QUERIES PRACTICE

NAME: Abdullah Mohsin

ALL QUERIES CONTENT:

TYPES OF DATA LANGUAGES:

There are three types of data models:
1.Data Definition Language
2.Data Manipulation Language
3.Data Control Language
4.Transactional Control Language
EXAMPLES: CONSIDER A ER-DIAGRAM with MART, FRUIT and BASKET:

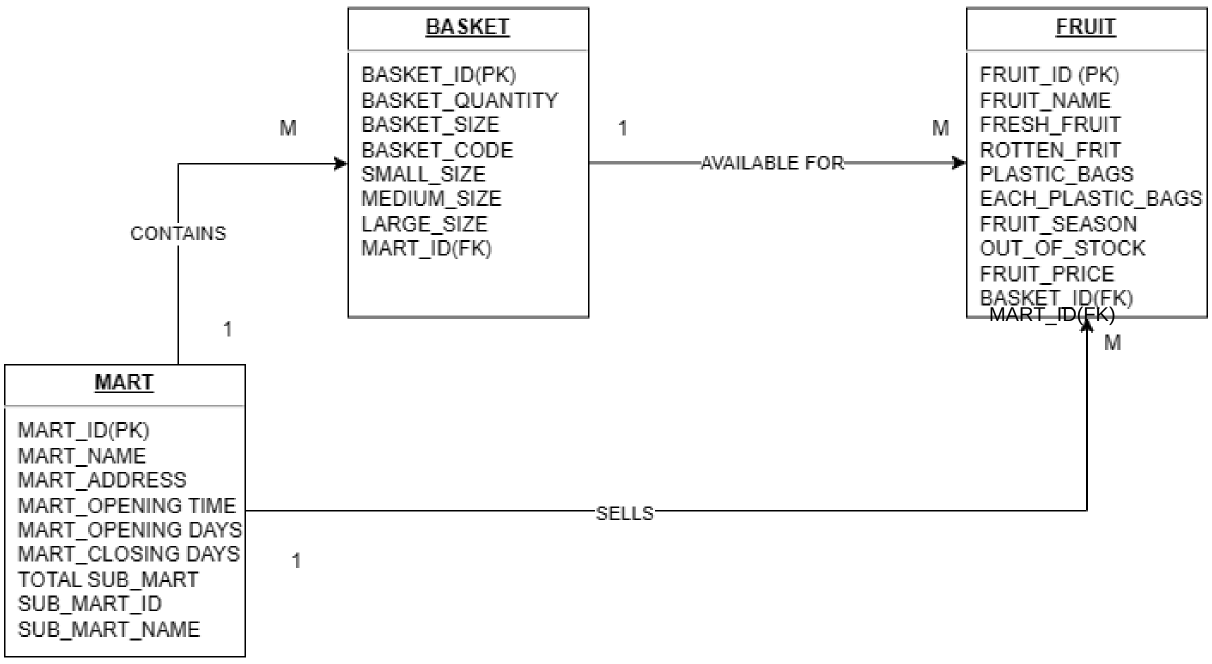


Figure:1.1(ENTITY RELATIONSHIP DIAGRAM)

RELATIONAL TABLES:

1. MART TABLE:

MART-ID(PK)	MART_NAME	MART_ADDRESS	MART_OPENING_TIME	MART_OPENING DAYS	MART_CLOSING DAYS	TOTAL_SUBMART	SUB_MART_ID	SUB_MART_NAME

2. BASKET TABLE:

BASKET-ID(PK)	BASKET_QUANTITY	BASKET_SIZE	BASKET_CODE	SMALL_SIZE	MEDIUM_SIZE	LARGE_SIZE	MART_ID(FK)

3. FRUIT TABLE:

FRUIT-ID(PK)	FRUIT_NAME	FRESH_FRUIT	ROTTEN_FRUIT	PLASTIC_BAGS	EACH_PLASTIC_BAGS	FRUIT_SEASON	OUT_OF_STOCK	FRUIT_PRICE	BASKET_ID (FK)

- # TUTORIAL:

We are familiar with database, database is the method of storing and managing data in hardware. So, we must need to use a software, this software is known as database management system. Here in this document we are studying SQL, it is relational database management system.
There are three model in order to create a database system:
1.Conceptual Model
2.Representational Model
3.Physical model
You can see in figure 1.1, ER-Diagram has been shown and from that diagram we have created relation schema. We will use above information in order to implement it physically. You can use SQL by downloading it or running queries online.
CLICK LINK: https://livesql.oracle.com/ords/f?p=590:1000
In order to download or use oracle live without any hurdle login in SQL and enjoy writing query, if any doubt contact me :03363736231

