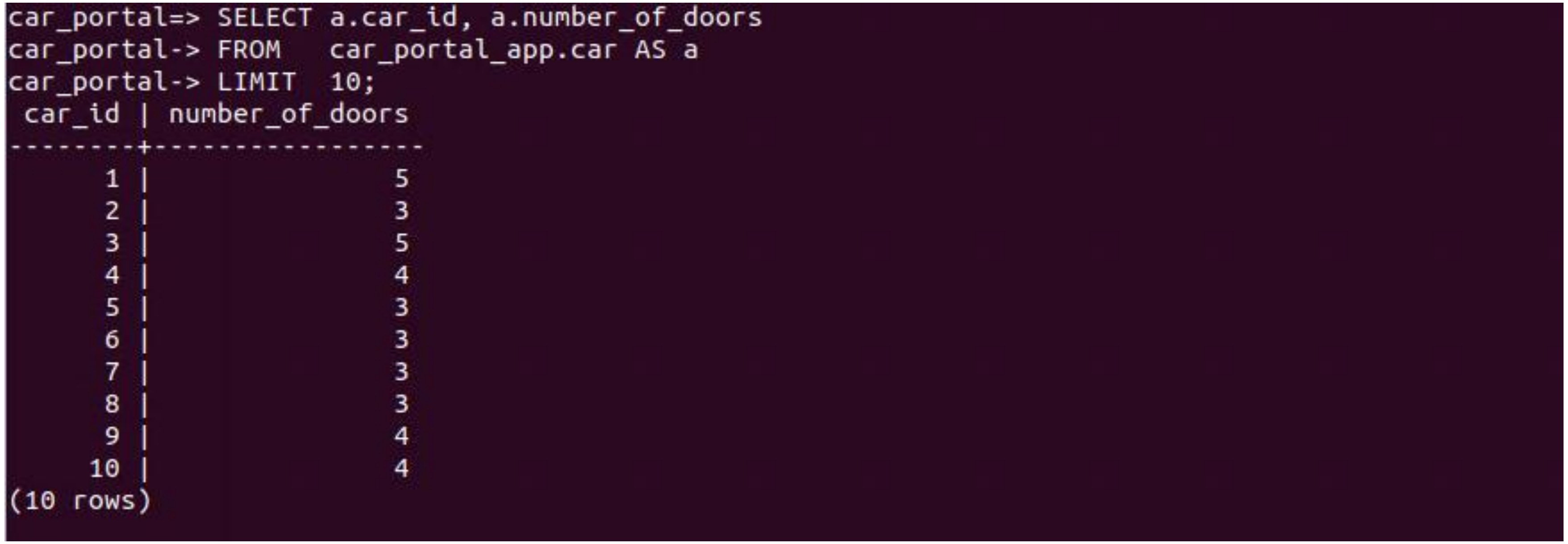
Querying Data with SELECT Statements

Part 2

The FROM Clause

- The source of the rows for a query is specified after the `FROM` keyword.
- It's called the `FROM` clause.
- The query can select rows from zero, one, or more sources.
- When no source is specified, the `FROM` keyword should be omitted.
- A source of the rows for a query can be any combination of the following:
    - Tables
    - Views
    - Functions
    - Subqueries
    - VALUES clauses
- When multiple sources are specified, they should be separated by a comma or the `JOIN` clause should be used.

- It's possible to set aliases for tables in the `FROM` clause.
- The optional `AS` keyword is used for that:

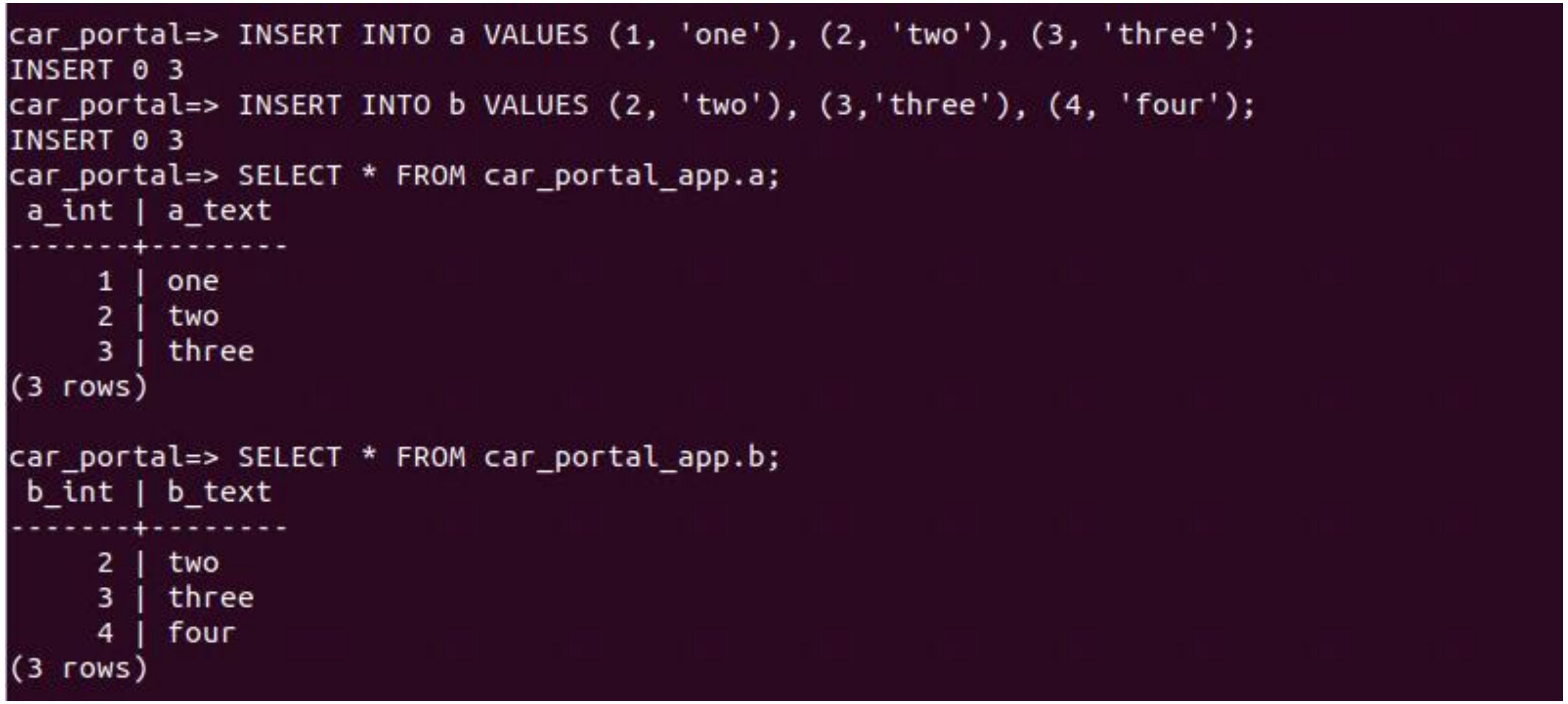


```
car_portal=> SELECT a.car_id, a.number_of_doors
car_portal-> FROM   car_portal_app.car AS a
car_portal-> LIMIT  10;
 car_id | number_of_doors
--------+-----------------
      1 |               5
      2 |               3
      3 |               5
      4 |               4
      5 |               3
      6 |               3
      7 |               3
      8 |               3
      9 |               4
     10 |               4
(10 rows)
```

