Changing the Data in a Database

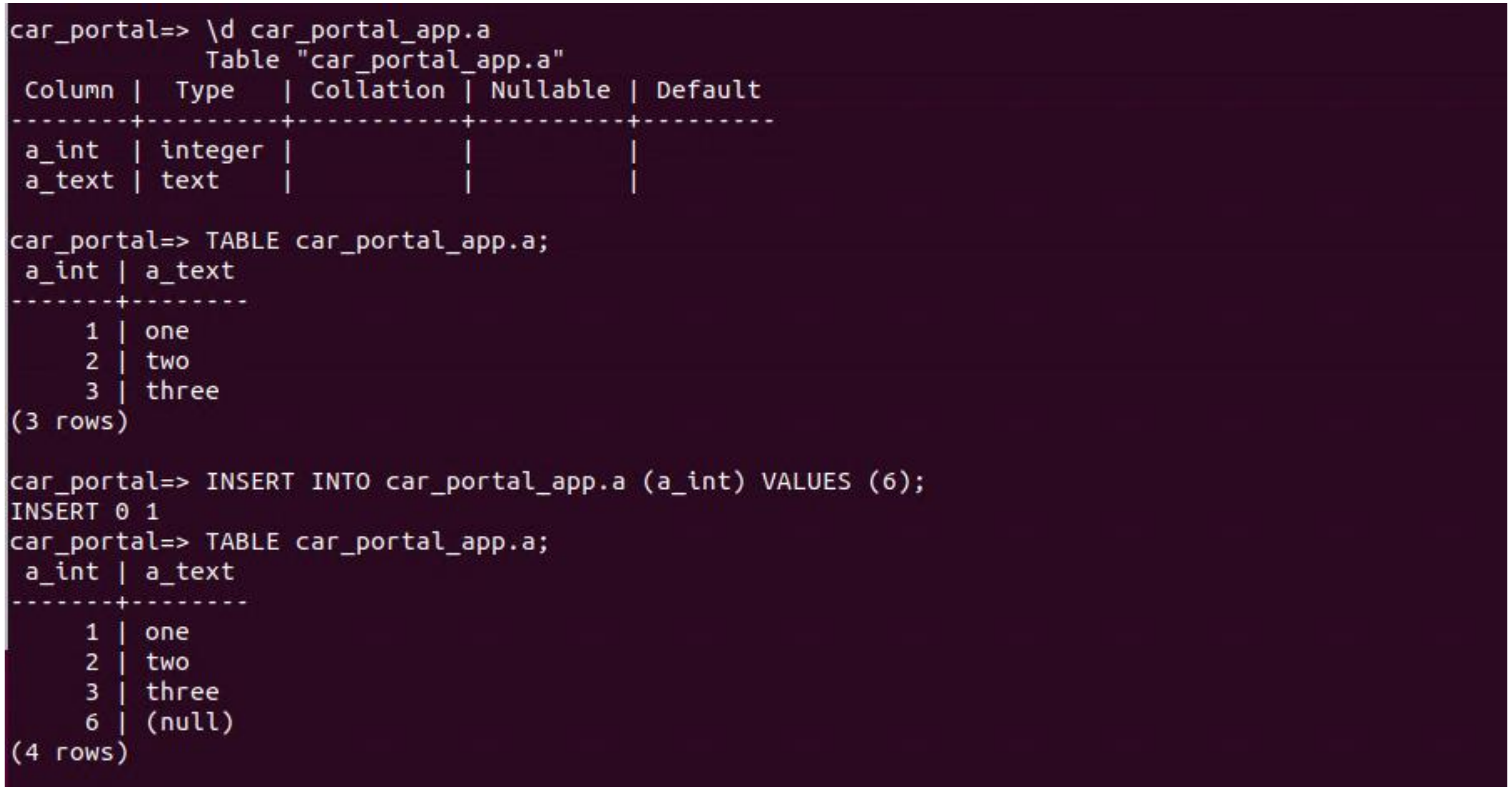
The INSERT Statement

- The `INSERT` statement is used to insert new data into tables in the database.
- Records are always inserted into only one table.
- The `INSERT` statement has the following syntax:
  - `INSERT INTO <table_name> [(<field_list>)] {VALUES (<expression_list>)[,...]}|{DEFAULT VALUES}|<SELECT query>;`

- There are two options when using the `INSERT` statement, which has a different syntax: to insert one or several individual records, or to insert a set of records produced by an SQL query.

- To insert one or several records, the `VALUES` clause is used.
- The list of the field values of a new record to insert is specified after the `VALUES` keyword.
- Items in the list correspond to the fields of the table according to their order.
- When it isn't necessary to set values for all the fields of a new record, the names of the fields whose values should be set should be given in parentheses after the table name.
- The missing fields will then get their default values, if defined, or they will be set to `NULL`.

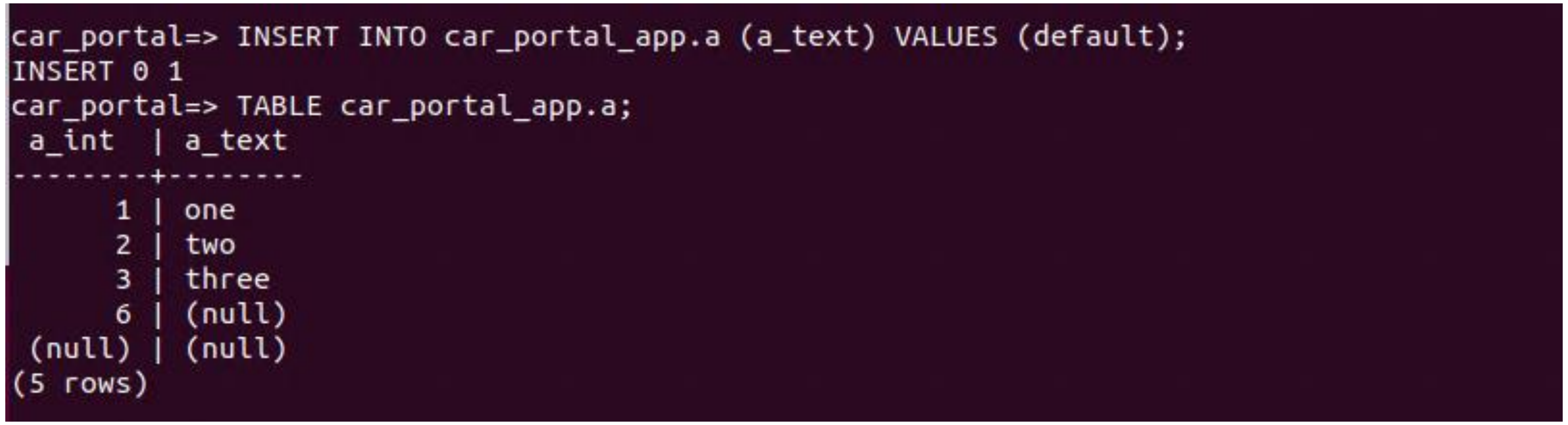
- The number of items in the VALUES list must be the same as the number of fields after the table name:



```
car_portal=> \d car_portal_app.a
             Table "car_portal_app.a"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 a_int  | integer |           |          |
 a_text | text    |           |          |

car_portal=> TABLE car_portal_app.a;
 a_int | a_text
-------+--------
     1 | one
     2 | two
     3 | three
(3 rows)

car_portal=> INSERT INTO car_portal_app.a (a_int) VALUES (6);
INSERT 0 1
car_portal=> TABLE car_portal_app.a;
 a_int | a_text
-------+--------
     1 | one
     2 | two
     3 | three
     6 | (null)
(4 rows)
```

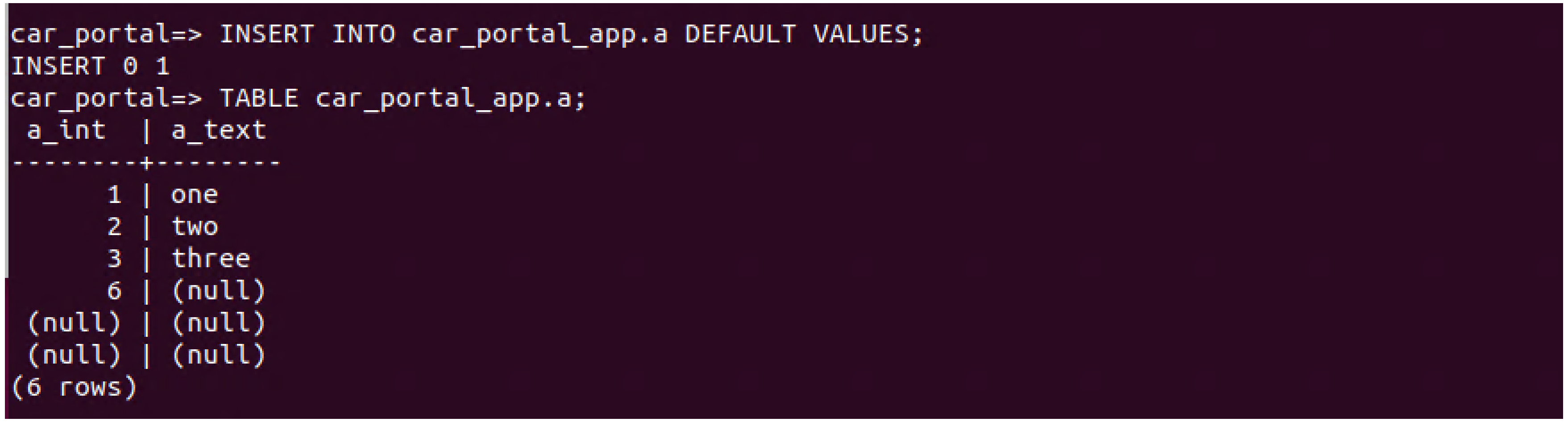
- The output of the `INSERT` command ,when it has successfully executed, is the word `INSERT` followed by the OID of the row that has been inserted (when only one row is inserted and OIDs are enabled for the table, otherwise, it is zero) and the number of records inserted.

