

CANDIDATE NUMBER (C-NUMBER)	C2096215
MODULE NAME	Data Design Management
WORD COUNT	---
SUBMISSION DATE	31/01/2024

DECLARATION

I certify that this assessment submission is entirely my work and I have fully referenced and correctly cited the work of others, where required. I also confirm the contents of my submission have not been generated by a third party, or through an Artificial Intelligence generative system*.

I have read the Student Discipline Regulations ([Student Discipline Regulations](#)) and understand any Assessment Related Offence/ Academic Misconduct may result penalties being applied.

By submitting this assessment submission, I am confirming that I am fit to sit according to the Assessment Regulations.

I declare that:

- This is my own unaided work.
Yes ☒
No ☐
- The word count stated by me is correct.
Yes ☒
No ☐
- I'm happy for my work to be retained on the Elite repository and made available to staff and future students**
Yes ☒
No ☐

*Please note that all the assignments are submitted to Turnitin.
**Please note personal information (such as names) will be deleted.

Data Design Assessment

Farzam Ferydooni

Student Number:0381952



(Lui M [Photograph] 2009)

Outline

- Requirements And Design Considerations
- Designing
- Implementation
- Data Entry
- Current State of Tables
- Assessment Tasks
- Use Cases
- Conclusion

Requirements And Design Considerations

Requirements

- Online shop
- Services: selling Books, Movies , Albums (Music Records)
- Location To service: World Wide
- Database Type: relational database, centralise, MySQL
- Environment for implementation: MySQL Workbench 8
- Design Environment: MS Visio
- Design: Chen ERD, Crow's Foot database notation

Design Considerations

- A database is the foundation of every service, it is crucial to invest extreme effort in its design(Silberschatz et al. 1–25). It could help us to reach sustainable service. Some of main considerations are:
 - **Performance and scalability:** One of the main factors in online business, especially in the worldwide market, is providing high-performance service for various scale customers. This could be achieve by deferent layers from high-performance network and processing power to modular design in database and also redundancy in those layers and designs.
 - **Security:** Encryption, data masking, and access Control (with segregation of duty) could be some example of ways to provide security over the database. In order that was mentioned before in other layers, backup and access control by firewall could be considered.
 - **Maintainability:** Designing a service is not an on time job. But it needs a lot of refer, to repair or even redesign with regard to service developments based on service owner or customer demands. For this reason, Databases should be designed and documented to facilitate these need.

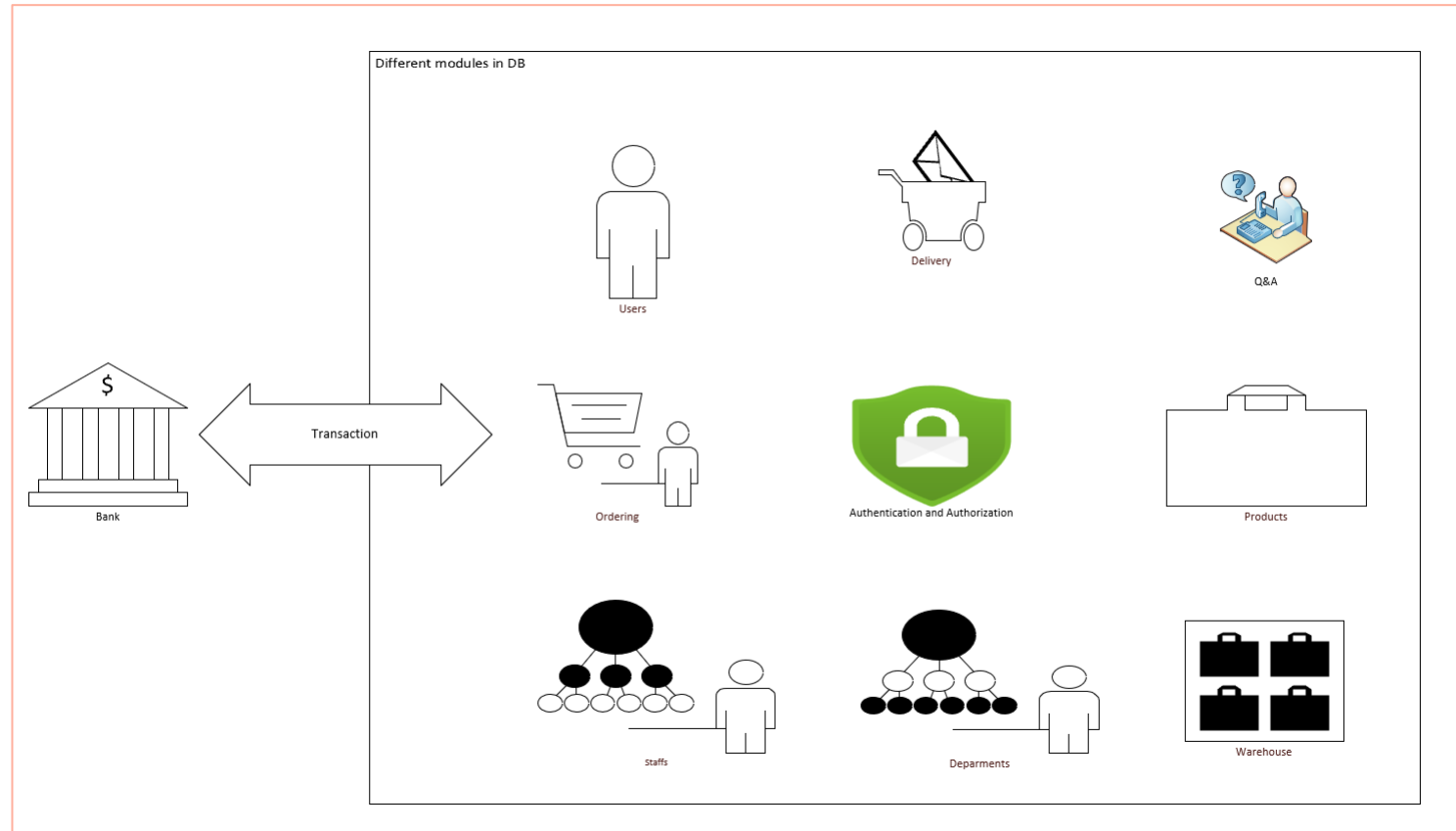
Lean Operation in Database design (Graupp, 2022)

- One systematic way to cover those demands is through of the Lean Operation. here are some practise for the lean design:
 - Value propositioning and targeting audience
 - Identifying unnecessary process in designing
 - Data Normalization
 - standardization in database design for example: Naming, indentation, documentation
 - Testing and validation procedures
 - Monitoring and evaluating

Lean Operation in Database design- benefits(Graupp, 2022)

- Because It is a systematic operation, Here are some benefit of using Lean operation in database design:
 - Increasing flexibility for fast and easy changes and updates
 - Increasing customer by covering their needs
 - Regular security and compliance control
 - Reduce cost of development with minimizing unnecessary features

Project-Macro View



In macro view, the project consist of external part (Bank and PSP for transaction) and internal part (Database and frontend and backend) and connectivity (private line and internet)

Project-Description

At least the following items should be considered in the programming of this project:

- User Section
 - Define users
 - Users activities logging
 - Review and rating
 - Shopping
- Employee section
 - Define employees
 - Assigning them to Department
 - Employee activities logging
 - Assigning to a task
- Departments
 - Define Departments
 - Staff management operation data

Project-Description

- Authentication and Authorization
 - Authentication and Authorization for users and staff
- Product
 - Define product
 - Product review and score
- Warehouse
 - All warehouse operation
- Ordering
 - Order listing
- Billing and Transaction
 - Bank payment and verification

Project-Description

- Financial Management
- Order Delivery
 - Delivery details
- Q&A and Review and rating
 - Every Q&A about products and services
- Feedback and complain centre
 - Submit customers and employees feedbacks and complains
 - Follow up feedbacks and complains
- Service Development
 - Improve service with users feedbacks

What is Crow's Foot Notation?

- A widely used convention for representing relationships in Entity Relationship Diagrams (ERDs).
- Employs distinct symbols to visually convey cardinality and modality between entities.
- Enhances clarity and understanding of data structures.

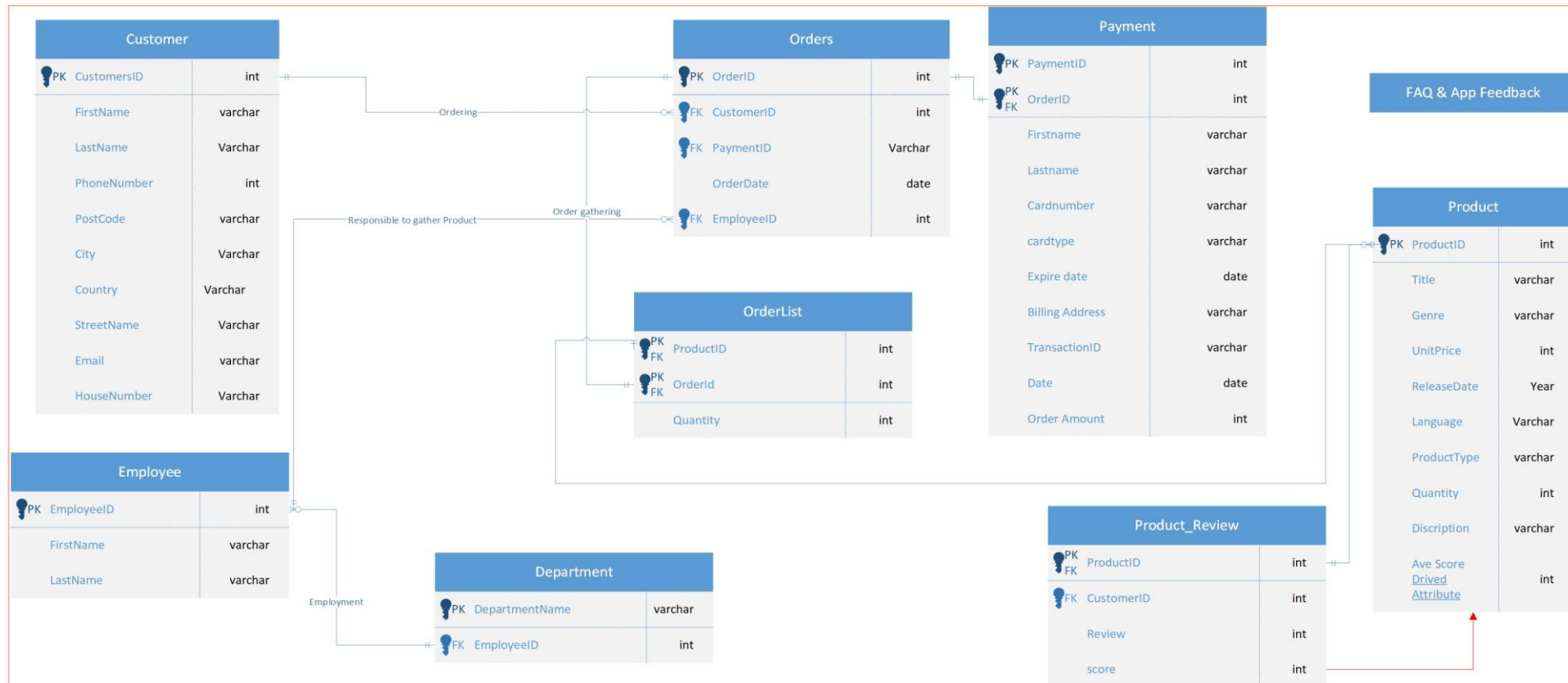
Cardinality-Modality

- Cardinality: Expresses the maximum number of instances of one entity that can be associated with another.
- Modality: Indicates whether participation in a relationship is mandatory or optional.
 - Mandatory: Represented by a solid circle.
 - Optional: Shown as an empty circle.