- If an alias is used for a table or view in the `FROM` clause, in the SELECT-list (or anywhere else), it's no longer possible to refer to the table by its name.
- Subqueries, when used in the `FROM` clause, must have an alias.
- Aliases are often used when a self-join is performed, which means using the same table several times in the `FROM` clause.

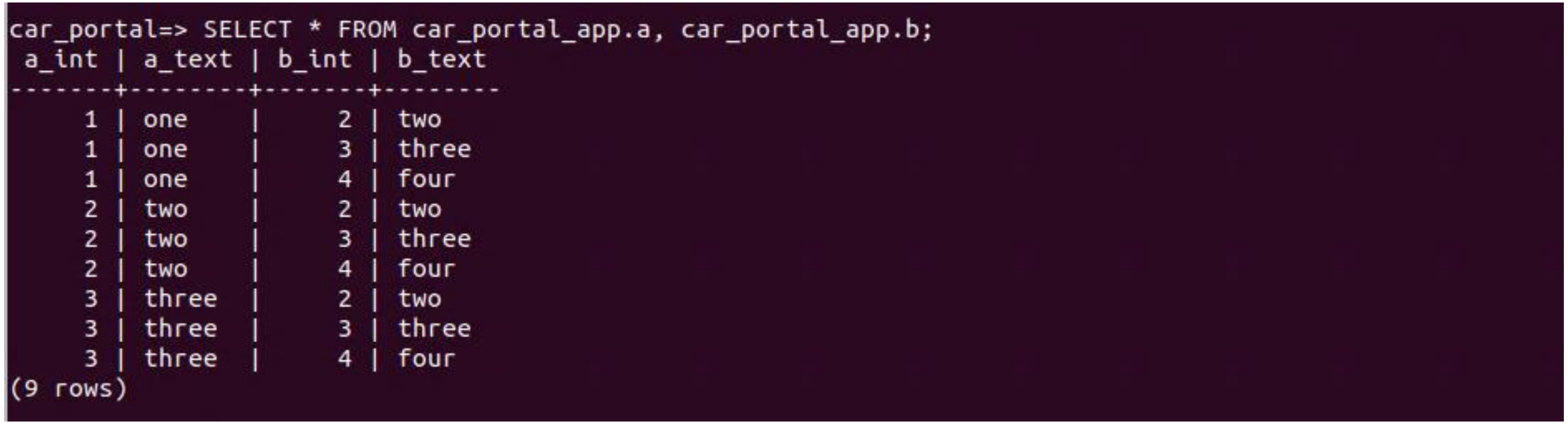
Selecting From Multiple Tables

- It's possible to select records from several sources at a time.
- Consider the following examples.
- There are two tables, each with three rows:



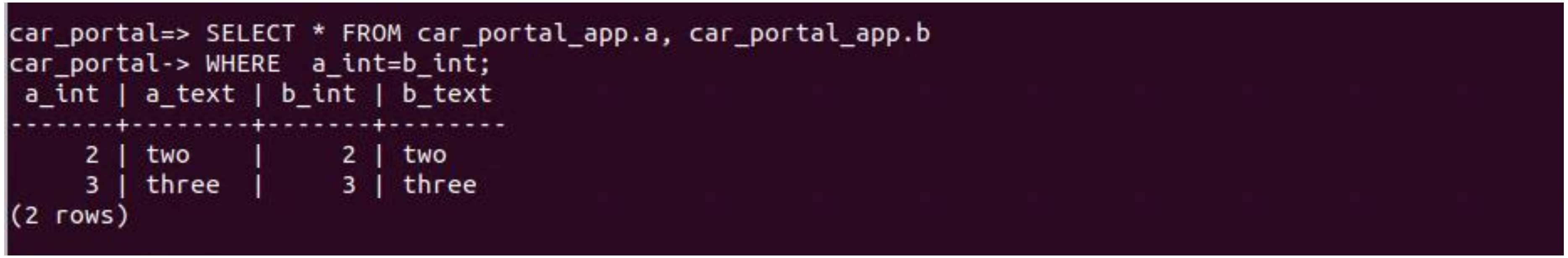
```
car_portal=> INSERT INTO a VALUES (1, 'one'), (2, 'two'), (3, 'three');
INSERT 0 3
car_portal=> INSERT INTO b VALUES (2, 'two'), (3,'three'), (4, 'four');
INSERT 0 3
car_portal=> SELECT * FROM car_portal_app.a;
 a_int | a_text
-------+--------
     1 | one
     2 | two
     3 | three
(3 rows)

car_portal=> SELECT * FROM car_portal_app.b;
 b_int | b_text
-------+--------
     2 | two
     3 | three
     4 | four
(3 rows)
```