- Another way to set default values to a field is to use the `DEFAULT` keyword in the `VALUES` list.
- If a default is not defined for the field, a `NULL` value will be set:



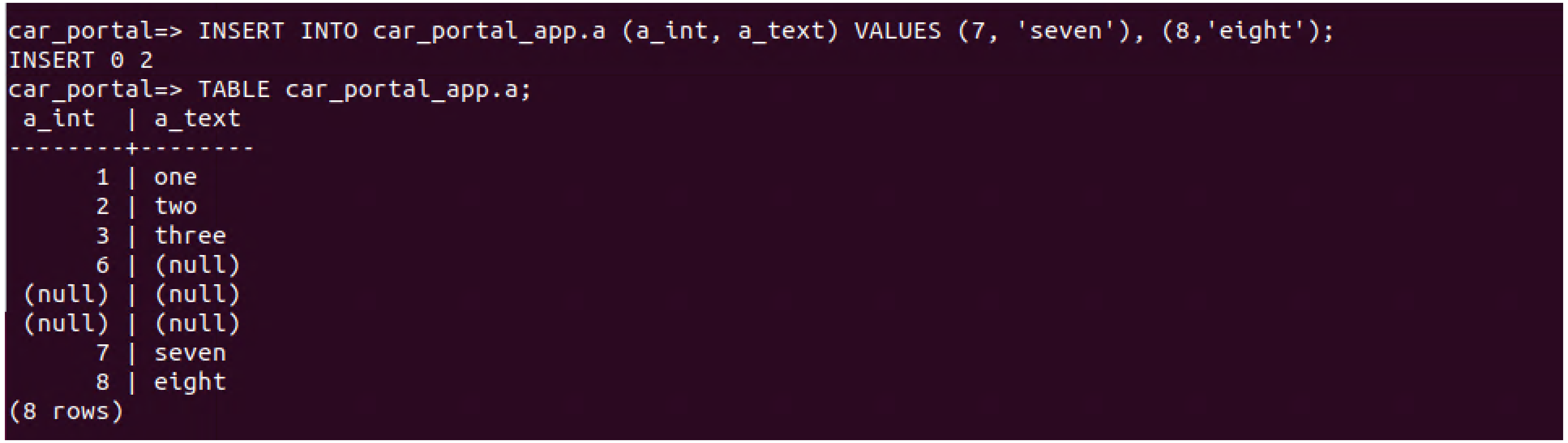
```
car_portal=> INSERT INTO car_portal_app.a (a_text) VALUES (default);
INSERT 0 1
car_portal=> TABLE car_portal_app.a;
 a_int  | a_text
--------+--------
      1 | one
      2 | two
      3 | three
      6 | (null)
 (null) | (null)
(5 rows)
```

- It's also possible to set all of the fields of the new record to their default values using the `DEFAULT VALUES` keyword:



```
car_portal=> INSERT INTO car_portal_app.a DEFAULT VALUES;
INSERT 0 1
car_portal=> TABLE car_portal_app.a;
 a_int  | a_text
--------+--------
      1 | one
      2 | two
      3 | three
      6 | (null)
 (null) | (null)
 (null) | (null)
(6 rows)
```

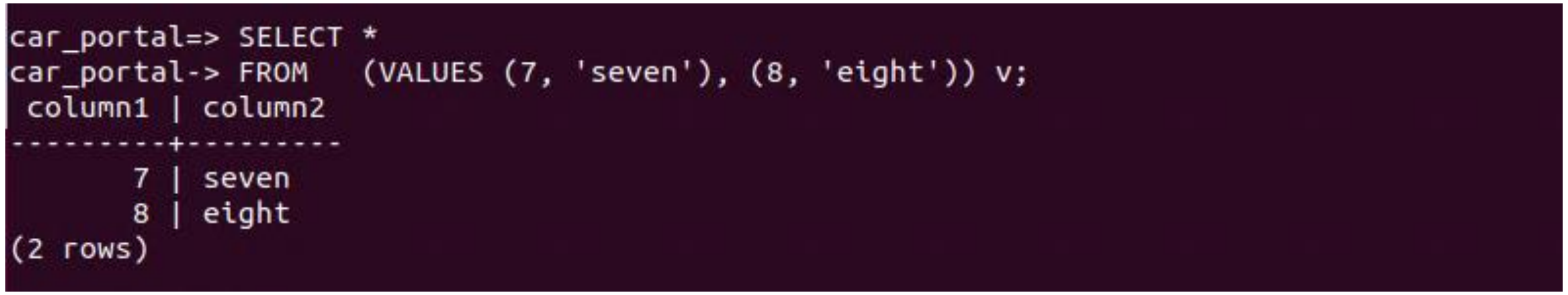
- It's possible to insert multiple records using the `VALUES` syntax, providing several lists of values, comma-separated:



```
car_portal=> INSERT INTO car_portal_app.a (a_int, a_text) VALUES (7, 'seven'), (8,'eight');
INSERT 0 2
car_portal=> TABLE car_portal_app.a;
 a_int  | a_text
--------+--------
      1 | one
      2 | two
      3 | three
      6 | (null)
 (null) | (null)
 (null) | (null)
      7 | seven
      8 | eight
(8 rows)
```

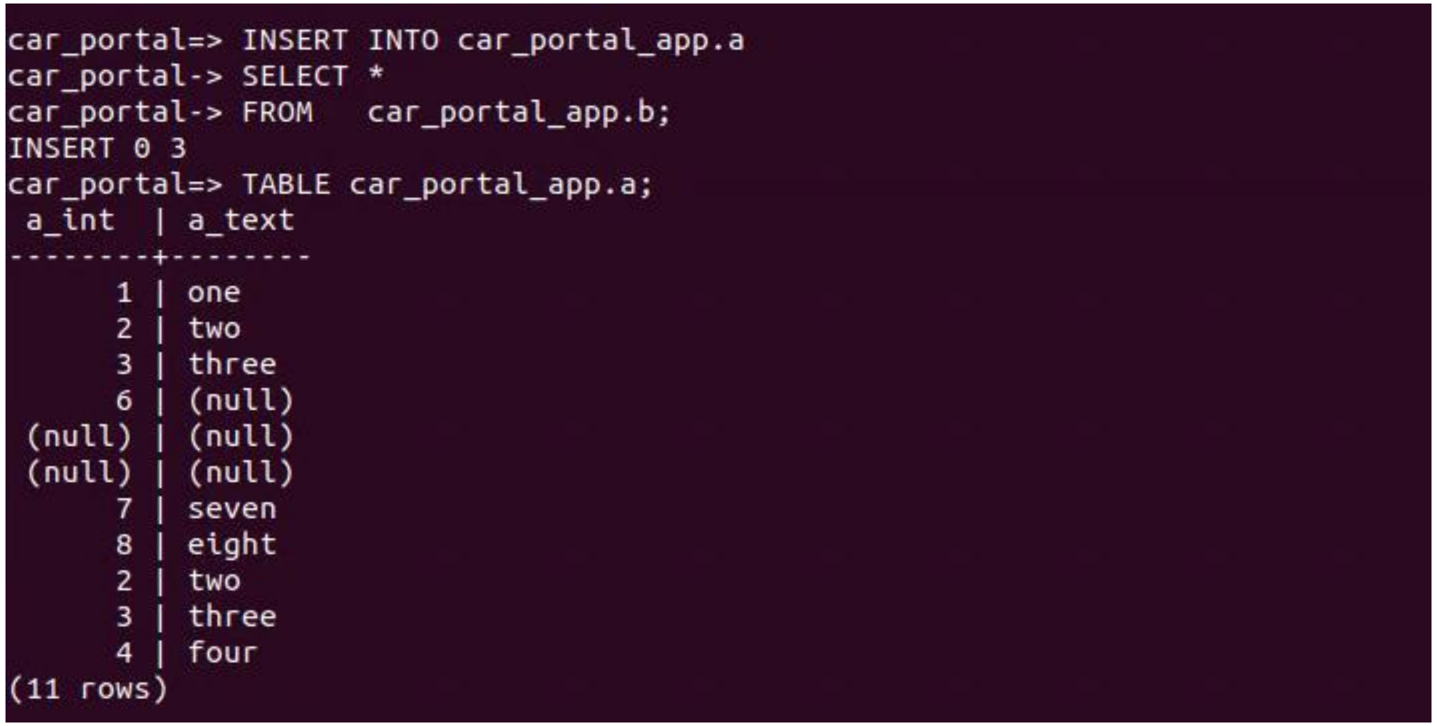
- This option is PostgreSQL-specific.
- Different databases might allow you to insert only one row at a time.