Now come to Database languages:

1. **DDL (DATA DEFINITION LANGUAGE):**

COMMANDS:

1. CREATE:

```
--DATABASE LANGUAGE COMMANDS
--1.DDL(DATA DEFINITION LANGUAGE)
--ENTITIES MART,BASKET,FRUIT
CREATE TABLE MART(
MART_ID INT  PRIMARY KEY,
MART_NAME VARCHAR(10),
MART_ADDRESS VARCHAR(30),
MART_OPENING_TIME TIMESTAMP,
MART_OPENING_DAYS VARCHAR(10),
MART_CLOSING_DAYS VARCHAR(10),
SUB_MART_ID INT UNIQUE,
SUB_MART_NAME VARCHAR(5)
);
CREATE TABLE BASKET(
    BASKET_ID INT PRIMARY KEY,
    BASKET_QUANTITY NUMBER(23,2),
    BASKET_SIZE INT,
    SMALL_SIZE INT,
    MEDIUM_SIZE INT,
    LARGE_SIZE INT,
    MART_ID INT,
    CONSTRAINT FK_MART FOREIGN KEY(MART_ID)REFERENCES MART(MART_ID),
    CONSTRAINT UNIQUE_BASKET_SIZE UNIQUE(BASKET_SIZE, SMALL_SIZE, MEDIUM_SIZE, LARGE_SIZE)
);
CREATE TABLE FRUIT(
    FRUIT_ID INT PRIMARY KEY,
    FRUIT_NAME VARCHAR(10),
    FRESH_FRUIT INT,
    ROTTEN_FRUIT INT,
    PLASTIC_BAGS NUMBER(10),
    EACH_PLASTIC_BAGS VARCHAR(1),
    FRUIT_SEASON VARCHAR(5),
    OUT_OF_STOCK NUMBER(20),
    FRUIT_PRICE NUMBER(5,2),
    BASKET_ID INT,
    MART_ID INT,
    CONSTRAINT FK_BASKETS FOREIGN KEY(BASKET_ID)REFERENCES BASKET(BASKET_ID),
    CONSTRAINT FK_MARTS FOREIGN KEY(MART_ID)REFERENCES MART(MART_ID)

);
```

2. ALTER:

```
-2.ALTER
ALTER TABLE MART ADD MANAGER_NAME VARCHAR(5);--ADD COLUMN
ALTER TABLE MART MODIFY MART_NAME VARCHAR(20);--MODIFY COLUMN
--CHANGEMENT IN DATATYPE
--RENAME TABLE
ALTER TABLE MART RENAME COLUMN MART_ID TO MART_NO;
--DROP COLUMN
ALTER TABLE MART DROP COLUMN MART_NAME;
```

11.DROP:

```
DROP TABLE BASKET;
```

13.TRUNCATE:

```
TRUNCATE TABLE BASKET;
```

15.RENAME:

15.ALTER TABLE BASKET RENAME TO SHOPPING_BASKET;

16.COMMENT:

16.COMMENT ON TABLE BASKET IS 'This table contains information about different baskets used in the store.';

17.USE:

17.USE my_database;

18.PARTITION:

```
CREATE TABLE MART (
MART_ID INT PRIMARY KEY,
MART_NAME VARCHAR(10),
MART_ADDRESS VARCHAR(30),
MART_OPENING_TIME TIMESTAMP,
MART_OPENING_DAYS VARCHAR(10),
MART_CLOSING_DAYS VARCHAR(10),
SUB_MART_ID INT UNIQUE,
SUB_MART_NAME VARCHAR(5)
)
PARTITION BY LIST (MART_OPENING_DAYS) (
PARTITION p_mon VALUES ('Monday'),
PARTITION p_tue VALUES ('Tuesday'),
PARTITION p_wed VALUES ('Wednesday'),
PARTITION p_thu VALUES ('Thursday'),
PARTITION p_fri VALUES ('Friday'),
PARTITION p_sat VALUES ('Saturday'),
PARTITION p_sun VALUES ('Sunday')
);
```

2. DML (Data Manipulation Language):

COMMANDS:

1. SELECT:

```
SELECT * FROM MART;
```

2. INSERT:

```
INSERT INTO MART (MART_ID, MART_NAME, MART_ADDRESS, MART_OPENING_TIME, MART_OPENING_DAYS, MART_CLOSING_DAYS, SUB_MART_ID, SUB_MART_NAME)
VALUES (2, 'FruitMart', '456 Elm St', TO_TIMESTAMP('2024-01-01 09:00:00', 'YYYY-MM-DD HH24:MI:SS'), 'Monday,Thursday', 'Friday,Saturday', 102, 'SubMart2');
```

3. UPDATE:

```
UPDATE BASKET
SET BASKET_QUANTITY = 150
WHERE BASKET_ID = 1;
```

4. DELETE:

```
DELETE FROM FRUIT

WHERE FRUIT_ID = 1;
```

5. MERGE:

```
MERGE INTO FRUIT f
USING (SELECT 2 AS FRUIT_ID, 'Grapes' AS FRUIT_NAME FROM DUAL) new_fruit
ON (f.FRUIT_ID = new_fruit.FRUIT_ID)
WHEN MATCHED THEN
  UPDATE SET f.FRUIT_NAME = new_fruit.FRUIT_NAME
WHEN NOT MATCHED THEN
  INSERT (FRUIT_ID, FRUIT_NAME) VALUES (new_fruit.FRUIT_ID, new_fruit.FRUIT_NAME);
```

6. CALL:

```
CALL UpdateFruitStock(1, 100);  -- Calls a stored procedure to update stock
```

7. EXPLAIN PLAN:

```
EXPLAIN PLAN FOR

SELECT * FROM MART WHERE MART_ID = 1;
```

8. LOCK TABLE:

```
LOCK TABLE MART IN EXCLUSIVE MODE;
```

- **DQL(DATA QUERY LANGUAGE):**

Here are the DQL (Data Query Language) commands and clauses:

COMMANDS:

1. SELECT:

```
SELECT MART_NAME, MART_ADDRESS FROM MART;
```

2. FROM:

```
SELECT * FROM BASKET;
```

3. WHERE:

```
SELECT FRUIT_NAME FROM FRUIT
WHERE FRUIT_PRICE > 1.00;
```

4. GROUP BY:

```
SELECT MART_ID, COUNT(*) AS FRUIT_COUNT
FROM FRUIT
GROUP BY MART_ID;
```

5. HAVING:

```
SELECT MART_ID, COUNT(*) AS FRUIT_COUNT
FROM FRUIT
GROUP BY MART_ID
HAVING COUNT(*) > 10;
```

6. ORDER BY:

```
SELECT FRUIT_NAME, FRUIT_PRICE
FROM FRUIT
ORDER BY FRUIT_PRICE DESC;
```

7. JOIN:

```
SELECT FRUIT.FRUIT_NAME, BASKET.BASKET_SIZE
FROM FRUIT
INNER JOIN BASKET ON FRUIT.BASKET_ID = BASKET.BASKET_ID;
```

i.INNER JOIN:

```
SELECT FRUIT.FRUIT_NAME, BASKET.BASKET_SIZE
FROM FRUIT
INNER JOIN BASKET ON FRUIT.BASKET_ID = BASKET.BASKET_ID;
```

ii.LEFT JOIN:

```
SELECT FRUIT.FRUIT_NAME, BASKET.BASKET_SIZE
FROM FRUIT
LEFT JOIN BASKET ON FRUIT.BASKET_ID = BASKET.BASKET_ID;
```

iii.RIGHT JOIN:

```
SELECT FRUIT.FRUIT_NAME, BASKET.BASKET_SIZE
FROM FRUIT
RIGHT JOIN BASKET ON FRUIT.BASKET_ID = BASKET.BASKET_ID;
```

iv. FULL JOIN:

```
SELECT FRUIT.FRUIT_NAME, BASKET.BASKET_SIZE
FROM FRUIT
FULL JOIN BASKET ON FRUIT.BASKET_ID = BASKET.BASKET_ID;
```

v.CROSS JOIN:

```
SELECT FRUIT.FRUIT_NAME, BASKET.BASKET_SIZE
FROM FRUIT
CROSS JOIN BASKET;
```

Vi .INNER JOIN:

```
SELECT A.FRUIT_NAME AS FRUIT1, B.FRUIT_NAME AS FRUIT2
FROM FRUIT A
INNER JOIN FRUIT B ON A.BASKET_ID = B.BASKET_ID
WHERE A.FRUIT_ID <> B.FRUIT_ID;
```