- When records are selected from both of them, we get all the possible combinations of all their rows:



```
car_portal=> SELECT * FROM car_portal_app.a, car_portal_app.b;
 a_int | a_text | b_int | b_text
-------+--------+-------+--------
     1 | one    |     2 | two
     1 | one    |     3 | three
     1 | one    |     4 | four
     2 | two    |     2 | two
     2 | two    |     3 | three
     2 | two    |     4 | four
     3 | three  |     2 | two
     3 | three  |     3 | three
     3 | three  |     4 | four
(9 rows)
```

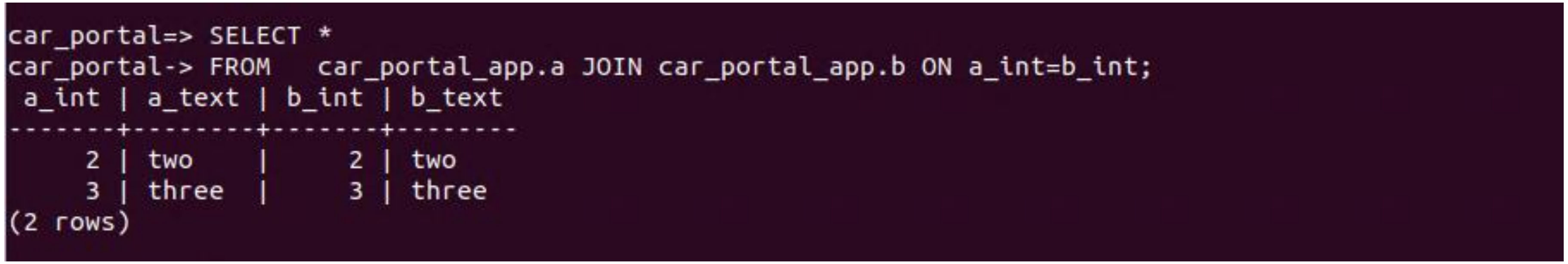
- All of the possible combinations of records from several tables is called a **Cartesian product** and, in many cases, it doesn't make much sense.
- In most cases, the user is interested in certain combinations of rows, when rows from one table match rows from another table based on some criteria.
- For example, it may be necessary to select only the combinations when the integer fields of both the tables have equal values.
- To get this, the query should be changed:



```
car_portal=> SELECT * FROM car_portal_app.a, car_portal_app.b
car_portal-> WHERE  a_int=b_int;
 a_int | a_text | b_int | b_text
-------+--------+-------+--------
     2 | two    |     2 | two
     3 | three  |     3 | three
(2 rows)
```

- The `a_int = b_int` condition joins the tables.
- The joining conditions could be specified in the `WHERE` clause, but in most cases, it's better to put them into the `FROM` clause to make it explicit that they are there for joining and not for filtering the result of the join, though there is no formal difference.

- The JOIN keyword is used to add join conditions to the FROM clause.
- The following query has the same logic and the same results as the previous one:



```
car_portal=> SELECT *
car_portal-> FROM   car_portal_app.a JOIN car_portal_app.b ON a_int=b_int;
 a_int | a_text | b_int | b_text
-------+--------+-------+--------
     2 | two    |     2 | two
     3 | three  |     3 | three
(2 rows)
```

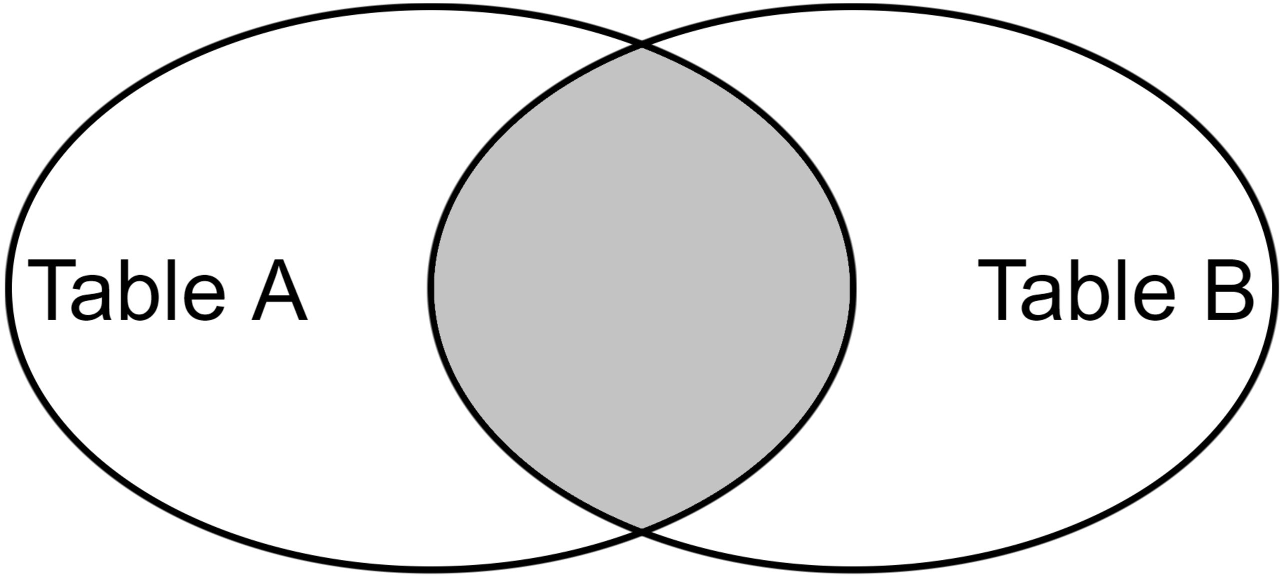
- The `JOIN` condition can be specified in any of the following three ways:
  - Using the `ON` keyword:
    `<first table> JOIN <second table> ON <condition>`
    The condition can be any SQL expression that returns a Boolean result. It isn't even necessary to include fields of the joined tables in the condition.
  - Using the `USING` keyword:
    `<first table> JOIN <second table> USING (<field list>)`
    The join is based on the equality of all the fields specified in the comma-separated `<field list>`. The fields should exist in both tables with the same name. So this syntax may be not flexible enough.
  - Performing a `NATURAL JOIN`:
    `<first table> NATURAL JOIN <second table>`
    Here, the join is based on the equality of all the fields that have the same name in both tables.