- In fact, in PostgreSQL, the `VALUES` clause is a standalone SQL command.
- Therefore, it can be used as a subquery in a `SELECT` query:



```
car_portal=> SELECT *
car_portal-> FROM   (VALUES (7, 'seven'), (8, 'eight')) v;
 column1 | column2
---------+---------
       7 | seven
       8 | eight
(2 rows)
```

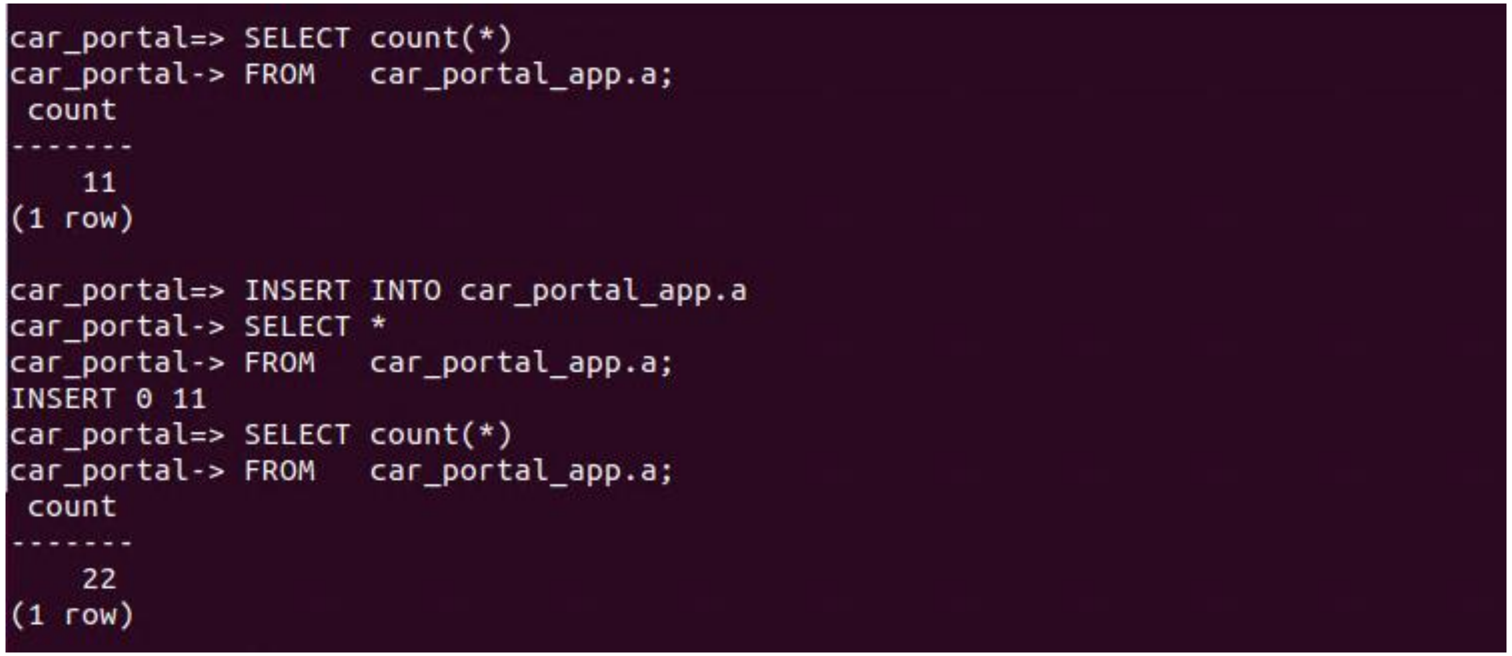
- When the records to insert are taken from another table or view, a `SELECT` query is used instead of the `VALUES` clause:



```
car_portal=> INSERT INTO car_portal_app.a
car_portal-> SELECT *
car_portal-> FROM   car_portal_app.b;
INSERT 0 3
car_portal=> TABLE car_portal_app.a;
 a_int  | a_text
--------+--------
      1 | one
      2 | two
      3 | three
      6 | (null)
 (null) | (null)
 (null) | (null)
      7 | seven
      8 | eight
      2 | two
      3 | three
      4 | four
(11 rows)
```

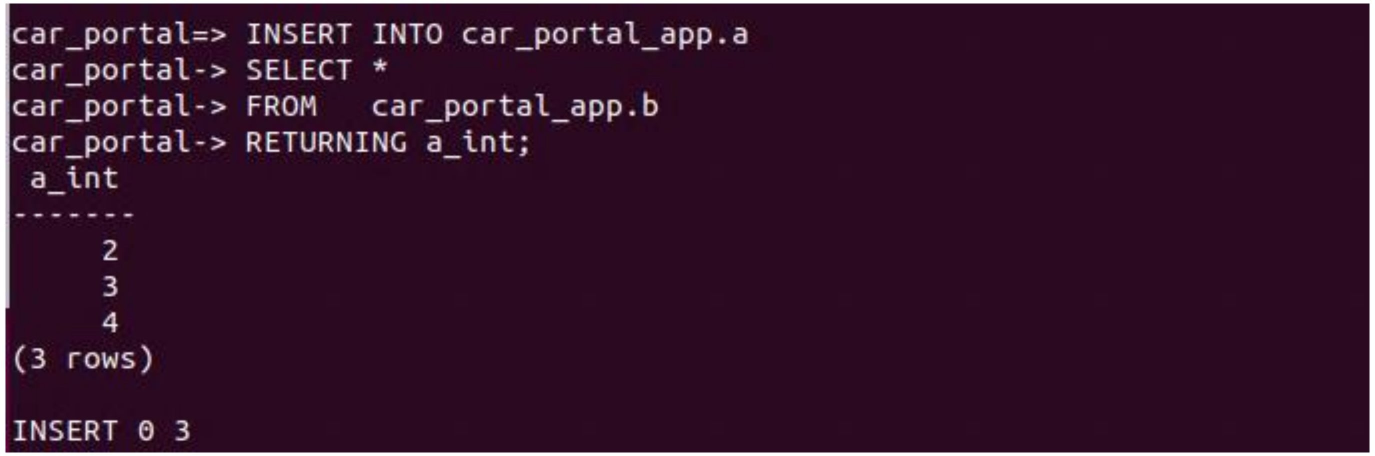
- The result of the query should match the structure of the table: have the same number of columns of compatible types in the same order.

- In the `SELECT` query, it's possible to use the table in which the records are inserted.
- For example, to duplicate records in a table, the following statement can be used:



```
car_portal=> SELECT count(*)
car_portal-> FROM   car_portal_app.a;
 count
-------
    11
(1 row)

car_portal=> INSERT INTO car_portal_app.a
car_portal-> SELECT *
car_portal-> FROM   car_portal_app.a;
INSERT 0 11
car_portal=> SELECT count(*)
car_portal-> FROM   car_portal_app.a;
 count
-------
    22
(1 row)
```

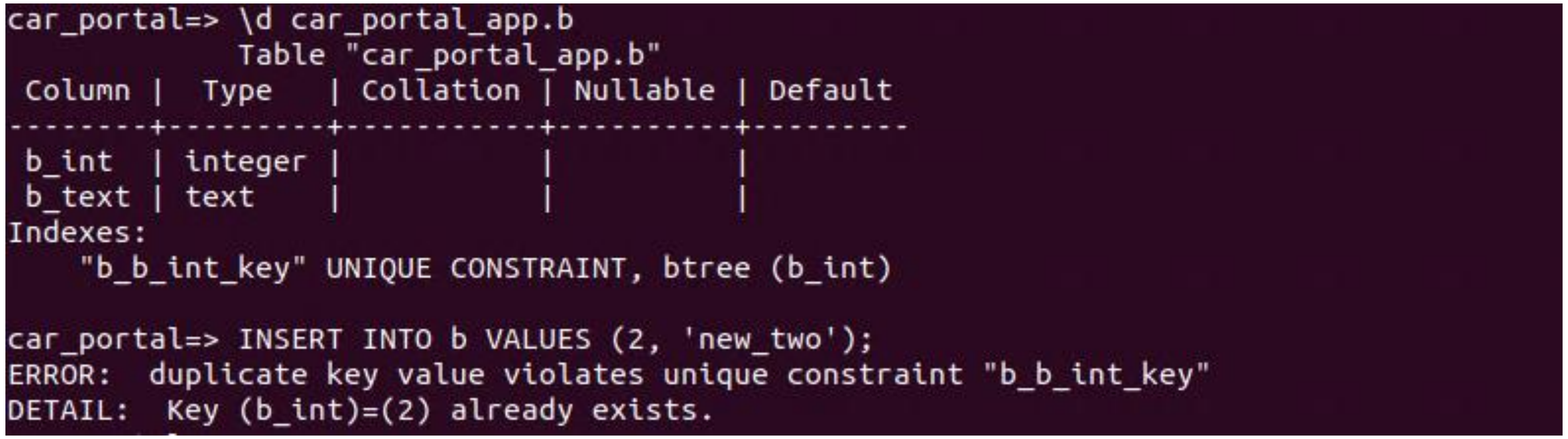
- By default, the `INSERT` statement returns the number of inserted records.
- It's also possible to return the inserted records themselves or some of their fields.
- The output of the statement is then similar to the output of the `SELECT` query.
- The `RETURNING` keyword, with the list of fields to return, is used for this:



```
car_portal=> INSERT INTO car_portal_app.a
car_portal-> SELECT *
car_portal-> FROM   car_portal_app.b
car_portal-> RETURNING a_int;
 a_int
-------
     2
     3
     4
(3 rows)

INSERT 0 3
```

