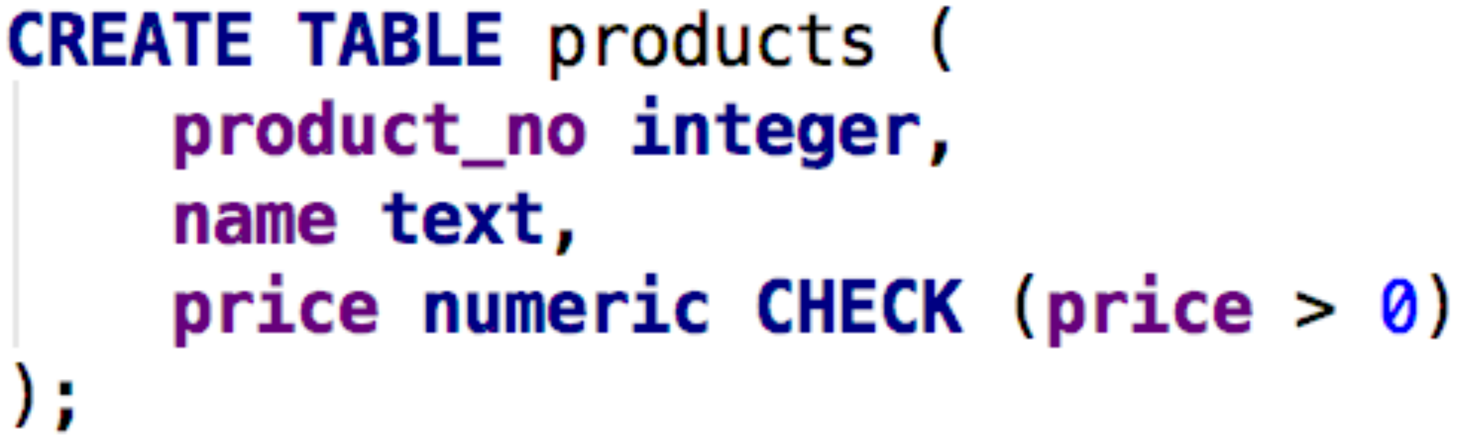
Constraints.

Overview

- Check constraints
- Not-Null constraints
- Unique constraints
- Primary keys
- Foreign keys

CHECK

- A check constraint is the most generic constraint type.
- It allows you to specify that the value in a certain column must satisfy a Boolean (truth-value) expression.

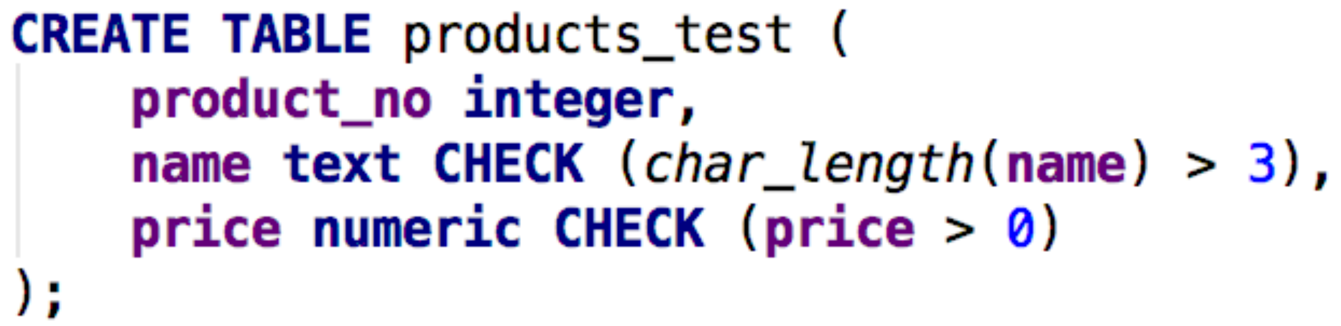


```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0)
);
```

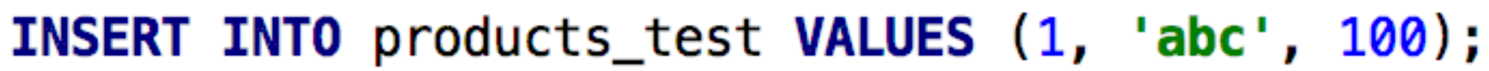
CHECK

- Constraint definition comes after the data type, just like default value definitions.
- Default values and constraints can be listed in any order.
- A check constraint consists of the key word CHECK followed by an expression in parentheses.
- The check constraint expression should involve the column thus constrained, otherwise the constraint would not make too much sense.

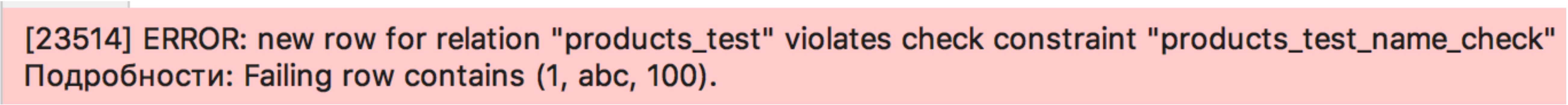
CHECK



```sql
CREATE TABLE products_test (
    product_no integer,
    name text CHECK (char_length(name) > 3),
    price numeric CHECK (price > 0)
);
```

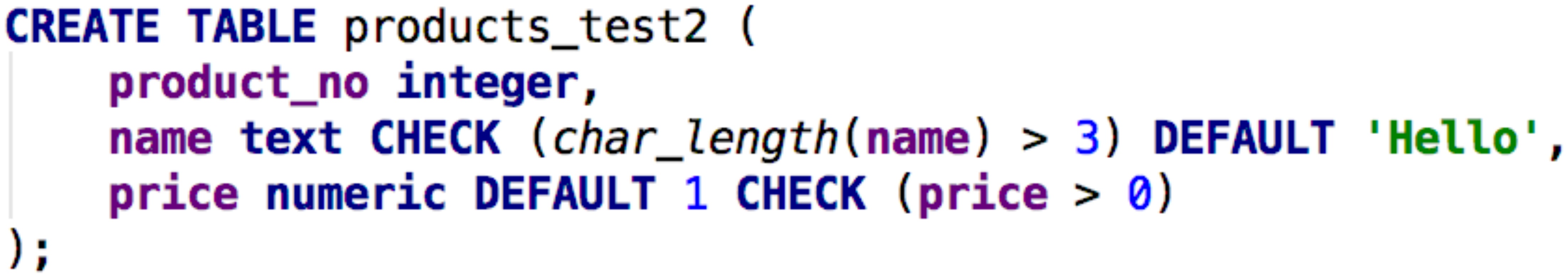


```sql
INSERT INTO products_test VALUES (1, 'abc', 100);
```



```
[23514] ERROR: new row for relation "products_test" violates check constraint "products_test_name_check"
Подробности: Failing row contains (1, abc, 100).
```

CHECK



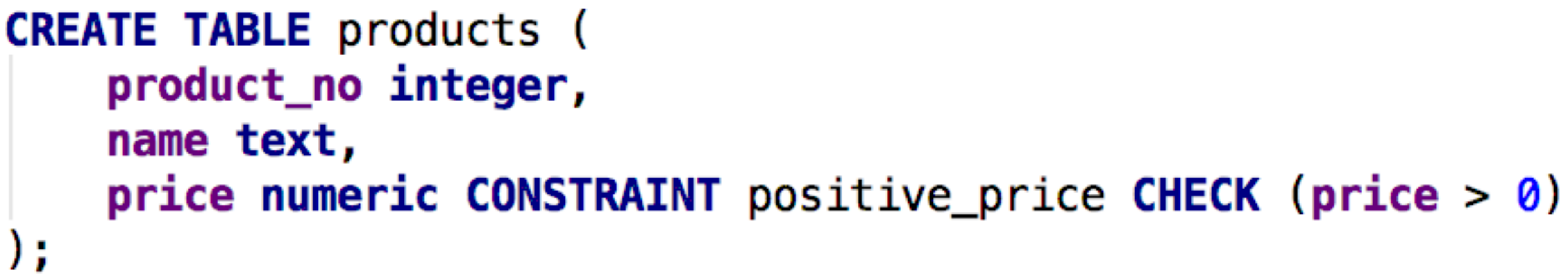
```
CREATE TABLE products_test2 (
    product_no integer,
    name text CHECK (char_length(name) > 3) DEFAULT 'Hello',
    price numeric DEFAULT 1 CHECK (price > 0)
);
```

CHECK

- You can also give the constraint a separate name.

- This clarifies error messages and allows you to refer to the constraint when you need to change it.

