

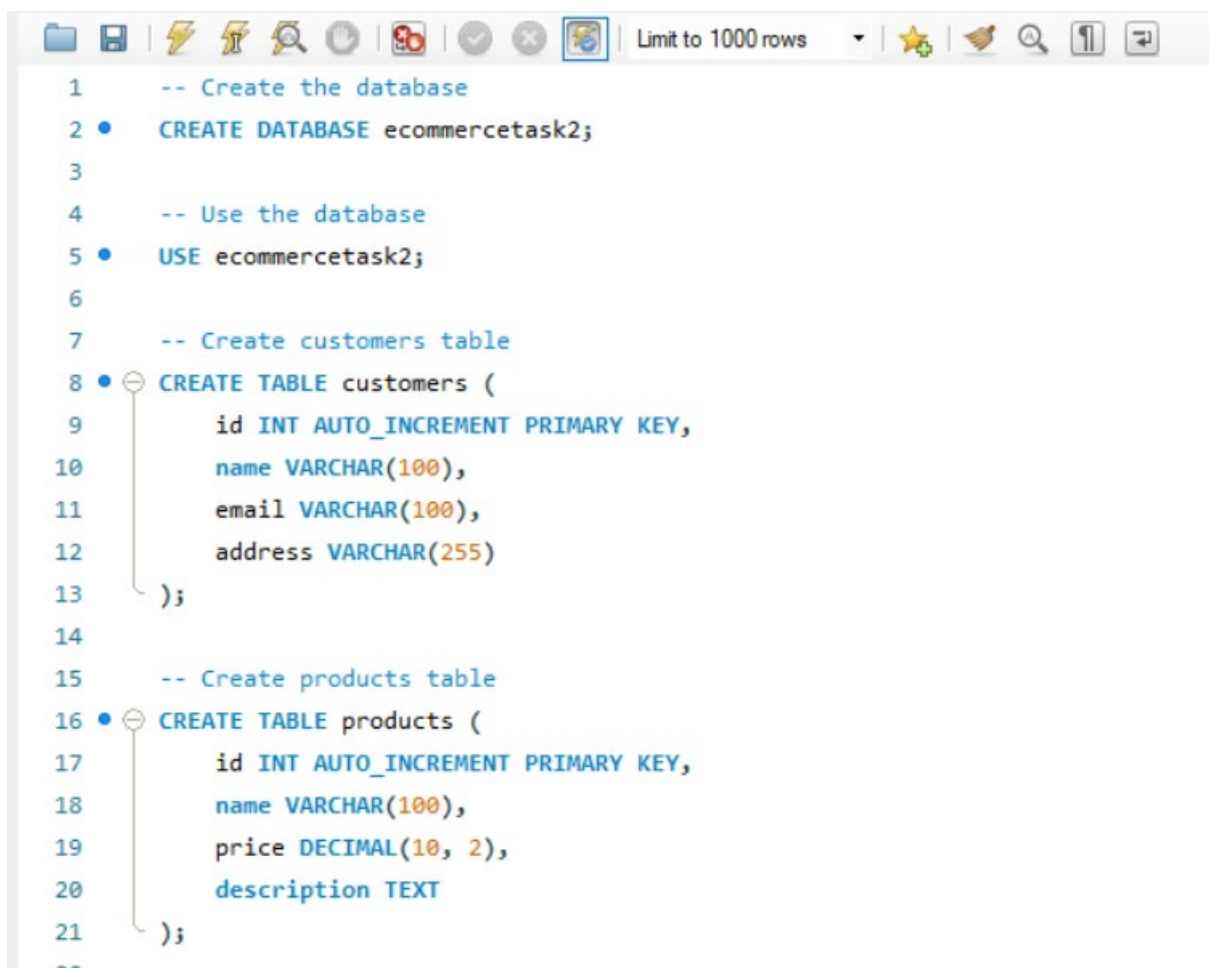
SQL Task

Task:

Create a database and tables to manage a simple e-commerce system.
The system should have three tables: customers, orders, and products.