Vii.NATURAL JOIN:

```
SELECT FRUIT_NAME, BASKET_SIZE
FROM FRUIT
NATURAL JOIN BASKET;
```

Viii.ANTI JOIN:

```
SELECT FRUIT_NAME
FROM FRUIT
WHERE BASKET_ID NOT IN (SELECT BASKET_ID FROM BASKET);
```

ix.SEMI JOIN:

```
SELECT FRUIT_NAME
FROM FRUIT
WHERE EXISTS (SELECT 1 FROM BASKET WHERE FRUIT.BASKET_ID = BASKET.BASKET_ID);
```

8. DISTINCT:

```
SELECT DISTINCT FRUIT_NAME
FROM FRUIT;
```

9. LIMIT:

```
SELECT * FROM FRUIT
LIMIT 5;
```

10.OFFSET:

```
SELECT * FROM FRUIT
LIMIT 5 OFFSET 10;
```

11.TOP:

```
SELECT TOP 5 * FROM FRUIT;
```

3. DCL (Data Control Language):

COMMANDS:

1. GRANT:

```
GRANT SELECT, INSERT ON MART TO user_name;
```

2. REVOKE:

```
REVOKE SELECT, INSERT ON MART FROM user_name;
```

4. TCL (Transactional Control Language):

COMMANDS:

1. COMMIT:

```
UPDATE BASKET
SET BASKET_QUANTITY = 200
WHERE BASKET_ID = 1;
COMMIT;
```

2. ROLLBACK:

```
UPDATE BASKET
SET BASKET_QUANTITY = 200
WHERE BASKET_ID = 1;
ROLLBACK;
```

3. SAVEPOINT:

```
SAVEPOINT sp1;
UPDATE BASKET
SET BASKET_QUANTITY = 200
WHERE BASKET_ID = 1;
SAVEPOINT sp2;
UPDATE BASKET
SET BASKET_QUANTITY = 300
WHERE BASKET_ID = 2;

ROLLBACK TO sp1;
```

4. SET TRANSACTION:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
UPDATE MART
SET MART_ADDRESS = '789 Oak St'
WHERE MART_ID = 1;
```

• OBJECTS IN SQL:

These are objects of SQL:

1. Tables:

```
CREATE TABLE Employees (
   Employee ID INT PRIMARY KEY,
   Name VARCHAR (50) NOT NULL,
   Department VARCHAR (50),
   Salary DECIMAL (10, 2)
);
```

2. Views:

```
CREATE VIEW HighSalaryEmployees AS
SELECT Name, Salary
FROM Employees
WHERE Salary > 50000;
```

3. Indexes:

```
CREATE INDEX idx_salary ON Employees(Salary);
```

4. Schemas:

```
CREATE SCHEMA Sales AUTHORIZATION sales_manager;
```

5. Sequences:

```
CREATE SEQUENCE emp_seq
START WITH 1
INCREMENT BY 1;
```

6. Triggers:

```
CREATE TRIGGER trg_salary_check
BEFORE INSERT ON Employees
FOR EACH ROW
WHEN (NEW.Salary < 0)
BEGIN
   RAISE_APPLICATION_ERROR(-20001, 'Salary cannot be negative.');
END;
```

7. Stored Procedures:

```
CREATE PROCEDURE UpdateSalary (emp_id INT, new_salary DECIMAL)
AS
BEGIN
   UPDATE Employees
   SET Salary = new_salary
   WHERE Employee_ID = emp_id;
```

```
END;
```

8. Functions:

```
CREATE FUNCTION CalculateBonus (salary DECIMAL)
RETURNS DECIMAL
AS
BEGIN
   RETURN salary * 0.1;
END;
```

9. Constraints (e.g., PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK, NOT NULL):

```
CREATE TABLE Departments (
   Dept_ID INT PRIMARY KEY,
   Dept_Name VARCHAR(50) NOT NULL,
   Manager_ID INT UNIQUE,
   CONSTRAINT chk_name CHECK (Dept_Name != '')
);
```

10. Synonyms:

```
CREATE SYNONYM EmpView FOR HighSalaryEmployees;
```

11. Users and Roles (for database security management):