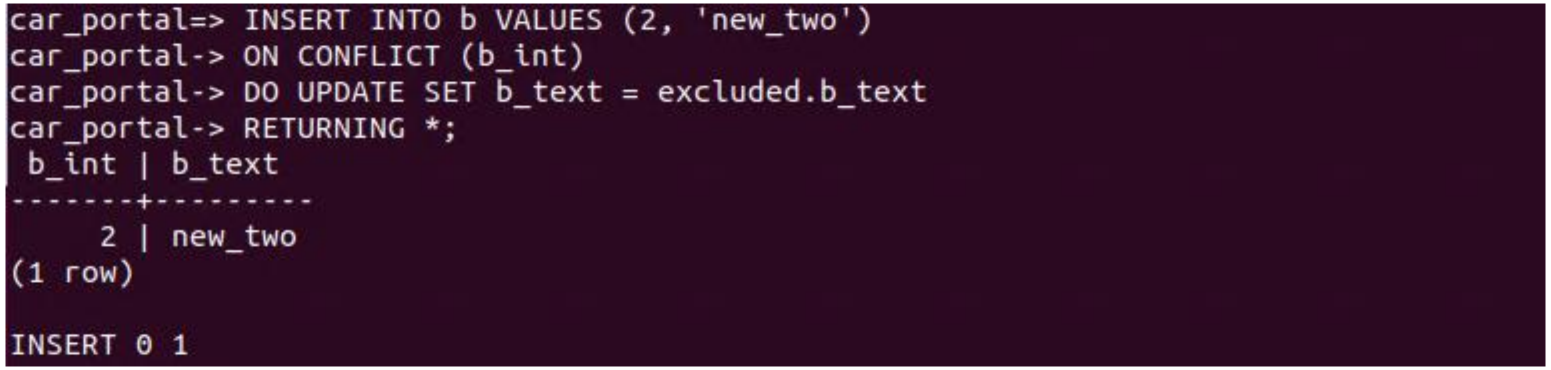
- If a unique constraint is defined on the table where the rows are inserted, `INSERT` will fail if it tries to insert records that conflict with existing ones.
- However, it's possible to let `INSERT` ignore such records or update them instead.

- Assuming that there is a unique constraint on the `b` table for the `b_int` field, consider the following example:



```
car_portal=> \d car_portal_app.b
              Table "car_portal_app.b"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 b_int  | integer |           |          |
 b_text | text    |           |          |
Indexes:
    "b_b_int_key" UNIQUE CONSTRAINT, btree (b_int)

car_portal=> INSERT INTO b VALUES (2, 'new_two');
ERROR:  duplicate key value violates unique constraint "b_b_int_key"
DETAIL:  Key (b_int)=(2) already exists.
```

- What if we want to change a record instead of inserting one, if it exists already:



```
car_portal=> INSERT INTO b VALUES (2, 'new_two')
car_portal-> ON CONFLICT (b_int)
car_portal-> DO UPDATE SET b_text = excluded.b_text
car_portal-> RETURNING *;
 b_int | b_text
-------+---------
     2 | new_two
(1 row)

INSERT 0 1
```

The UPDATE Statement

- The `UPDATE` statement is used to change the data or records of a table without changing their number.
- It has the following syntax:

  -
    ```
    UPDATE <table_name>
    SET <field_name> = <expression>[, ...]
    [FROM <table_name> [JOIN clause]]
    [WHERE <condition>];
    ```

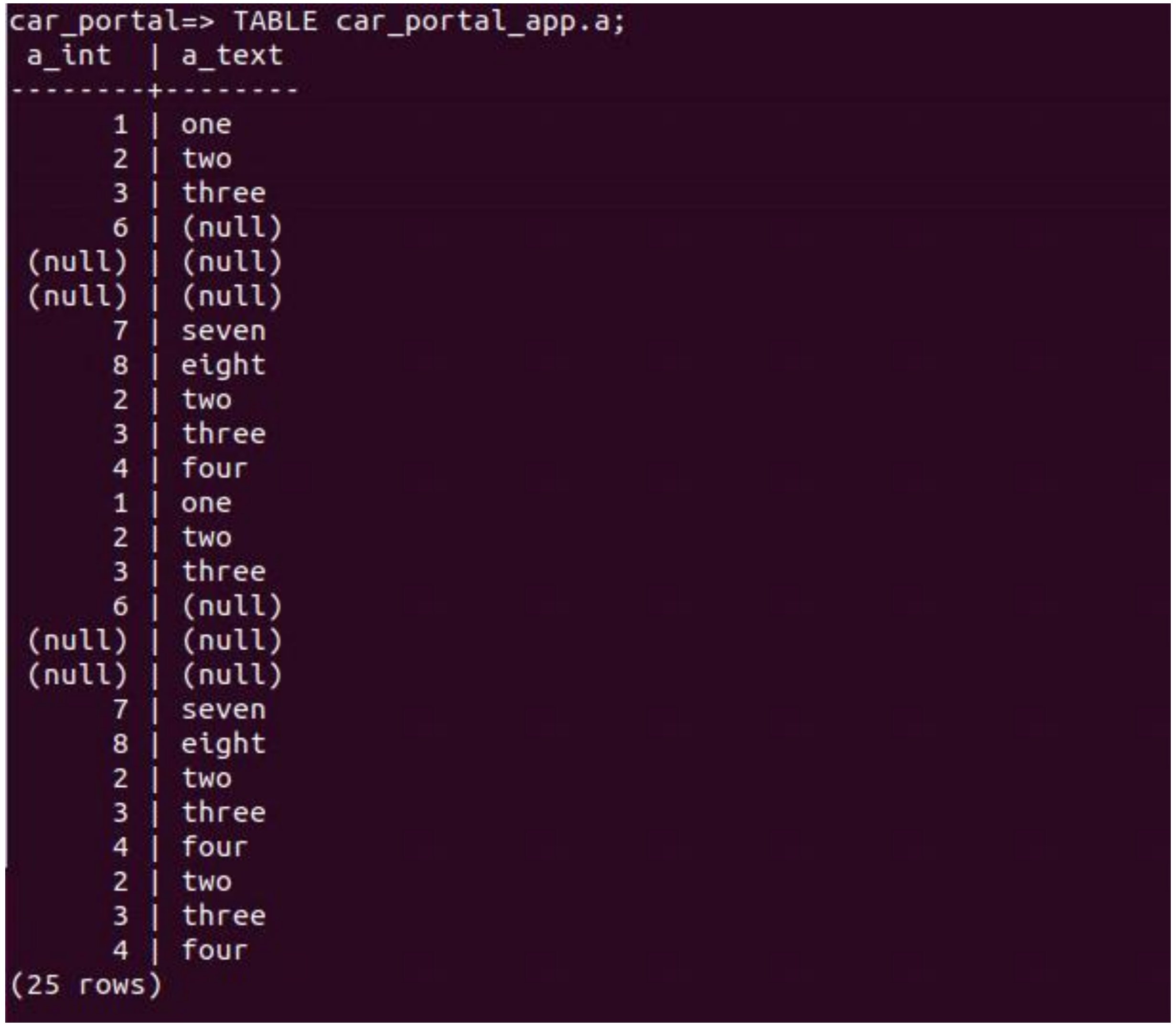
- There are two ways of using the `UPDATE` statement.
- The first is similar to a simple `SELECT` statement and is called `sub-select`.
- The second is based on other tables and is similar to a `SELECT` statement with multiple tables.
- In most cases, the same result can be achieved using any of these methods.

- In PostgreSQL, only one table can be updated at a time.
- Other databases may allow you to update multiple tables at the same time under certain conditions.

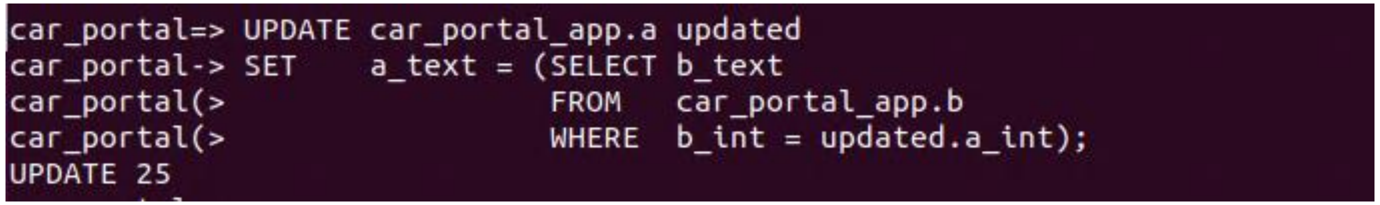
UPDATE Using Sub-Select

- The expression for a new value is the usual SQL expression.
- It's possible to refer to the same field in the expression.
- In that case, the old value is used:
  - `UPDATE t SET f = f + 1 WHERE a = 5;`

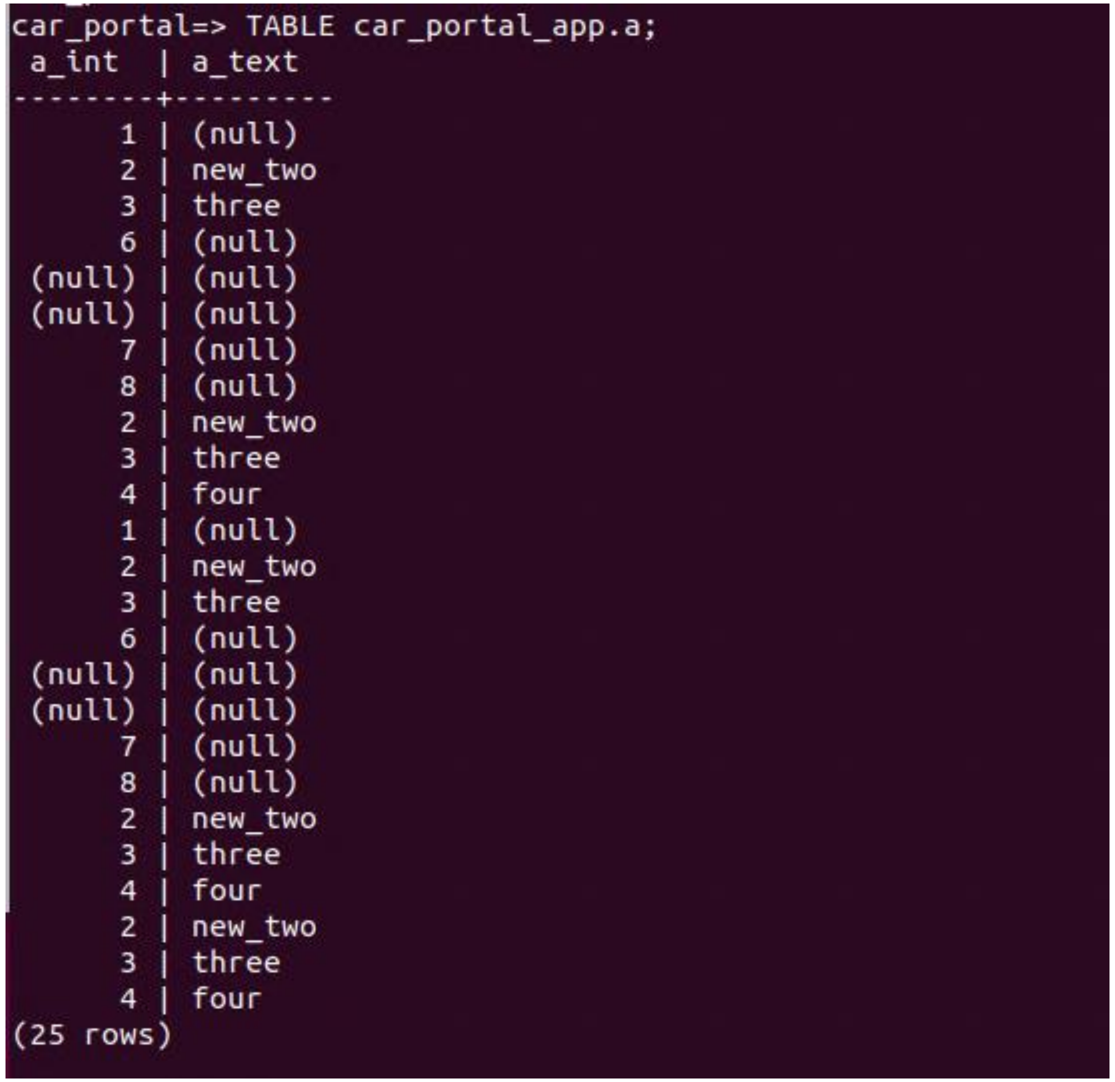
- It's common to use a subquery in the `UPDATE` statements.
- To be able to refer to the table being updated from a subquery, the table should get an alias.