Designing

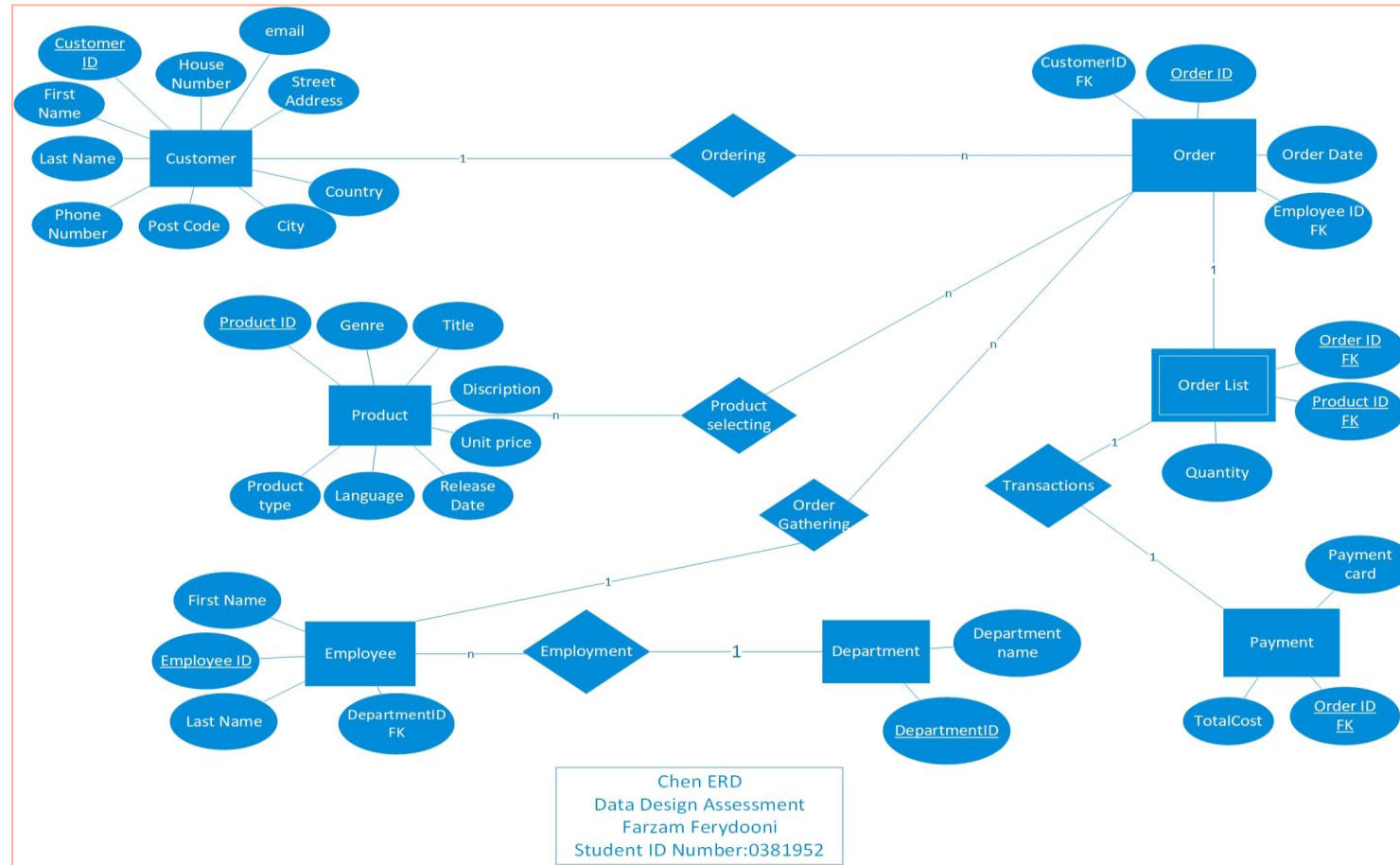
Crow's Foot Notation

Comprehensive ERD-For Lean Design

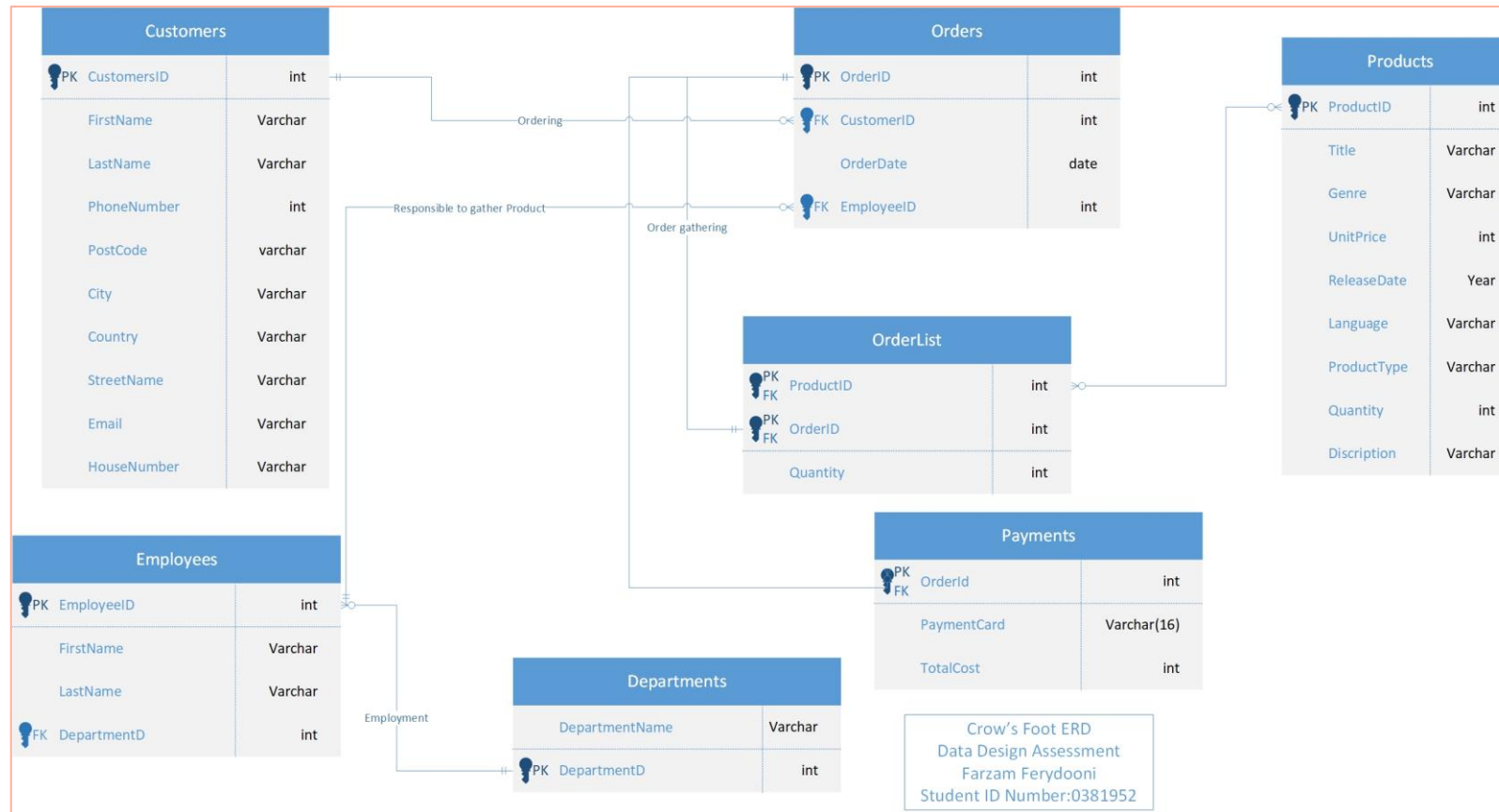


The parts of the assessment question that are desired will be the primary focus of attention since the process of incorporating the features stated on the slide into the framework of the course test is time-consuming. The following slide, which will cover the modules that are necessary for evaluation, is now ready for us to move on to.

Chen Based ERD

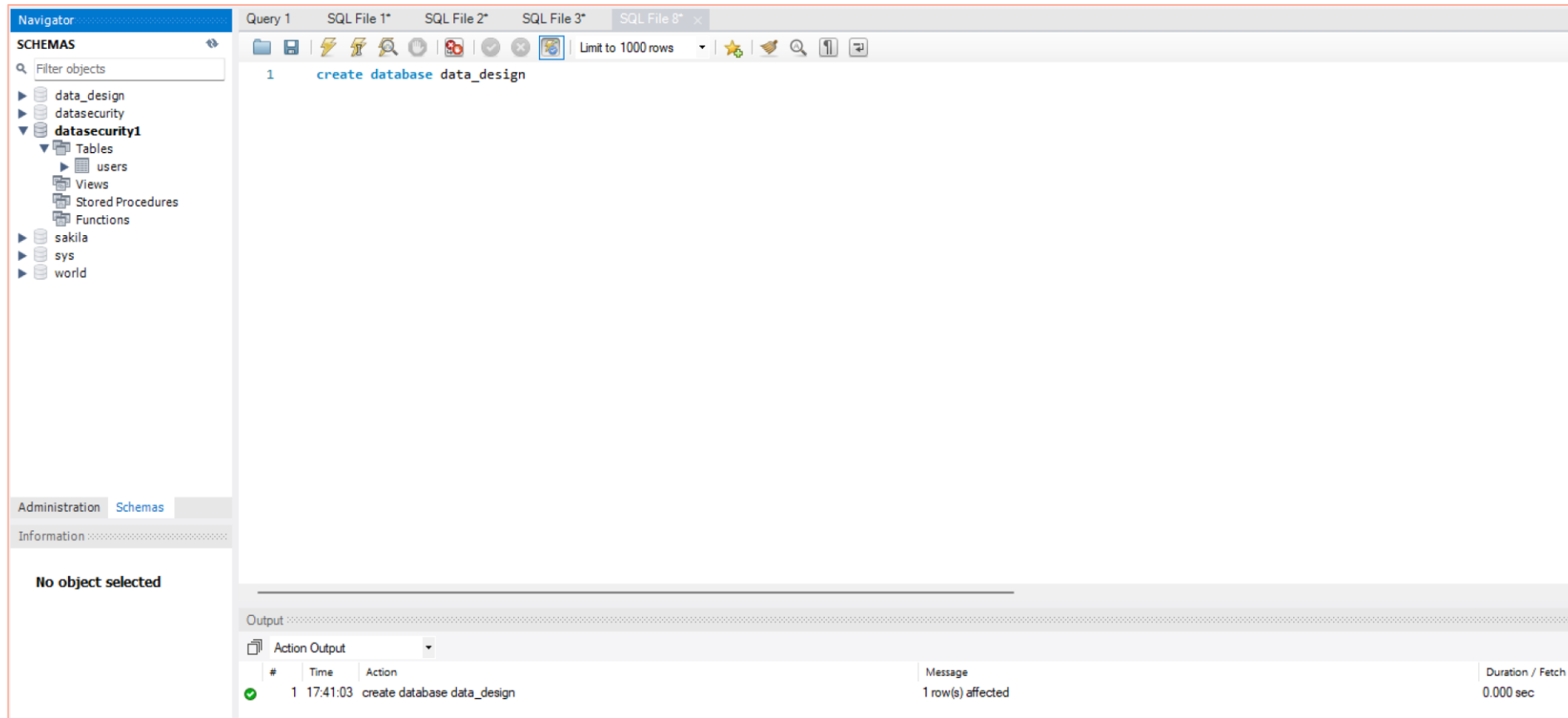


Crow's Foot based ERD



Implementation

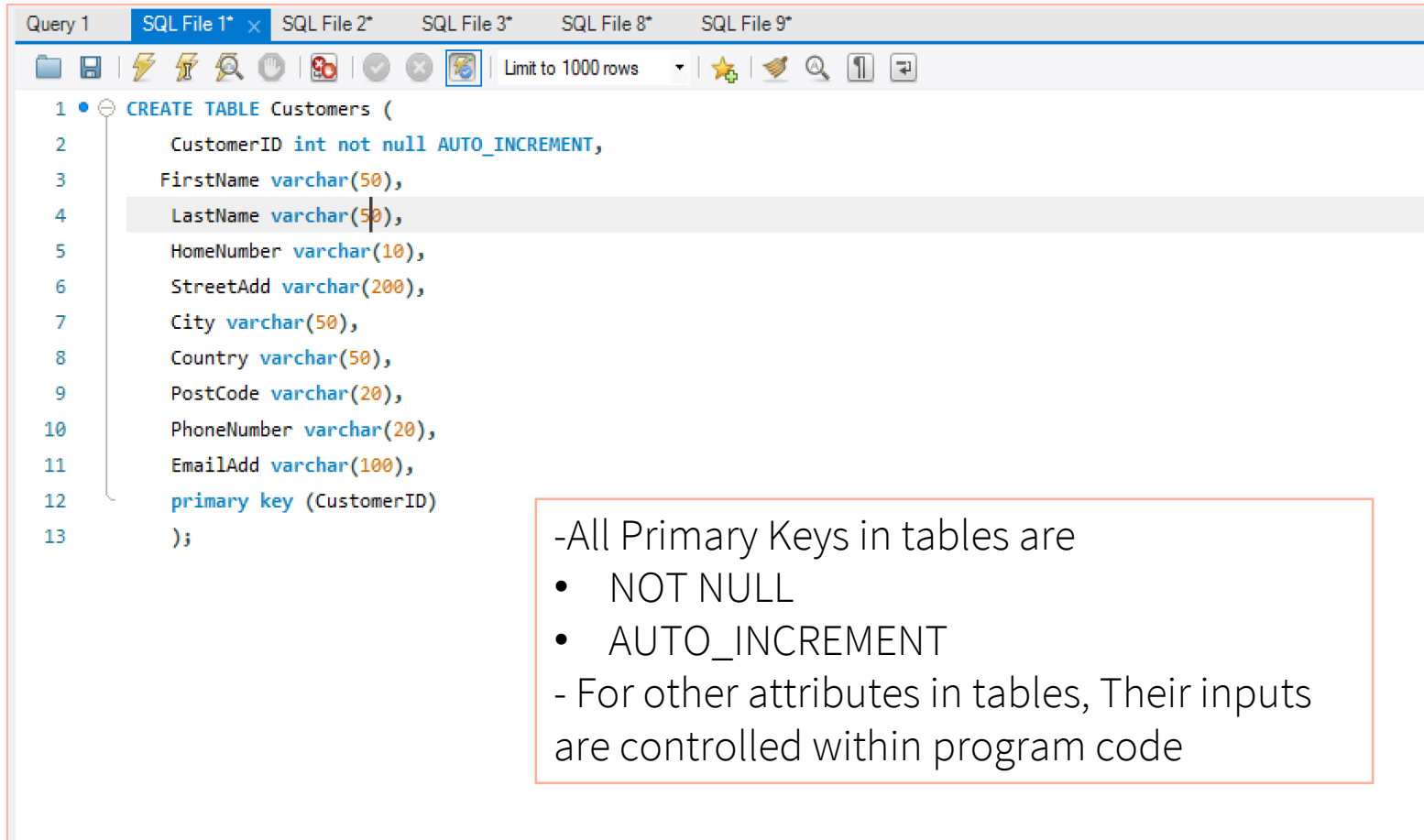
Creation Phase Database



First step is making data base “data_design”

Creation Phase

Customers Table



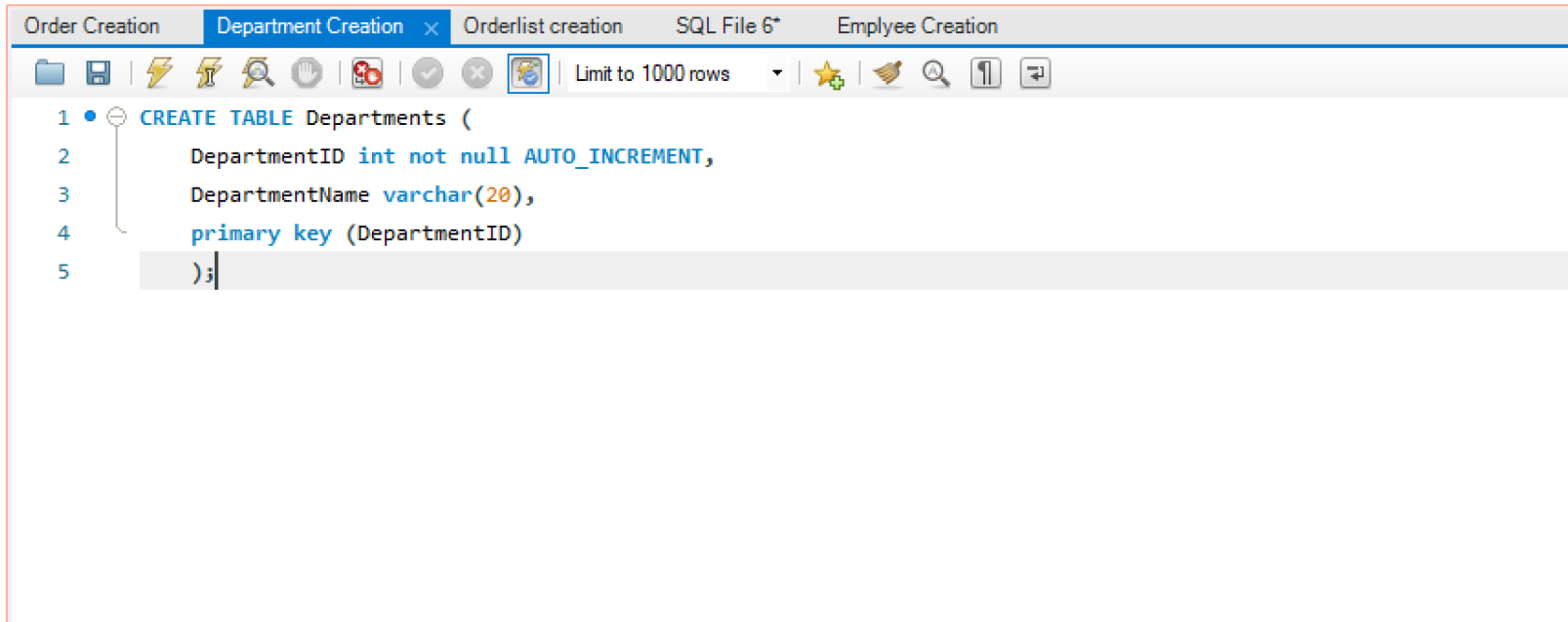
```
1 • CREATE TABLE Customers (  
2     CustomerID int not null AUTO_INCREMENT,  
3     FirstName varchar(50),  
4     LastName varchar(50),  
5     HomeNumber varchar(10),  
6     StreetAdd varchar(200),  
7     City varchar(50),  
8     Country varchar(50),  
9     PostCode varchar(20),  
10    PhoneNumber varchar(20),  
11    EmailAdd varchar(100),  
12    primary key (CustomerID)  
13    );
```

- All Primary Keys in tables are
 - NOT NULL
 - AUTO_INCREMENT
- For other attributes in tables, Their inputs are controlled within program code

It is urged that we begin the process of creating tables by beginning with a table that has no or a minimal dependent on other tables.