- The syntax is:



```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CONSTRAINT positive_price CHECK (price > 0)
);
```

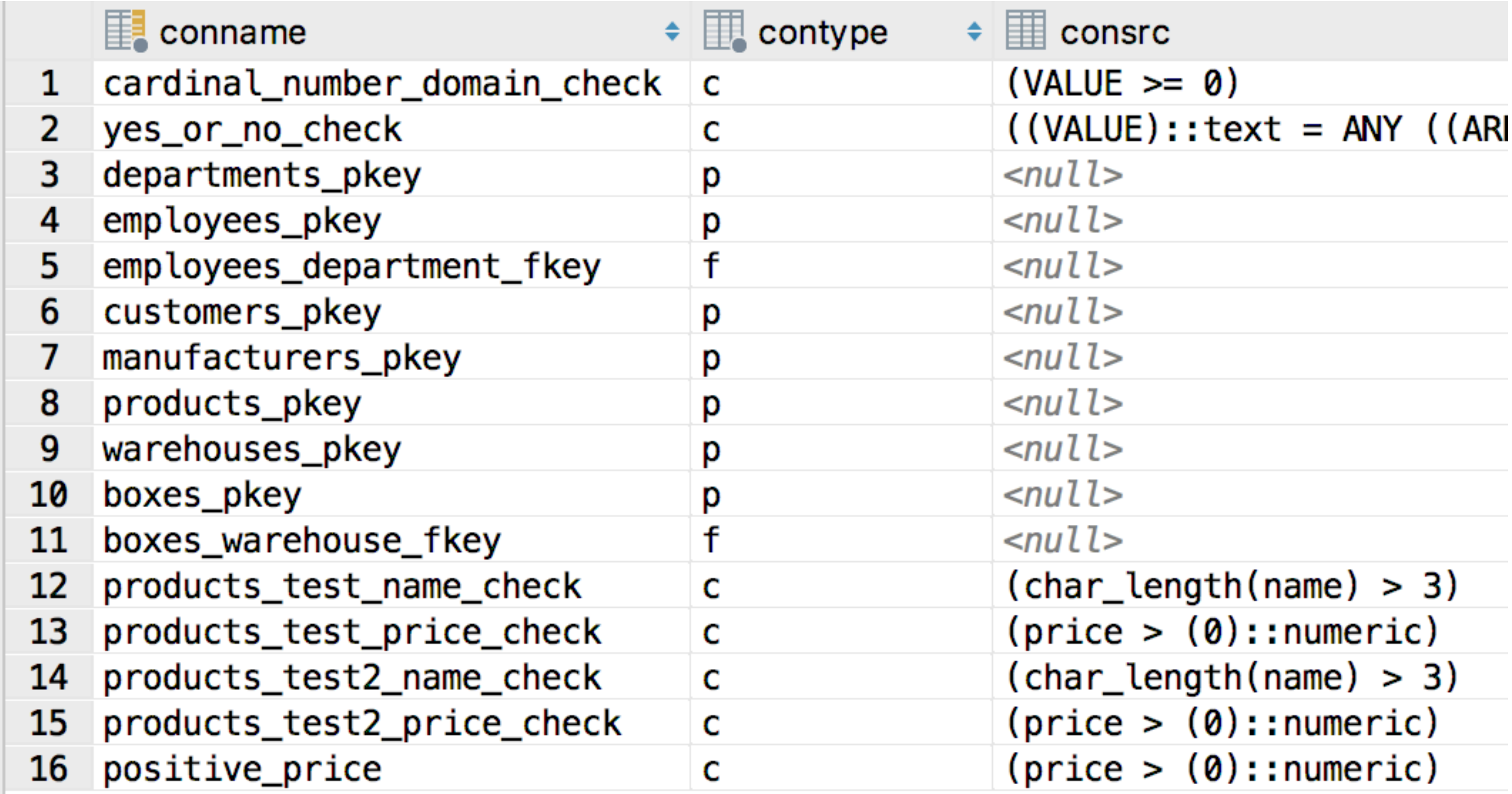
CHECK

- To specify a named constraint, use the key word CONSTRAINT followed by an identifier followed by the constraint definition.

- If you don't specify a constraint name in this way, the system chooses a name for you.

CHECK

```
SELECT conname, contype, consrc FROM pg_constraint;
```

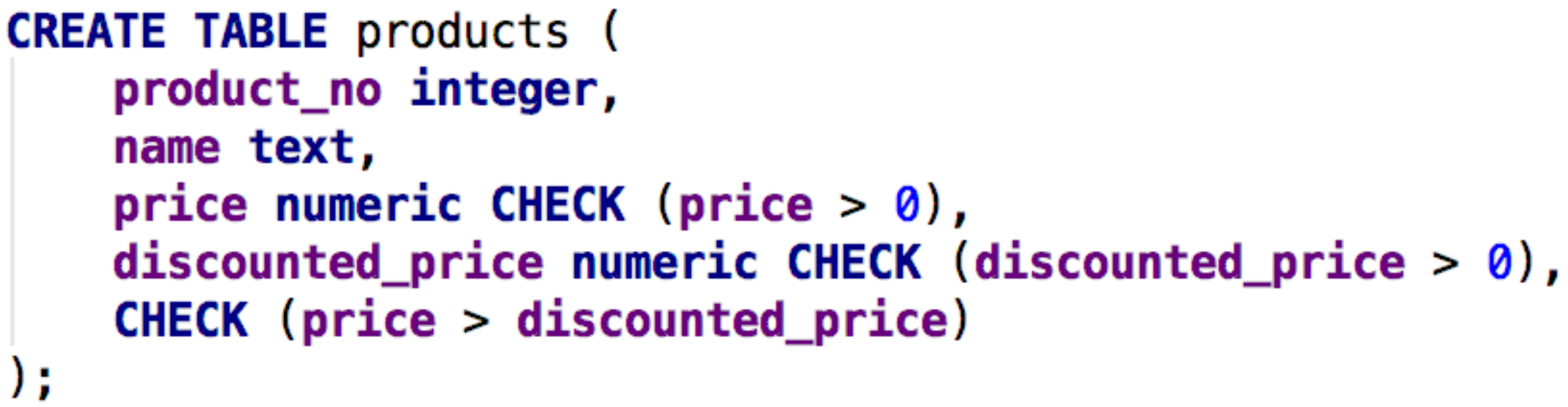


	conname	contype	consrc
1	cardinal_number_domain_check	c	(VALUE >= 0)
2	yes_or_no_check	c	((VALUE)::text = ANY ((AR
3	departments_pkey	p	<null>
4	employees_pkey	p	<null>
5	employees_department_fkey	f	<null>
6	customers_pkey	p	<null>
7	manufacturers_pkey	p	<null>
8	products_pkey	p	<null>
9	warehouses_pkey	p	<null>
10	boxes_pkey	p	<null>
11	boxes_warehouse_fkey	f	<null>
12	products_test_name_check	c	(char_length(name) > 3)
13	products_test_price_check	c	(price > (0)::numeric)
14	products_test2_name_check	c	(char_length(name) > 3)
15	products_test2_price_check	c	(price > (0)::numeric)
16	positive_price	c	(price > (0)::numeric)

CHECK

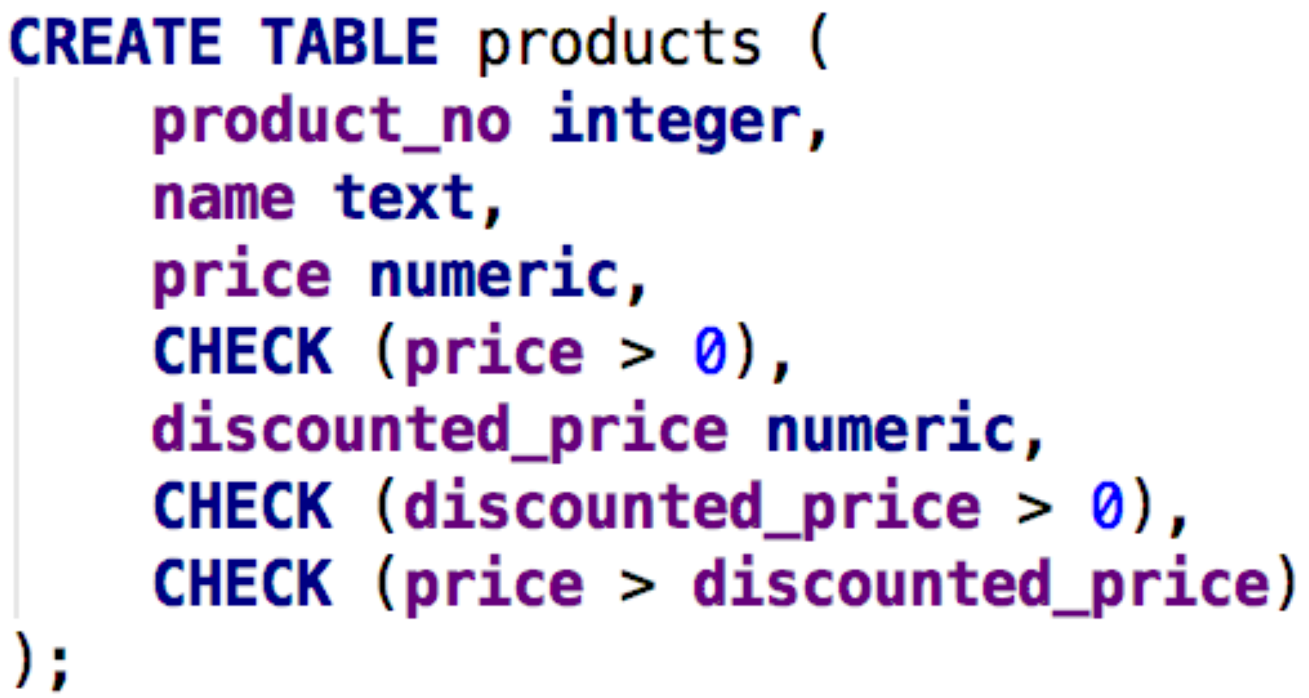
- A check constraint can also refer to several columns.

