

AMERICAN INTERNATIONAL UNIVERSITY BANGLADESH
(AIUB)

FACULTY OF SCIENCE & TECHNOLOGY



Course Title
INTRODUCTION TO DATABASE (2108)

Semester: Fall (2024-2025)

Section: [J]

TITLE

Restaurant Management System

Supervised By
MD SAJID BIN-FAISAL

Submitted By: Group no: 03

Name	ID
ADIBA HOSSAIN	23-52389-2
SANZIDA BINTA HUQ	23-52959-2
ABEAZ MOHTASIM	23-54631-3

TABLE OF CONTENTS

TOPICS	Page no.
Title Page	1
Table of Content	2
1. Introduction	3
2. Case Study	3
3. ER Diagram	4
4. Normalization	5-6
5. Finalization	6
6. Table Creation	6-10
7. Data Insertion	11
8. Query Test	12-17
9. DB connection	17-19
10. Conclusion	20

Introduction

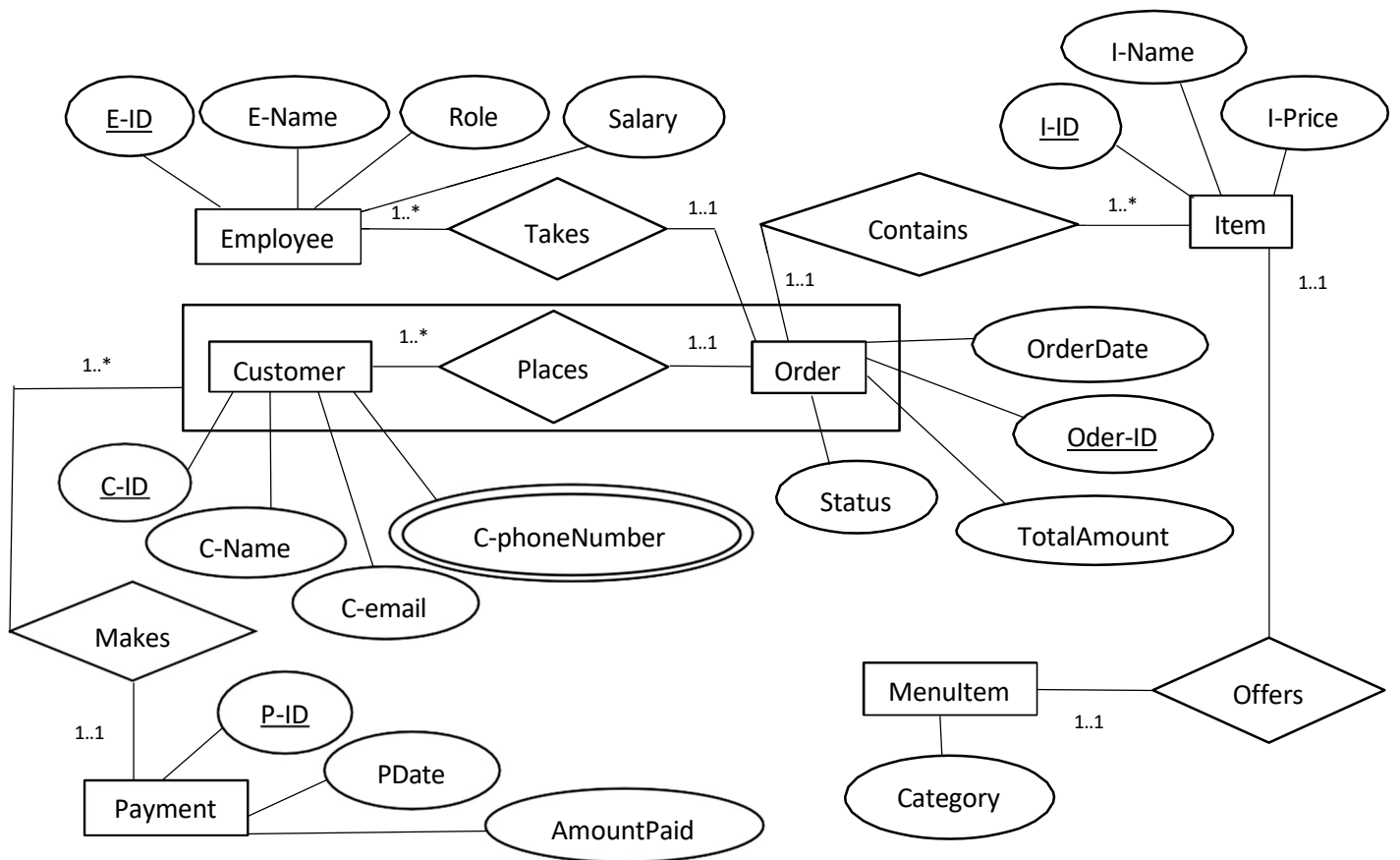
The Restaurant Management System aims to streamline restaurant operations by efficiently managing customer orders, employee tasks, payments, and inventory. Utilizing technologies like databases, front-end frameworks, and back-end servers, the system provides a seamless experience for customers and staff alike. The project's goal is to enhance customer satisfaction, improve order accuracy, and optimize restaurant workflows. It benefits restaurant owners, employees, and customers by offering an organized and automated approach to daily operations. Targeted users include restaurant staff, management, and customers who seek a quick and hassle-free dining experience.