Creation Phase

Department Table



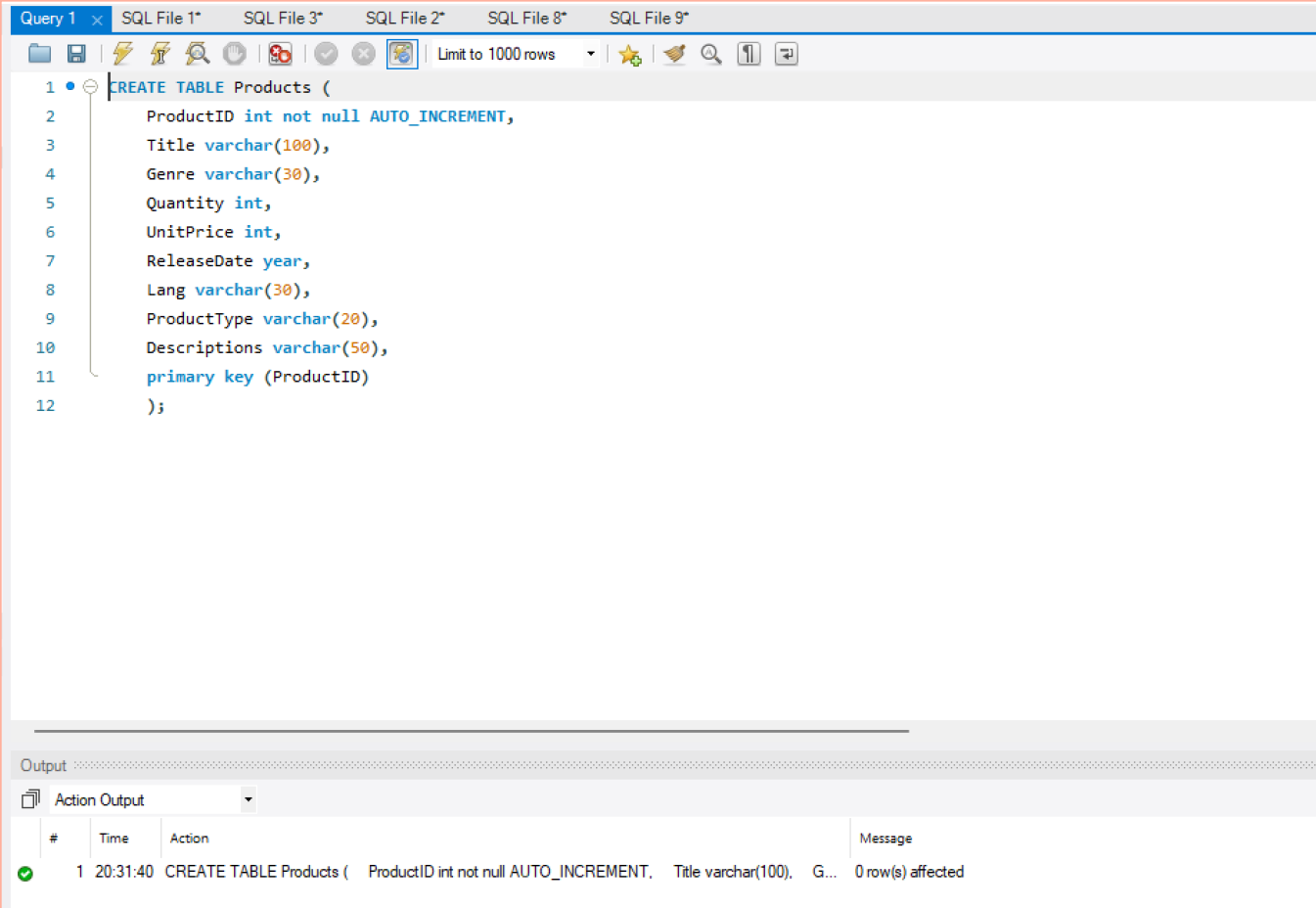
The screenshot shows a database management tool interface with a tab labeled "Department Creation". The SQL editor displays the following code:

```
1 CREATE TABLE Departments (  
2     DepartmentID int not null AUTO_INCREMENT,  
3     DepartmentName varchar(20),  
4     primary key (DepartmentID)  
5 );
```

The interface includes a toolbar with various icons and a dropdown menu set to "Limit to 1000 rows".

Creation Phase

Products Table



```
1 CREATE TABLE Products (  
2     ProductID int not null AUTO_INCREMENT,  
3     Title varchar(100),  
4     Genre varchar(30),  
5     Quantity int,  
6     UnitPrice int,  
7     ReleaseDate year,  
8     Lang varchar(30),  
9     ProductType varchar(20),  
10    Descriptions varchar(50),  
11    primary key (ProductID)  
12 );
```

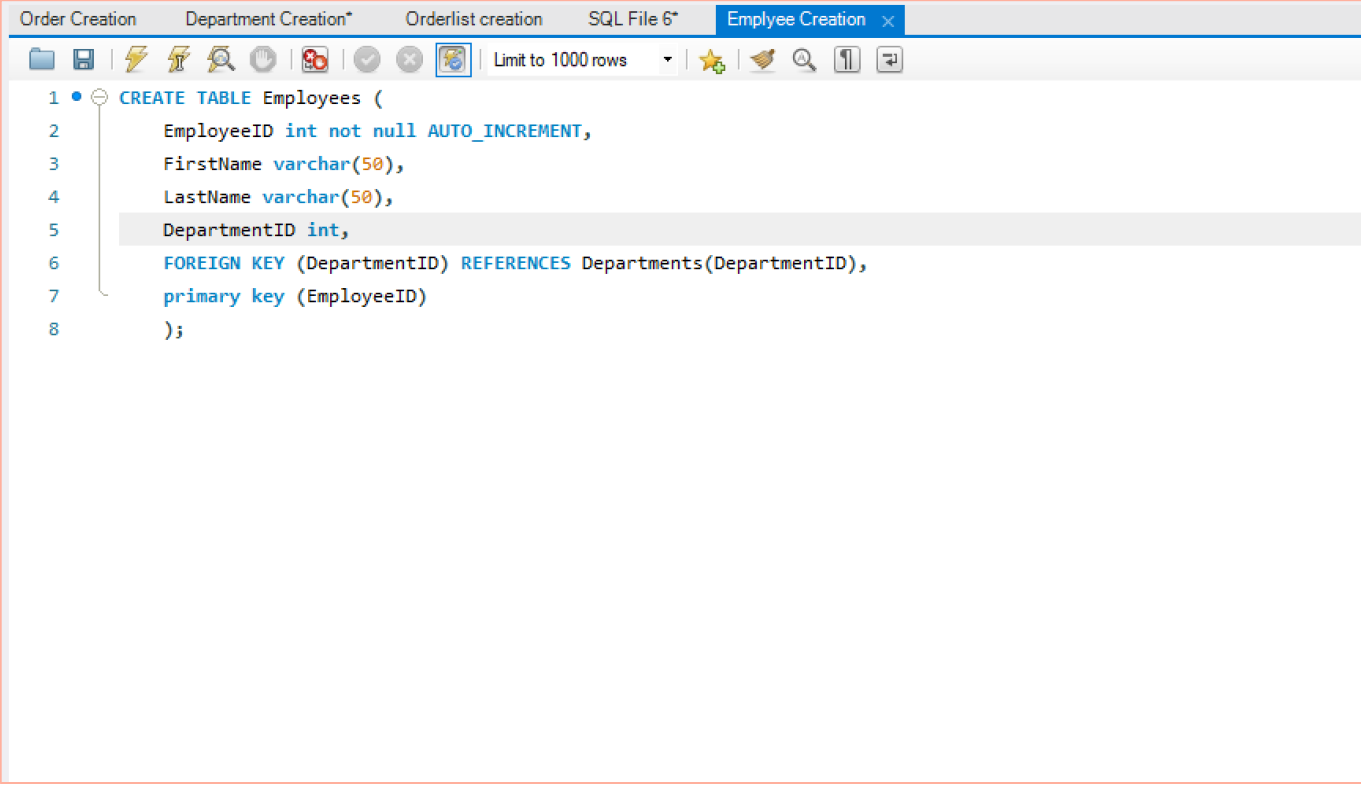
Output

Action Output

#	Time	Action	Message
✓ 1	20:31:40	CREATE TABLE Products (ProductID int not null AUTO_INCREMENT, Title varchar(100), G...	0 row(s) affected

Creation Phase

Employee Table



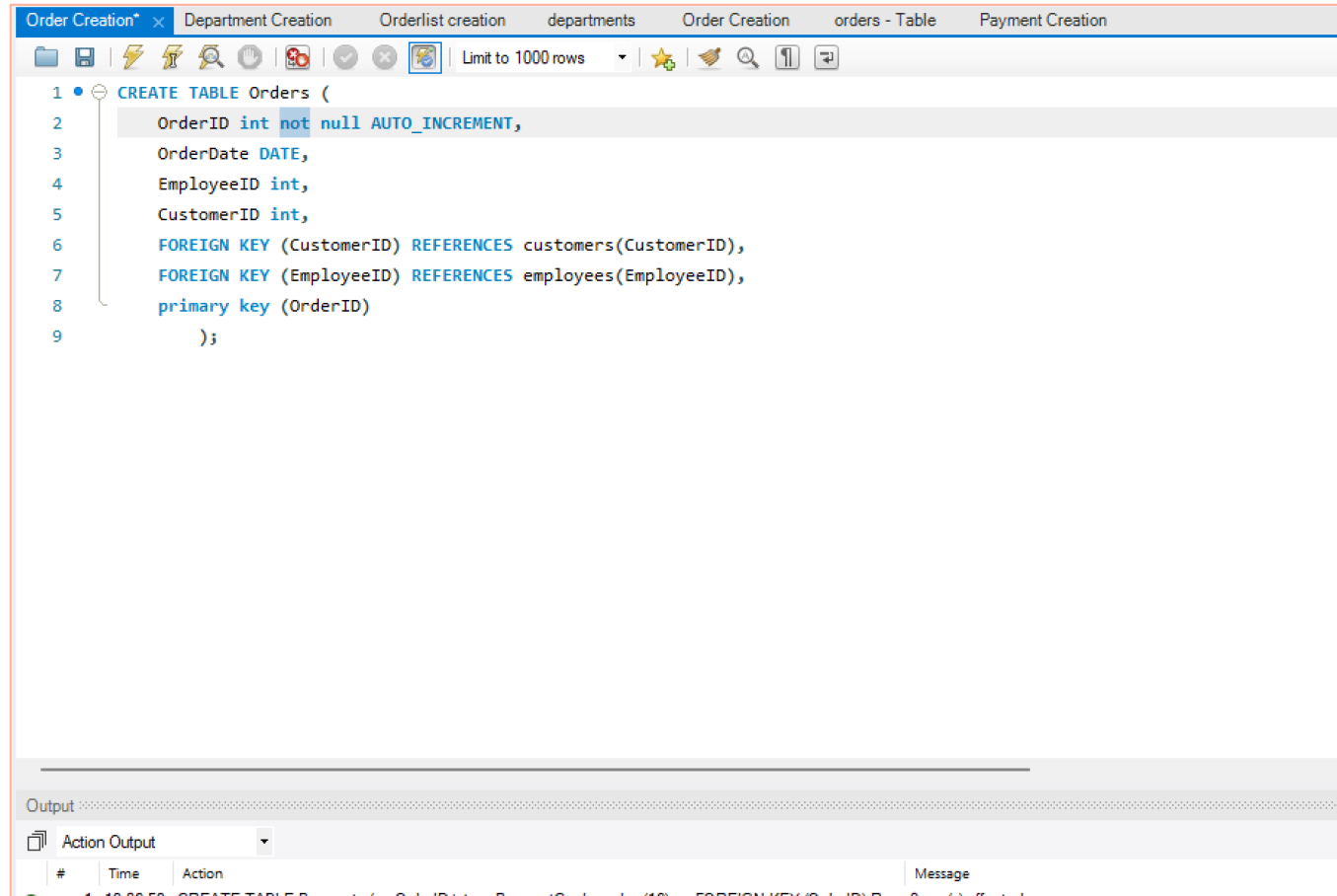
The screenshot shows a database management tool with multiple tabs: 'Order Creation', 'Department Creation*', 'Orderlist creation', 'SQL File 6*', and 'Employee Creation'. The 'Employee Creation' tab is active. The SQL editor displays the following code:

```
1 CREATE TABLE Employees (  
2     EmployeeID int not null AUTO_INCREMENT,  
3     FirstName varchar(50),  
4     LastName varchar(50),  
5     DepartmentID int,  
6     FOREIGN KEY (DepartmentID) REFERENCES Departments(DepartmentID),  
7     primary key (EmployeeID)  
8 );
```

Because each employee should connect to one of the departments, the department table is linked to the employee table with a foreign key.

Creation Phase

Orders Table



The screenshot shows a database management tool interface with multiple tabs at the top: "Order Creation*", "Department Creation", "Orderlist creation", "departments", "Order Creation", "orders - Table", and "Payment Creation". The "orders - Table" tab is active. Below the tabs is a toolbar with various icons, including a "Limit to 1000 rows" dropdown. The main area displays a SQL script for creating the "Orders" table. The script is as follows:

```
1 CREATE TABLE Orders (  
2     OrderID int not null AUTO_INCREMENT,  
3     OrderDate DATE,  
4     EmployeeID int,  
5     CustomerID int,  
6     FOREIGN KEY (CustomerID) REFERENCES customers(CustomerID),  
7     FOREIGN KEY (EmployeeID) REFERENCES employees(EmployeeID),  
8     primary key (OrderID)  
9 );
```

At the bottom of the window is an "Output" section with a dropdown menu set to "Action Output". Below this is a table with columns for "#", "Time", "Action", and "Message". The first row of the table shows the execution of the SQL script.