- Say you store a regular price and a discounted price, and you want to ensure that the discounted price is lower than the regular price:



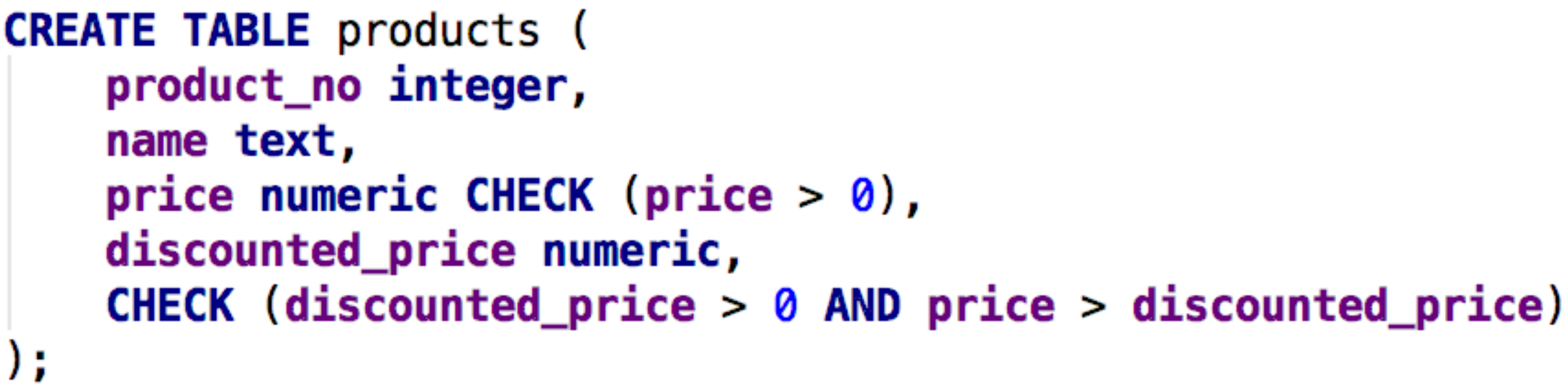
```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0),
    discounted_price numeric CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);
```

CHECK



```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    CHECK (price > 0),
    discounted_price numeric,
    CHECK (discounted_price > 0),
    CHECK (price > discounted_price)
);
```

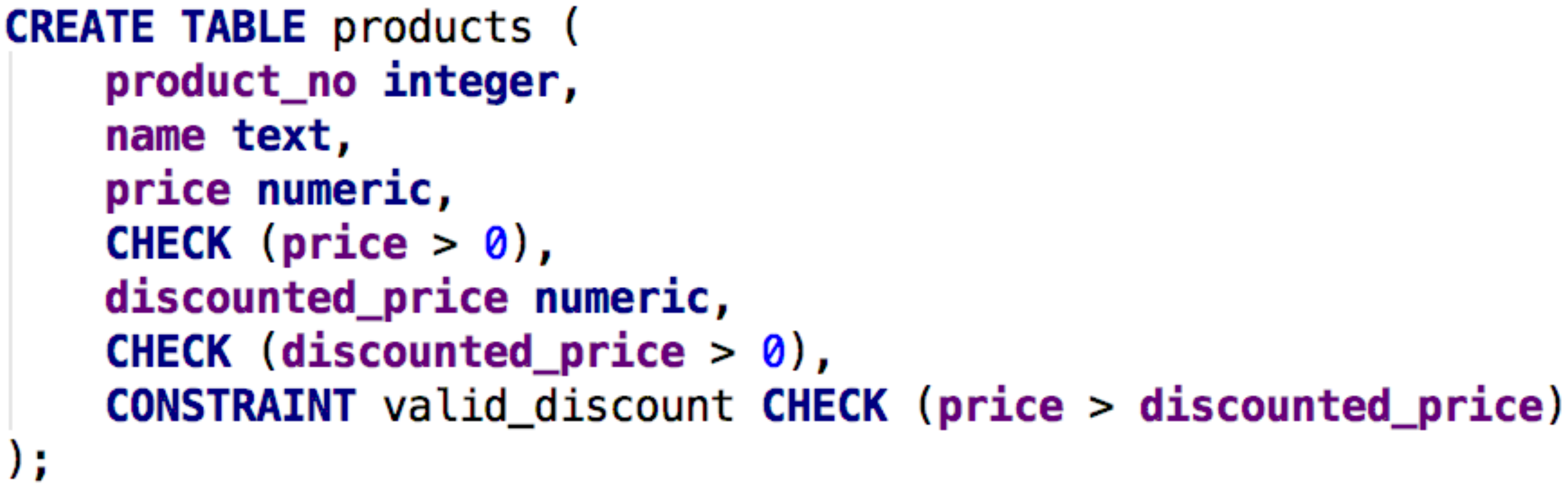
CHECK



```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric CHECK (price > 0),
    discounted_price numeric,
    CHECK (discounted_price > 0 AND price > discounted_price)
);
```

CHECK

- Names can be assigned to table constraints in the same way as column constraints:

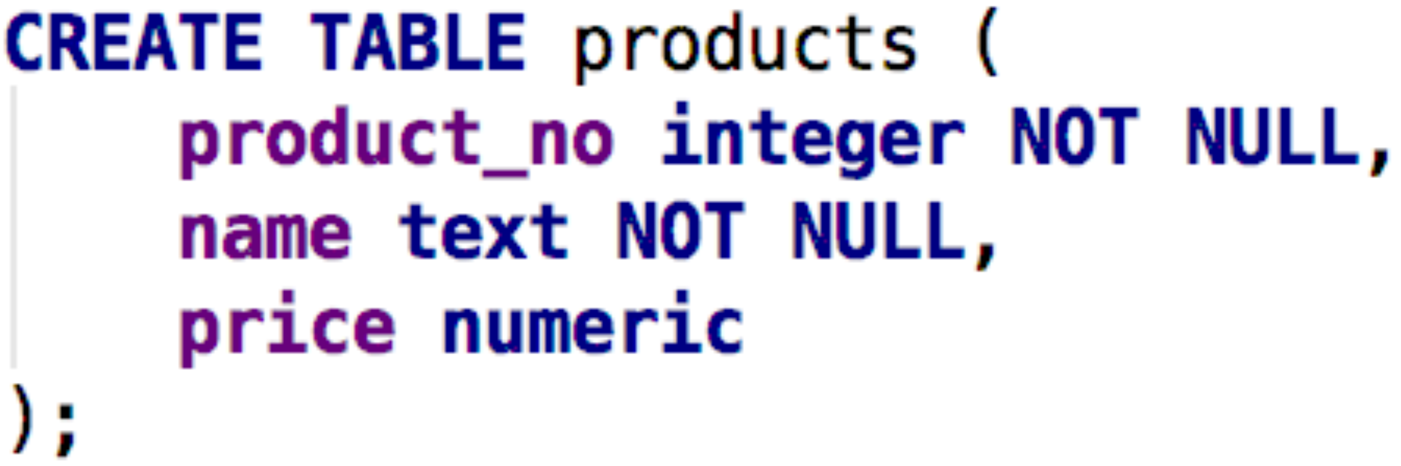


```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    CHECK (price > 0),
    discounted_price numeric,
    CHECK (discounted_price > 0),
    CONSTRAINT valid_discount CHECK (price > discounted_price)
);
```

Not-Null

- A not-null constraint simply specifies that a column must not assume the null value.

- A syntax example:



```
CREATE TABLE products (
    product_no integer NOT NULL,
    name text NOT NULL,
    price numeric
);
```

Not-Null

- A not-null constraint is always written as a column constraint.

