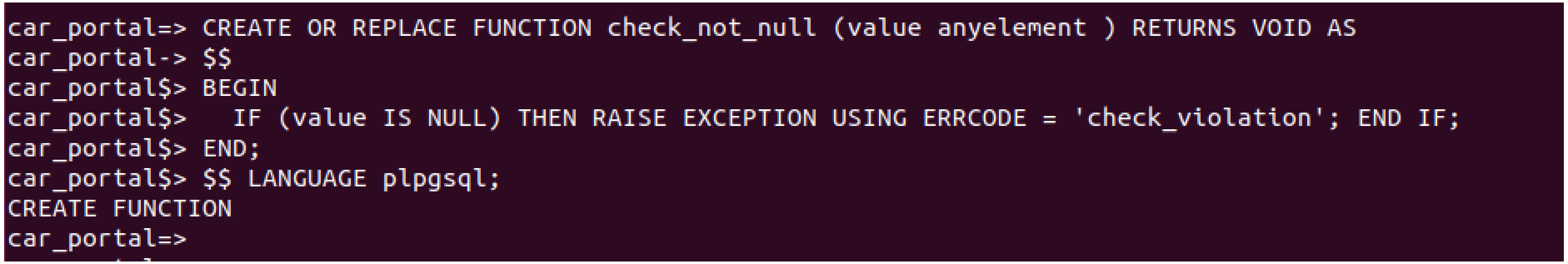
Exception Handling

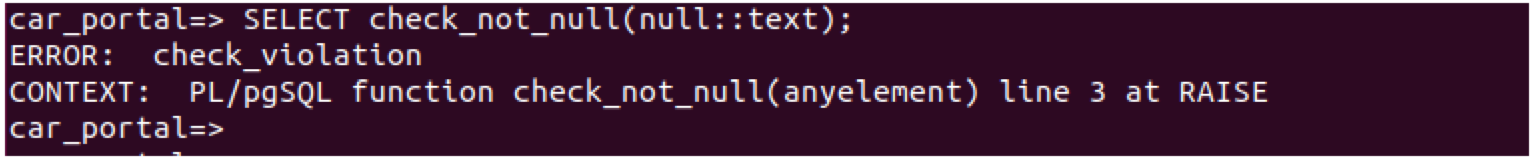
- You can trap and raise errors in PostgreSQL by using the `EXCEPTION` and `RAISE` statements.
- Errors can be raised by violating data integrity constraints or by performing illegal operations, such as assigning text to integers, dividing an integer or float by zero, and out-of-range assignments.
- By default, any error occurrences inside of a PL/pgSQL function cause the function to abort the execution and roll back the changes.
- To be able to recover from errors, PL/pgSQL can trap the errors, using the `EXCEPTION` clause.

- To understand exception handling, let's consider the following helping function:



```
car_portal=> CREATE OR REPLACE FUNCTION check_not_null (value anyelement ) RETURNS VOID AS
car_portal-> $$
car_portal$> BEGIN
car_portal$>   IF (value IS NULL) THEN RAISE EXCEPTION USING ERRCODE = 'check_violation'; END IF;
car_portal$> END;
car_portal$> $$ LANGUAGE plpgsql;
CREATE FUNCTION
car_portal=>
```

- The `check_not_null` statement is a `polymorphic` function, which simply raises an error with `check_violation` SQLSTATE.
- Calling this function and passing the `NULL` value as an argument will cause an error, as follows:



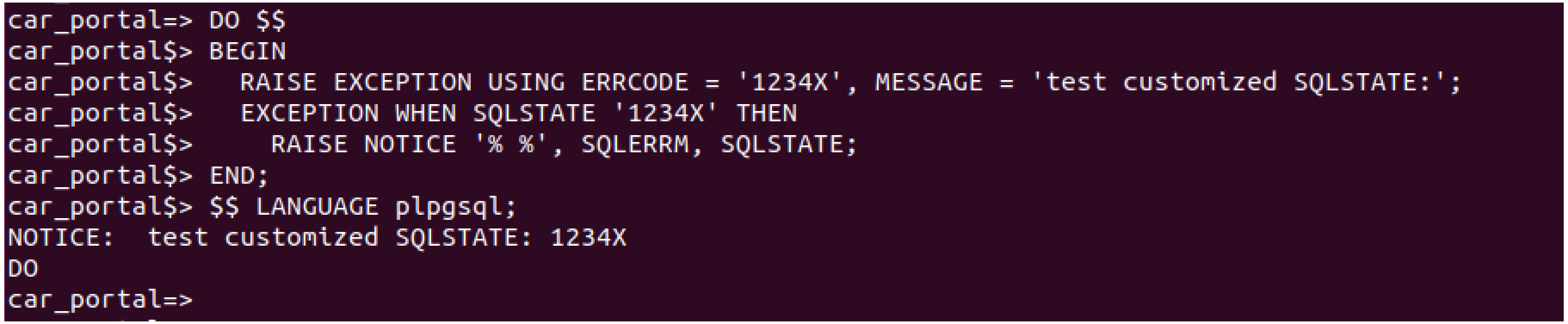
```
car_portal=> SELECT check_not_null(null::text);
ERROR:  check_violation
CONTEXT:  PL/pgSQL function check_not_null(anyelement) line 3 at RAISE
car_portal=>
```

- In order to properly determine when the exception is raised and why, PostgreSQL defines several categories of error codes.
- PostgreSQL error codes can be found at http://www.postgresql.org/docs/current/interactive/errcodes-appendix.html
- For example, raising an exception by the user without specifying `ERRCODE` will set the `SQLSTATE` to `P001`, while a unique violation exception will set `SQLSTATE` to `23505`.

- Errors can be matched in the `EXCEPTION` clause, by either `SQLSTATE` or the condition name, as follows:

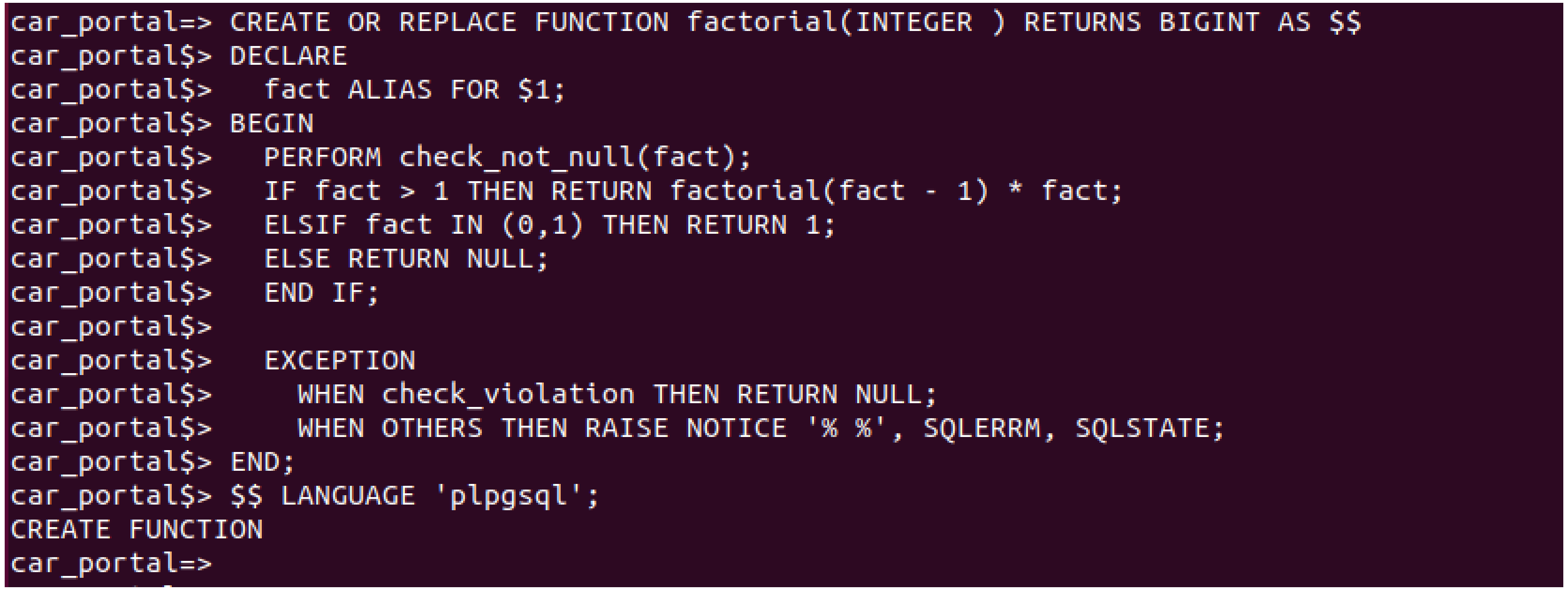
  ```
  WHEN unique_violation THEN ...
  WHEN SQLSTATE '23505' THEN ...
  ```