Case Study / Scenario

RESTAURANT MANAGEMENT SYSTEM

In a Restaurant Management System, a Customer can place multiple Orders, but each Order is associated with only one Customer. The Customer is identified by CustomerID, Name, PhoneNumber, and Email. The Order is identified by OrderID and includes details such as OrderDate, TotalAmount, and the status of the order (e.g., pending, completed). An Order can contain multiple Items, and each Item is identified by ItemID, ItemName, and Price. The restaurant offers various MenuItem s like appetizers, main courses, and desserts. Each MenuItem can belong to a specific category (e.g., appetizer, dessert, main course). The restaurant also has Employees who are assigned to take orders and serve customers. Each Employee has an EmployeeID, Name, Role, and Salary. Employees can serve multiple Orders, but each Order is served by one Employee at a time. Finally, Payments are made by the Customer for each Order. Each Payment is identified by PaymentID, PaymentDate, and AmountPaid. A Customer can make multiple Payments, and each Payment is linked to a specific Order.

ER Diagram



Normalization

Places: 1 to many

UNF: C-ID, C-Name, C-Email, C-phoneNumber, order-ID, orderDate, totalAmount, status

1NF: C-ID, C-Name, C-Email, C-phoneNumber, Order-ID, orderDate, totalAmount, status

2NF: 1) C-ID, C-Name, C-Email, C-phoneNumber, order-ID(FK)

2) Order-ID, orderDate, totalAmount, status

3NF: same as 2NF

Makes: 1 to many

UNF: C-ID, C-Name, C-Email, C-phoneNumber, order-ID, orderDate, totalAmount, status, P-ID, P-Date, Amount-Paid

1NF: C-ID, C-Name, C-Email, C-phoneNumber, Order-ID, orderDate, totalAmount, status, P-ID, P-Date, Amount-Paid

2NF: 1) C-ID, C-Name, C-Email, C-phoneNumber, Order-ID, orderDate, totalAmount, status, P-ID(FK)

2) P-ID, P-Date, Amount-Paid

3NF: same as 2NF

Takes: 1 to many

UNF: E-ID, E-Name, Role, Salary, Order-ID, OrderDate, TotalAmount, Status

1NF: E-ID, E-Name, Role, Salary, Order-ID, OrderDate, TotalAmount, Status

2NF: 1) E-ID, E-Name, Role, Salary, Order-ID(FK)