```
car_portal=> TABLE car_portal_app.a;
 a_int  | a_text
--------+--------
      1 | one
      2 | two
      3 | three
      6 | (null)
 (null) | (null)
 (null) | (null)
      7 | seven
      8 | eight
      2 | two
      3 | three
      4 | four
      1 | one
      2 | two
      3 | three
      6 | (null)
 (null) | (null)
 (null) | (null)
      7 | seven
      8 | eight
      2 | two
      3 | three
      4 | four
      2 | two
      3 | three
      4 | four
(25 rows)
```



```
car_portal=> UPDATE car_portal_app.a updated
car_portal-> SET    a_text = (SELECT b_text
car_portal(>                 FROM   car_portal_app.b
car_portal(>                 WHERE  b_int = updated.a_int);
UPDATE 25
```



```
car_portal=> TABLE car_portal_app.a;
 a_int  | a_text
--------+---------
      1 | (null)
      2 | new_two
      3 | three
      6 | (null)
 (null) | (null)
 (null) | (null)
      7 | (null)
      8 | (null)
      2 | new_two
      3 | three
      4 | four
      1 | (null)
      2 | new_two
      3 | three
      6 | (null)
 (null) | (null)
 (null) | (null)
      7 | (null)
      8 | (null)
      2 | new_two
      3 | three
      4 | four
      2 | new_two
      3 | three
      4 | four
(25 rows)
```

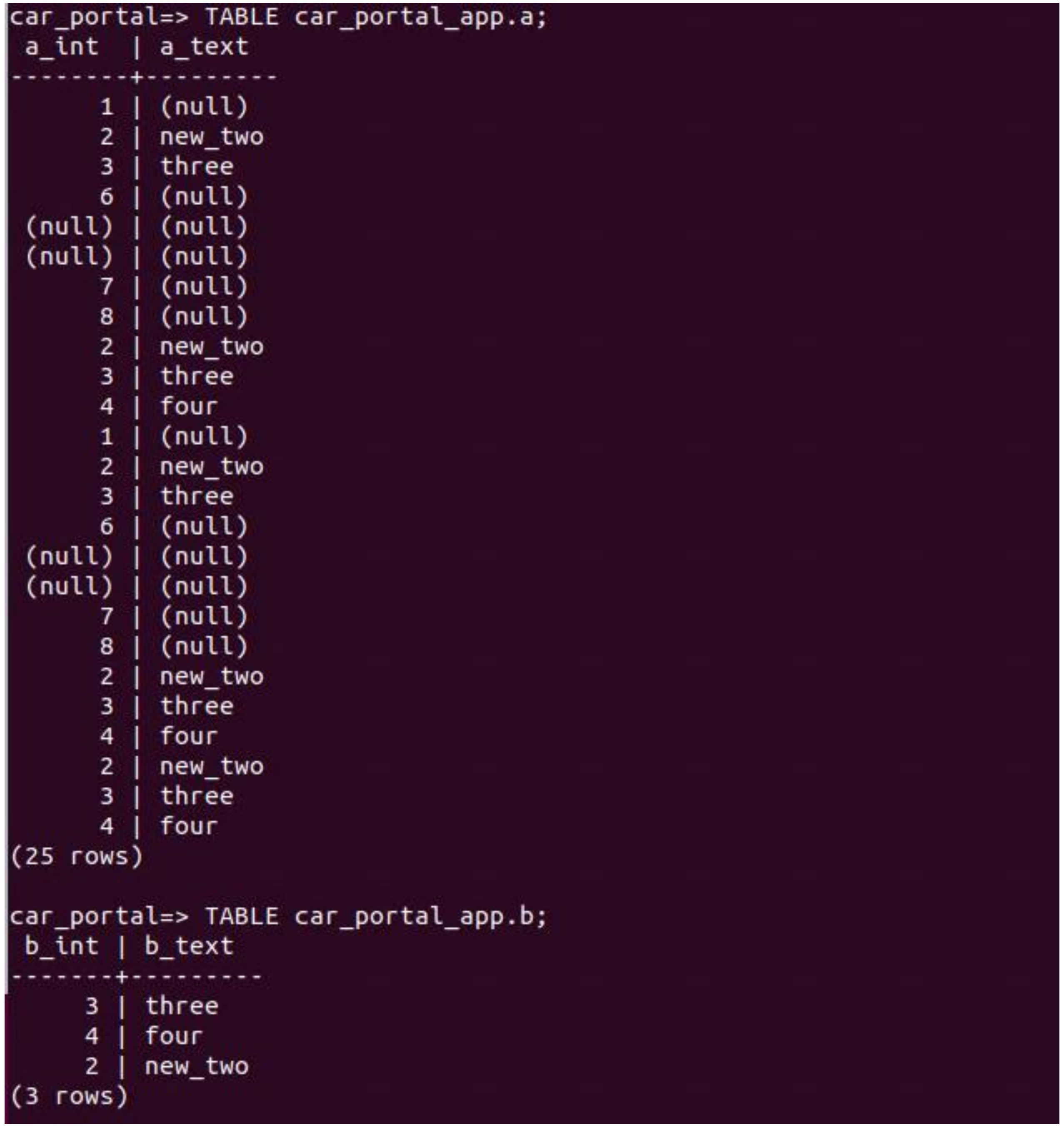
- If the subquery returns no result, the field value is set to `NULL`.

- The `WHERE` clause is similar to the one used in the `SELECT` statement.
- If the `WHERE` statement is not specified, all of the records are updated.

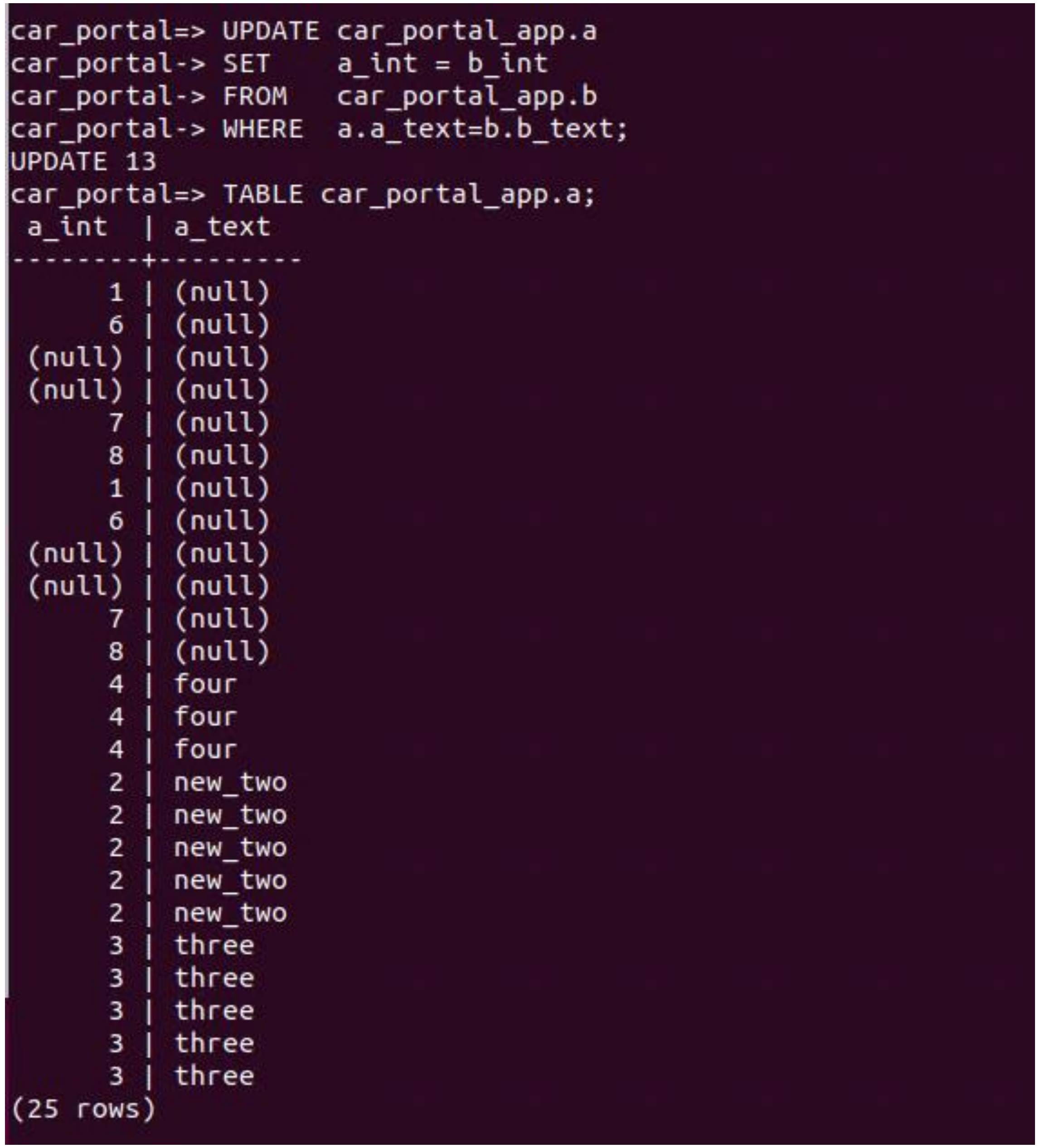
UPDATE Using Additional Tables

- The second way of updating rows in the table is to use the `FROM` clause in a similar manner as in the `SELECT` statement:

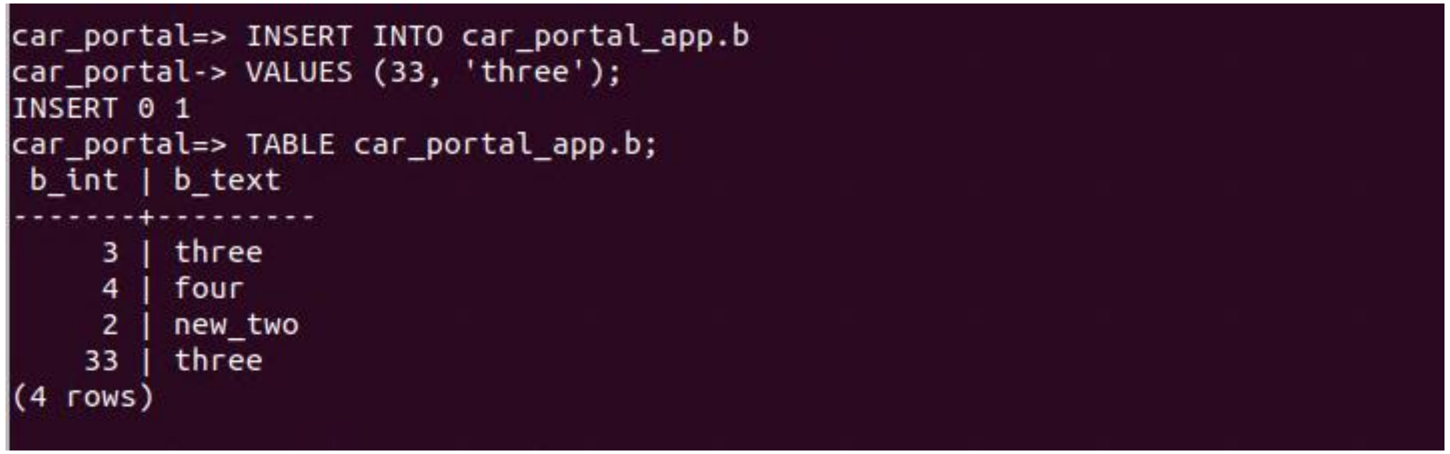
  -
    ```
    UPDATE car_portal_app.a
    SET    a_int = b_int
    FROM   car_portal_app.b
    WHERE  a.a_text=b.b_text;
    ```
- Every row from `a`, when there are rows in `b` with the same value of the text field, will be updated.
- The new value for the numeric field is taken from the `b` table.