- Usage of the `USING` or `NATURAL JOIN` syntax has a drawback that is similar to the usage of `*` in the SELECT-list.
- It's possible to change the structure of the tables, for example, by adding another column or renaming them, in a way that does not make the query invalid, but changes the logic of the query.
- This will cause errors that are very difficult to find.

- What if not all of the rows from the first table can be matched to rows in the second table?

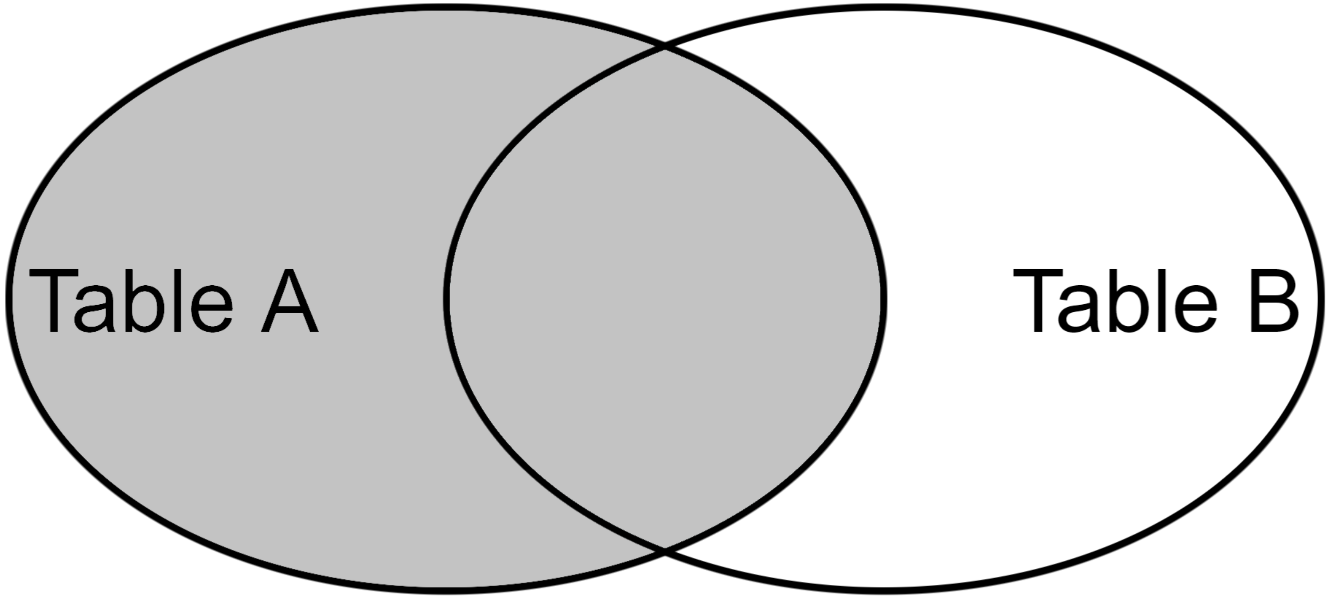
- In our example, only rows with integer values of `2` and `3` exist in both tables.
- When we join on the `a_int=b_int` condition, only those two rows are selected from the tables.
- The rest of the rows are not selected.
- This kind of join is called an **inner join**.

- It is shown as the filled area in the following diagram:

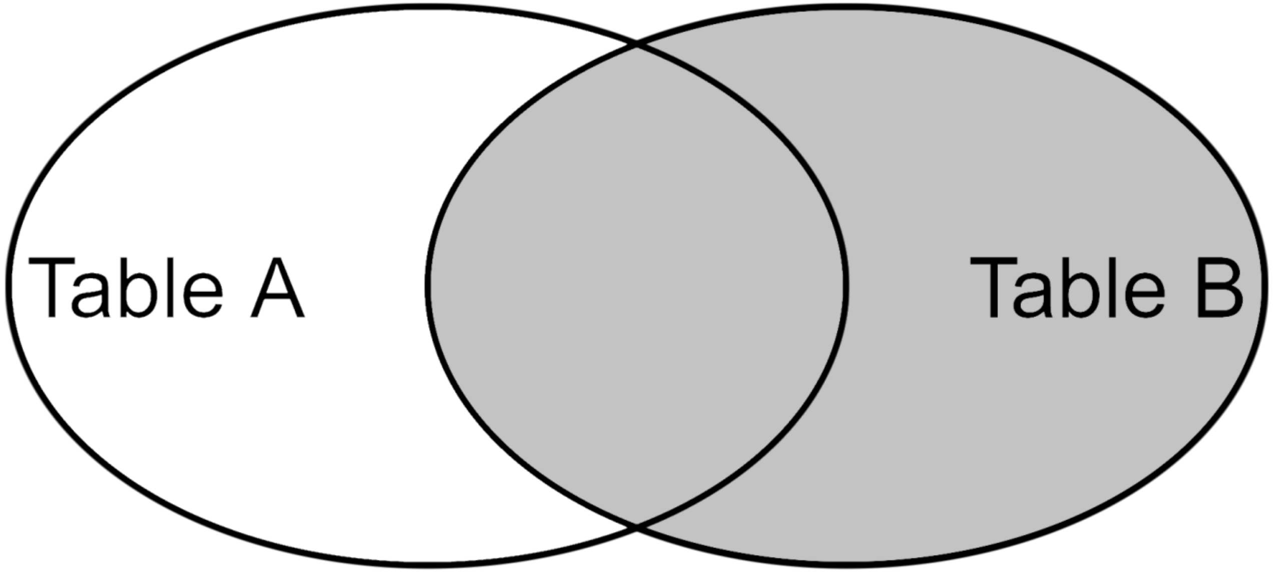


- When all the records from one table are selected, regardless of the existence of matching records in the other table, it's called an **outer join**.