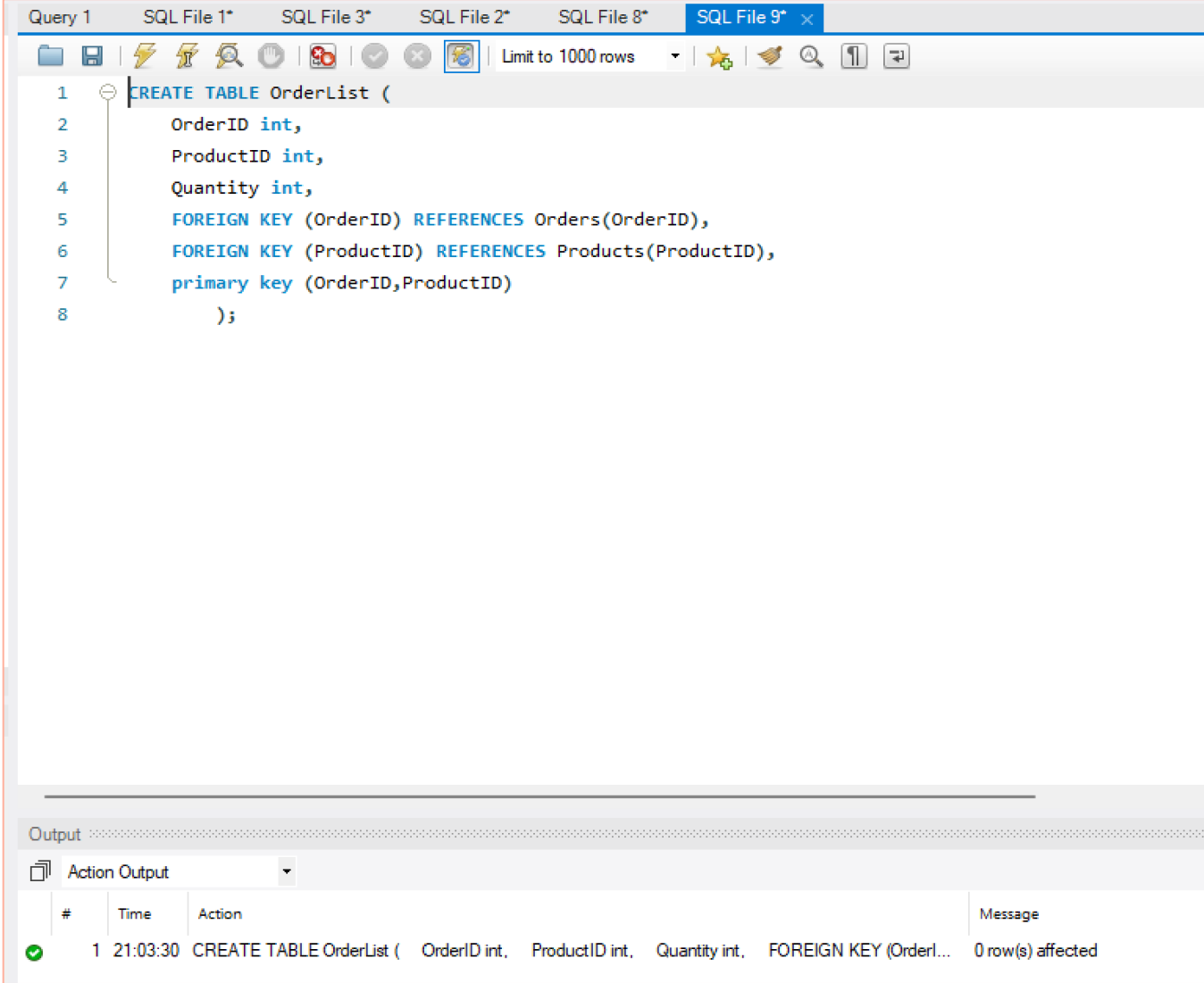
#	Time	Action	Message
1	10:05:50	CREATE TABLE Orders (OrderID int not null AUTO_INCREMENT, OrderDate DATE, EmployeeID int, CustomerID int, FOREIGN KEY (CustomerID) REFERENCES customers(CustomerID), FOREIGN KEY (EmployeeID) REFERENCES employees(EmployeeID), primary key (OrderID));	

In Orders table we have two foreign keys, one connects each order to a specific customer and other connect each order to one employee to gathers required products

Creation Phase

OrdersList Table

In this table customers orders should be list, so this table have foreign key to products and orders.



The screenshot shows a SQL IDE window with multiple tabs. The active tab is 'SQL File 9*'. The main editor displays the following SQL code:

```
1 CREATE TABLE OrderList (  
2     OrderID int,  
3     ProductID int,  
4     Quantity int,  
5     FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),  
6     FOREIGN KEY (ProductID) REFERENCES Products(ProductID),  
7     primary key (OrderID,ProductID)  
8 );
```

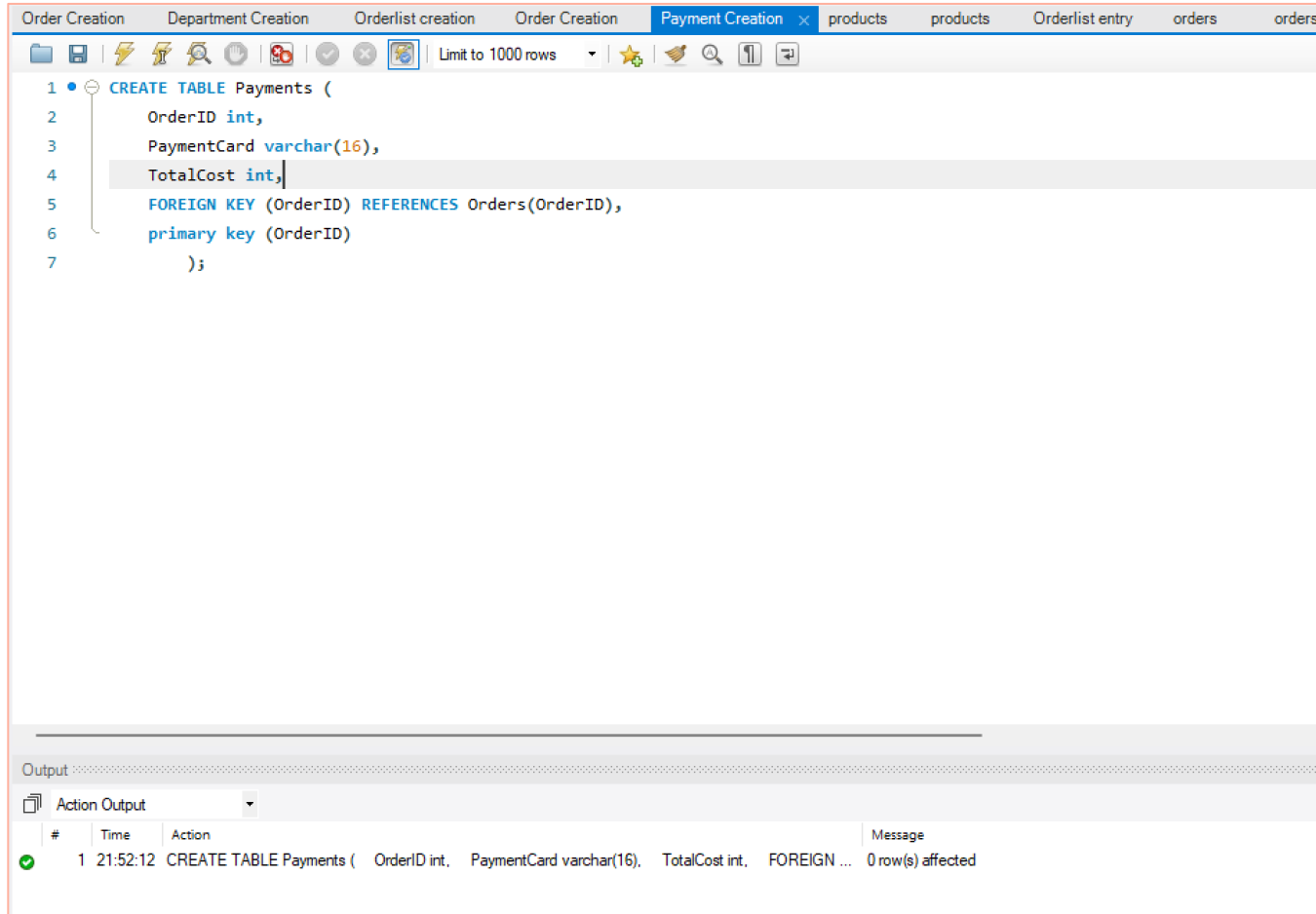
The bottom of the window features an 'Output' pane with a dropdown menu set to 'Action Output'. Below this is a table showing the execution results:

#	Time	Action	Message
✓ 1	21:03:30	CREATE TABLE OrderList (OrderID int, ProductID int, Quantity int, FOREIGN KEY (OrderID...	0 row(s) affected

Creation Phase

Payments Table

This table's TotalCost attribute is driven by the OrderList attribute, which gives a list of orders along with their quantities and prices from the Product table.



The screenshot shows a database management interface with a 'Payment Creation' tab selected. The main area displays the SQL command to create the 'Payments' table. The command is as follows:

```
1 CREATE TABLE Payments (  
2     OrderID int,  
3     PaymentCard varchar(16),  
4     TotalCost int,  
5     FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),  
6     primary key (OrderID)  
7 );
```

Below the SQL editor, the 'Output' pane shows the 'Action Output' for the execution of the command. The output table has columns for '#', 'Time', 'Action', and 'Message'. The first row shows a successful execution at 21:52:12, indicating that 0 row(s) were affected.

#	Time	Action	Message
1	21:52:12	CREATE TABLE Payments (OrderID int, PaymentCard varchar(16), TotalCost int, FOREIGN ...	0 row(s) affected

Data Entry

Data Entry Phase Product

The screenshot shows a MySQL IDE interface with a left-hand sidebar and a main editor area. The sidebar contains a 'SCHEMAS' panel with a tree view of databases, including 'data_design' (containing tables like customers, departments, employees, orderlist, orders, products) and 'sakila'. Below this is a 'Connection Details' panel showing the connection to 'Local instance MySQL80' on 'localhost' port '3306' as user 'root'. The main editor area displays an SQL script for 'Query 1' in 'SQL File 10'. The script is an 'INSERT INTO' statement for the 'Products' table, with columns (Title, Genre, Quantity, UnitPrice, ReleaseDate, Lang, ProductType, Descriptions). It includes comments for 'Movies' and 'Series' and lists 15 records. The bottom of the IDE shows an 'Output' panel with a table summarizing the execution results.

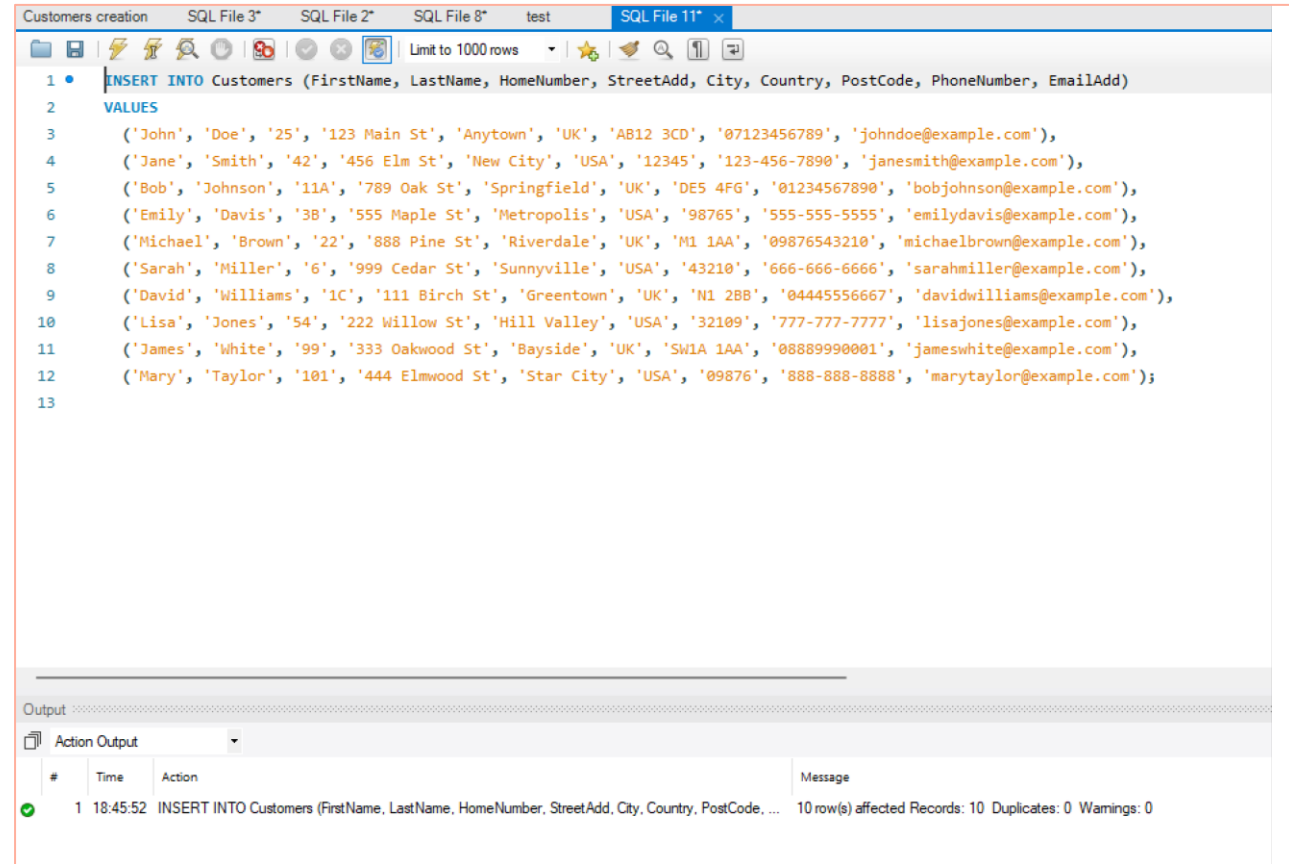
```
1 • INSERT INTO Products (Title, Genre, Quantity, UnitPrice, ReleaseDate, Lang, ProductType, Descriptions)
2 VALUES
3 -- Movies
4 ('The Lord of the Rings: The Fellowship of the Ring', 'Fantasy', 5, 4, 2001, 'English', 'Movie', 'An epic fantasy adventure'),
5 ('The Shawshank Redemption', 'Drama', 3, 3, 1994, 'English', 'Movie', 'A critically acclaimed drama about hope and redemption'),
6 ('The Dark Knight', 'Action', 8, 5, 2008, 'English', 'Movie', 'A gritty and intense superhero crime thriller'),
7 ('The Godfather', 'Crime Drama', 2, 4, 1972, 'English', 'Movie', 'A classic gangster film about family, power, and betrayal'),
8 ('The Matrix', 'Sci-Fi', 7, 3, 1999, 'English', 'Movie', 'A mind-bending action film that explores the nature of reality'),
9 -- Series
10 ('Friends', 'Comedy', 10, 2, 1994, 'English', 'TV Series', 'A sitcom about six friends in New York City'),
11 ('The Office', 'Mockumentary Comedy', 6, 3, 2005, 'English', 'TV Series', 'A mockumentary about office life in Scranton, Pennsylvania'),
12 ('Breaking Bad', 'Crime Drama', 4, 4, 2008, 'English', 'TV Series', 'A high-stakes drama about a chemistry teacher turned drug kingpin'),
13 ('Game of Thrones', 'Fantasy', 8, 5, 2011, 'English', 'TV Series', 'An epic fantasy drama about power struggles in a medieval world'),
14 ('The Wire', 'Crime Drama', 2, 4, 2002, 'English', 'TV Series', 'A gritty and realistic portrayal of the drug trade in Baltimore'),
15 -- Music
16 ('Abbey Road', 'Rock', 15, 2, 1969, 'English', 'Album', 'The Beatles final studio album'),
17 ('Thriller', 'Pop', 12, 3, 1982, 'English', 'Album', 'Michael Jackson best-selling album'),
18 ('The Dark Side of the Moon', 'Progressive Rock', 10, 4, 1973, 'English', 'Album', 'Pink Floyd landmark album'),
19 ('Kind of Blue', 'Jazz', 8, 3, 1959, 'English', 'Album', 'Miles Davis masterpiece'),
20 ('Nevermind', 'Grunge', 6, 4, 1991, 'English', 'Album', 'Nirvana iconic album that changed the course of rock music');
```

#	Time	Action	Message
✓ 1	18:25:18	INSERT INTO Products (Title, Genre, Quantity, UnitPrice, ReleaseDate, Lang, ProductType, Descript...	15 row(s) affected Records: 15 Duplicates: 0 Warnings: 0

Data Entry Phase

Customers

As mentioned earlier, when a row in a table is given a value, the primary key is automatically assigned.