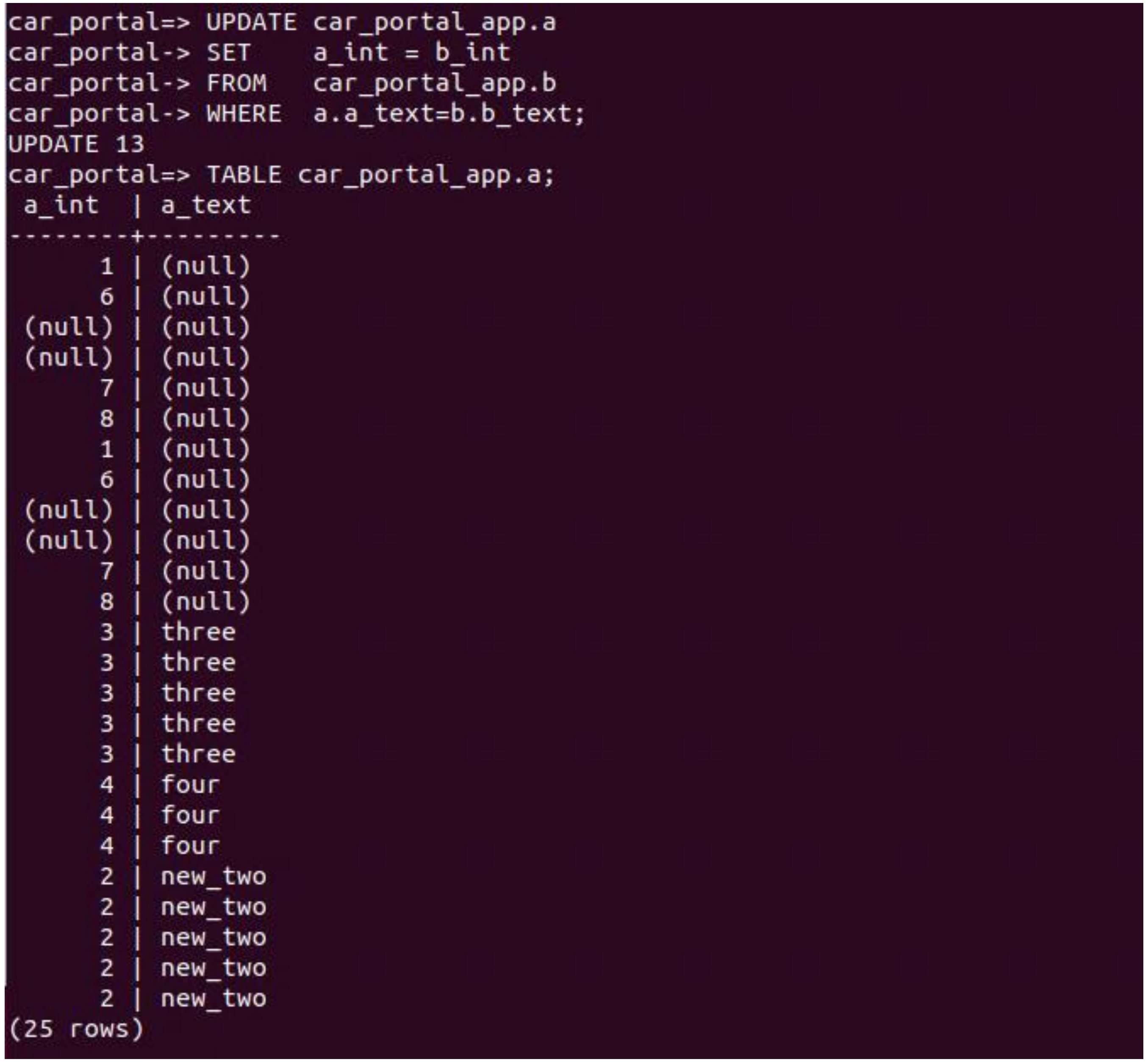
```
car_portal=> TABLE car_portal_app.a;
 a_int  | a_text
--------+---------
      1 | (null)
      2 | new_two
      3 | three
      6 | (null)
 (null) | (null)
 (null) | (null)
      7 | (null)
      8 | (null)
      2 | new_two
      3 | three
      4 | four
      1 | (null)
      2 | new_two
      3 | three
      6 | (null)
 (null) | (null)
 (null) | (null)
      7 | (null)
      8 | (null)
      2 | new_two
      3 | three
      4 | four
      2 | new_two
      3 | three
      4 | four
(25 rows)

car_portal=> TABLE car_portal_app.b;
 b_int | b_text
-------+---------
     3 | three
     4 | four
     2 | new_two
(3 rows)
```


```
car_portal=> UPDATE car_portal_app.a
car_portal-> SET    a_int = b_int
car_portal-> FROM   car_portal_app.b
car_portal-> WHERE  a.a_text=b.b_text;
UPDATE 13
car_portal=> TABLE car_portal_app.a;
 a_int  | a_text
--------+---------
      1 | (null)
      6 | (null)
 (null) | (null)
 (null) | (null)
      7 | (null)
      8 | (null)
      1 | (null)
      6 | (null)
 (null) | (null)
 (null) | (null)
      7 | (null)
      8 | (null)
      4 | four
      4 | four
      4 | four
      2 | new_two
      2 | new_two
      2 | new_two
      2 | new_two
      2 | new_two
      3 | three
      3 | three
      3 | three
      3 | three
      3 | three
(25 rows)
```

- When a `FROM` clause is present, what essentially happens is that the target table is joined to the tables mentioned in the *from_list*, and each output row of the join represents an update operation for the target table.
- When using `FROM` you should ensure that the join produces at most one output row for each row to be modified.
- In other words, a target row shouldn't join to more than one row from the other table(s).
- If it does, then only one of the join rows will be used to update the target row, but which one will be used is not readily predictable.