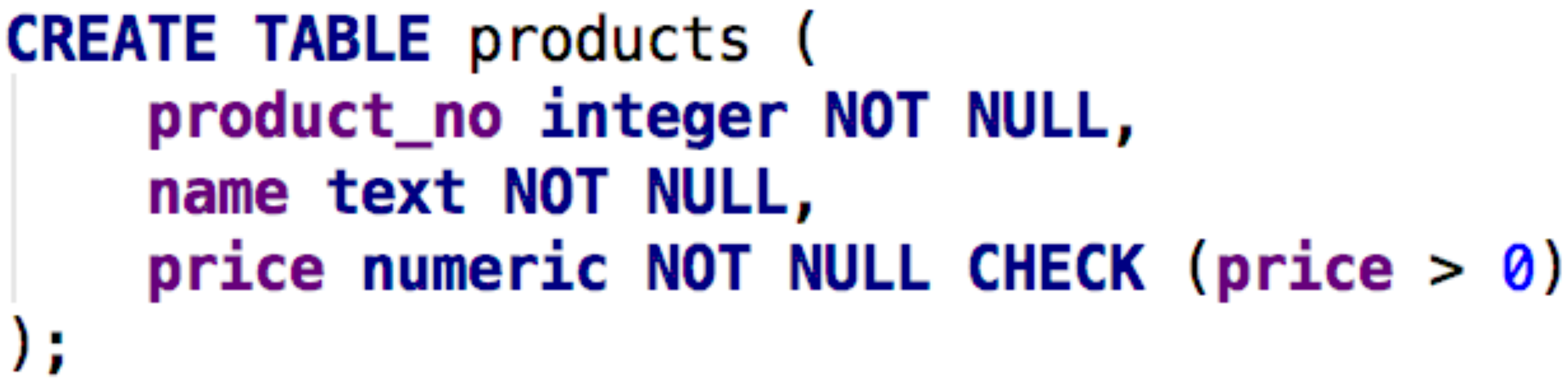
- A not-null constraint is functionally equivalent to creating a check constraint CHECK (column_name IS NOT NULL)

- But in PostgreSQL creating an explicit not-null constraint is more efficient. T

- he drawback is that you cannot give explicit names to not-null constraints created this way.

Not-Null

- Column can have more than one constraint.
- Just write the constraints one after another:

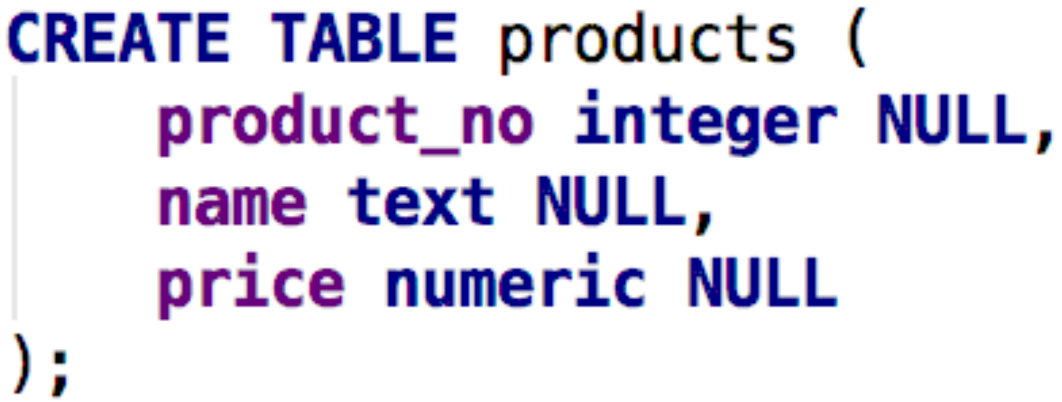


```
CREATE TABLE products (
    product_no integer NOT NULL,
    name text NOT NULL,
    price numeric NOT NULL CHECK (price > 0)
);
```

Not-Null

- The NOT NULL constraint has an inverse: the NULL constraint.

- This simply selects the default behavior that the column might be null.

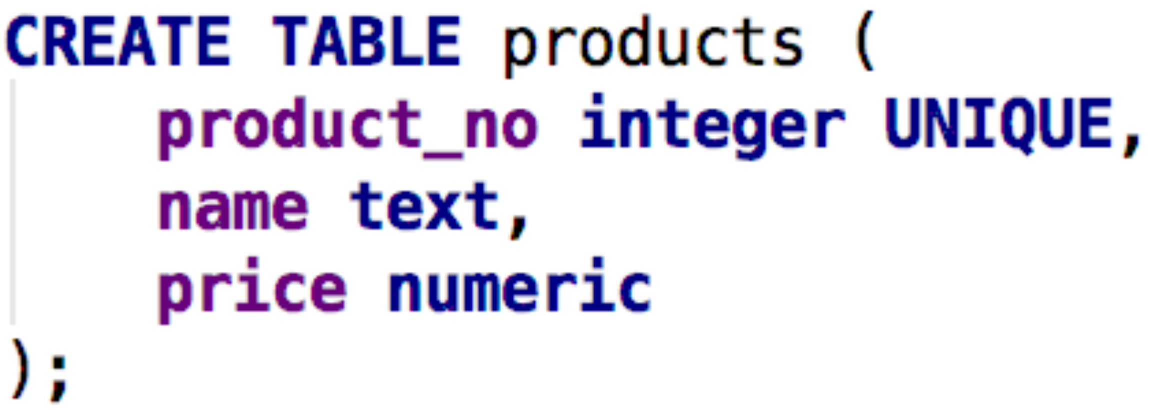


```
CREATE TABLE products (
    product_no integer NULL,
    name text NULL,
    price numeric NULL
);
```

Unique

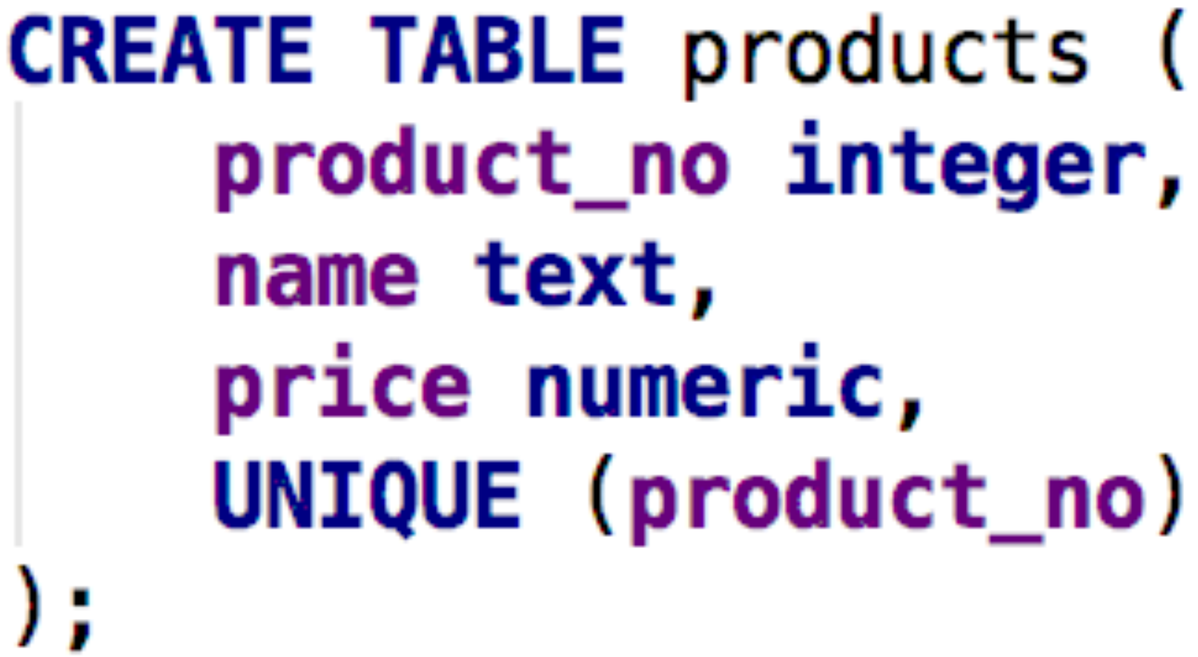
- Unique constraints ensure that the data contained in a column, or a group of columns, is unique among all the rows in the table.