2) Order-ID, OrderDate, TotalAmount, Status

3NF: same as 2NF

Offers: 1 to 1

UNF: I-ID, I-Name, I-Price, Category

1NF: I-ID, I-Name, I-Price, Category

2NF: 1) I-ID, I-Name, I-Price

2) Category

3NF: same as 2NF

Contains: 1 to many**UNF:** Order-ID, OrderDate, TotalAmount, Status, I-ID, I-Name, I-Price**1NF:** Order-ID, OrderDate, TotalAmount, Status, I-ID, I-Name, I-Price**2NF:** 1) Order-ID, OrderDate, TotalAmount, Status2) I-ID, I-Name, I-Price, Order-ID(FK)**3NF:** same as 2NF**Finalization**

 Table name
  Primary key
  Foreign key

1. C-ID, C-Name, C-Email, C-phoneNumber, order-ID(FK) [Customer]
2. Order-ID, orderDate, totalAmount, status [OrderC]
3. P-ID, P-Date, Amount-Paid [Payment]
4. E-ID, E-Name, Role, Salary, Order-ID(FK) [Employee]
5. I-ID, I-Name, I-Price [Item]
6. Category [Category]
7. C-ID, C-Name, C-Email, C-phoneNumber, Order-ID, orderDate, totalAmount, status, P-ID(FK) [Makes]
8. I-ID, I-Name, I-Price, Order-ID(FK) [Contains]
9. ~~Order-ID, orderDate, totalAmount, status~~
10. ~~Order-ID, orderDate, totalAmount, status~~

Table Creation (DDL Operations)

StudentID1: 23-52959-2 Name: SANZIDA BINTA HUQ	StudentID3: 23-54631-3 Name: ABEAZ MOHTASIM
StudentID2: 23-52389-2 Name: ADIBA HOSSAIN	X
CO4: Creating DML, DDL using Oracle and connection with ODBC/JDBC for existing JAVA application	
PO-e-2: Use modern engineering and IT tools for prediction and modeling of complex computer science and engineering problem	Marks

(next page) 

Table creation:

ORACLE® Database Express Edition

User: RESTAURANT

Home > SQL > SQL Commands

☒ Autocommit Display 10 ▼

```
create table "Order" (oid number(4) primary key, odate DATE, tamount
number(6, 2), status VARCHAR2(20))
rename "Order" to OrderC
describe OrderC
```

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
ORDERC	OID	Number	-	4	0	1	-	-	-
	ODATE	Date	7	-	-	-	✓	-	-
	TAMOUNT	Number	-	6	2	-	✓	-	-
	STATUS	Varchar2	20	-	-	-	✓	-	-

1 - 4

Fig 1: Table (2) creation command & description of OrderC

ORACLE® Database Express Edition

User: RESTAURANT

Home > SQL > SQL Commands

☒ Autocommit Display 10 ▼

```
create table Customer (cid number(4) primary key, cname VARCHAR2(30),
cemail VARCHAR2(30), cphoneNumber VARCHAR2(15), oid
number(4), constraint oi foreign key (oid) references OrderC (oid))
describe Customer
```

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
CUSTOMER	CID	Number	-	4	0	1	-	-	-
	CNAME	Varchar2	30	-	-	-	✓	-	-
	CEMAIL	Varchar2	30	-	-	-	✓	-	-
	CPHONENUMBER	Varchar2	15	-	-	-	✓	-	-
	OID	Number	-	4	0	-	✓	-	-

1 - 5

Fig 2: Table (1) creation command & description of Customer

ORACLE® Database Express Edition

User: RESTAURANT

Home > SQL > SQL Commands

☒ Autocommit Display 10 ▼

```
create table Payment (pid number(4) primary key, pdate DATE,
amountPaid number(6, 2))
describe Payment
```

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
PAYMENT	PID	Number	-	4	0	1	-	-	-
	PDATE	Date	7	-	-	-	✓	-	-
	AMOUNT PAID	Number	-	6	2	-	✓	-	-

1 - 3

Fig 3: Table (3) creation command & description of Payment

ORACLE® Database Express Edition

User: RESTAURANT

Home > SQL > SQL Commands

☒ Autocommit Display 10 ▼

```
create table Employee (eid number (6) primary key, ename
varchar2(30), role varchar2(10), salary varchar2(5), oid number (4),
constraint od foreign key(oid) references OrderC(oid))
describe Employee

alter table Employee modify (eid number (3), salary varchar2(4))
describe Employee
```