```
CREATE USER john IDENTIFIED BY password123;
GRANT CONNECT, CREATE TABLE TO john;
CREATE ROLE SalesRole;
GRANT SELECT, INSERT ON Employees TO SalesRole;
GRANT SalesRole TO john;
```

• PL/SQL COMMANDS & OBJECTS IN SQL :

These are objects of PL/SQL:

1. Anonymous Blocks:

```
BEGIN
   DBMS_OUTPUT.PUT_LINE('Hello, PL/SQL!');
END;
```

2. Stored Procedures:

```
CREATE PROCEDURE UpdateEmployeeSalary (emp_id INT, increment DECIMAL)
AS
BEGIN
   UPDATE Employees
   SET Salary = Salary + increment
   WHERE Employee_ID = emp_id;
END;
```

3. Functions:

```
CREATE FUNCTION GetTotalSalary
RETURN DECIMAL
AS
   total_salary DECIMAL;
BEGIN
   SELECT SUM(Salary) INTO total_salary FROM Employees;
   RETURN total_salary;
END;
```

4. Packages:

```
CREATE PACKAGE EmployeePackage AS
   PROCEDURE UpdateEmployeeSalary(emp_id INT, increment DECIMAL);
   FUNCTION GetEmployeeCount RETURN INT;
END EmployeePackage;

CREATE PACKAGE BODY EmployeePackage AS
   PROCEDURE UpdateEmployeeSalary(emp_id INT, increment DECIMAL) AS
   BEGIN
      UPDATE Employees
      SET Salary = Salary + increment
      WHERE Employee_ID = emp_id;
   END;

   FUNCTION GetEmployeeCount RETURN INT AS
      emp_count INT;
   BEGIN
      SELECT COUNT(*) INTO emp_count FROM Employees;
      RETURN emp_count;
   END;
END EmployeePackage;
```

5. Triggers:

```
CREATE TRIGGER trg_before_insert
BEFORE INSERT ON Employees
FOR EACH ROW
BEGIN
   IF :NEW.Salary < 0 THEN
```

```
      RAISE_APPLICATION_ERROR(-20001, 'Salary cannot be negative.');
   END IF;
END;
```

6. Views:

```
CREATE VIEW ActiveEmployees AS
SELECT Name, Department, Salary
FROM Employees
WHERE Salary > 0;
```

7. Cursors (Implicit and Explicit):

```
BEGIN
   SELECT Salary INTO total_salary FROM Employees WHERE Employee_ID = 1;
   DBMS_OUTPUT.PUT_LINE('Total Salary: ' || total_salary);
END;
DECLARE
   CURSOR emp_cursor IS SELECT Name, Salary FROM Employees;
   emp_name Employees.Name%TYPE;
   emp_salary Employees.Salary%TYPE;
BEGIN
   OPEN emp_cursor;
   LOOP
      FETCH emp_cursor INTO emp_name, emp_salary;
      EXIT WHEN emp_cursor%NOTFOUND;
      DBMS_OUTPUT.PUT_LINE(emp_name || ': ' || emp_salary);
   END LOOP;
   CLOSE emp_cursor;
END;
```

8. Records:

```
DECLARE
   emp_rec Employees%ROWTYPE;
BEGIN
   SELECT * INTO emp_rec FROM Employees WHERE Employee_ID = 1;
   DBMS_OUTPUT.PUT_LINE('Name: ' || emp_rec.Name || ', Salary: ' || emp_rec.Salary);
END;
```

9. Exceptions:

```
BEGIN
   UPDATE Employees SET Salary = -1 WHERE Employee_ID = 1;
EXCEPTION
   WHEN OTHERS THEN
      DBMS_OUTPUT.PUT_LINE('An error occurred: ' || SQLERRM);
END;
```

10.Variables:

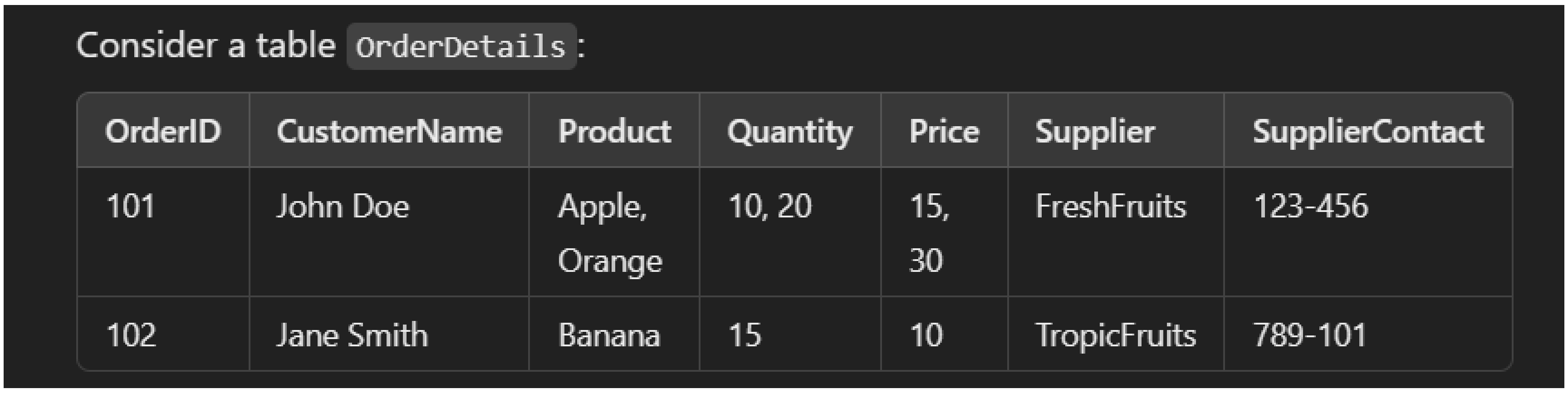
```
DECLARE
   emp_name VARCHAR2(50);
   emp_salary DECIMAL(10, 2);
BEGIN
   emp_name := 'John Doe';
   emp_salary := 50000;
   DBMS_OUTPUT.PUT_LINE(emp_name || ': ' || emp_salary);
END;
```

11.Collections (Associative Arrays, Nested Tables, Varrays):

```
DECLARE

   TYPE salary_array IS TABLE OF DECIMAL INDEX BY PLS_INTEGER;

   salaries salary_array;

BEGIN

   salaries(1) := 50000;

   salaries(2) := 60000;

   DBMS_OUTPUT.PUT_LINE('First Salary: ' || salaries(1));

END;

--NESTED TABLE

DECLARE

   TYPE salary_table IS TABLE OF DECIMAL;

   salaries salary_table := salary_table(50000, 60000, 70000);

BEGIN

   FOR i IN 1..salaries.COUNT LOOP

      DBMS_OUTPUT.PUT_LINE('Salary ' || i || ': ' || salaries(i));

   END LOOP;

END;