The screenshot shows a SQL IDE window with multiple tabs. The active tab is 'SQL File 11*'. The editor contains an SQL INSERT statement for a table named 'Customers'. The statement lists 10 rows of data, including first and last names, home numbers, street addresses, cities, countries, postcodes, phone numbers, and email addresses. Below the editor, the 'Output' pane is visible, showing the 'Action Output' for the executed statement. The output indicates that 10 rows were affected, with no duplicates or warnings.

```
1 • INSERT INTO Customers (FirstName, LastName, HomeNumber, StreetAdd, City, Country, PostCode, PhoneNumber, EmailAdd)
2 VALUES
3 ('John', 'Doe', '25', '123 Main St', 'Anytown', 'UK', 'AB12 3CD', '07123456789', 'johndoe@example.com'),
4 ('Jane', 'Smith', '42', '456 Elm St', 'New City', 'USA', '12345', '123-456-7890', 'janesmith@example.com'),
5 ('Bob', 'Johnson', '11A', '789 Oak St', 'Springfield', 'UK', 'DE5 4FG', '01234567890', 'bobjohnson@example.com'),
6 ('Emily', 'Davis', '3B', '555 Maple St', 'Metropolis', 'USA', '98765', '555-555-5555', 'emilydavis@example.com'),
7 ('Michael', 'Brown', '22', '888 Pine St', 'Riverdale', 'UK', 'M1 1AA', '09876543210', 'michaelbrown@example.com'),
8 ('Sarah', 'Miller', '6', '999 Cedar St', 'Sunnyville', 'USA', '43210', '666-666-6666', 'sarahmiller@example.com'),
9 ('David', 'Williams', '1C', '111 Birch St', 'Greentown', 'UK', 'N1 2BB', '04445556667', 'davidwilliams@example.com'),
10 ('Lisa', 'Jones', '54', '222 Willow St', 'Hill Valley', 'USA', '32109', '777-777-7777', 'lisajones@example.com'),
11 ('James', 'White', '99', '333 Oakwood St', 'Bayside', 'UK', 'SW1A 1AA', '08889990001', 'jameswhite@example.com'),
12 ('Mary', 'Taylor', '101', '444 Elmwood St', 'Star City', 'USA', '09876', '888-888-8888', 'marytaylor@example.com');
13
```

#	Time	Action	Message
1	18:45:52	INSERT INTO Customers (FirstName, LastName, HomeNumber, StreetAdd, City, Country, PostCode, ...	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0

Data Entry Phase

Employee

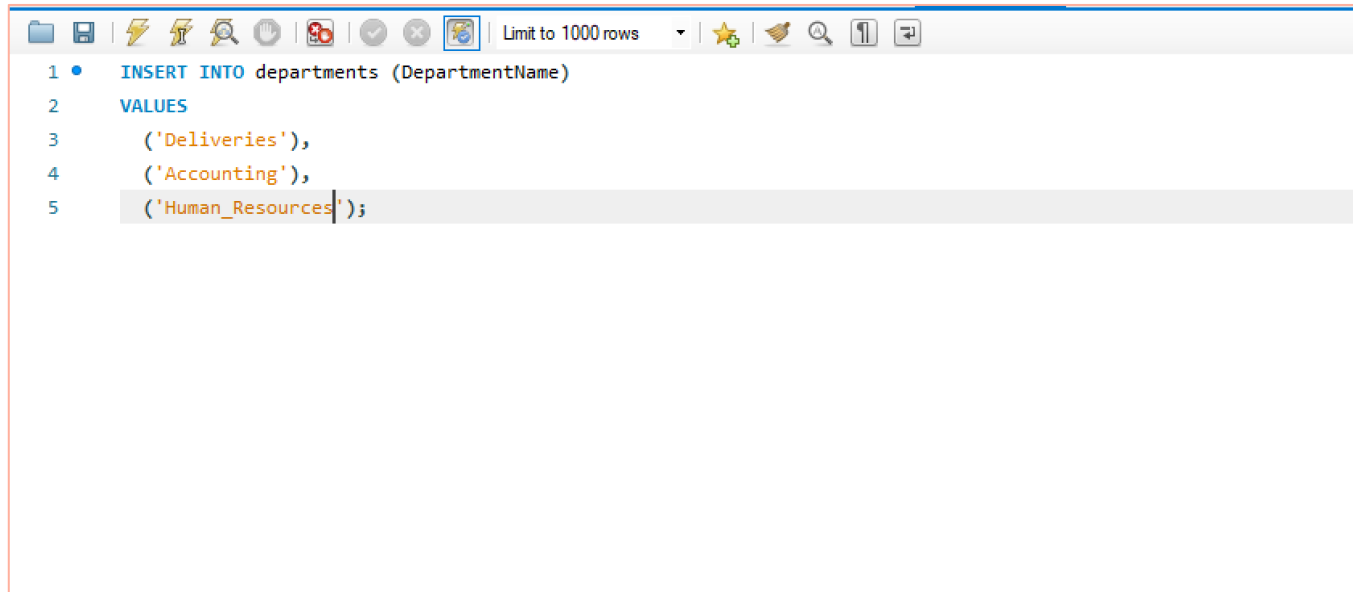
The screenshot displays a database management interface with multiple tabs. The active tab is 'Employee entry'. The main area shows an SQL script for inserting employee data into the 'Employees' table. The script consists of seven lines: a blue '1' followed by 'INSERT INTO Employees (FirstName, LastName)', a blue '2' followed by 'VALUES', and lines 3 through 7 containing five rows of employee data in parentheses, separated by commas. The data includes Alice Smith, Bob Johnson, Charlie Davis, Emily Brown, and David Miller. Below the script, an 'Output' section is visible, showing a table with execution results. The table has columns for a status icon, a sequence number, time, action, and a message. Two rows are shown, both with green checkmark icons, indicating successful execution. The first row shows the insertion into the 'Customers' table, and the second row shows the insertion into the 'Employees' table.

```
1 • INSERT INTO Employees (FirstName, LastName)
2 VALUES
3     ('Alice', 'Smith'),
4     ('Bob', 'Johnson'),
5     ('Charlie', 'Davis'),
6     ('Emily', 'Brown'),
7     ('David', 'Miller');
```

#	Time	Action	Message
✓ 1	18:45:52	INSERT INTO Customers (FirstName, LastName, HomeNumber, StreetAdd, City, Country, PostCode, ...	10 row(s) affected Records: 10 Duplicates: 0 Warnings: 0
✓ 2	19:04:23	INSERT INTO Employees (FirstName, LastName) VALUES ('Alice', 'Smith'), ('Bob', 'Johnson'), ('Ch...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0

Data Entry Phase

Departments



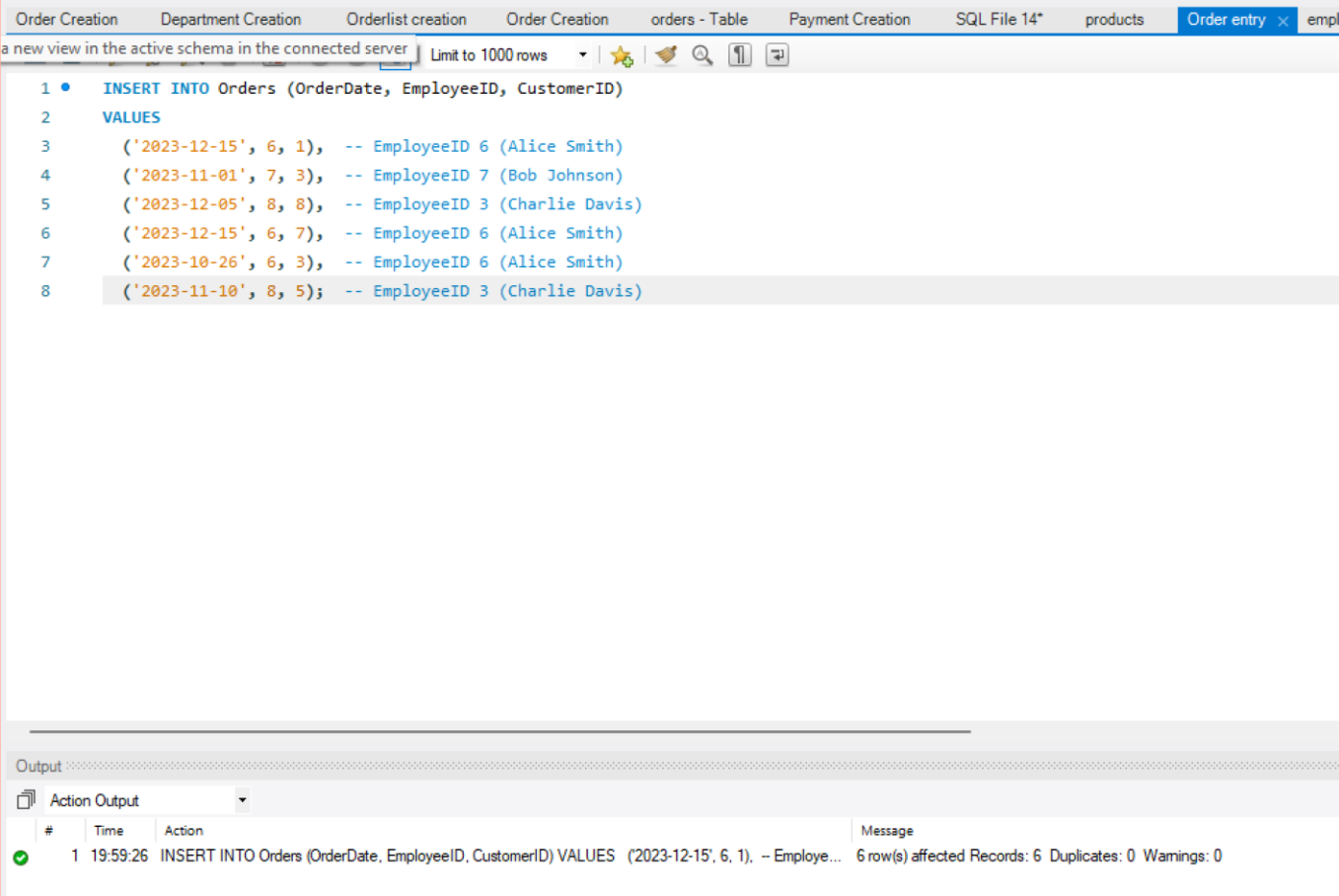
The screenshot shows a SQL IDE window with a toolbar at the top. The toolbar includes icons for file operations (folder, save, lightning bolt, copy, paste, undo, redo, refresh), a 'Limit to 1000 rows' dropdown, and other utility icons. The main text area contains the following SQL code:

```
1 • INSERT INTO departments (DepartmentName)
2 VALUES
3     ('Deliveries'),
4     ('Accounting'),
5     ('Human_Resources');
```


Data Entry Phase

Orders

In this section, in order to make each row clear, the given value has been related to the Employee table with explanations.

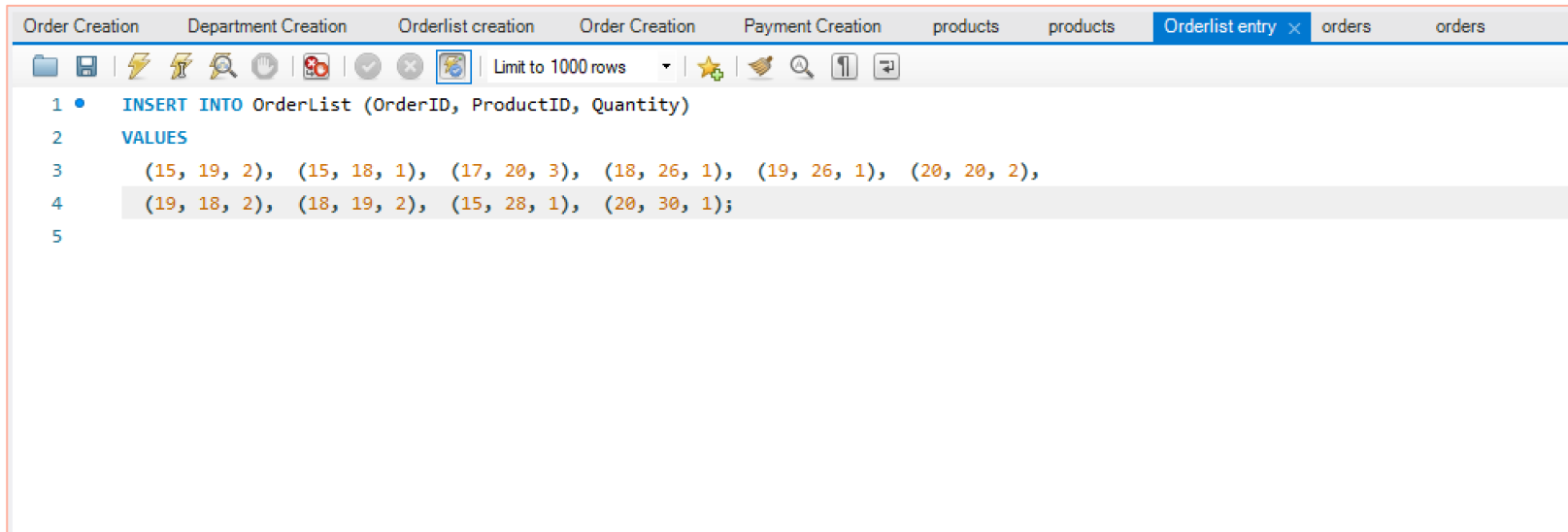


```
1 • INSERT INTO Orders (OrderDate, EmployeeID, CustomerID)
2 VALUES
3 ('2023-12-15', 6, 1), -- EmployeeID 6 (Alice Smith)
4 ('2023-11-01', 7, 3), -- EmployeeID 7 (Bob Johnson)
5 ('2023-12-05', 8, 8), -- EmployeeID 3 (Charlie Davis)
6 ('2023-12-15', 6, 7), -- EmployeeID 6 (Alice Smith)
7 ('2023-10-26', 6, 3), -- EmployeeID 6 (Alice Smith)
8 ('2023-11-10', 8, 5); -- EmployeeID 3 (Charlie Davis)
```

Output

#	Time	Action	Message
1	19:59:26	INSERT INTO Orders (OrderDate, EmployeeID, CustomerID) VALUES ('2023-12-15', 6, 1), -- Employee...	6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0