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
EMPLOYEE	EID	Number	-	3	0	1	-	-	-
	ENAME	Varchar2	30	-	-	-	✓	-	-
	ROLE	Varchar2	10	-	-	-	✓	-	-
	SALARY	Varchar2	4	-	-	-	✓	-	-
	OID	Number	-	4	0	-	✓	-	-

1 - 5

Fig 4: Table (4) creation command & description of Employee

ORACLE® Database Express Edition

User: RESTAURANT

Home > SQL > **SQL Commands**

☒ Autocommit Display 10 ▼

```
create table Item (iid number (35) primary key, iname varchar2(40),
iprice varchar2(7))
describe Item

alter table Item modify (iid number (4), iprice varchar2(3))
describe Item
```

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
ITEM	IID	Number	-	4	0	1	-	-	-
	INAME	Varchar2	40	-	-	-	✓	-	-
	IPRICE	Varchar2	3	-	-	-	✓	-	-

1 - 3

Fig 5: Table (5) creation command & description of Item

ORACLE® Database Express Edition

User: RESTAURANT

Home > SQL > **SQL Commands**

☒ Autocommit Display 10 ▼

```
create table Category (category varchar2(12))
describe Category
```

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
CATEGORY	CATEGORY	Varchar2	12	-	-	-	✓	-	-

1 - 1

Fig 6: Table (6) creation command & description of Category

User: RESTAURANT

Home > SQL > SQL Commands

☒ Autocommit Display 10 ▼

```
create table Makes (cid number(4), cname VARCHAR2(30), cemail
VARCHAR2(30), cphoneNumber VARCHAR2(15), oid number(4) primary key,
odate DATE, tamount number(6, 2), status VARCHAR2(20), pid number(4),
constraint pi foreign key(pid) references payment(pid))
describe Makes
```

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
MAKES	CID	Number	-	4	0	-	✓	-	-
	CNAME	Varchar2	30	-	-	-	✓	-	-
	CEMAIL	Varchar2	30	-	-	-	✓	-	-
	CPHONENUMBER	Varchar2	15	-	-	-	✓	-	-
	OID	Number	-	4	0	1	-	-	-
	ODATE	Date	7	-	-	-	✓	-	-
	TAMOUNT	Number	-	6	2	-	✓	-	-
	STATUS	Varchar2	20	-	-	-	✓	-	-
	PID	Number	-	4	0	-	✓	-	-

1 - 9

Fig 7: Table (7) creation command & description of Makes

User: RESTAURANT

Home > SQL > SQL Commands

☒ Autocommit Display 10 ▼

```
varchar2(40), iprice varchar2(7), oid number (4), constraint oii
foreign key(oid) references OrderC(oid))
describe Contains

alter table Contains modify (iid number (4), iprice varchar2(3))
describe Contains
```

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
CONTAINS	IID	Number	-	4	0	1	-	-	-
	INAME	Varchar2	40	-	-	-	✓	-	-
	IPRICE	Varchar2	3	-	-	-	✓	-	-
	OID	Number	-	4	0	-	✓	-	-

1 - 4

Fig 8: Table (8) creation command & description of Contains

Inserted Values in the tables

INSERTION

OID	ODATE	TAMOUNT	STATUS
1001	11-OCT-24	1200.21	Pending
1002	12-NOV-24	1150.52	Completed
1003	15-DEC-24	1010.43	Pending

Table2(OrderC)

CID	CNAME	CEMAIL	CPHONENUMBER	OID
2222	Adiba	Adiba@gmail.com	01722222222	1001
2223	Abeaz	Abeaz@gmail.com	01833333333	1002
2224	Sanzida	Sanzida@gmail.com	01944444444	-

Table1 (Customer)

PID	PDATE	AMOUNTPAID
3132	11-OCT-24	1250.11
3133	12-OCT-24	1200.52
3134	15-DEC-24	1100.51

Table3(Payment)