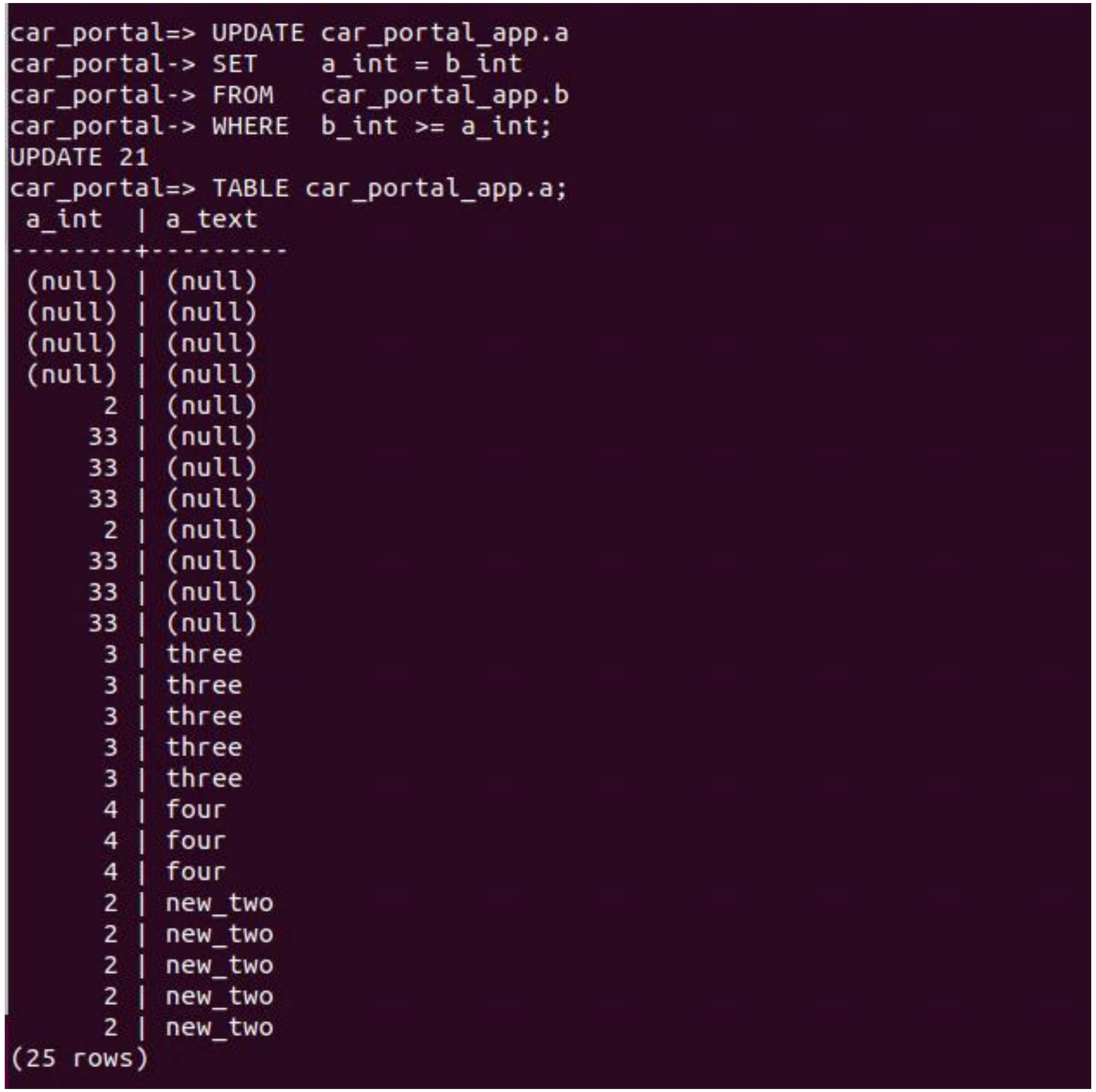


```
car_portal=> INSERT INTO car_portal_app.b
car_portal-> VALUES (33, 'three');
INSERT 0 1
car_portal=> TABLE car_portal_app.b;
 b_int | b_text
-------+---------
     3 | three
     4 | four
     2 | new_two
    33 | three
(4 rows)
```



```
car_portal=> UPDATE car_portal_app.a
car_portal-> SET    a_int = b_int
car_portal-> FROM   car_portal_app.b
car_portal-> WHERE  a.a_text=b.b_text;
UPDATE 13
car_portal=> TABLE car_portal_app.a;
 a_int  | a_text
--------+---------
      1 | (null)
      6 | (null)
 (null) | (null)
 (null) | (null)
      7 | (null)
      8 | (null)
      1 | (null)
      6 | (null)
 (null) | (null)
 (null) | (null)
      7 | (null)
      8 | (null)
      3 | three
      3 | three
      3 | three
      3 | three
      3 | three
      4 | four
      4 | four
      4 | four
      2 | new_two
      2 | new_two
      2 | new_two
      2 | new_two
      2 | new_two
(25 rows)
```

- Another example:

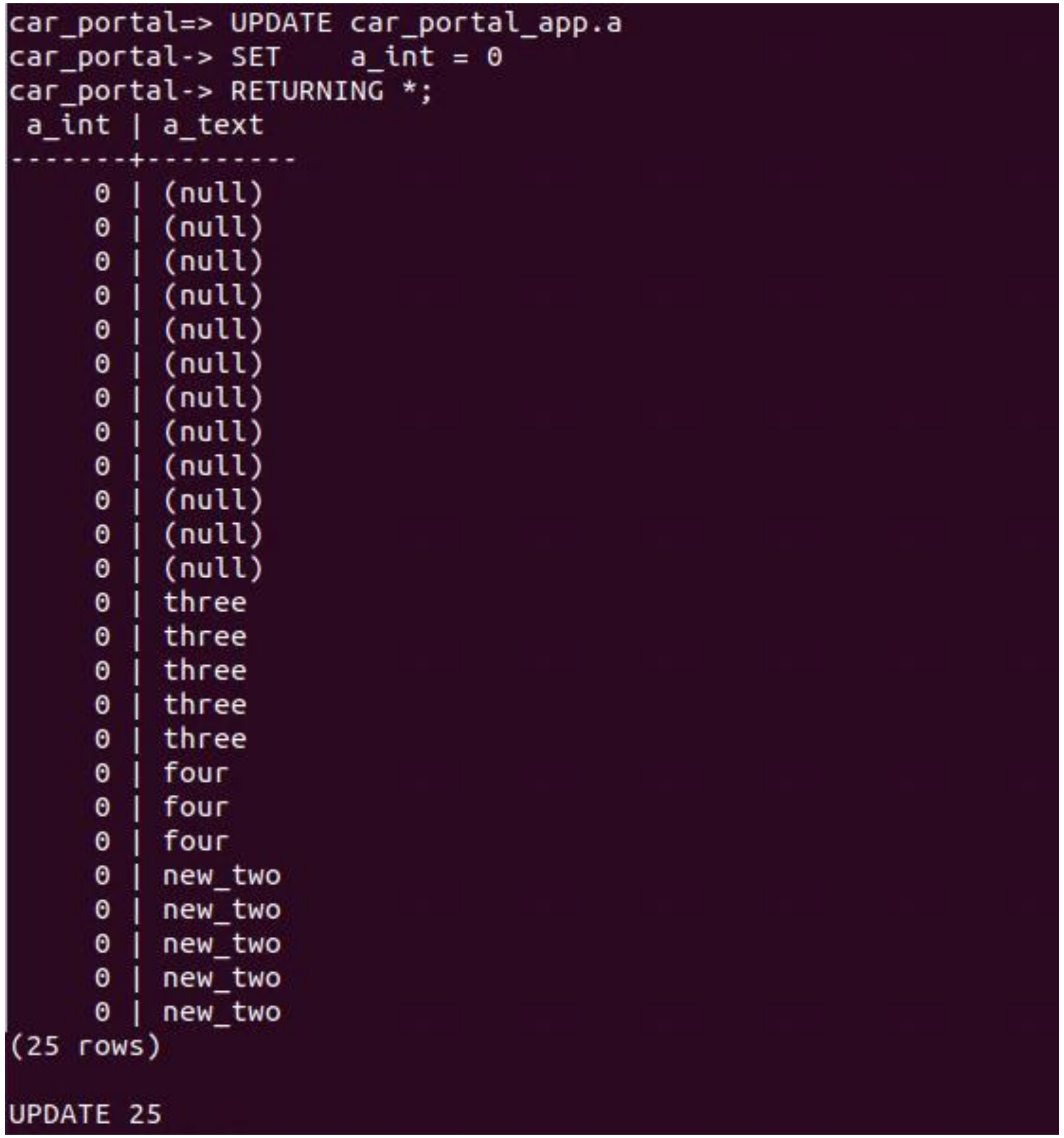


```
car_portal=> UPDATE car_portal_app.a
car_portal-> SET    a_int = b_int
car_portal-> FROM   car_portal_app.b
car_portal-> WHERE  b_int >= a_int;
UPDATE 21
car_portal=> TABLE car_portal_app.a;
 a_int  | a_text
--------+---------
 (null) | (null)
 (null) | (null)
 (null) | (null)
 (null) | (null)
      2 | (null)
     33 | (null)
     33 | (null)
     33 | (null)
      2 | (null)
     33 | (null)
     33 | (null)
     33 | (null)
      3 | three
      3 | three
      3 | three
      3 | three
      3 | three
      4 | four
      4 | four
      4 | four
      2 | new_two
      2 | new_two
      2 | new_two
      2 | new_two
      2 | new_two
(25 rows)
```

- For each record of the a table, there's more than one record from the `b` table where `b_int` is greater than or equal to `a_int`.
- That's why the result of this update is undefined.
- However, PostgreSQL will allow this to be executed.