Data Entry Phase



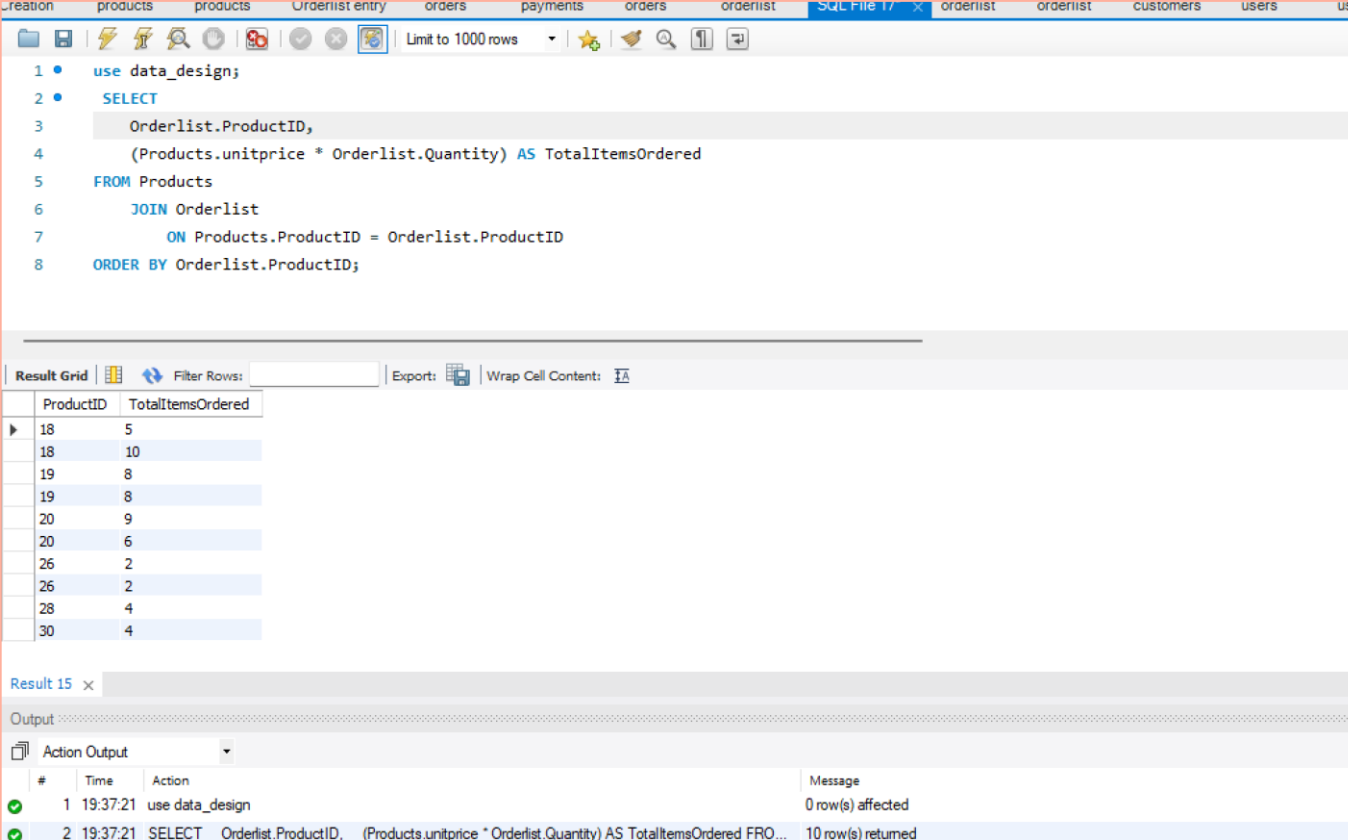
The screenshot shows a database management tool interface with a tabbed menu at the top. The tabs are: Order Creation, Department Creation, Orderlist creation, Order Creation, Payment Creation, products, products, Orderlist entry (selected), orders, and orders. Below the tabs is a toolbar with various icons for file operations, editing, and viewing. A dropdown menu shows 'Limit to 1000 rows'. The main area displays a SQL query for inserting data into the 'OrderList' table. The query is as follows:

```
1 • INSERT INTO OrderList (OrderID, ProductID, Quantity)
2 VALUES
3     (15, 19, 2), (15, 18, 1), (17, 20, 3), (18, 26, 1), (19, 26, 1), (20, 20, 2),
4     (19, 18, 2), (18, 19, 2), (15, 28, 1), (20, 30, 1);
5
```

Data Entry Phase

Payments-Part1

As mentioned before, the Payments table is calculated from the combination of OrderList and Products. Therefore, in this section, the cost of each order item is calculated using a JOIN operation and then multiplied. Then, the total order amount is calculated using the programmer's instructions (as shown in the next slide).



The screenshot displays a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL code:

```
1 • use data_design;
2 • SELECT
3     Orderlist.ProductID,
4     (Products.unitprice * Orderlist.Quantity) AS TotalItemsOrdered
5 FROM Products
6 JOIN Orderlist
7     ON Products.ProductID = Orderlist.ProductID
8 ORDER BY Orderlist.ProductID;
```

The results pane shows a table with two columns: ProductID and TotalItemsOrdered. The data is as follows:

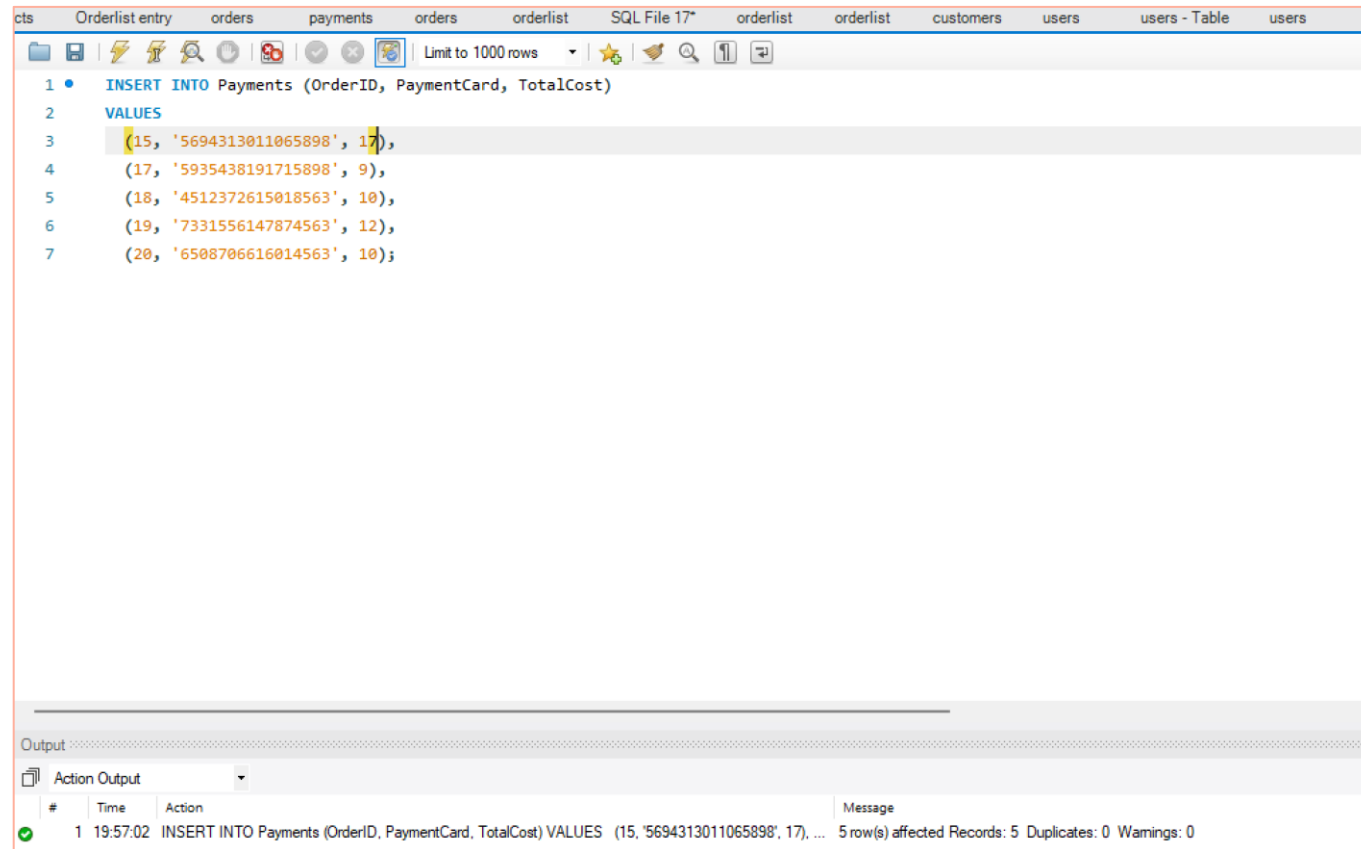
ProductID	TotalItemsOrdered
18	5
18	10
19	8
19	8
20	9
20	6
26	2
26	2
28	4
30	4

Below the results table, the 'Output' pane shows the execution log:

#	Time	Action	Message
1	19:37:21	use data_design	0 row(s) affected
2	19:37:21	SELECT Orderlist.ProductID, (Products.unitprice * Orderlist.Quantity) AS TotalItemsOrdered FRO...	10 row(s) returned

Data Entry Phase

Payments-Part2



The screenshot shows a SQL IDE window with a tab labeled "SQL File 17*". The main editor contains an SQL statement to insert five records into the "Payments" table. The statement is as follows:

```
1 • INSERT INTO Payments (OrderID, PaymentCard, TotalCost)
2 VALUES
3 (15, '5694313011065898', 17),
4 (17, '5935438191715898', 9),
5 (18, '4512372615018563', 10),
6 (19, '7331556147874563', 12),
7 (20, '6508706616014563', 10);
```

Below the editor, the "Output" pane shows the "Action Output" for the executed statement. It indicates that 5 rows were affected, with 0 duplicates and 0 warnings.

#	Time	Action	Message
✓ 1	19:57:02	INSERT INTO Payments (OrderID, PaymentCard, TotalCost) VALUES (15, '5694313011065898', 17), ...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0

Current State of Tables

Tables State Customers

Result Grid										
		Filter Rows:			Edit:		Export/Import:		Wrap Cell Content:	
	CustomerID	FirstName	LastName	HomeNumber	StreetAdd	City	Country	PostCode	PhoneNumber	EmailAdd
▶	1	John	Doe	25	123 Main St	Anytown	UK	AB 12 3CD	07123456789	johndoe@example.com
	2	Jane	Smith	42	456 Elm St	New City	USA	12345	123-456-7890	janesmith@example.com
	3	Bob	Johnson	11A	789 Oak St	Bayside	UK	DE5 4FG	01234567890	bobjohnson@example.com
	4	Emily	Davis	3B	555 Maple St	Metropolis	USA	98765	555-555-5555	emilydavis@example.com
	5	Michael	Brown	22	888 Pine St	Riverdale	UK	M1 1AA	09876543210	michaelbrown@example.com
	6	Sarah	Miller	6	999 Cedar St	Sunnyville	USA	43210	666-666-6666	sarahmiller@example.com
	7	David	Williams	1C	111 Birch St	Greentown	UK	N1 2BB	04445556667	davidwilliams@example.com
	8	Lisa	Jones	54	222 Willow St	Hill Valley	USA	32109	777-777-7777	lisajones@example.com
	9	James	White	99	333 Oakwood St	Bayside	UK	SW1A 1AA	08889990001	jameswhite@example.com
	10	Mary	Taylor	101	444 Elmwood St	Star City	USA	09876	888-888-8888	marytaylor@example.com
✱	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL
customers 1 ×										

Tables State

Departments-Employees-Orders

Result Grid		
	Filter Rows:	Edit:
DepartmentID	DepartmentName	
1	Deliveries	
2	Accounting	
3	Human_Resources	
* NULL	NULL	

departments 2 ×

Result Grid				
	Filter Rows:		Edit:	
EmployeeID	FirstName	LastName	DepartmentID	
6	Alice	Smith	1	
7	Bob	Johnson	3	
8	Charlie	Davis	1	
9	Emily	Brown	2	
10	David	Miller	3	
* NULL	NULL	NULL	NULL	

employees 3 ×

Result Grid				
	Filter Rows:		Edit:	
OrderID	OrderDate	EmployeeID	CustomerID	
15	2023-12-15	6	1	
16	2023-11-01	7	3	
17	2023-12-05	8	8	
18	2023-12-15	6	7	
19	2023-10-26	6	3	
20	2023-11-10	8	5	
* NULL	NULL	NULL	NULL	

orders 4 ×

Tables State

OrderList-Payments









Result Grid			
	OrderID	ProductID	Quantity
▶	15	18	1
	15	19	2
	15	28	1
	17	20	3
	18	19	2
	18	26	1
	19	18	2
	19	26	1
	20	20	2
	20	30	1
*	NULL	NULL	NULL

orderlist5 ×

Result Grid			
	OrderID	PaymentCard	TotalCost
▶	15	5694313011065898	17
	17	5935438191715898	9
	18	4512372615018563	10
	19	7331556147874563	12
	20	6508706616014563	10
*	NULL	NULL	NULL

payments 6 ×

Tables State Products

Result Grid   Filter Rows: <input type="text"/> Edit:    Export/Import:   Wrap Cell Content: 									
	ProductID	Title	Genre	Quantity	UnitPrice	ReleaseDate	Lang	ProductType	Descriptions
▶	16	The Lord of the Rings: The Fellowship of the Ring	Fantasy	5	4	2001	English	Movie	An epic fantasy adventure
	17	The Shawshank Redemption	Drama	3	3	1994	English	Movie	A critically acclaimed drama about hope and red...
	18	The Dark Knight	Action	8	5	2008	English	Book	A gritty and intense superhero crime thriller
	19	The Godfather	Crime Drama	2	4	1972	English	Movie	A classic gangster film about family, power, and...
	20	The Matrix	Sci-Fi	7	3	1999	English	Movie	A mind-bending action film that explores the nat...
	21	Friends	Comedy	10	2	1994	English	TV Series	A sitcom about six friends in New York City
	22	The Office	Mockumentary Comedy	6	3	2005	English	TV Series	A mockumentary about office life in Scranton, P...
	23	Breaking Bad	Crime Drama	4	4	2008	English	TV Series	A high-stakes drama about a chemistry teacher ...
	24	Game of Thrones	Fantasy	8	5	2011	English	TV Series	An epic fantasy drama about power struggles in...
	25	The Wire	Crime Drama	2	4	2002	English	TV Series	A gritty and realistic portrayal of the drug trade...
	26	Abbey Road	Rock	15	2	1969	English	Album	The Beatles final studio album
	27	Thriller	Pop	12	3	1982	English	Album	Michael Jackson best-selling album
	28	The Dark Side of the Moon	Progressive Rock	10	4	1973	English	Album	Pink Floyd landmark album
	29	Kind of Blue	Jazz	8	3	1959	English	Album	Miles Davis masterpiece
	30	Nevermind	Grunge	6	4	1991	English	Album	Nirvana iconic album that changed the course o...
✱	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

products 7 ×

Assessment Tasks

A-Extract all the customers from a specific city

The screenshot shows a SQL query editor with the following components:

- Query Editor:** Contains the SQL query: `SELECT AVG(UnitPrice) AS AvePrice FROM products`. The editor has a toolbar with icons for file operations, query execution, and a "Limit to 1000 rows" dropdown.
- Result Grid:** Displays the query results in a table with one column, "AvePrice", and one row with the value "3.5333".
- Output Panel:** Shows the execution log with a single entry: "1 20:55:40 SELECT AVG(UnitPrice) AS AvePrice FROM products LIMIT 0, 1000". The message indicates "1 row(s) returned".