- The syntax is:



```
CREATE TABLE products (
    product_no integer UNIQUE,
    name text,
    price numeric
);
```

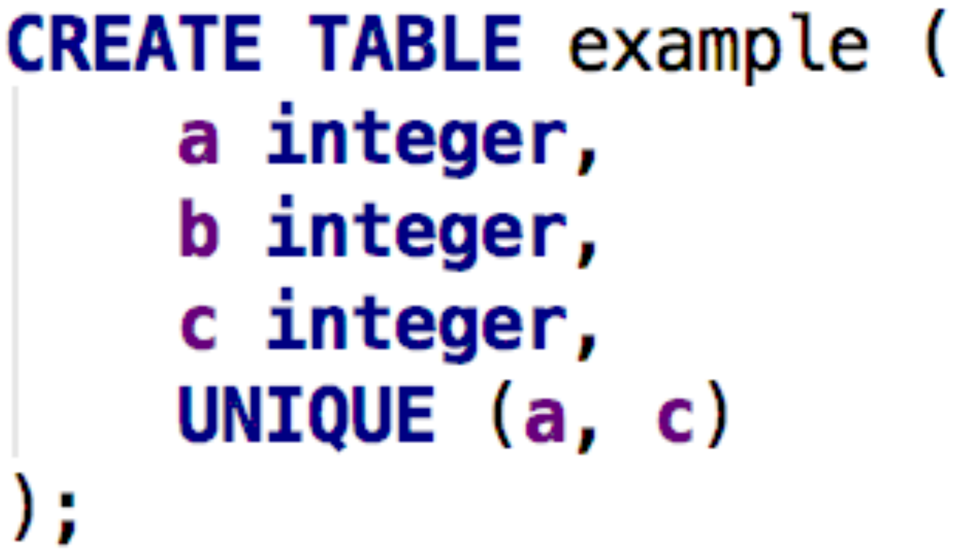
Unique



```
CREATE TABLE products (
    product_no integer,
    name text,
    price numeric,
    UNIQUE (product_no)
);
```

Unique

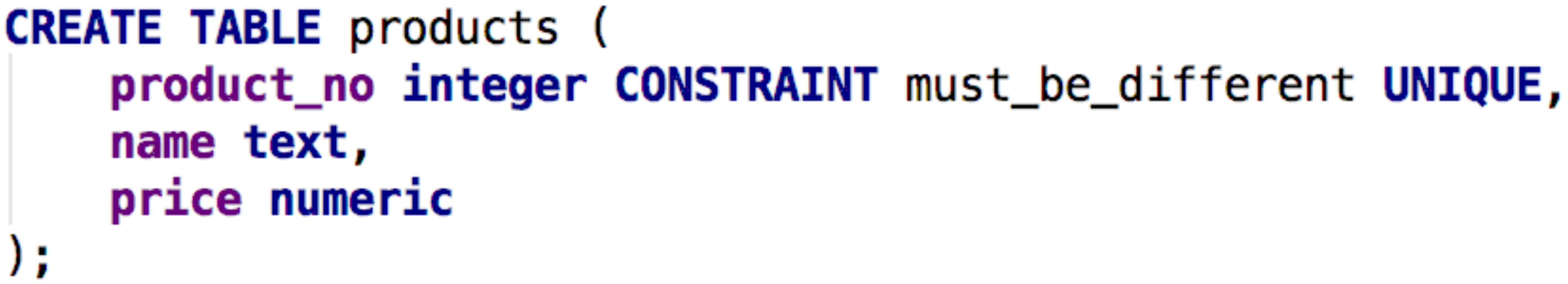
- To define a unique constraint for a group of columns, write it as a table constraint with the column names separated by commas:



```
CREATE TABLE example (
    a integer,
    b integer,
    c integer,
    UNIQUE (a, c)
);
```

Unique

- You can assign your own name for a unique constraint, in the usual way:



```
CREATE TABLE products (
    product_no integer CONSTRAINT must_be_different UNIQUE,
    name text,
    price numeric
);
```

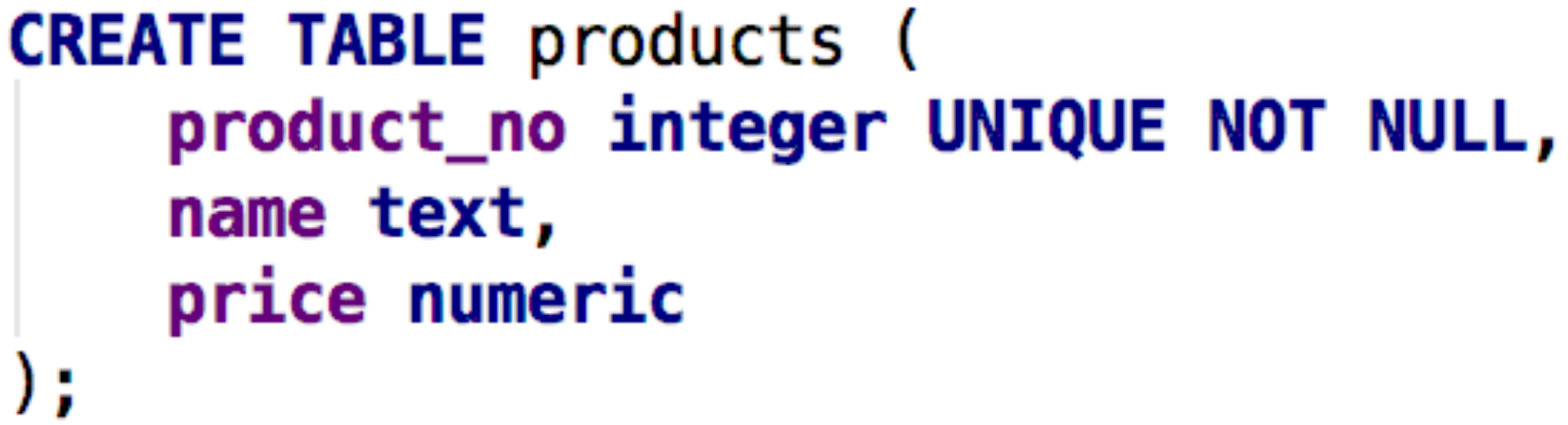
Primary key

- A primary key constraint indicates that a column, or group of columns, can be used as a unique identifier for rows in the table.

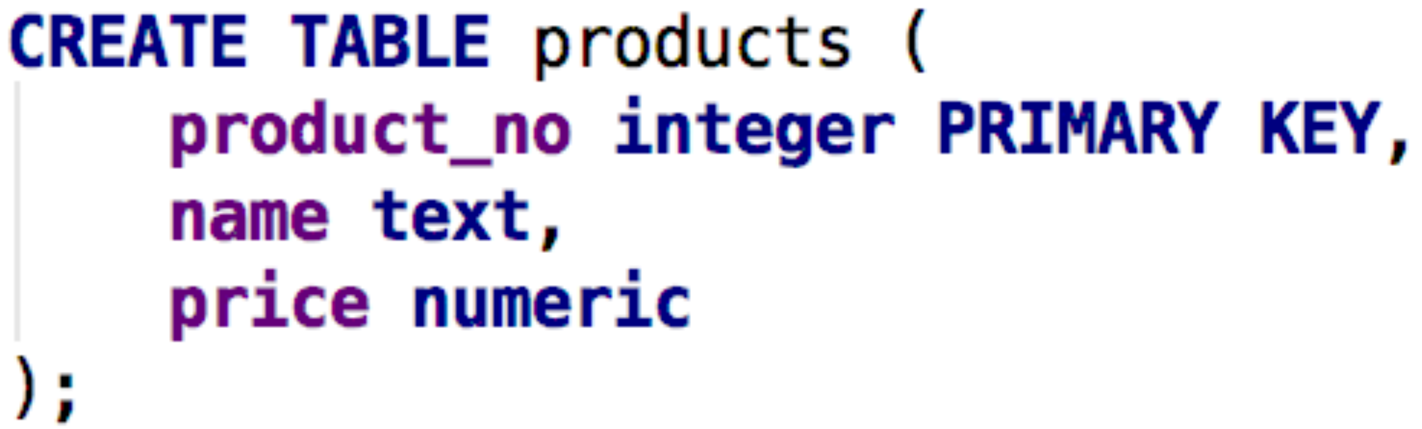
- This requires that the values be both unique and not null.

- So, the following two table definitions accept the same data:

Primary key



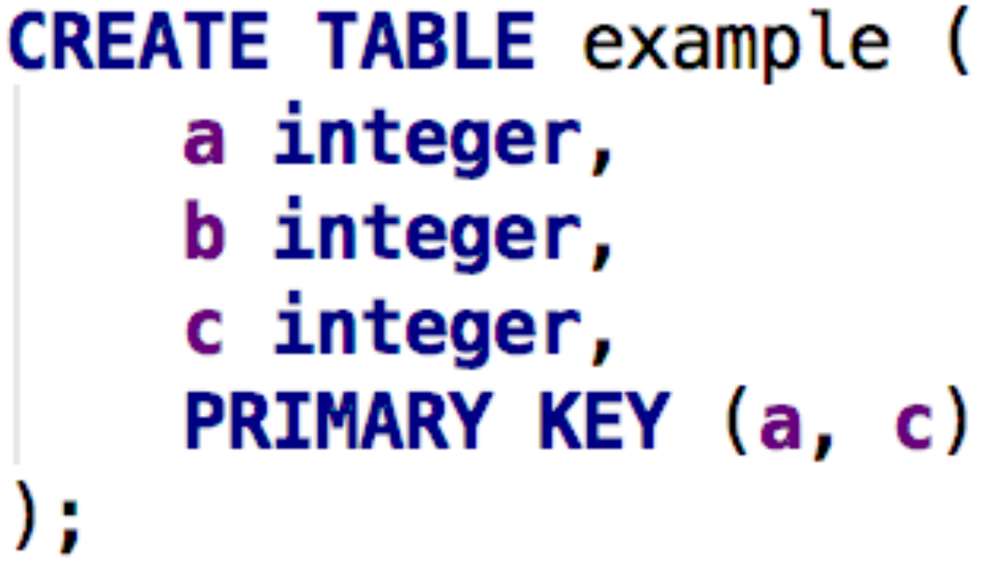
```
CREATE TABLE products (
    product_no integer UNIQUE NOT NULL,
    name text,
    price numeric
);
```



```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
```