- Finally, you can provide a customized error message and SQLSTATE when raising an exception, so that ERRCODE is five digits and/or uppercase ASCII letters (other than 00000), as follows:



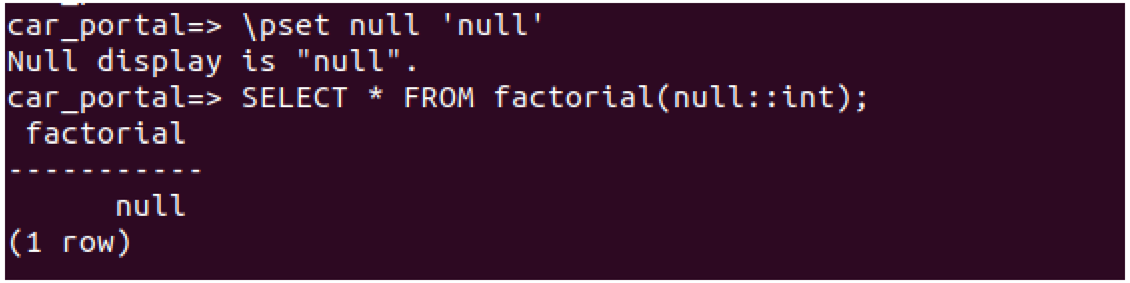
```
car_portal=> DO $$
car_portal$> BEGIN
car_portal$>   RAISE EXCEPTION USING ERRCODE = '1234X', MESSAGE = 'test customized SQLSTATE:';
car_portal$>   EXCEPTION WHEN SQLSTATE '1234X' THEN
car_portal$>     RAISE NOTICE '% %', SQLERRM, SQLSTATE;
car_portal$> END;
car_portal$> $$ LANGUAGE plpgsql;
NOTICE:  test customized SQLSTATE: 1234X
DO
car_portal=>
```

- To trap an exception, let's rewrite the `factorial` function, and let's suppose that the `factorial` function should return null if the provided argument is null:



```
car_portal=> CREATE OR REPLACE FUNCTION factorial(INTEGER ) RETURNS BIGINT AS $$
car_portal$> DECLARE
car_portal$>   fact ALIAS FOR $1;
car_portal$> BEGIN
car_portal$>   PERFORM check_not_null(fact);
car_portal$>   IF fact > 1 THEN RETURN factorial(fact - 1) * fact;
car_portal$>   ELSIF fact IN (0,1) THEN RETURN 1;
car_portal$>   ELSE RETURN NULL;
car_portal$>   END IF;
car_portal$>
car_portal$>   EXCEPTION
car_portal$>     WHEN check_violation THEN RETURN NULL;
car_portal$>     WHEN OTHERS THEN RAISE NOTICE '% %', SQLERRM, SQLSTATE;
car_portal$> END;
car_portal$> $$ LANGUAGE 'plpgsql';
CREATE FUNCTION
car_portal=>
```

- To test the function handling exception, let's call the function and pass a `NULL` value to it, as follows:



```
car_portal=> \pset null 'null'
Null display is "null".
car_portal=> SELECT * FROM factorial(null::int);
 factorial
-----------
      null
(1 row)
```

- In handling exceptions, if `SQLERRM` and `SQLSTATE` are not deterministic enough to know the exception cause, you can get more information about the exception by using `GET STACKED DIAGNOSTICS`:

```
GET STACKED DIAGNOSTICS variable { = | := } item [ , ... ];
```

  `item` is a keyword identifying a status value related to the exception.

- For example, the item keywords `COLUMN_NAME`, `TABLE_NAME`, and `SCHEMA_NAME` indicate the names of the column, table, and schema involved in the exception.