Result Grid	
	AvePrice
▶	3.5333

#	Time	Action	Message
✓ 1	20:55:40	SELECT AVG(UnitPrice) AS AvePrice FROM products LIMIT 0, 1000	1 row(s) returned

B-Search for a product of a specific genre

The screenshot shows a database query tool interface. At the top, a tab labeled 'products' is active, and a sub-tab 'B_Genre' is selected. The SQL query editor contains the following query:

```
1 • SELECT * FROM products WHERE Genre='Crime Drama'
```

Below the query editor, the 'Result Grid' section displays the query results in a table format. The table has the following columns: ProductID, Title, Genre, Quantity, UnitPrice, ReleaseDate, Lang, ProductType, and Descriptions. The results are as follows:

ProductID	Title	Genre	Quantity	UnitPrice	ReleaseDate	Lang	ProductType	Descriptions
19	The Godfather	Crime Drama	2	4	1972	English	Movie	A classic gangster film about family, power, and...
23	Breaking Bad	Crime Drama	4	4	2008	English	TV Series	A high-stakes drama about a chemistry teacher ...
25	The Wire	Crime Drama	2	4	2002	English	TV Series	A gritty and realistic portrayal of the drug trade...
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

At the bottom of the interface, the 'Output' section shows the execution log. It contains three entries, each with a status icon, a sequence number, a timestamp, the SQL query, and the number of rows returned:

#	Time	Action	Message
✓ 1	20:42:30	SELECT count(FirstName) AS CountOfrepeatedCity FROM Customers WHERE City='Bayside' LIMIT 0,...	1 row(s) returned
✓ 2	20:44:52	SELECT * FROM data_design.products LIMIT 0, 1000	15 row(s) returned
✓ 3	20:50:27	SELECT * FROM products WHERE Genre='Crime Drama' LIMIT 0, 1000	3 row(s) returned

C-Count how many customers are from a specific city

The screenshot shows a SQL IDE interface with a query editor and a results pane. The query editor contains the following SQL statement:

```
1 • SELECT count(FirstName) AS CountOfrepeatedCity FROM Customers WHERE City='Bayside'
```

The results pane displays a single row in a table:

CountOfrepeatedCity
2

Below the results pane, the 'Output' section shows the execution log:

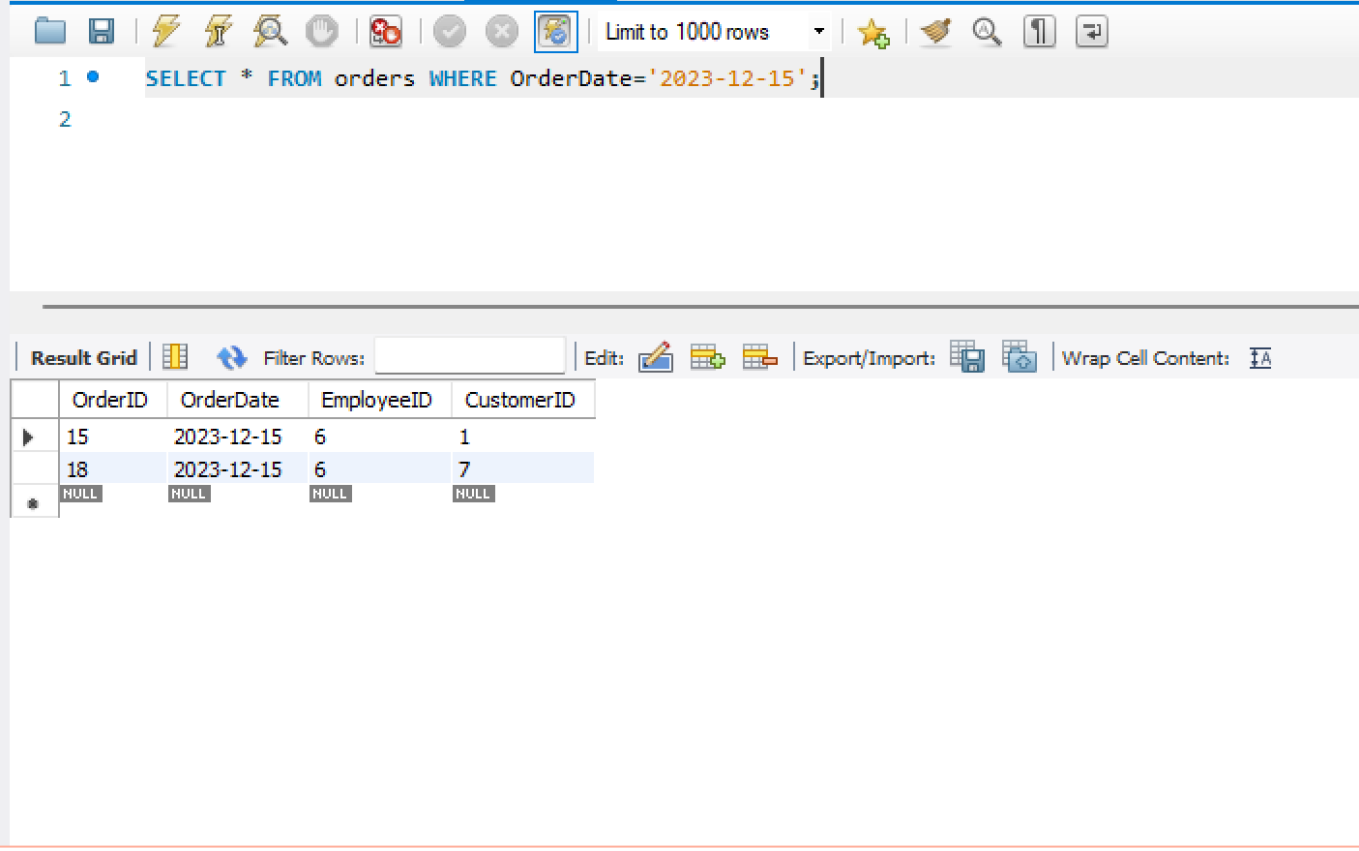
#	Time	Action	Message
1	20:42:30	SELECT count(FirstName) AS CountOfrepeatedCity FROM Customers WHERE City='Bayside' LIMIT 0...	1 row(s) returned

D-Calculate the average of the unit price

The screenshot shows a database query tool interface. At the top, there are tabs for 'products', 'B_Genre', and 'D_AvePrice'. Below the tabs is a toolbar with various icons and a 'Limit to 1000 rows' dropdown. The main area contains a SQL query: `1 • SELECT AVG(UnitPrice) AS AvePrice FROM products`. Below the query is a 'Result Grid' section with a 'Filter Rows' input and an 'Export' button. The grid shows a single row with the column 'AvePrice' and the value '3.5333'. At the bottom, there is a 'Result 1' tab and an 'Output' section. The 'Output' section has a dropdown menu set to 'Action Output' and a table with columns '#', 'Time', 'Action', and 'Message'. The table contains one row:

#	Time	Action	Message
1	20:55:40	SELECT AVG(UnitPrice) AS AvePrice FROM products LIMIT 0, 1000	1 row(s) returned

E-Extract all current orders.



1 • `SELECT * FROM orders WHERE OrderDate='2023-12-15';`

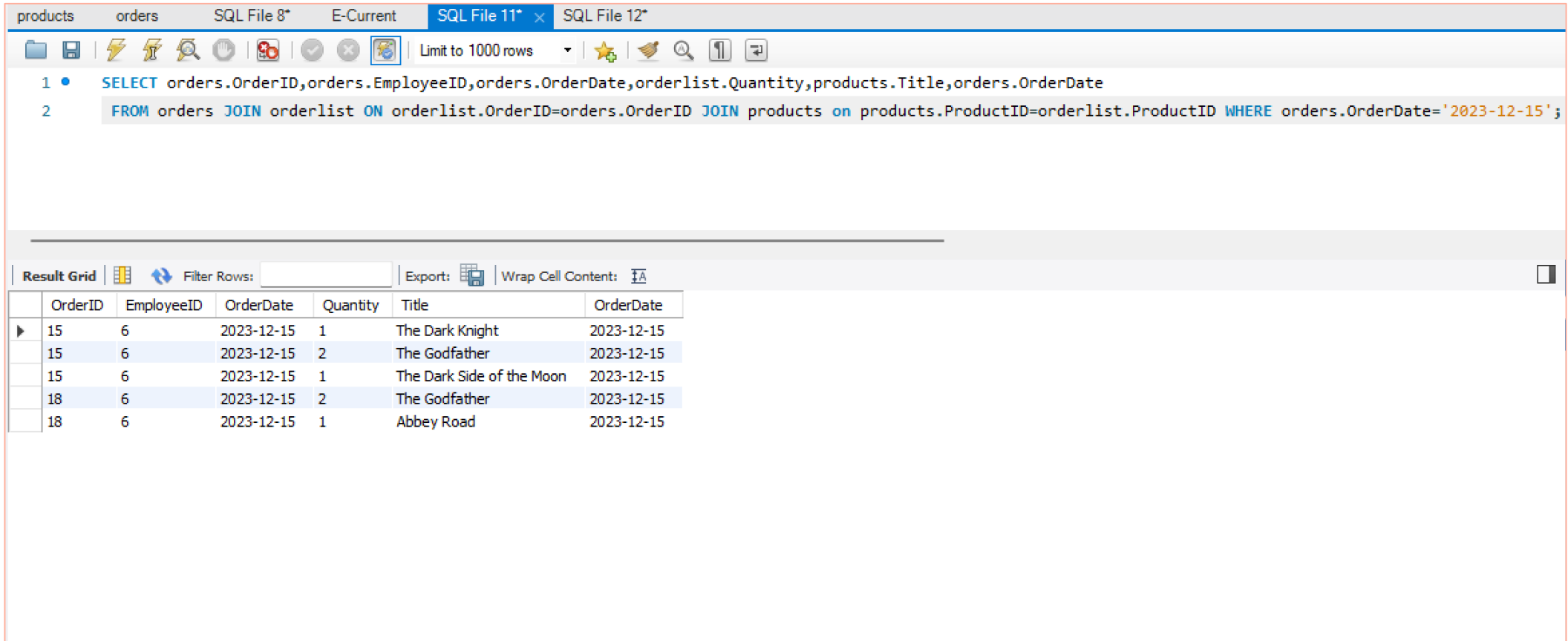
2

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

	OrderID	OrderDate	EmployeeID	CustomerID
▶	15	2023-12-15	6	1
	18	2023-12-15	6	7
*	NULL	NULL	NULL	NULL

Assumption: 2023-12-15 is current date

E-Extract all current orders-Complit Tables.



The screenshot shows a database management interface with multiple tabs: 'products', 'orders', 'SQL File 8*', 'E-Current', 'SQL File 11*', and 'SQL File 12*'. The 'SQL File 11*' tab is active, displaying an SQL query. The query is as follows:

```
1 • SELECT orders.OrderID,orders.EmployeeID,orders.OrderDate,orderlist.Quantity,products.Title,orders.OrderDate
2 FROM orders JOIN orderlist ON orderlist.OrderID=orders.OrderID JOIN products on products.ProductID=orderlist.ProductID WHERE orders.OrderDate='2023-12-15';
```

Below the query editor, the 'Result Grid' is visible, showing the results of the query. The grid has columns for OrderID, EmployeeID, OrderDate, Quantity, Title, and OrderDate. The results are as follows:

	OrderID	EmployeeID	OrderDate	Quantity	Title	OrderDate
▶	15	6	2023-12-15	1	The Dark Knight	2023-12-15
	15	6	2023-12-15	2	The Godfather	2023-12-15
	15	6	2023-12-15	1	The Dark Side of the Moon	2023-12-15
	18	6	2023-12-15	2	The Godfather	2023-12-15
	18	6	2023-12-15	1	Abbey Road	2023-12-15

Assumption: 2023-12-15 is current date

F-Extract all orders for books that have the keyword “the” in their description

The screenshot shows a SQL query editor with a tab labeled 'F_The'. The query is as follows:

```
1 • SELECT OrderList.OrderID, Products.Title
2 FROM Products
3 INNER JOIN OrderList ON OrderList.ProductID = Products.ProductID
4 WHERE Products.Title LIKE 'The%' AND Products.ProductType='Book' ;
```

Below the query, the 'Result Grid' shows two rows of data:

	OrderID	Title
▶	15	The Dark Knight
	19	The Dark Knight

At the bottom, the 'Output' section shows the execution log:

#	Time	Action	Message
✓ 1	22:56:30	SELECT OrderList.OrderID, Products.Title FROM Products INNER JOIN OrderList ON OrderList.Produ...	2 row(s) returned

G-Extract all payments with credit cards for music records

The screenshot shows a database management interface with a tab labeled 'G-PaymentCard'. The SQL query editor contains the following query:

```
1 • SELECT orders.OrderID,orders.EmployeeID,orders.OrderDate,orderlist.Quantity,products.Title,orders.OrderDate,payments.PaymentCard FROM orders JOIN orderlist ON orderlist.OrderID=orders.OrderID JOIN products on products.ProductID=orderlist.ProductID JOIN payments ON orderlist.OrderID=payments.OrderID WHERE products.ProductType='Album' and LENGTH (payments.PaymentCard) = 16 AND payments.PaymentCard REGEXP '^[0-9]+$';
```

Below the query editor, the 'Result Grid' is displayed, showing the following data:

	OrderID	EmployeeID	OrderDate	Quantity	Title	OrderDate	PaymentCard
▶	18	6	2023-12-15	1	Abbey Road	2023-12-15	4512372615018563
	19	6	2023-10-26	1	Abbey Road	2023-10-26	7331556147874563
	15	6	2023-12-15	1	The Dark Side of the Moon	2023-12-15	5694313011065898
	20	8	2023-11-10	1	Nevermind	2023-11-10	6508706616014563

The interface also includes a toolbar with various icons, a 'Limit to 1000 rows' dropdown, and buttons for 'Export' and 'Wrap Cell Content'.

H-Count how many employees handle music records

The screenshot shows a SQL IDE window with the title bar 'H-CountEmployee'. The query editor contains the following SQL statement:

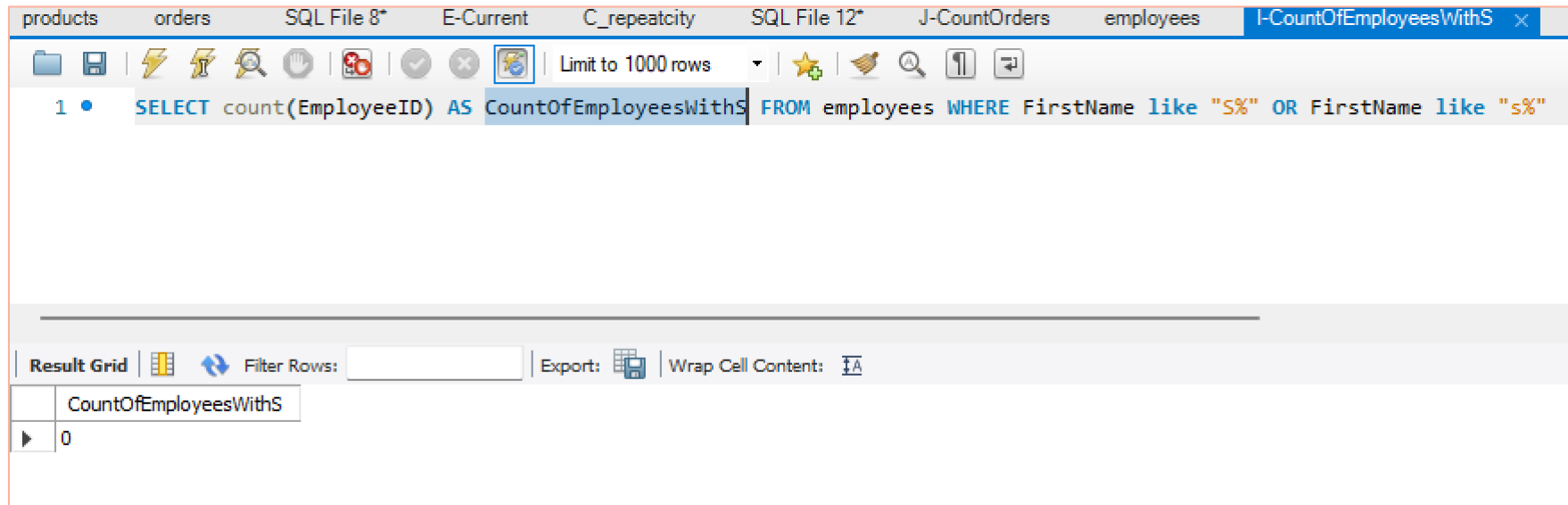
```
1 • SELECT count(DISTINCT orders.EmployeeID) as SUMER FROM orders JOIN orderlist ON orderlist.OrderID=orders.  
OrderID JOIN products on products.ProductID=orderlist.ProductID WHERE products.ProductType='Album' ;
```