Requirements:

- Create a database named ecommerce.
- Create three tables: customers, orders, and products.
- Insert some sample data into the tables.



```
1  -- Create the database
2  • CREATE DATABASE ecommerce2;
3
4  -- Use the database
5  • USE ecommerce2;
6
7  -- Create customers table
8  • CREATE TABLE customers (
9      id INT AUTO_INCREMENT PRIMARY KEY,
10     name VARCHAR(100),
11     email VARCHAR(100),
12     address VARCHAR(255)
13 );
14
15 -- Create products table
16 • CREATE TABLE products (
17     id INT AUTO_INCREMENT PRIMARY KEY,
18     name VARCHAR(100),
19     price DECIMAL(10, 2),
20     description TEXT
21 );
22
23
```

```

22
23      -- Create orders table
24  ● ○ CREATE TABLE orders (
25      id INT AUTO_INCREMENT PRIMARY KEY,
26      customer_id INT,
27      order_date DATE,
28      total_amount DECIMAL(10, 2),
29      FOREIGN KEY (customer_id) REFERENCES customers(id)
30  );
31
32
33
34
35
36
37
38
39
40

```

Output :

📄 Action Output ▾

	#	Time	Action
✓	1	19:18:25	CREATE DATABASE ecommerceTask2
✓	2	19:18:25	USE ecommerceTask2
✓	3	19:18:25	CREATE TABLE customers (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100), email V...
✓	4	19:18:25	CREATE TABLE products (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(100), price DE...
✓	5	19:18:25	CREATE TABLE orders (id INT AUTO_INCREMENT PRIMARY KEY, customer_id INT, order_date DAT...

```
1  -- Insert customers
2  • INSERT INTO customers (name, email, address) VALUES
3    ('Alice', 'alice@example.com', '123 Apple St'),
4    ('Bob', 'bob@example.com', '456 Banana Ave'),
5    ('Charlie', 'charlie@example.com', '789 Cherry Blvd');
6
7  -- Insert products
8  • INSERT INTO products (name, price, description) VALUES
9    ('Product A', 50.00, 'Description A'),
10   ('Product B', 100.00, 'Description B'),
11   ('Product C', 40.00, 'Description C'),
12   ('Product D', 200.00, 'Description D'),
13   ('Product E', 150.00, 'Description E');
14
15  -- Insert orders
16  • INSERT INTO orders (customer_id, order_date, total_amount) VALUES
17    (1, CURDATE(), 150.00),
18    (2, CURDATE() - INTERVAL 10 DAY, 250.00),
19    (1, CURDATE() - INTERVAL 40 DAY, 75.00),
20    (3, CURDATE() - INTERVAL 5 DAY, 180.00);
21
22
```

Output

Action Output

#	Time	Action
✓ 1	19:21:24	INSERT INTO customers (name, email, address) VALUES ('Alice', 'alice@example.com', '123 Apple St'), ('Bob', b...
✓ 2	19:21:24	INSERT INTO products (name, price, description) VALUES ('Product A', 50.00, 'Description A'), ('Product B', 100....
✓ 3	19:21:24	INSERT INTO orders (customer_id, order_date, total_amount) VALUES (1, CURDATE(), 150.00), (2, CURDATE() ...

Table Structure:

Customers

Id (primary key, auto-increment): unique identifier for each customer
Name: customer's name
Email: customer's email address
Address: customer's address

Orders

Id (primary key, auto-increment): unique identifier for each order
customer_id (foreign key referencing customers.id): a customer who placed the order
order_date: date the order was placed
total_amount: total amount of the order

Products

Id (primary key, auto-increment): unique identifier for each product
Name: product's name

Price: product's price
Description: product's description

Queries to Write:

- Retrieve all customers who have placed an order in the last 30 days.

The screenshot shows a database query editor interface. The top toolbar includes icons for file operations, execution, and settings, along with a 'Limit to 1000 rows' dropdown. The SQL editor contains the following query:

```
1  -- Retrieve all customers who have placed an order in the last 30 days
2  •  SELECT DISTINCT c.*
3     FROM customers c
4     JOIN orders o ON c.id = o.customer_id
5     WHERE o.order_date >= CURDATE() - INTERVAL 30 DAY;
```

Below the editor, the 'Result Grid' tab is active, displaying a table with 4 columns: id, name, email, and address. It contains 3 rows of data:

	id	name	email	address
▶	1	Alice	alice@example.com	123 Apple St
	2	Bob	bob@example.com	456 Banana Ave
	3	Charlie	charlie@example.com	789 Cherry Blvd

On the right side, there are buttons for 'Result Grid', 'Form Editor', and 'Field Types'. At the bottom, the 'Output' pane shows 'Action Output' with a table of execution details:

#	Time	Action	Message
✓ 1	19:22:25	SELECT DISTINCT c.* FROM customers c JOIN orders o ON c.id = o.customer_id WHERE o.order_date >= CU...	3 row(s) returned

- Get the total amount of all orders placed by each customer.