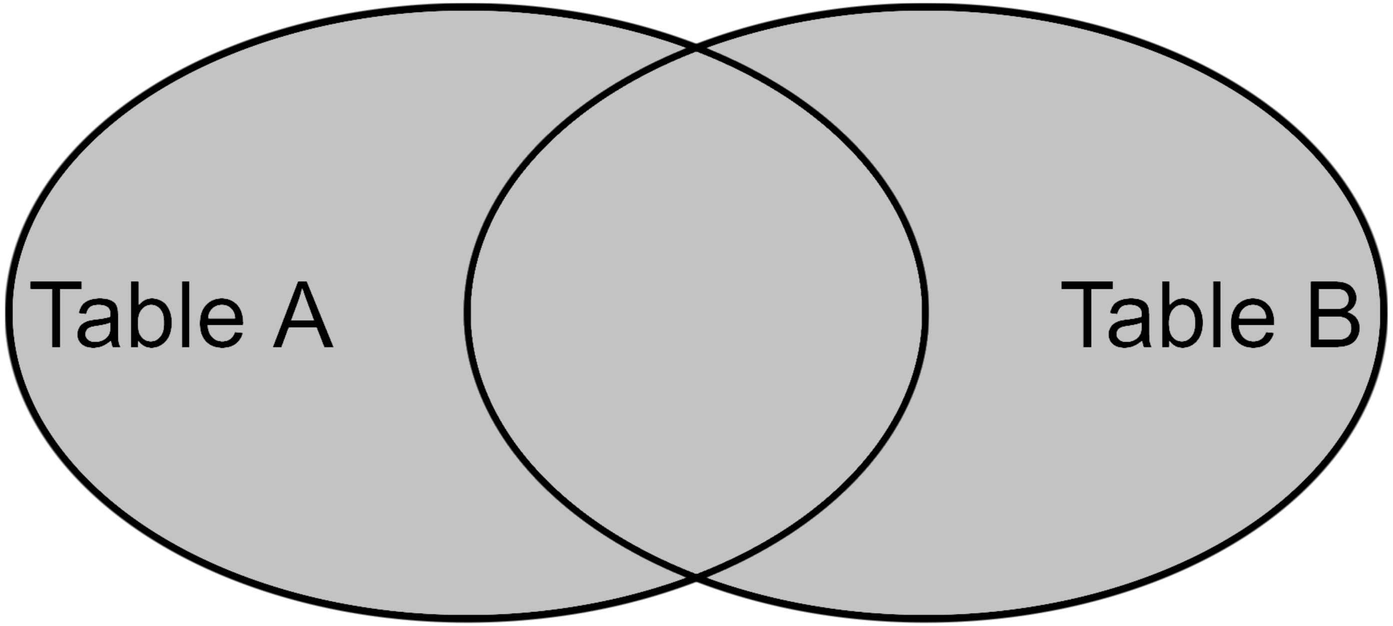
- If all of the records are selected from the first table, along with only those records that match the joining condition from the second, it's a **left outer join**:



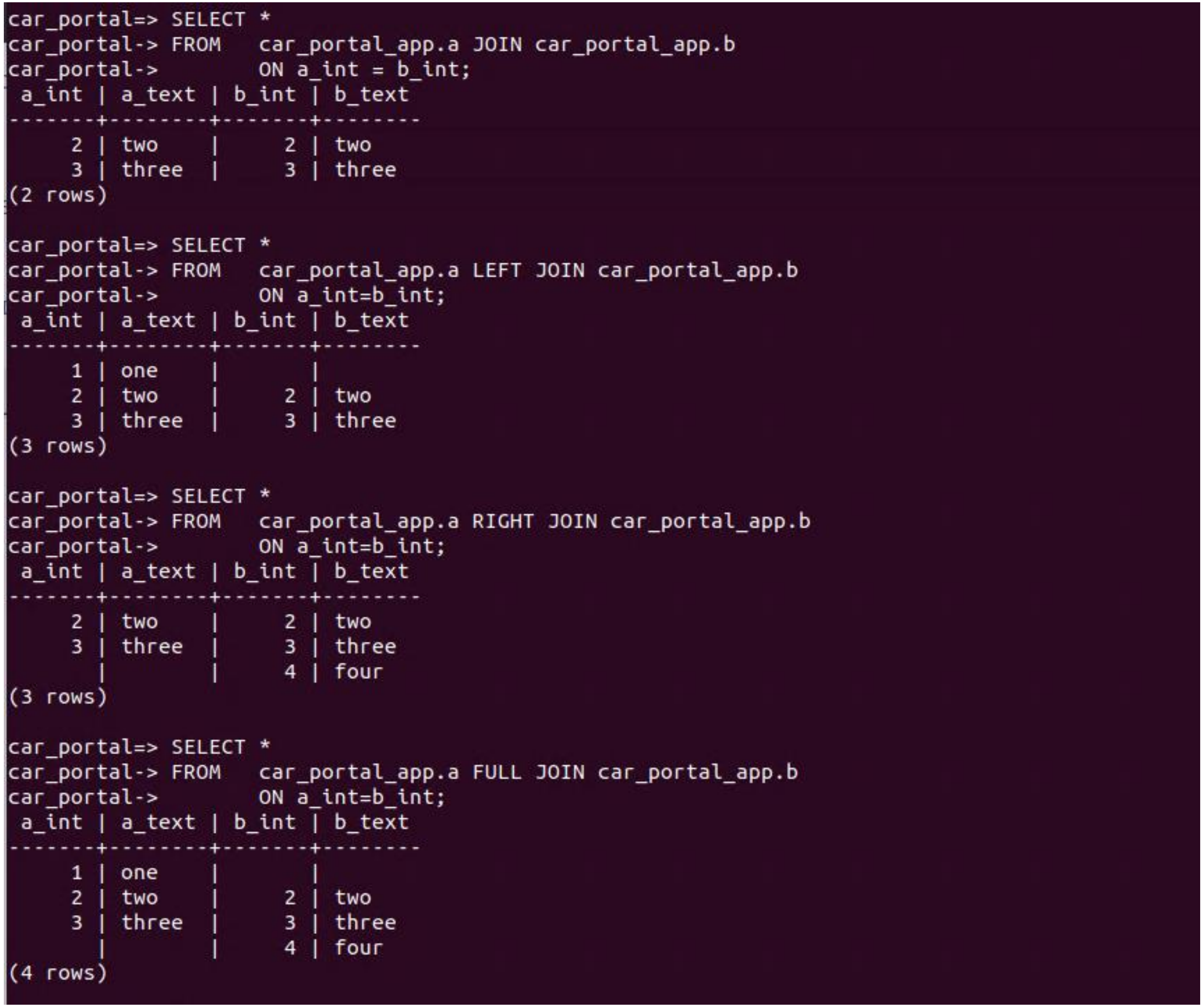
- When all records from the second table are selected, along with only the matching records from the first table, it's a **right outer join**:



- And when all the records from both tables are selected, it's a **full outer join**:

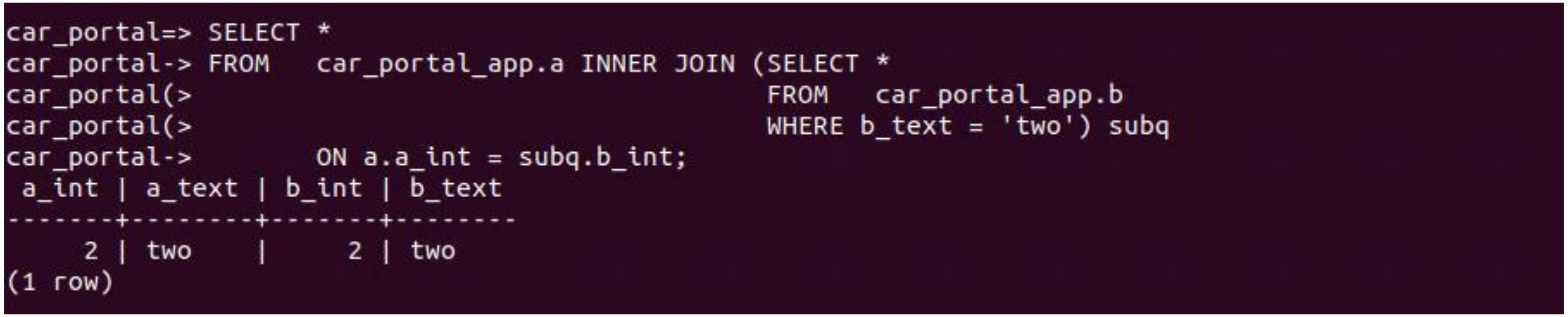


- In SQL syntax, the words INNER and OUTER are optional.



```
car_portal=> SELECT *
car_portal-> FROM   car_portal_app.a JOIN car_portal_app.b
car_portal->        ON a_int = b_int;
 a_int | a_text | b_int | b_text
-------+--------+-------+--------
     2 | two    |     2 | two
     3 | three  |     3 | three
(2 rows)

car_portal=> SELECT *
car_portal-> FROM   car_portal_app.a LEFT JOIN car_portal_app.b
car_portal->        ON a_int=b_int;
 a_int | a_text | b_int | b_text
-------+--------+-------+--------
     1 | one    |       |
     2 | two    |     2 | two
     3 | three  |     3 | three
(3 rows)

car_portal=> SELECT *
car_portal-> FROM   car_portal_app.a RIGHT JOIN car_portal_app.b
car_portal->        ON a_int=b_int;
 a_int | a_text | b_int | b_text
-------+--------+-------+--------
     2 | two    |     2 | two
     3 | three  |     3 | three
       |        |     4 | four
(3 rows)

car_portal=> SELECT *
car_portal-> FROM   car_portal_app.a FULL JOIN car_portal_app.b
car_portal->        ON a_int=b_int;
 a_int | a_text | b_int | b_text
-------+--------+-------+--------
     1 | one    |       |
     2 | two    |     2 | two
     3 | three  |     3 | three
       |        |     4 | four
(4 rows)
```

- As it's possible to query not only tables but also views, functions, and subqueries, it's also possible to join them using the same syntax as when joining tables:



```
car_portal=> SELECT *
car_portal-> FROM   car_portal_app.a INNER JOIN (SELECT *
car_portal(>                                    FROM   car_portal_app.b
car_portal(>                                    WHERE b_text = 'two') subq
car_portal->        ON a.a_int = subq.b_int;
 a_int | a_text | b_int | b_text
-------+--------+-------+--------
     2 | two    |     2 | two
(1 row)
```

- It's also possible to join more than two tables.
- In fact, every `JOIN` clause joins all the tables before the `JOIN` keyword with the table right after the keyword.

