5);  
3  
4 create table dept (部署id integer, 部署 text);  
5 insert into dept values(1, '総務'), (2, '企画'), (3, '営業'), (4, '開発');  
6  
7 select \* from users;

Data Output

Explain

Messages

Notifications

	id integer	名前 text	部署id integer
1	1	鈴木	2
2	2	佐藤	3
3	3	田中	4
4	4	渡辺	5

student/postgres@student

Query EditorQuery History

1 select \* from dept;

Data Output


Explain

Messages

Notifications

	部署id integer	部署 text
1	1	総務
2	2	企画
3	3	営業
4	4	開発

## INNER JOIN

 student/postgres@student

[Query Editor](#)
[Query History](#)

---

```


1  select *
2  from users inner join dept on users.部署id=dept.部署id;

```

---

[Data Output](#)
[Explain](#)
[Messages](#)
[Notifications](#)

	id integer	名前 text	部署id integer	部署id integer	部署 text
1	1	鈴木	2	2	企画
2	2	佐藤	3	3	営業
3	3	田中	4	4	開発

 student/postgres@student

[Query Editor](#)
[Query History](#)

---

```

1  select *
2  from users inner join dept using(部署id);

```

---

[Data Output](#)
[Explain](#)
[Messages](#)
[Notifications](#)

	部署id integer	id integer	名前 text	部署 text
1	2	1	鈴木	企画
2	3	2	佐藤	営業
3	4	3	田中	開発

NATURALを用いた場合、結合に使用された列は一番左の列としてまとめられる。



 student/postgres@student

Query EditorQuery History

1 select \*  
2 from users natural inner join dept;

Data Output


Explain

Messages

Notifications

	部署id integer	id integer	名前 text	部署 text
1	2	1	鈴木	企画
2	3	2	佐藤	営業
3	4	3	田中	開発

CROSS JOIN

 student/postgres@student

Query EditorQuery History

1 select \*  
2 from users cross join dept;

Data Output

Explain

Messages

Notifications

	id integer	名前 text	部署id integer	部署id integer	部署 text
1	1	鈴木	2	1	総務
2	2	佐藤	3	1	総務
3	3	田中	4	1	総務
4	4	渡辺	5	1	総務
5	1	鈴木	2	2	企画
6	2	佐藤	3	2	企画
7	3	田中	4	2	企画

右から2番目の部署idを見ると、全ての組み合わせが結合されていることが分かりやすい。

student/postgres@student

Query EditorQuery History

1

```
select *
from users cross join dept
where users.部署id=2;
```

Data Output

Explain

Messages

Notifications

	<div>id</div> <div>integer</div>	<div>名前</div> <div>text</div>	<div>部署id</div> <div>integer</div>	<div>部署id</div> <div>integer</div>	<div>部署</div> <div>text</div>
1	1	鈴木	2	1	総務
2	1	鈴木	2	2	企画
3	1	鈴木	2	3	営業
4	1	鈴木	2	4	開発

LEFT OUTER JOIN

student/postgres@student

Query EditorQuery History

1

```
select *
from users left outer join dept on users.部署id=dept.部署id;
```

Data Output

Explain

Messages

Notifications

	<div>id</div> <div>integer</div>	<div>名前</div> <div>text</div>	<div>部署id</div> <div>integer</div>	<div>部署id</div> <div>integer</div>	<div>部署</div> <div>text</div>
1	1	鈴木	2	2	企画
2	2	佐藤	3	3	営業
3	3	田中	4	4	開発
4	4	渡辺	5	[null]	[null]

RIGHT OUTER JOIN

student/postgres@student

Query Editor Query History

```

1 select *
2 from users right outer join dept on users.部署id=dept.部署id;

```

Data Output

Explain

Messages

Notifications

	id integer	名前 text	部署id integer	部署id integer	部署 text
1	[null]	[null]	[null]	1	総務
2	1	鈴木	2	2	企画
3	2	佐藤	3	3	営業
4	3	田中	4	4	開発