EID	ENAME	ROLE	SALARY	OID
441	Robi	Waiter	5000	1001
442	Sakib	Waiter	5100	1002
443	Himu	Waiter	3100	1003

Table4(Employee)

IID	INAME	IPRICE
5152	Pizza	400
5153	Coffee	250
5154	Tea	100

Table5(Item)

CATEGORY
Appetizer
Dessert
MainCourse

Table6(Category)

IID	INAME	IPRICE	OID
5152	Pizza	400	1001
5153	Coffee	250	1002
5154	Tea	100	1003
5155	Burger	150	-

Table8(Contains)

CID	CNAME	CEMAIL	CPHONENUMBER	OID	ODATE	TAMOUNT	STATUS	PID
2222	Adiba	Adiba@gmail.com	01722222222	1001	11-OCT-24	1200.21	Pending	3132
2223	Abeaz	Abeaz@gmail.com	01833333333	1002	12-NOV-24	1150.52	Completed	3133
2224	Sanzida	Sanzida@gmail.com	01944444444	1003	15-DEC-24	1010.43	Pending	3134

Table7(Makes)

Query Test in DB

SINGLE ROW FUNCTION:

→ Write a query to show the email address of the customer with CustomerID = 2223 and rename the email column to CustomerEmail.

User: RESTAURANT

Home > SQL > **SQL Commands**

☒ Autocommit Display 10 ▼

```
select cemail from Customer where cid = 2223
```

Fig :Single row function command

CEMAIL
Abeaz@gmail.com

Fig : Result of the single row function

MULTIPLE ROW FUNCTION:

→ Write a query to show the average salary of all waiters in your database and give an appropriate name to that column.

```
select AVG(salary) from Employee where role = 'Waiter'
```

Fig :Multiple row function command

AVG(SALARY)
4400

Fig : Result of the multiple row function

SINGLE ROW SUBQUERY:

→ Write a query to show the name of the employee with the highest salary in your database.

```
select ename from Employee where salary = (select MAX(salary) from Employee)
```

Fig :Single row subquery command

ENAME
Sakib

Fig : Result of the single row subquery

MULTIPLE ROW SUBQUERY:

→ Write a query to retrieve the details of all customers who have placed orders with a total amount greater than the average total amount of all orders.

```
select c.cid, c.cname as CustomerName, o.oid, o.odate, o.tamount from Customer c join OrderC o on  
c.oid = o.oid where o.tamount > (select AVG(tamount) from OrderC)
```

Fig : Multiple row subquery command

CID	CUSTOMERNAME	OID	ODATE	TAMOUNT
2222	Adiba	1001	11-OCT-24	1200.21
2223	Abeaz	1002	12-NOV-24	1150.52

Fig : Result of the Multiple row subquery

TWO TYPES OF JOINING:

Equijoin:

→ Write a query to display the names of customers, the order details (OrderID, OrderDate, TotalAmount), and the names of the employees who served those orders.

```
select o.oid, o.odate, o.tamount, e.ename as EmployeeName from OrderC o join
Employee e on o.oid = e.oid where o.tamount > (select AVG(tamount) from
OrderC)
```

Fig : Equijoin command

OID	ODATE	TAMOUNT	EMPLOYEENAME
1001	11-OCT-24	1200.21	Robi
1002	12-NOV-24	1150.52	Sakib

Fig : Result of the equijoin

Non Equijoin:

→ Write a query to retrieve the details of orders with a total amount greater than the average total amount of all orders.

```
select o.oid, o.odate, o.tamount, e.ename as EmployeeName from OrderC o join
Employee e on o.oid = e.oid where o.tamount > (select AVG(tamount) from
OrderC)
```

Fig : Non equijoin command

OID	ODATE	TAMOUNT	EMPLOYEENAME
1001	11-OCT-24	1200.21	Robi
1002	12-NOV-24	1150.52	Sakib

Fig : Result of the non equijoin

SIMPLE VIEW:

→ Create a view named CustomerOrderView that shows the Customer's Name, OrderID, OrderDate, TotalAmount, Employee Name, and Payment Amount for all orders placed by customers. The view should include details of the customer, the order they placed, the employee who served the order, and the payment made for the order.

```
create view CustomerOrderView as select c.cname as CustomerName, o.oid as OrderID, o.odate as OrderDate, o.tamount as TotalAmount, e.ename as EmployeeName, p.amountPaid as PaymentAmount from Customer c join Makes m on c.cid = m.cid join OrderC o on m.oid = o.oid join Employee e on o.oid = e.oid join Payment p on m.pid = p.pid
```

Fig: Simple view creation command

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
CUSTOMERORDER VIEW	CUSTOMER NAME	Varchar 2	30	-	-	-	✓	-	-
	ORDERID	Number	-	4	0	-	-	-	-
	ORDERDATE	Date	7	-	-	-	✓	-	-
	TOTALAMOUNT	Number	-	6	2	-	✓	-	-
	EMPLOYEE NAME	Varchar 2	30	-	-	-	✓	-	-
	PAYMENTAMOUNT	Number	-	6	2	-	✓	-	-
1 - 6									

Fig: Simple view creation command

CUSTOMERNAME	ORDERID	ORDERDATE	TOTALAMOUNT	EMPLOYEE NAME	PAYMENTAMOUNT
Adiba	1001	11-OCT-24	1200.21	Robi	1250.11
Abeaz	1002	12-NOV-24	1150.52	Sakib	1200.52
Sanzida	1003	15-DEC-24	1010.43	Himu	1100.51

Fig: Result of the simple view as a whole table

COMPLEX VIEW:

→ Create a view for customer orders so that the user can see the following information - Customer Name, Customer Phone Number, Customer Email, Order ID, Order Date, Total Amount, Order Status, Employee Name (who served the order), Menu Item Name, Item Price, Payment Date, Amount Paid.

```
Create view CustomerOrderDetails as select c.cname as CustomerName, c.cphoneNumber as
CustomerPhone, c.cemail as CustomerEmail, o.oid as OrderID, o.odate as OrderDate, o.tamount as
TotalAmount, o.status as OrderStatus, e.ename as EmployeeName, i.iname as MenuItemName,
i.iprice as ItemPrice, p.pdate as PaymentDate, p.amountPaid as AmountPaid from Customer c join
OrderC o on c.oid = o.oid join Employee e on o.oid = e.oid join Contains ci on o.oid = ci.oid
join Item i on ci.iid = i.iid join Makes m on o.oid = m.oid join Payment p on m.pid = p.pid
```

Fig: Complex view creation command

Table	Column	Data Type	Length	Precision	Scale	Primary Key	Nullable	Default	Comment
CUSTOMERORDERDETAILS	CUSTOMERNAME	Varchar 2	30	-	-	-	✓	-	-
	CUSTOMERPHONE	Varchar 2	15	-	-	-	✓	-	-
	CUSTOMEREMAIL	Varchar 2	30	-	-	-	✓	-	-
	ORDERID	Number	-	4	0	-	-	-	-
	ORDERDATE	Date	7	-	-	-	✓	-	-
	TOTALAMOUNT	Number	-	6	2	-	✓	-	-
	ORDERSTATUS	Varchar 2	20	-	-	-	✓	-	-
	EMPLOYEENAME	Varchar 2	30	-	-	-	✓	-	-
	MENUITEMNAME	Varchar 2	40	-	-	-	✓	-	-
	ITEMPRICE	Varchar 2	3	-	-	-	✓	-	-
	PAYMENTDATE	Date	7	-	-	-	✓	-	-
	AMOUNTPAID	Number	-	6	2	-	✓	-	-

1 - 12

Fig: complex view creation command

CUST OMER NAME	CUST OMER PHON E	CUST OMER EMAIL	OR DE RID	ORD ERD ATE	TOT ALA MOU NT	ORD ERST ATU S	EMPL OYEE NAME	MENU ITEM NAME	ITE MP RIC E	PAY MEN TDATE	AMO UNT PAID
Adiba	0172222 2222	Adiba@gmail.com	100 1	11-OCT-24	1200.2 1	Pending	Robi	Pizza	400	11-OCT-24	1250. 11
Abeaz	0183333 3333	Abeaz@gmail.com	100 2	12-NOV-24	1150.5 2	Completed	Sakib	Coffee	250	12-OCT-24	1200. 52