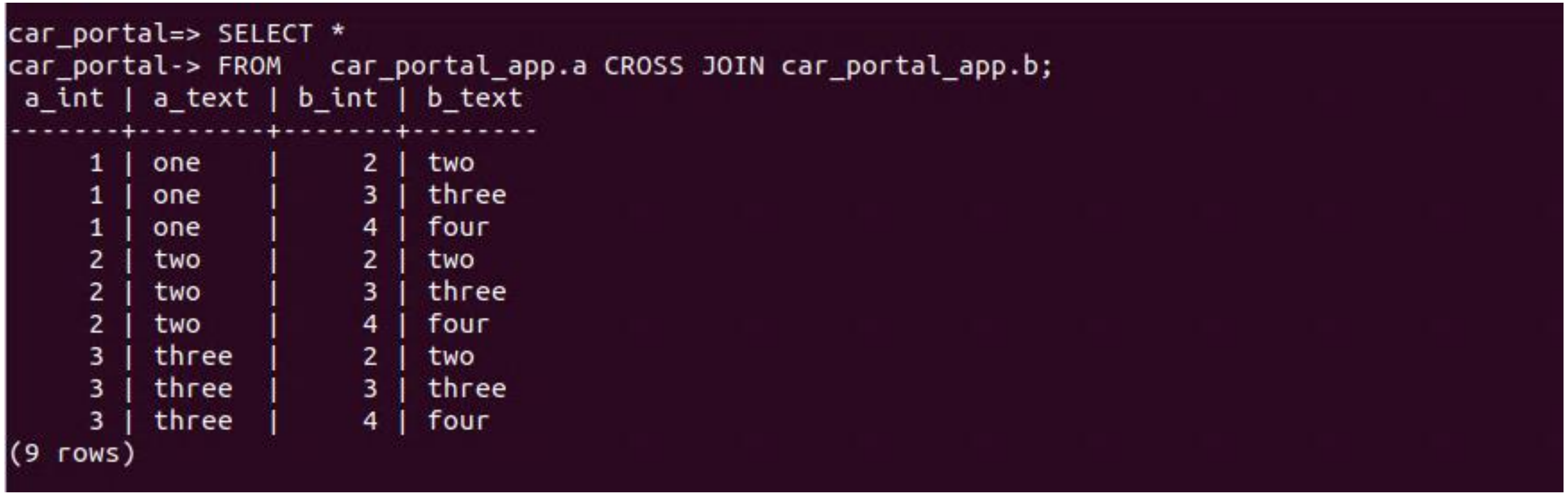
- For example, this is correct:

  -
    ```
    SELECT *
    FROM table_a JOIN table_b
         ON table_a.field1 = table_b.field1
         JOIN table_c
         ON table_a.field2 = table_c.field2 AND
            table_b.field3 = table_c.field3;
    ```
- At the point of joining the `table_c` table, the `table_a` table has been mentioned already in the `FROM` clause, therefore it is possible to refer to that table.

- However, this is not correct:

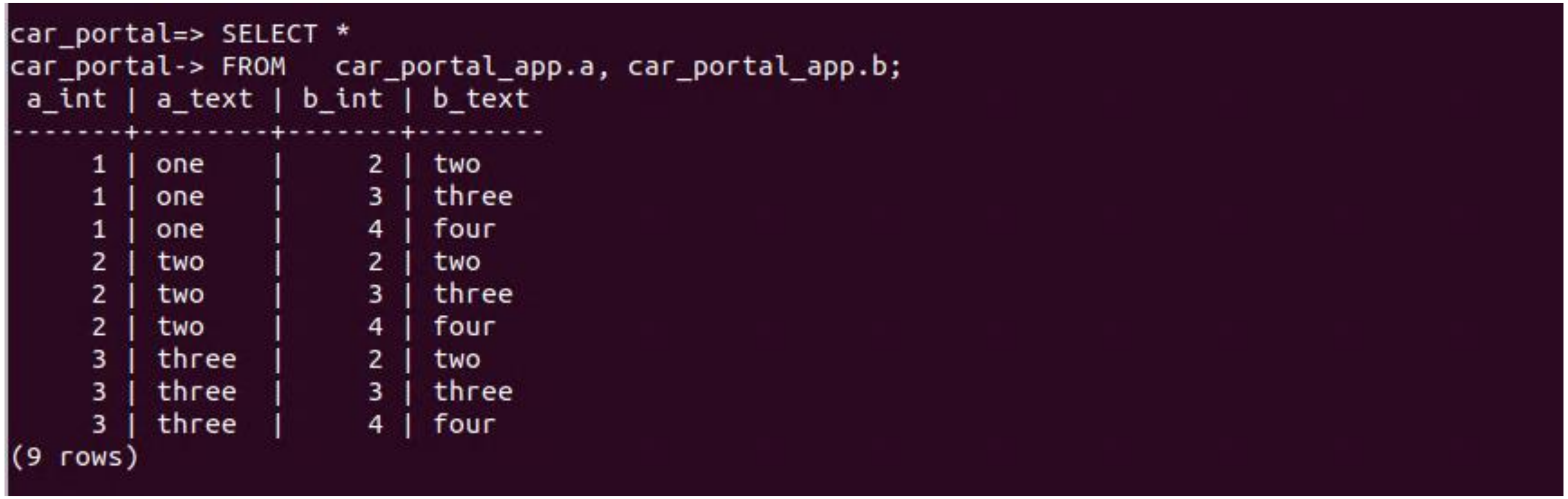
  -
    ```
    SELECT *
    FROM   table_a JOIN table_b
           ON table_b.field3 = table_c.field3
           JOIN table_c
           ON table_a.field2 = table_c.field2
    ```
- The code will cause an error because at JOIN table_b, the table_c table has not been there yet.