- For this reason, you should be careful when doing updates this way.

- The UPDATE query can return the records that were changed if the RETURNING clause is used, just as it is in the INSERT statement:



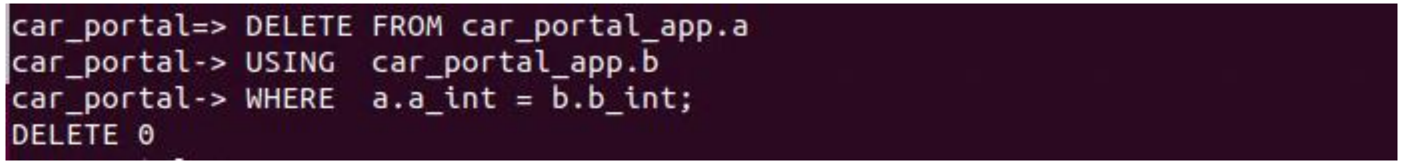
```
car_portal=> UPDATE car_portal_app.a
car_portal-> SET    a_int = 0
car_portal-> RETURNING *;
 a_int | a_text
-------+---------
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | three
     0 | three
     0 | three
     0 | three
     0 | three
     0 | four
     0 | four
     0 | four
     0 | new_two
     0 | new_two
     0 | new_two
     0 | new_two
     0 | new_two
(25 rows)

UPDATE 25
```

The DELETE Statement

- The `DELETE` statement is used to remove records from the database.
- As with `UPDATE`, there are two ways of deleting: using sub-select or using another table or tables.

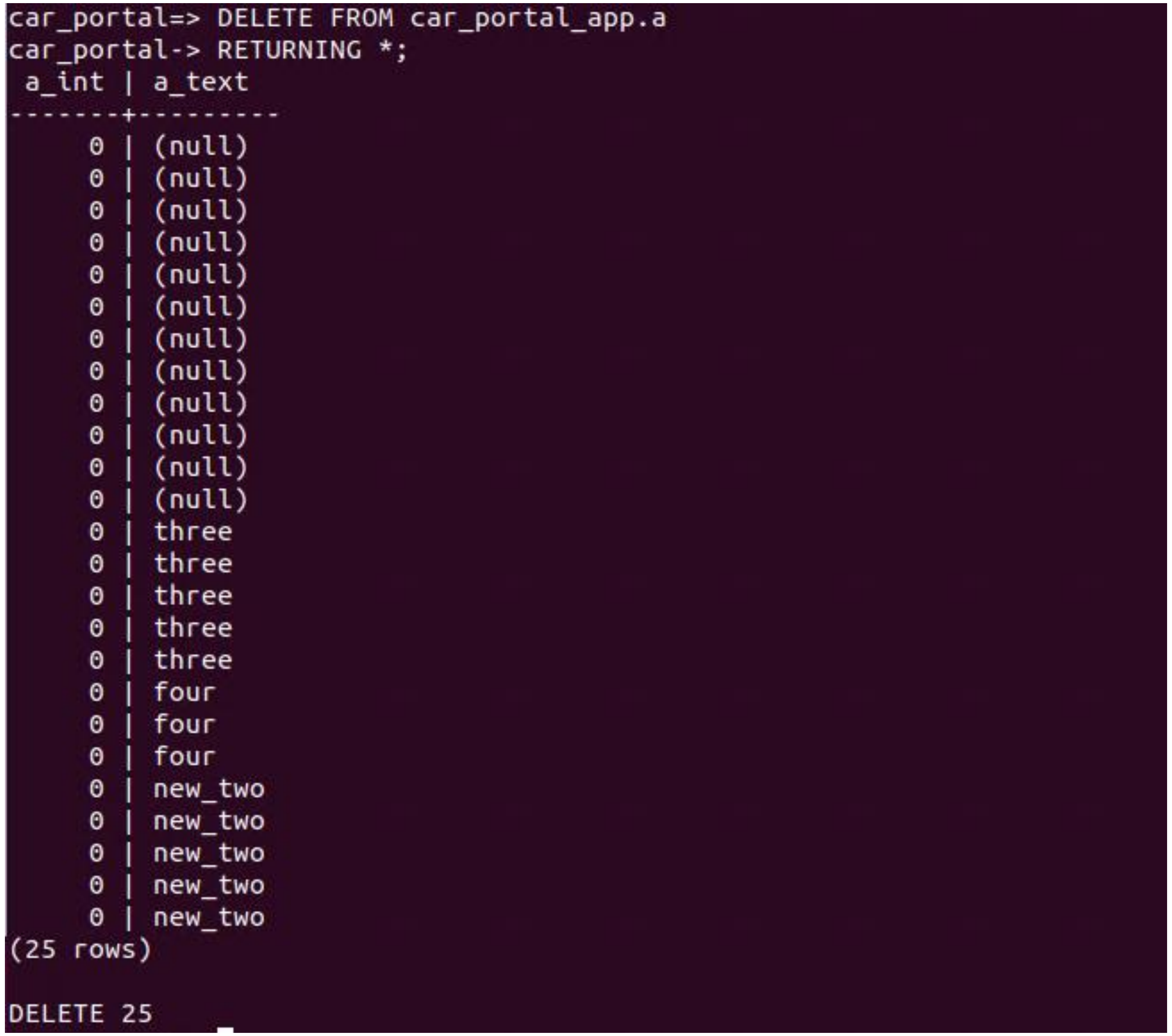
- The sub-select syntax is as follows:
  - `DELETE FROM <table_name> [WHERE <condition>];`
- Records that follow the condition will be removed from the table.
- If the WHERE clause is omitted, all of the records will be deleted.

- DELETE based on another table is similar to using the FROM clause of the UPDATE statement.
- Instead of FROM, the USING keyword should be used because FROM is already used in the syntax of the DELETE statement:



```
car_portal=> DELETE FROM car_portal_app.a
car_portal-> USING  car_portal_app.b
car_portal-> WHERE  a.a_int = b.b_int;
DELETE 0
```

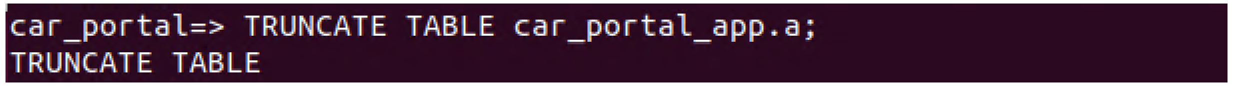
- As well as UPDATE and INSERT, the DELETE statement can return deleted rows when the RETURNING keyword is used:



```
car_portal=> DELETE FROM car_portal_app.a
car_portal-> RETURNING *;
 a_int | a_text
-------+---------
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | (null)
     0 | three
     0 | three
     0 | three
     0 | three
     0 | three
     0 | four
     0 | four
     0 | four
     0 | new_two
     0 | new_two
     0 | new_two
     0 | new_two
     0 | new_two
(25 rows)

DELETE 25
```

The TRUNCATE Statement

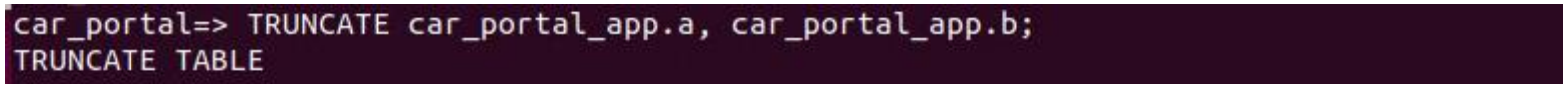
- Another statement that can also change the data, but not the data structure, is `TRUNCATE`.
- It clears a table completely and almost instantly.
- It has the same effect as the `DELETE` statement without the `WHERE` clause. So, it's useful on large tables:



```
car_portal=> TRUNCATE TABLE car_portal_app.a;
TRUNCATE TABLE
```

- The `TABLE` keyword is optional.

- `TRUNCATE` can clear several tables at a time.
- To do so, use a list of tables in the command, as follows:



```
car_portal=> TRUNCATE car_portal_app.a, car_portal_app.b;
TRUNCATE TABLE
```

- If there are sequences used to auto generate values that are owned by fields of the table being cleared, the `TRUNCATE` command can reset those sequences.
- To do this, the following command could be used:
  - `TRUNCATE some_table RESTART IDENTITY;`

- If there are tables that have foreign keys that reference the table being cleared, those tables could also be cleared with the same command, as follows:
  - `TRUNCATE some_table CASCADE;`