Fig: Result of the complex view as a whole table

Description of a Successful DB connection

DATABASE CONNECTION:

SANZIDA BINTA HUQ (23-52959-2)

For my part of the project, I will be connecting to an Oracle database through NetBeans. The first step I took was ensuring that the Oracle Database is running. I then made sure to add the JDBC driver (ojdbc8.jar) to my project's classpath. After that, I created a new Java class in NetBeans to handle the connection. I used `DriverManager.getConnection()` to establish the connection, providing the correct URL, username, and password for the Oracle database.

The table in oracle –

CID	CNAME	CEMAIL	CPHONENUMBER	OID
2222	Adiba	Adiba@gmail.com	01722222222	1001
2223	Abeaz	Abeaz@gmail.com	01833333333	1002
2224	Sanzida	Sanzida@gmail.com	01944444444	-

The output will be –

```
- RESTAURANT (run) x
run:
cid: 2222 cname: Adibacemail: Adiba@gmail.comcphoneNumber: 1722222222oid: 1001
cid: 2223 cname: Abeazcemail: Abeaz@gmail.comcphoneNumber: 1833333333oid: 1002
cid: 2224 cname: Sanzidacemail: Sanzida@gmail.comcphoneNumber: 1944444444oid: 0
BUILD SUCCESSFUL (total time: 0 seconds)
```

ADIBA HOSSAIN (23-52389-2)

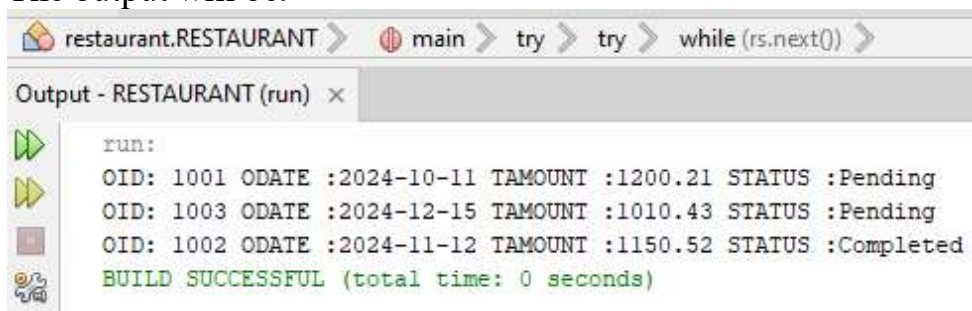
1. I started by downloading and installing Oracle Database from the Oracle website.
2. I also downloaded NetBeans IDE to write the Java code.
3. I searched for the Oracle JDBC driver (ojdbc8.jar) from the official Oracle website and downloaded the correct version for my Oracle database.
4. Then I created tables, inserted the values and opened NetBeans IDE to create a new project which is named as RESTAURANT.

OID	ODATE	TAMOUNT	STATUS
1001	11-OCT-24	1200.21	Pending
1003	15-DEC-24	1010.43	Pending
1002	12-NOV-24	1150.52	Completed

The table OrderC in Oracle

5. I created a new library named OracleJDBC in Tools > Libraries, added the ojdbc14.jar as well as ojdbc14_g.jar file to it and included the library in my project's classpath to ensure the JDBC driver was recognized during execution.
6. I implemented the Java code to connect to the database, retrieve data through a query, and display the results, and when I executed it, the output appeared successfully in the console.