## FULL OUTER JOIN

student/postgres@student

Query Editor Query History

```

1 select *
2 from users full outer join dept on users.部署id=dept.部署id;

```

Data Output

Explain

Messages

Notifications

	id integer	名前 text	部署id integer	部署id integer	部署 text
1	[null]	[null]	[null]	1	総務
2	1	鈴木	2	2	企画
3	2	佐藤	3	3	営業
4	3	田中	4	4	開発
5	4	渡辺	5	[null]	[null]

## 8-1-12 EXISTS と NOT EXISTS

student/postgres@student

Query EditorQuery History

1 select \* from users where exists (select \* from dept);

Data Output

Explain

Messages

Notifications

	id integer	名前 text	部署id integer
1	1	鈴木	2
2	2	佐藤	3
3	3	田中	4
4	4	渡辺	5

student/postgres@student

Query EditorQuery History

1 select \* from users where exists (select 1 from dept where users.部署id=dept.部署id);

Data Output

Explain


Messages

Notifications

	id integer	名前 text	部署id integer
1	1	鈴木	2
2	2	佐藤	3
3	3	田中	4

EXISTSは行を返すかどうかの指定しかしないため、副問い合わせのselectでは何を指定しても問題ない。副問い合わせ内の条件に合致するものがあれば、usersの中から抽出するだけである。

逆にNOT EXISTSは、条件に合致しないものを、usersの中から抽出する。

 student/postgres@student

Query Editor
Query History

---

1
select \* from users where not exists (select 1 from dept where users.部署id=dept.部署id);

---

Data Output
Explain
Messages
Notifications

	id integer	名前 text	部署id integer
1	4	渡辺	5

8-1-13 UNION/EXCEPT/INTERSECT

 student/postgres@student

Query Editor
Query History

---

1 create table tbl1 (id integer, name text);
2 insert into tbl1 values(1, 'AAA'), (2, 'BBB'), (3, 'CCC');
3 select \* from tbl1;

---

Data Output
Explain
Messages
Notifications

	id integer	name text
1	1	AAA
2	2	BBB
3	3	CCC

student/postgres@student

Query EditorQuery History

```
1 create table tbl2 (id integer, name text);
2 insert into tbl2 values(2, 'BBB'), (3, 'CCC'), (4, 'DDD');
3 select * from tbl2;
```

Data OutputExplainMessagesNotifications

	id integer	name text
1	2	BBB
2	3	CCC
3	4	DDD

UNIONした結果に対して、ORDER BYを実行している。

student/postgres@student

Query EditorQuery History

```
1 select * from tbl1 union select * from tbl2 order by id;
```

Data OutputExplainMessagesNotifications

	id integer	name text
1	1	AAA
2	2	BBB
3	3	CCC
4	4	DDD

ALLを付与すると、重複行がそのまま出力される。

student/postgres@student

Query EditorQuery History

1 select \* from tbl1 union all select \* from tbl2 order by id;

Data OutputExplainMessagesNotifications

	id	name
	integer	text
1	1	AAA
2	2	BBB
3	2	BBB
4	3	CCC
5	3	CCC
6	4	DDD

EXCEPTは、2つのテーブルのうち、左側のテーブルにあるデータだけを抽出する。

student/postgres@student

Query EditorQuery History

1 select \* from tbl1 except select \* from tbl2 order by id;

Data OutputExplainMessagesNotifications

	id	name
	integer	text
1	1	AAA

INTERSECTは、2つのテーブルのうち、両方とも存在するデータのみを抽出する。

student/postgres@student

Query EditorQuery History

1 select \* from tbl1 intersect select \* from tbl2 order by id;

Data OutputExplainMessagesNotifications

	id	name
	integer	text
1	1	AAA

UNION／EXCEPT／INTERSECTを同時に使用した場合、最優先となるのはINTERSECTとなり  
それ以外は左から順に処理される。