- The Cartesian product can also be implemented using the `JOIN` syntax.
- The `CROSS JOIN` keywords are used for that.
- Take a look at the following code:



```
car_portal=> SELECT *
car_portal-> FROM   car_portal_app.a CROSS JOIN car_portal_app.b;
 a_int | a_text | b_int | b_text
-------+--------+-------+--------
     1 | one    |     2 | two
     1 | one    |     3 | three
     1 | one    |     4 | four
     2 | two    |     2 | two
     2 | two    |     3 | three
     2 | two    |     4 | four
     3 | three  |     2 | two
     3 | three  |     3 | three
     3 | three  |     4 | four
(9 rows)
```

- The preceding code is equivalent to the following:



```
car_portal=> SELECT *
car_portal-> FROM   car_portal_app.a, car_portal_app.b;
 a_int | a_text | b_int | b_text
-------+--------+-------+--------
     1 | one    |     2 | two
     1 | one    |     3 | three
     1 | one    |     4 | four
     2 | two    |     2 | two
     2 | two    |     3 | three
     2 | two    |     4 | four
     3 | three  |     2 | two
     3 | three  |     3 | three
     3 | three  |     4 | four
(9 rows)
```

- The join condition in `INNER JOIN` in the logic of the query has the same meaning as a condition to filter the rows in the `WHERE` clause.
- So, the following two queries are, in fact, the same:

  -
    ```
    SELECT *
    FROM   car_portal_app.a INNER JOIN car_portal_app.b
           ON a.a_int = b.b_int;
    ```

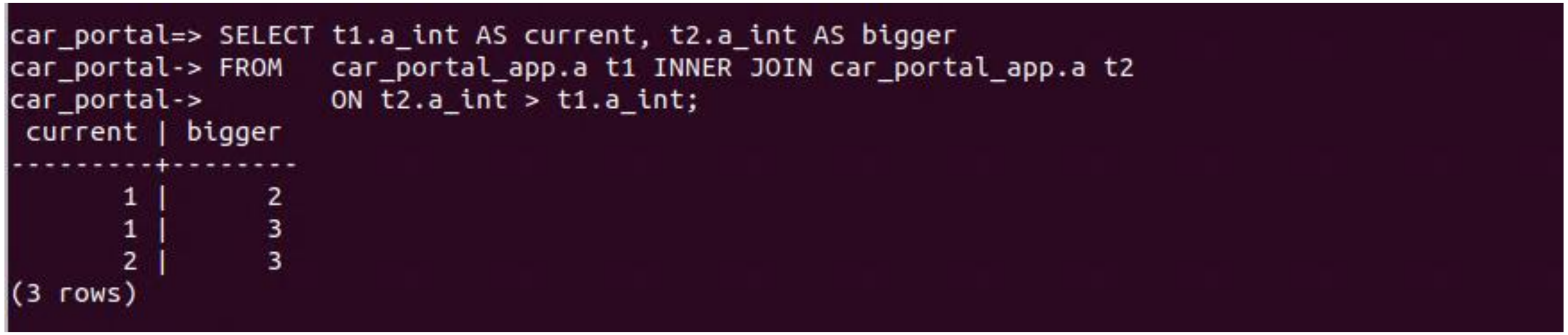
  -
    ```
    SELECT *
    FROM   car_portal_app.a, car_portal_app.b
    WHERE  a.a_int = b.b_int;
    ```

- However, this is not the case for outer joins.
- There is no way to implement an `OUTER JOIN` with the `WHERE` clause in PostgreSQL, though it may be possible in other databases.

Self-Joins

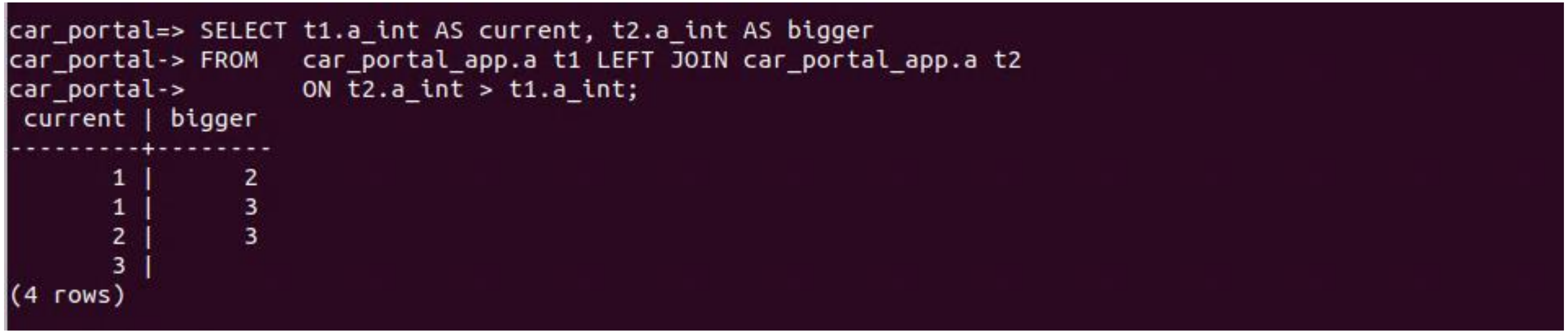
- It's possible to join a table with itself.
- This is called a self-join.
- A self-join has no special syntax.
- In fact, all the data sources in a query are independent, even though they could be the same physically.

- Imagine you want to find out for the a table, for every record, if there are other records with a bigger value of the `a_int` field.
- The following query can be used for this:



```
car_portal=> SELECT t1.a_int AS current, t2.a_int AS bigger
car_portal-> FROM   car_portal_app.a t1 INNER JOIN car_portal_app.a t2
car_portal->        ON t2.a_int > t1.a_int;
 current | bigger
---------+--------
       1 |      2
       1 |      3
       2 |      3
(3 rows)
```

- The value of 3 does not appear in the current column because there are no values greater than 3.
- However, if you want to explicitly show that, LEFT JOIN could be used:



```
car_portal=> SELECT t1.a_int AS current, t2.a_int AS bigger
car_portal-> FROM   car_portal_app.a t1 LEFT JOIN car_portal_app.a t2
car_portal->        ON t2.a_int > t1.a_int;
 current | bigger
---------+--------
       1 |      2
       1 |      3
       2 |      3
       3 |
(4 rows)
```