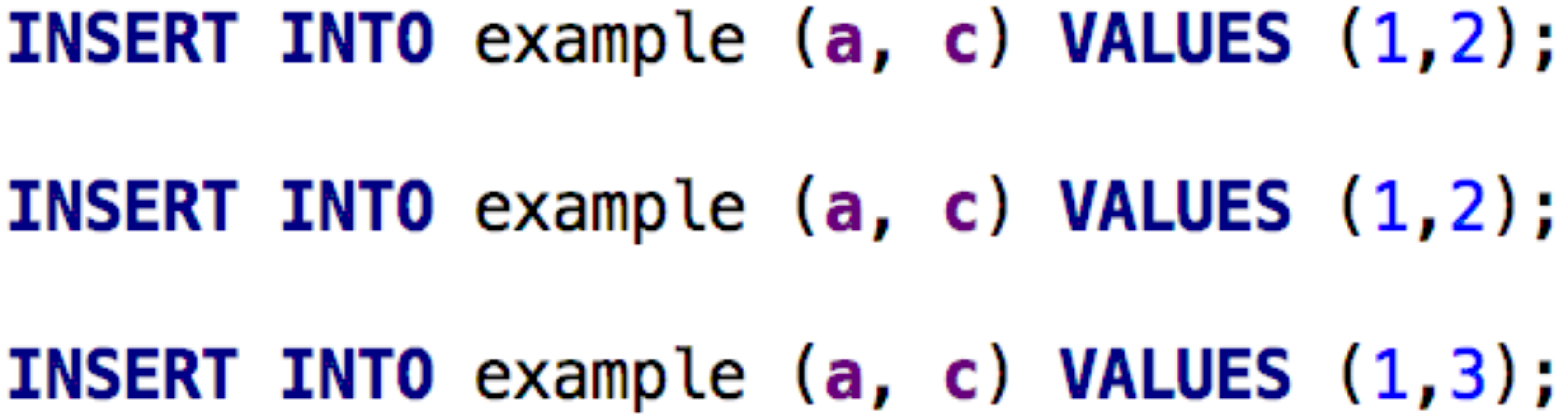
Primary key

- Primary keys can span more than one column; the syntax is similar to unique constraints:



```
CREATE TABLE example (
    a integer,
    b integer,
    c integer,
    PRIMARY KEY (a, c)
);
```

Primary key



```
INSERT INTO example (a, c) VALUES (1,2); 🟢
INSERT INTO example (a, c) VALUES (1,2); 🔴
INSERT INTO example (a, c) VALUES (1,3); 🟢
```

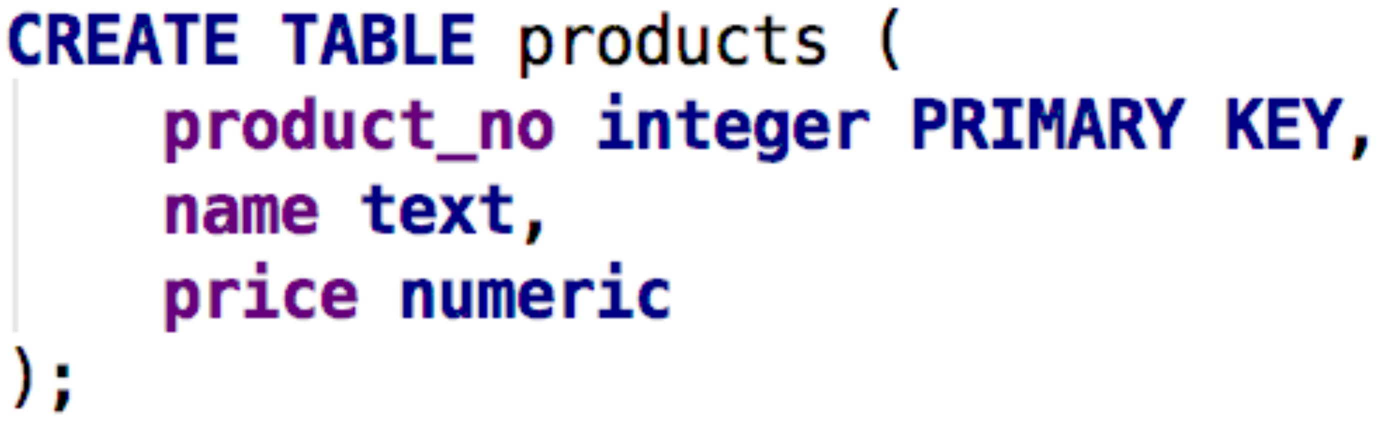
Foreign keys

- A foreign key constraint specifies that the values in a column (or a group of columns) must match the values appearing in some row of another table.

- We say this maintains the *referential integrity* between two related tables.

Foreign keys

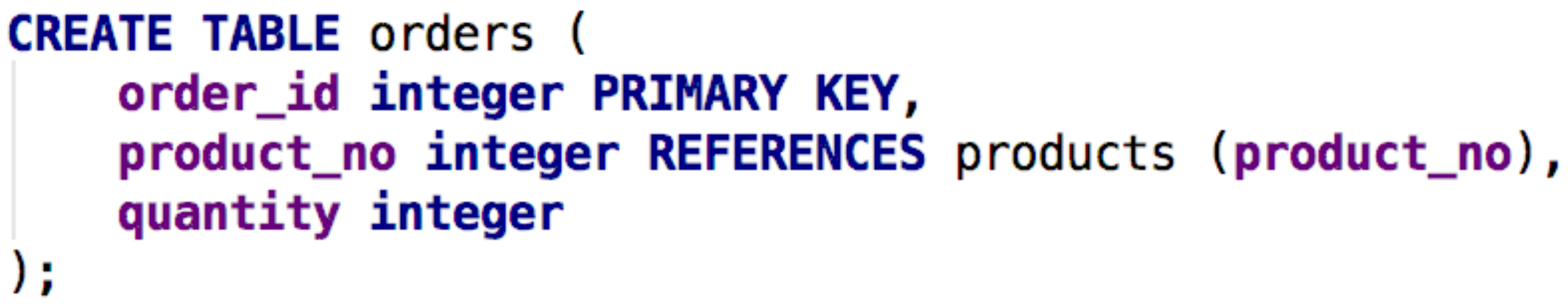
- Say you have the product table that we have used several times already:



```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);
```

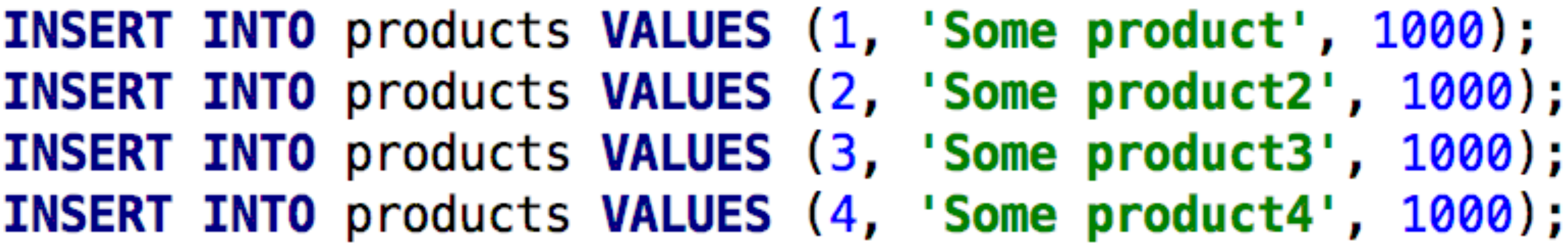
Foreign keys

- Let's also assume you have a table storing orders of those products.



```
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products (product_no),
    quantity integer
);
```