The output will be:



```

restaurant.RESTAURANT > main > try > try > while (rs.next()) >
Output - RESTAURANT (run) x
run:
OID: 1001 ODATE :2024-10-11 TAMOUNT :1200.21 STATUS :Pending
OID: 1003 ODATE :2024-12-15 TAMOUNT :1010.43 STATUS :Pending
OID: 1002 ODATE :2024-11-12 TAMOUNT :1150.52 STATUS :Completed
BUILD SUCCESSFUL (total time: 0 seconds)

```

ABEAZ MOHTASIM (23-54631-3)

A successful database connection occurs when an application establishes a stable communication link with a database using valid credentials (username, password) and connection details. Once connected, the application can execute queries, retrieve data, or make updates as needed. Successful connections typically return a confirmation message or status without errors.

•What do we need?

Steps to Connect to Oracle Database

1. Setup Your Oracle Database

- Ensure Oracle Database is installed and running.
- Note your connection details: Host, Port, Service Name/SID, Username, and Password.

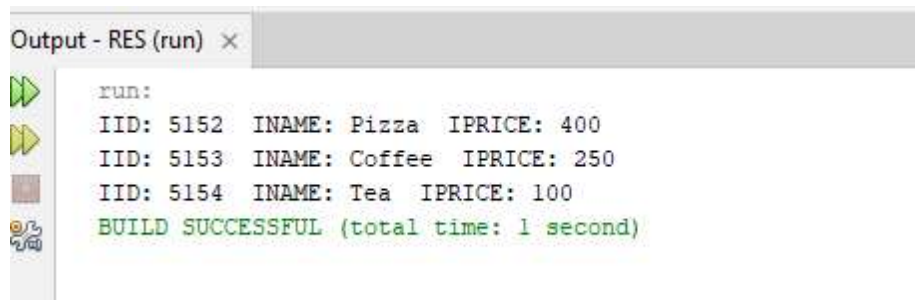
2. Add Oracle JDBC Driver

- Download ojdbc8.jar from Oracle's site.
- In NetBeans, right-click your project > **Properties** > **Libraries** > **Add JAR/Folder** > Add ojdbc8.jar.

3. Use the following java code for connection

4. Run the program

5. If successful we can see “Oracle database connection successful”



```

Output - RES (run) x
run:
IID: 5152  INAME: Pizza  IPRICE: 400
IID: 5153  INAME: Coffee IPRICE: 250
IID: 5154  INAME: Tea   IPRICE: 100
BUILD SUCCESSFUL (total time: 1 second)
  
```

IID	INAME	IPRICE
5152	Pizza	400
5153	Coffee	250
5154	Tea	100

The table in oracle (Item)

That's it. Use this connection to execute queries as needed.

Conclusion

The **Restaurant Management System (RMS)** project successfully demonstrates the application of modern database design principles to streamline restaurant operations. By leveraging Oracle's relational database capabilities, the system manages various facets of restaurant operations such as customer orders, employee assignments, inventory, and payments. The comprehensive **ER diagram**, data normalization, and **SQL queries** (including single-row and multiple-row functions, joins, and views) ensure that the system is both efficient and user-friendly, addressing the key needs of restaurant owners, staff, and customers.

Throughout the project, we effectively created and populated tables representing the relationships between customers, orders, employees, menu items, and payments. By employing normalized database structures, we ensured data consistency and minimized redundancy. The practical implementation of **DML (Data Manipulation Language)** and **DDL (Data Definition Language)** operations using SQL facilitated seamless interactions with the database. Furthermore, **views** and **subqueries** were applied to simplify complex data retrieval tasks, providing enhanced reporting and decision-making capabilities.

In terms of **future aspects**, the system could be expanded to incorporate real-time order tracking, integration with point-of-sale (POS) systems, and advanced analytics for inventory management and customer preferences. Additionally, the platform could be enhanced with **mobile app support** to allow customers to place orders, make payments, and track order status from their devices. Integrating **AI** to predict demand and suggest personalized menu items to customers could also further enhance the system's utility.

In summary, the project not only demonstrates the power of relational databases in business management but also lays the foundation for future enhancements that could make restaurant operations more agile, accurate, and customer-centric. The effective use of Oracle, SQL, and Java integration ensures the RMS is both robust and scalable, positioning it for real-world application and future upgrades.