The screenshot shows a database management interface. At the top, a toolbar includes icons for file operations, a 'Limit to 1000 rows' dropdown, and other utility icons. Below the toolbar, a SQL editor contains the following query:

```
1 -- Get the total amount of all orders placed by each customer
2 • SELECT c.name, SUM(o.total_amount) AS total_spent
3 FROM customers c
4 JOIN orders o ON c.id = o.customer_id
5 GROUP BY c.id;
```

Below the editor, the 'Result Grid' tab is active, displaying a table with the following data:

	name	total_spent
▶	Alice	225.00
	Bob	250.00
	Charlie	180.00

To the right of the result grid is a vertical toolbar with icons for 'Result Grid', 'Form Editor', and 'Field Types'. Below the result grid, a 'Result 2' tab is visible with a 'Read Only' status. At the bottom, an 'Output' section shows the 'Action Output' for the executed query:

#	Time	Action	Message
✓ 1	19:25:03	SELECT c.name, SUM(o.total_amount) AS total_spent FROM customers c JOIN orders o ON c.id = o.customer_id...	3 row(s) returned

- Update the price of Product C to 45.00.

The screenshot shows a database management tool interface. The top section contains a SQL editor with the following code:

```

1  -- Update the price of Product C to 45.00
2  • UPDATE products
3    SET price = 45.00
4    WHERE id = 3;
5
6  • SELECT id FROM products WHERE name = 'Product C';
7
8
9
10

```

Below the editor is a toolbar with icons for various actions. The 'Result Grid' tab is active, showing a table with one row:

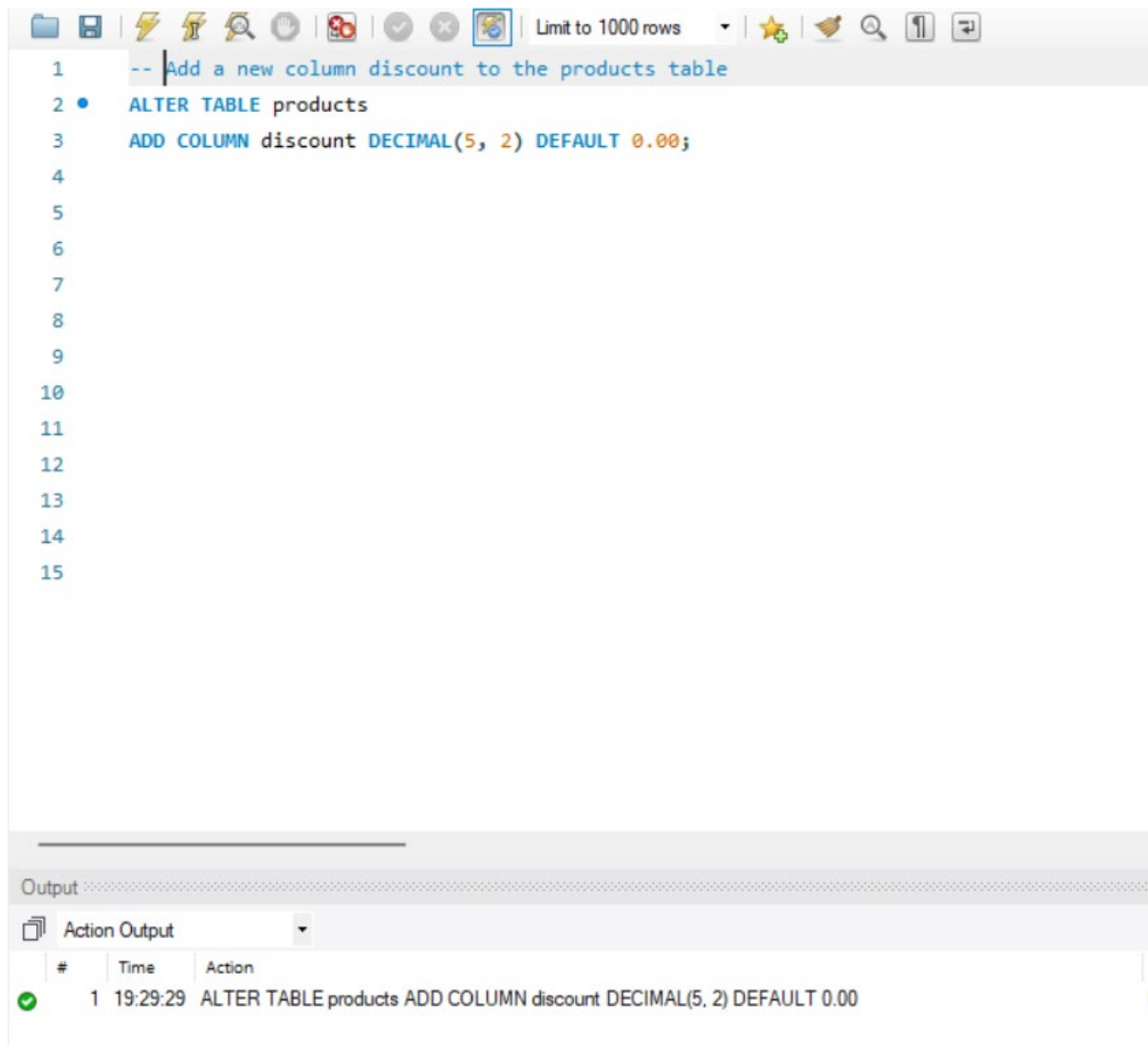
id
3

Below the table is a 'Filter Rows' input field and buttons for 'Edit', 'Export/Import', and 'Wrap Cell Content'. To the right of the table is a vertical toolbar with icons for 'Result Grid', 'Form Editor', and 'Field Types'.

At the bottom, the 'Output' section is visible, showing a table with two rows of execution results:

#	Time	Action	Message
1	19:28:00	UPDATE products SET price = 45.00 WHERE id = 3	1 row(s) affected Row
2	19:28:00	SELECT id FROM products WHERE name = 'Product C' LIMIT 0, 1000	1 row(s) returned

- Add a new column discount to the products table.



The screenshot shows a database management interface. The top toolbar includes icons for file operations, execution, and search, along with a dropdown menu set to "Limit to 1000 rows". The main text area contains the following SQL code:

```
1  -- Add a new column discount to the products table
2  • ALTER TABLE products
3    ADD COLUMN discount DECIMAL(5, 2) DEFAULT 0.00;
4
5
6
7
8
9
10
11
12
13
14
15
```

Below the code editor is an "Output" section with a dropdown menu set to "Action Output". The output table displays the execution results:

#	Time	Action
✓ 1	19:29:29	ALTER TABLE products ADD COLUMN discount DECIMAL(5, 2) DEFAULT 0.00

- Retrieve the top 3 products with the highest price.

The screenshot shows a database management tool interface. At the top, a toolbar includes icons for file operations, execution, and search, along with a 'Limit to 1000 rows' dropdown. Below the toolbar is a SQL editor with the following code:

```
1 -- Retrieve the top 3 products with the highest price
2 • SELECT * FROM products
3 ORDER BY price DESC
4 LIMIT 3;
5
6
7
8
9
10
```

Below the editor is a 'Result Grid' section. It includes a 'Filter Rows' input field and buttons for 'Edit', 'Export/Import', and 'Wrap Cell Content'. The grid displays the following data:

	id	name	price	description	discount
▶	4	Product D	200.00	Description D	0.00
	5	Product E	150.00	Description E	0.00
	2	Product B	100.00	Description B	0.00
*	NULL	NULL	NULL	NULL	NULL

On the right side of the interface, there is a vertical toolbar with buttons for 'Result Grid', 'Form Editor', and 'Field Types'. At the bottom, there is a tab labeled 'products 4' with an 'Apply' button and a 'Cont' button. Below the tabs is an 'Output' section with a dropdown menu set to 'Action Output'. The output table shows the following entry:

#	Time	Action	Message
✓ 1	19:31:31	SELECT * FROM products ORDER BY price DESC LIMIT 3	3 row(s) returned

- Get the names of customers who have ordered Product A.
- Get the names of customers who have ordered Product A
-- Requires normalization with order items first (see below)
-- So this query comes after normalization

The screenshot shows a database query editor interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The query editor contains the following SQL code:

```
1
2  -- Retrieve the average total of all orders
3  • SELECT AVG(total_amount) AS avg_order_total
4    FROM orders;
```

Below the query editor is the 'Result Grid' section, which displays the query results in a table:

avg_order_total
163.750000

On the right side of the interface, there is a vertical toolbar with buttons for 'Result Grid', 'Form Editor', and 'Field Types'. At the bottom, the 'Output' section shows the 'Action Output' for the query, indicating it was executed successfully at 19:41:33 and returned 1 row(s).