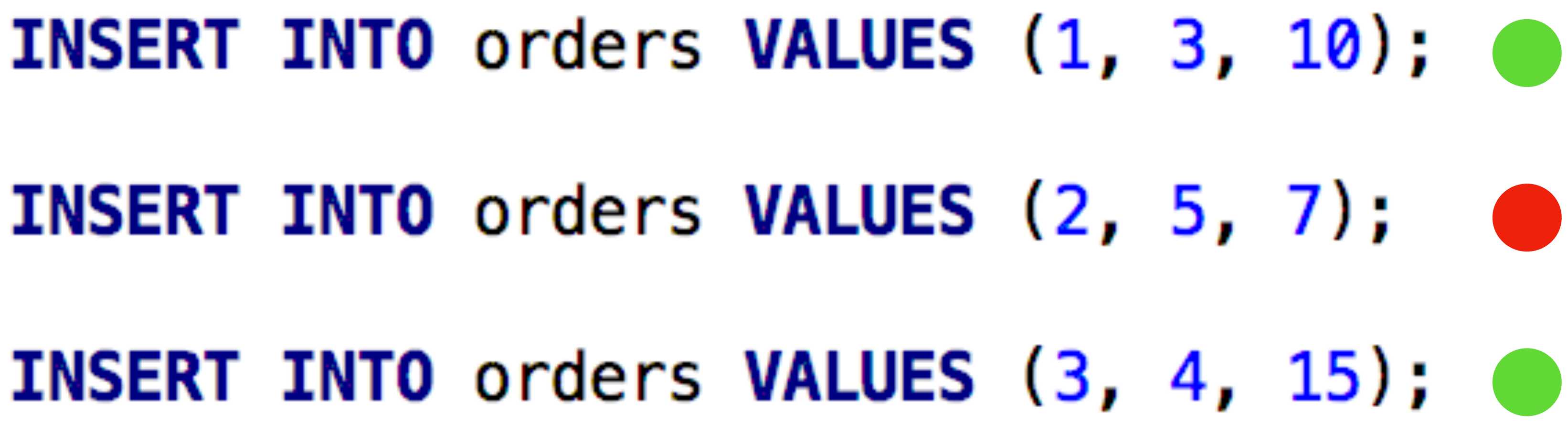
Foreign keys



```
INSERT INTO products VALUES (1, 'Some product', 1000);
INSERT INTO products VALUES (2, 'Some product2', 1000);
INSERT INTO products VALUES (3, 'Some product3', 1000);
INSERT INTO products VALUES (4, 'Some product4', 1000);
```

Foreign keys

- Now it is impossible to create orders with non-NULL product_no entries that do not appear in the products table.



```
INSERT INTO orders VALUES (1, 3, 10);  🟢
INSERT INTO orders VALUES (2, 5, 7);   🔴
INSERT INTO orders VALUES (3, 4, 15);  🟢
```

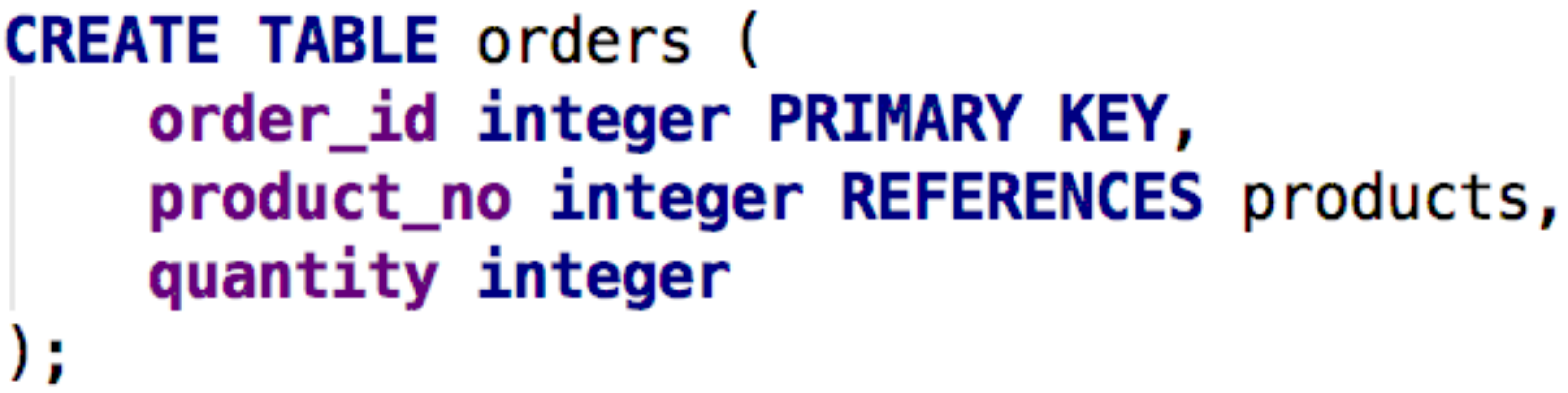
Foreign keys

- We say that in this situation the orders table is the referencing table and the products table is the referenced table.

- Similarly, there are referencing and referenced columns.

Foreign keys

- You can also shorten the above command to:



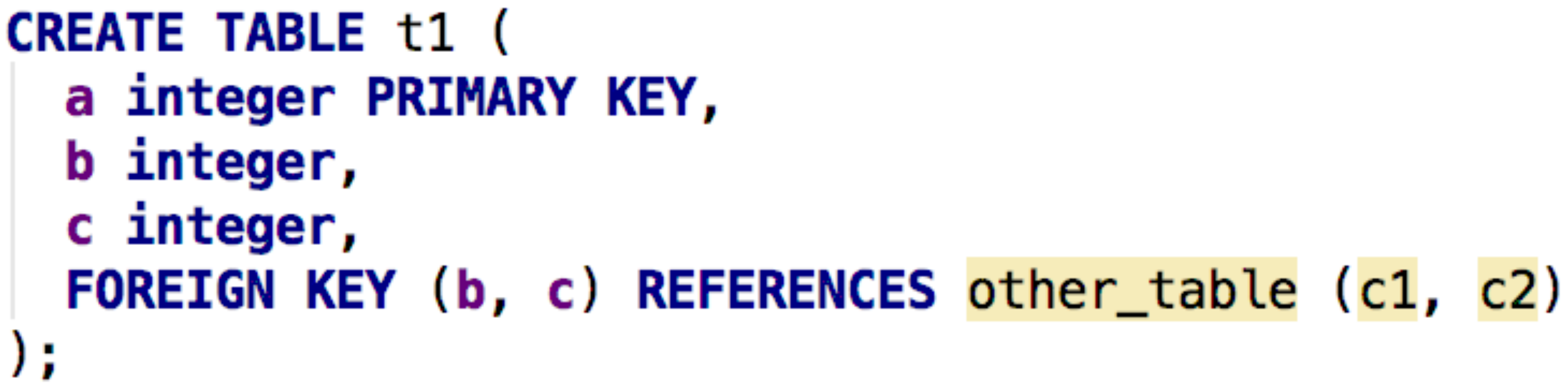
```
CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    product_no integer REFERENCES products,
    quantity integer
);
```

Foreign keys

- A foreign key can also constrain and reference a group of columns.

- As usual, it then needs to be written in table constraint form.

- Here is a contrived syntax example:

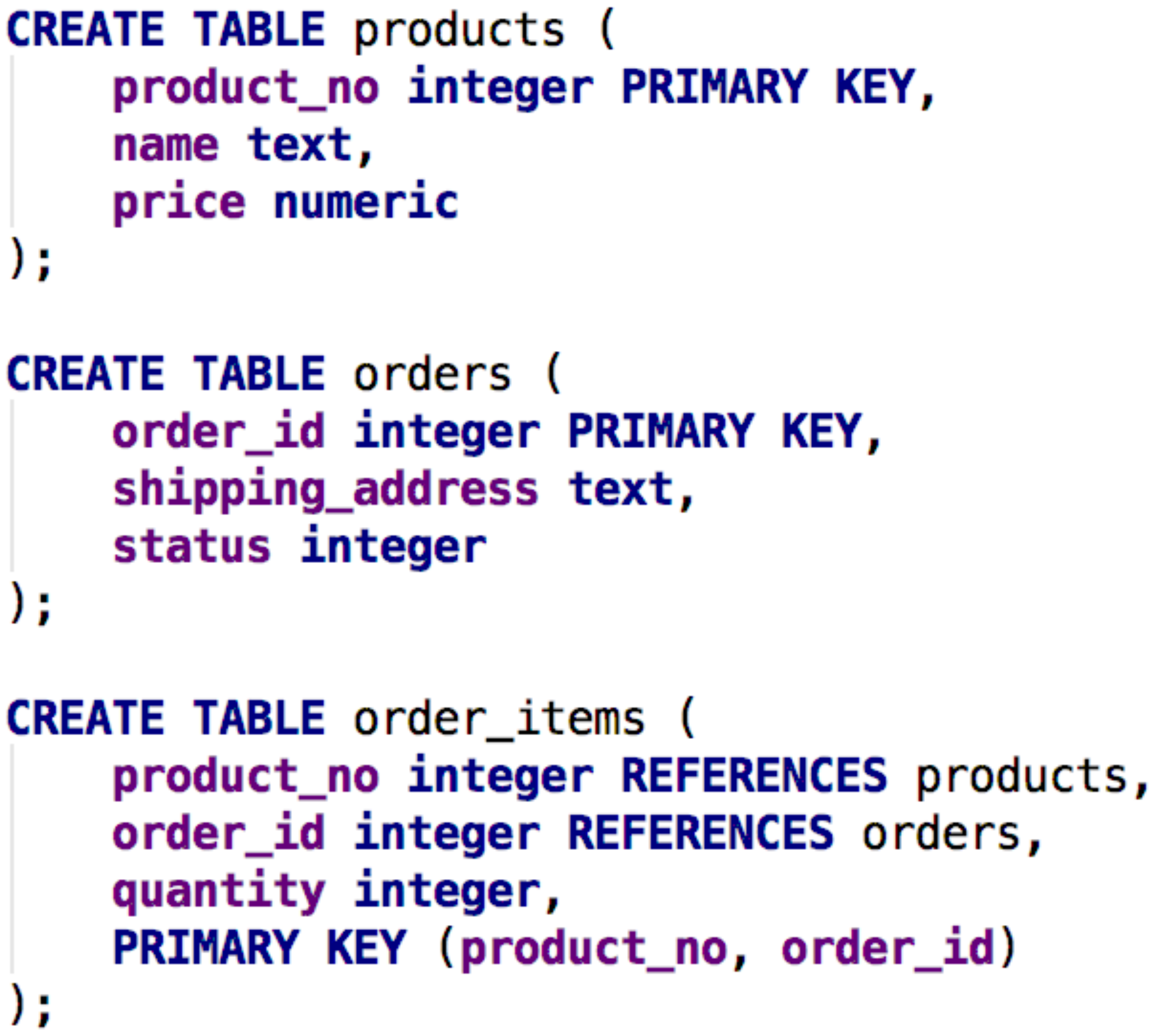


```
CREATE TABLE t1 (
  a integer PRIMARY KEY,
  b integer,
  c integer,
  FOREIGN KEY (b, c) REFERENCES other_table (c1, c2)
);
```