Below the query editor, the 'Result Grid' tab is active, displaying the following result:

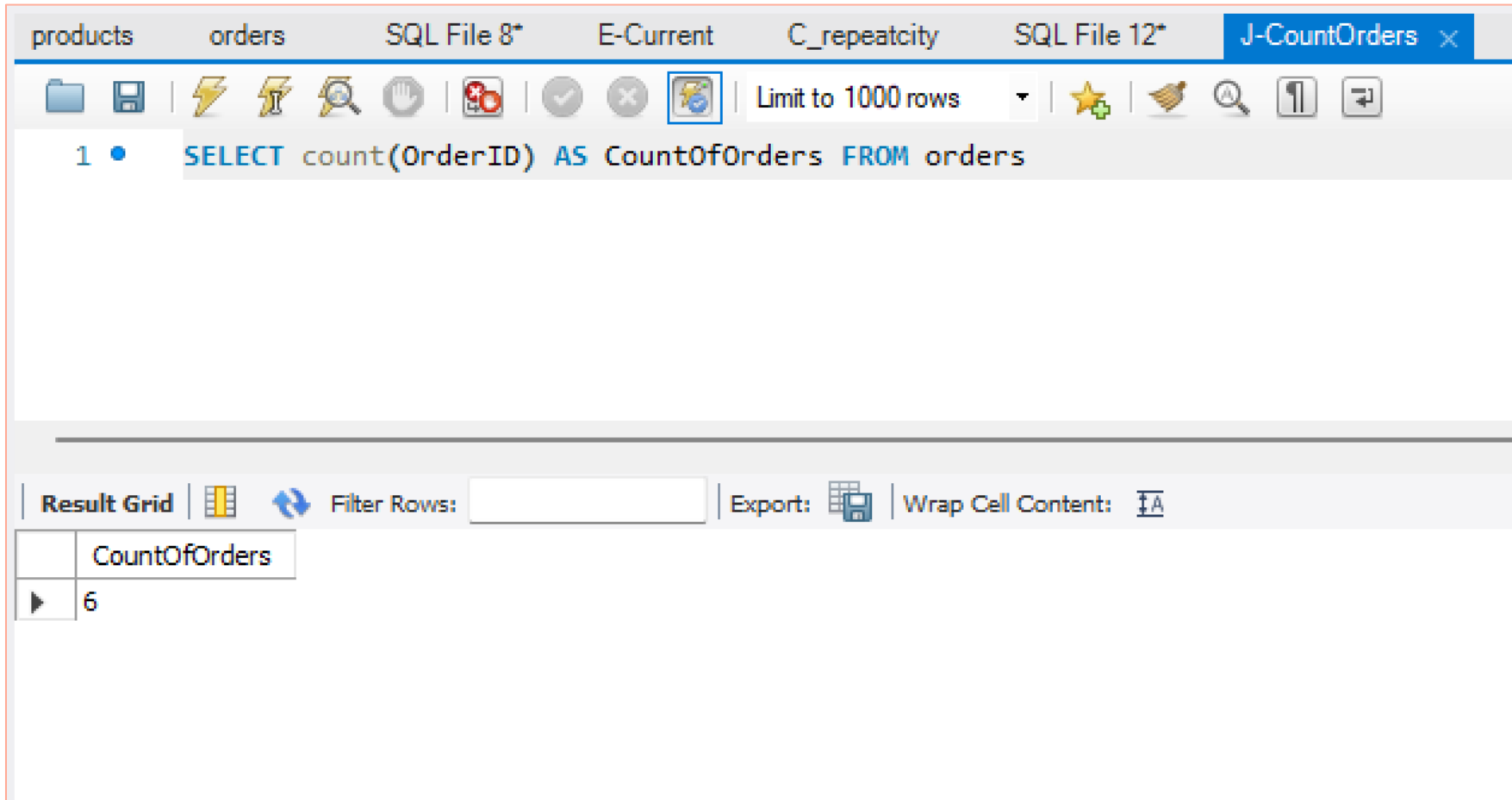
SUMER
2

The interface includes a toolbar with icons for file operations, a 'Limit to 1000 rows' dropdown, and buttons for 'Filter Rows', 'Export', and 'Wrap Cell Content'.

I-Count the employees with the first name starting with the letter S



J-Count how many orders are in the system



The screenshot shows a database application window with multiple tabs. The active tab is 'J-CountOrders'. The SQL editor displays the following query:

```
1 • SELECT count(OrderID) AS CountOfOrders FROM orders
```

The query is executed, and the result is displayed in a table with one row and one column, 'CountOfOrders'.

	CountOfOrders
▶	6

The interface includes a toolbar with various icons for file operations, a 'Limit to 1000 rows' dropdown, and a 'Result Grid' section with a 'Filter Rows' input field and an 'Export' button.

Use Cases

Use Cases-1

Get details of a customer order

The screenshot displays a database query tool interface. At the top, a toolbar contains various icons for file operations, execution, and viewing. Below the toolbar, a SQL query is entered in a text area:

```
1 • SELECT customers.FirstName, customers.LastName, orders.OrderDate, payments.TotalCost
2 FROM customers
3 JOIN orders ON orders.CustomerID = Customers.CustomerID
4 JOIN payments ON payments.OrderID = orders.OrderID
5 WHERE orders.OrderID = 15;
```

Below the query editor, a "Result Grid" section shows the query results. It includes a "Filter Rows" input field, an "Export" button, and a "Wrap Cell Content" button. The results are displayed in a table with the following data:

	FirstName	LastName	OrderDate	TotalCost
▶	John	Doe	2023-12-15	17

At the bottom of the interface, a "Result 1" tab is active. Below it, an "Output" section shows the execution details. The "Action Output" dropdown is set to "Message". The output table shows the following information:

#	Time	Action	Message
✓ 1	17:31:10	SELECT customers.FirstName, customers.LastName, orders.OrderDate, payments.TotalCost FROM c...	1 row(s) returned

Use Cases-2

Get total quantity sold of each product:

The screenshot shows a SQL query editor with a toolbar at the top. The query is as follows:

```
1 SELECT products.ProductID, products.Title, SUM(orderlist.Quantity) AS TotalSold
2 FROM orderlist
3 JOIN products ON products.ProductID = orderlist.ProductID
4 GROUP BY orderlist.ProductID;
```

Below the query editor, there is a 'Result Grid' section. It includes a 'Filter Rows' input field, an 'Export' button, and a 'Wrap Cell Content' checkbox. The results are displayed in a table with the following data:

ProductID	Title	TotalSold
18	The Dark Knight	3
19	The Godfather	4
28	The Dark Side of the Moon	1
20	The Matrix	5
26	Abbey Road	2
30	Nevermind	1

At the bottom, there is a 'Result 1' tab and an 'Output' section. The 'Output' section shows the execution of the query, with a message indicating that 6 row(s) were returned.

Result 1 x

Output

Action Output

#	Time	Action	Message
1	17:47:03	SELECT products.ProductID, products.Title, SUM(orderlist.Quantity) AS TotalSold FROM orderlist JO...	6 row(s) returned

Use Cases-3

Get customers who have made at least 2 purchases in the last 3 months:

- The DATE_SUB() function is used to subtract a time/date interval from a given date and retrieve the resulting date(MySQL DATE_SUB() Function, no date).
- The CURDATE() function returns the current date in numeric or string format(MySQL CURDATE() Function, no date).
- Selected INTERVAL is 3 month to show query functionality.

```
2 FROM customers
3 JOIN orders ON customers.CustomerID = orders.CustomerID
4 WHERE orders.OrderDate >= DATE_SUB(CURDATE(), INTERVAL 3 MONTH)
5 GROUP BY customers.CustomerID
6 HAVING COUNT(orders.OrderID) >= 2;
```

Result Grid

	CustomerID	FirstName	LastName	OrderCount
▶	3	Bob	Johnson	2

Result 7 x

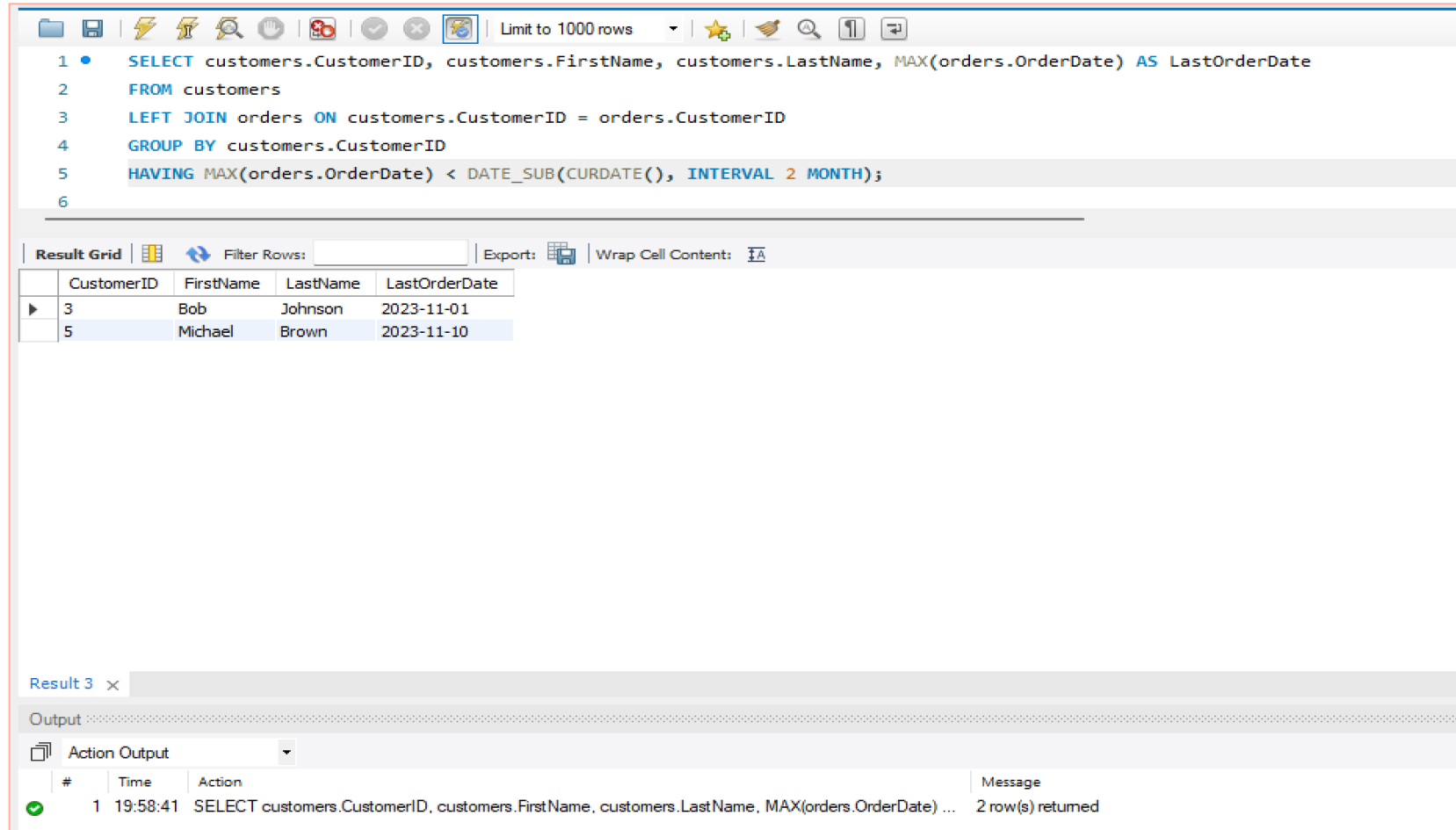
Output

Action Output

#	Time	Action	Message
✓ 1	19:43:43	SELECT customers.CustomerID, FirstName, LastName, COUNT(OrderID) AS OrderCount FROM custo...	1 row(s) returned

Use Cases-4

Identify customers who haven't made a purchase in the last 2 month



```
1 • SELECT customers.CustomerID, customers.FirstName, customers.LastName, MAX(orders.OrderDate) AS LastOrderDate
2 FROM customers
3 LEFT JOIN orders ON customers.CustomerID = orders.CustomerID
4 GROUP BY customers.CustomerID
5 HAVING MAX(orders.OrderDate) < DATE_SUB(CURDATE(), INTERVAL 2 MONTH);
6
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

	CustomerID	FirstName	LastName	LastOrderDate
▶	3	Bob	Johnson	2023-11-01
	5	Michael	Brown	2023-11-10

Result 3 x

Output

Action Output

#	Time	Action	Message
✓ 1	19:58:41	SELECT customers.CustomerID, customers.FirstName, customers.LastName, MAX(orders.OrderDate) ...	2 row(s) returned

MAX() is a function with find maximum in a list.

Use Cases-5

get the name and ID of the department that has the most employees

The screenshot shows a SQL IDE interface. The query editor contains the following SQL code:

```
1 • SELECT DepartmentName, DepartmentID FROM departments
2 WHERE DepartmentID = (SELECT DepartmentID FROM employees GROUP BY DepartmentID ORDER BY COUNT(*) DESC LIMIT 1)
```

Below the query editor, the 'Result Grid' tab is active, displaying the following data:

	DepartmentName	DepartmentID
▶	Deliveries	1
*	NULL	NULL

At the bottom of the IDE, the 'Output' pane shows the 'Action Output' for the query execution:

#	Time	Action	Message
✓ 1	20:21:19	SELECT DepartmentName, DepartmentID FROM departments WHERE DepartmentID = (SELECT De...	1 row(s) returned

Finally

Conclusion-1

- In summary, this project involved designing a database to support an online shop selling books, movies, and music records worldwide. Several key considerations were taken into account during the design process, including performance, scalability, security, and maintainability. Lean principles were applied to streamline the design and ensure it delivers maximum value to customers.

Conclusion-2

- Prior to executing example use cases to get significant insights, test data was inputted to populate the tables. In summary, the well-considered design allows the database to effectively meet the operational and analytical requirements of the online shop.

Reflection

- This database design showed me how to use classroom principles of requirements analysis, data modelling, normalisation, and SQL queries. I learned how to design methodically around end-user needs. Lean concepts also helped me maximise value rather than merely finish features.
- In the future, I would focus more on security and access control than during this deployment. Considering flexibility and extensibility beforehand has positioned the database for future expansion as the online shop's offers grow. Use case testing yielded valuable ideas, but more validation is needed. This project greatly improved my database design skills through hands-on learning.

References

- Lui M [Photograph] 2009 Available at: <https://www.timeout.com/sydney/shopping/title-music-film-books-surry-hills>.
- Silberschatz, Abraham, et al. Database System Concepts. New York, Ny, Mcgraw-Hill Education, 2020, pp. 1–25.
- Graupp, P. (2022) What Are Lean Operations? Everything You Need to Know, TWI Institute. Available at: <https://www.twi-institute.com/lean-operations/> (Accessed: 10 January 2024).
- MySQL DATE_SUB() Function (no date). Available at: https://www.w3schools.com/sql/func_mysql_date_sub.asp (Accessed: 23 January 2024).
- MySQL CURDATE() Function (no date). Available at: https://www.w3schools.com/sql/func_mysql_curdate.asp (Accessed: 23 January 2024).