Result 8 x Read Only Cont

Output

Action Output

#	Time	Action	Message
1	19:41:33	SELECT AVG(total_amount) AS avg_order_total FROM orders LIMIT 0, 1000	1 row(s) returned

- Join the orders and customers tables to retrieve the customer's name and order date for each order.

The screenshot shows a database query editor interface. The top section contains a SQL query:

```
1 -- Join orders and customers to get customer name and order date
2 • SELECT c.name, o.order_date
3 FROM orders o
4 JOIN customers c ON o.customer_id = c.id;
5
6
7
8
9
10
```

Below the query editor, the "Result Grid" is displayed, showing the results of the query:

	name	order_date
▶	Alice	2025-07-19
	Alice	2025-06-09
	Bob	2025-07-09
	Charlie	2025-07-14

On the right side of the interface, there are buttons for "Result Grid", "Form Editor", and "Field Types".

At the bottom, the "Output" section shows the execution details:

#	Time	Action	Message
✓ 1	19:34:17	SELECT c.name, o.order_date FROM orders o JOIN customers c ON o.customer_id = c.id LIMIT 0, 1000	4 row(s) returned

- Retrieve the orders with a total amount greater than 150.00.

The screenshot shows a database management interface. At the top, a toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The SQL editor contains the following query:

```
1 -- Retrieve the orders with total amount greater than 150.00
2 • SELECT * FROM orders
3 WHERE total_amount > 150.00;
4
5
6
7
8
9
10
```

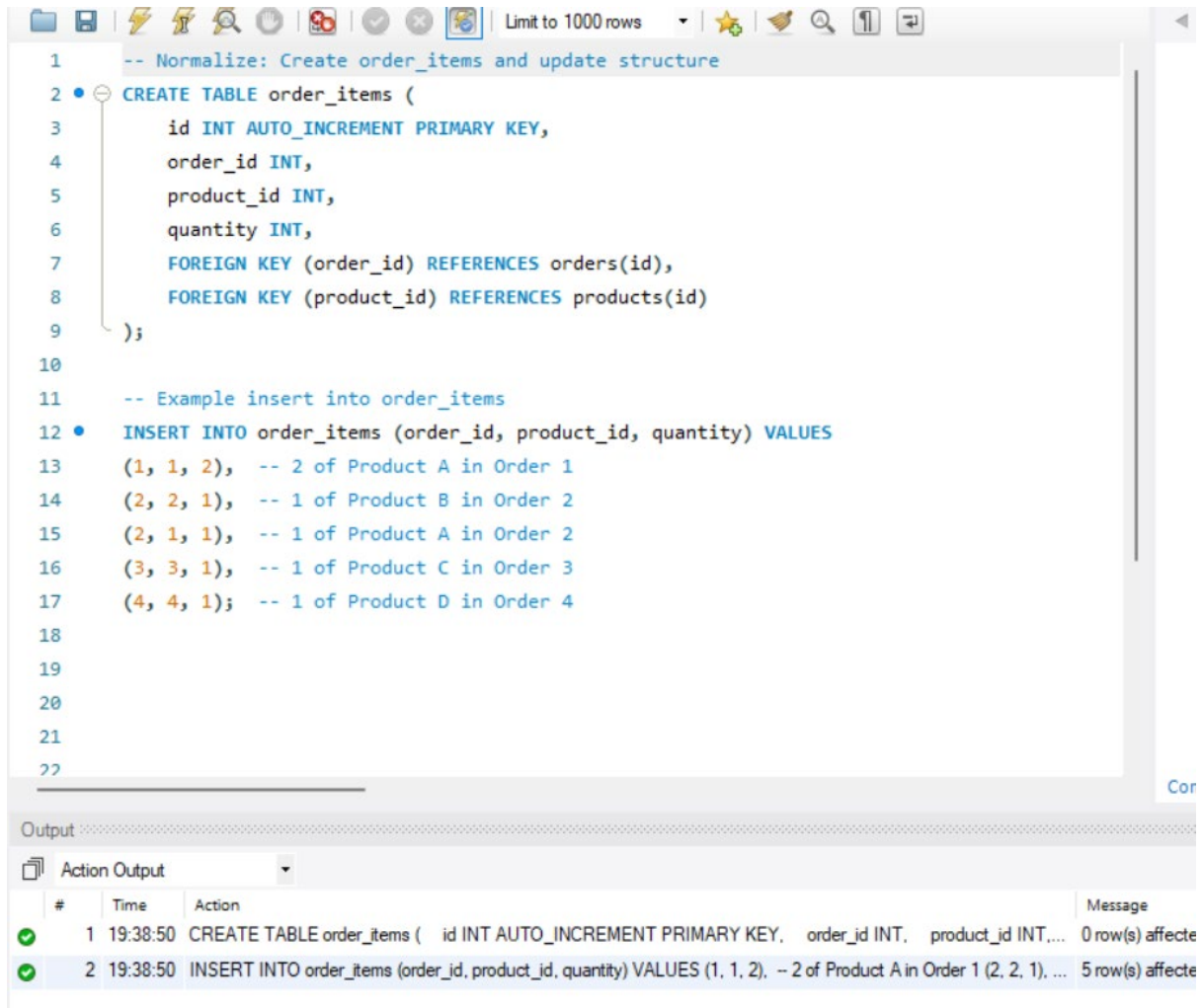
Below the editor is the 'Result Grid' section, which includes a 'Filter Rows' input and various action icons. It displays a table with the following data:

	id	customer_id	order_date	total_amount
▶	2	2	2025-07-09	250.00
	4	3	2025-07-14	180.00
*	NULL	NULL	NULL	NULL

On the right side of the interface, there is a vertical toolbar with buttons for 'Result Grid', 'Form Editor', and 'Field Types'. At the bottom, the 'Output' section shows the 'Action Output' for the executed query:

#	Time	Action	Message
✓ 1	19:36:00	SELECT * FROM orders WHERE total_amount > 150.00 LIMIT 0, 1000	2 row(s) returned

- Normalize the database by creating a separate table for order items and updating the orders table to reference the order items table.



The screenshot shows a database management interface with a SQL editor and an output window. The SQL editor contains the following code:

```

1  -- Normalize: Create order_items and update structure
2  • CREATE TABLE order_items (
3      id INT AUTO_INCREMENT PRIMARY KEY,
4      order_id INT,
5      product_id INT,
6      quantity INT,
7      FOREIGN KEY (order_id) REFERENCES orders(id),
8      FOREIGN KEY (product_id) REFERENCES products(id)
9  );
10
11  -- Example insert into order_items
12  • INSERT INTO order_items (order_id, product_id, quantity) VALUES
13      (1, 1, 2), -- 2 of Product A in Order 1
14      (2, 2, 1), -- 1 of Product B in Order 2
15      (2, 1, 1), -- 1 of Product A in Order 2
16      (3, 3, 1), -- 1 of Product C in Order 3
17      (4, 4, 1); -- 1 of Product D in Order 4
18
19
20
21
22

```

The output window shows the execution results of the SQL statements:

#	Time	Action	Message
✓ 1	19:38:50	CREATE TABLE order_items (id INT AUTO_INCREMENT PRIMARY KEY, order_id INT, product_id INT,...	0 row(s) affected
✓ 2	19:38:50	INSERT INTO order_items (order_id, product_id, quantity) VALUES (1, 1, 2), -- 2 of Product A in Order 1 (2, 2, 1), ...	5 row(s) affected

- Retrieve the average total of all orders.

The screenshot shows a database query editor interface. The top toolbar includes icons for file operations, a search icon, and a 'Limit to 1000 rows' dropdown. The SQL editor contains the following query:

```
1
2 -- Get names of customers who have ordered Product A (after normalization)
3 • SELECT DISTINCT c.name
4 FROM customers c
5 JOIN orders o ON c.id = o.customer_id
6 JOIN order_items oi ON o.id = oi.order_id
7 JOIN products p ON oi.product_id = p.id
8 WHERE p.name = 'Product A';
9
10
```

Below the editor, the 'Result Grid' tab is active, displaying the query results in a table:

name
Alice
Bob

On the right side, there are buttons for 'Result Grid', 'Form Editor', and 'Field Types'. At the bottom, the 'Output' section shows the execution log:

#	Time	Action	Message
1	19:40:09	SELECT DISTINCT c.name FROM customers c JOIN orders o ON c.id = o.customer_id JOIN order_items oi ON ...	2 row(s)