Foreign keys

- A table can have more than one foreign key constraint.
- This is used to implement many-to-many relationships between tables.
- Say you have tables about products and orders, but now you want to allow one order to contain possibly many products

Foreign keys



```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    shipping_address text,
    status integer
);

CREATE TABLE order_items (
    product_no integer REFERENCES products,
    order_id integer REFERENCES orders,
    quantity integer,
    PRIMARY KEY (product_no, order_id)
);
```

Foreign keys

- Notice that the primary key overlaps with the foreign keys in the last table.

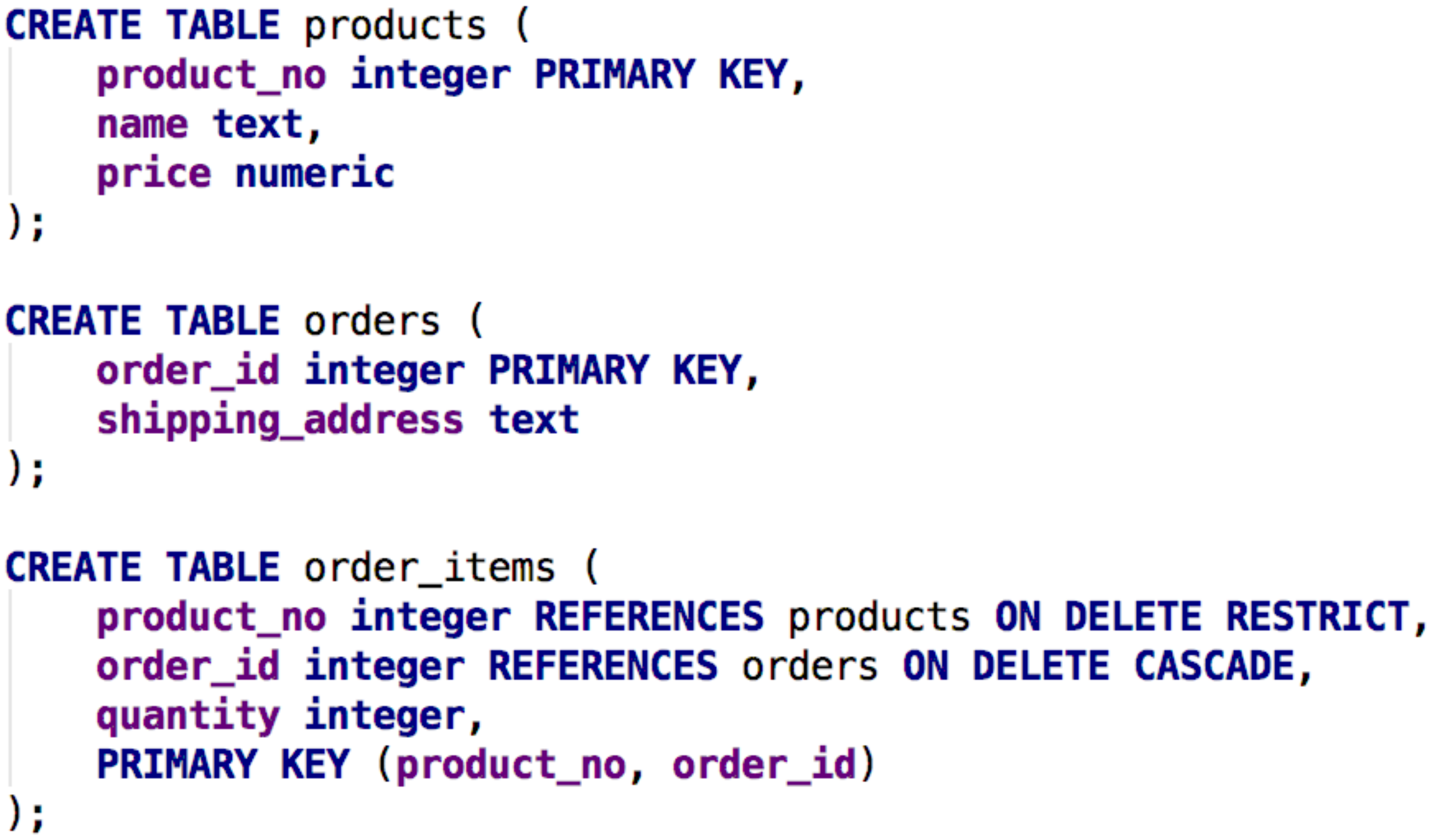
- We know that the foreign keys disallow creation of orders that do not relate to any products.

- But what if a product is removed after an order is created that references it?

Foreign keys

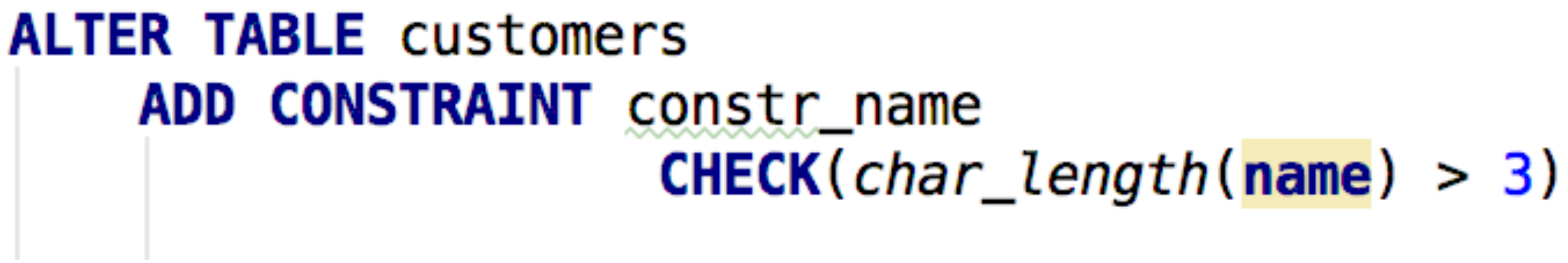
- Disallow deleting a referenced product
- Delete the orders as well
- Something else?

Foreign keys

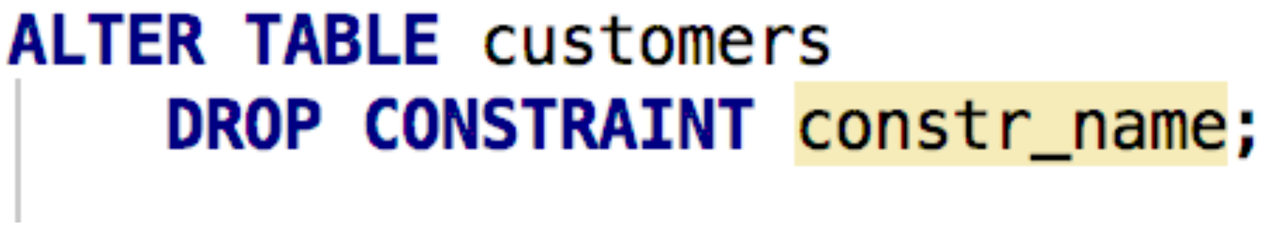


```
CREATE TABLE products (
    product_no integer PRIMARY KEY,
    name text,
    price numeric
);

CREATE TABLE orders (
    order_id integer PRIMARY KEY,
    shipping_address text
);

CREATE TABLE order_items (
    product_no integer REFERENCES products ON DELETE RESTRICT,
    order_id integer REFERENCES orders ON DELETE CASCADE,
    quantity integer,
    PRIMARY KEY (product_no, order_id)
);
```

Add constraint



```
ALTER TABLE customers
    ADD CONSTRAINT constr_name
                    CHECK(char_length(name) > 3)
```



```
ALTER TABLE customers
    DROP CONSTRAINT constr_name;
```

Questions?