DECLARE

   TYPE salary_varray IS VARRAY(3) OF DECIMAL;
```

```
  salaries salary_varray := salary_varray(50000, 60000, 70000);
BEGIN
  FOR i IN 1..salaries.COUNT LOOP
    DBMS_OUTPUT.PUT_LINE('Salary ' || i || ': ' || salaries(i));
  END LOOP;
END;
```

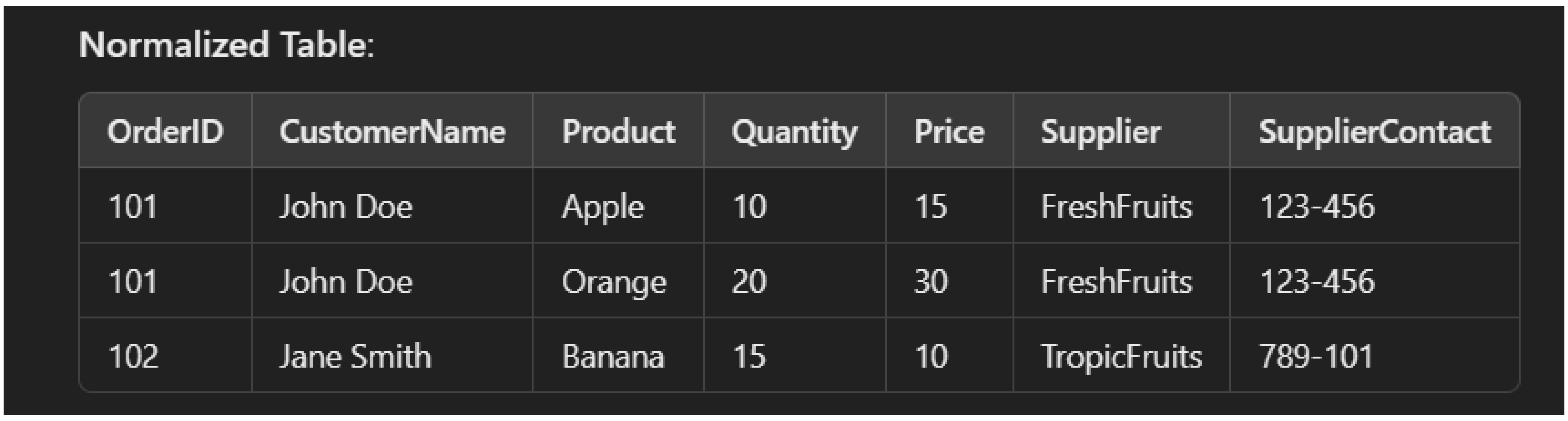
NORMALIZATION:



Consider a table `OrderDetails`:

OrderID	CustomerName	Product	Quantity	Price	Supplier	SupplierContact
101	John Doe	Apple, Orange	10, 20	15, 30	FreshFruits	123-456
102	Jane Smith	Banana	15	10	TropicFruits	789-101

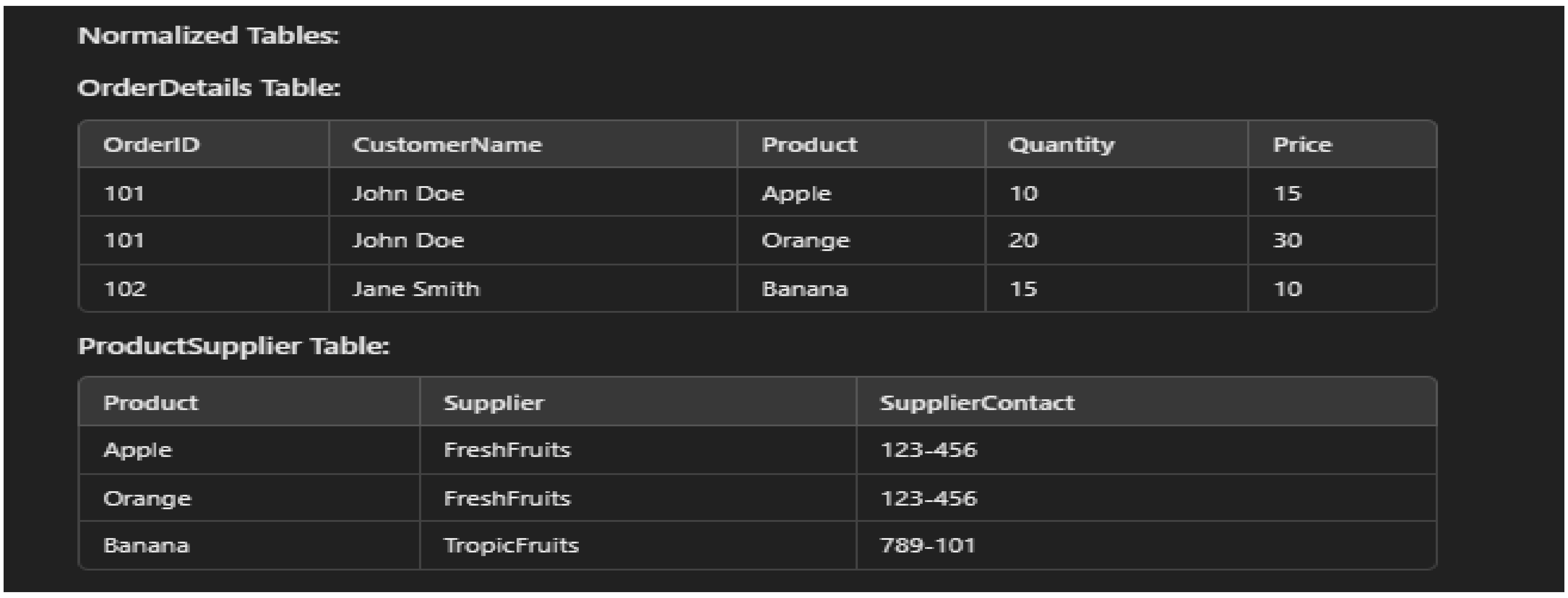
1NF:



Normalized Table:

OrderID	CustomerName	Product	Quantity	Price	Supplier	SupplierContact
101	John Doe	Apple	10	15	FreshFruits	123-456
101	John Doe	Orange	20	30	FreshFruits	123-456
102	Jane Smith	Banana	15	10	TropicFruits	789-101

2NF



Normalized Tables:

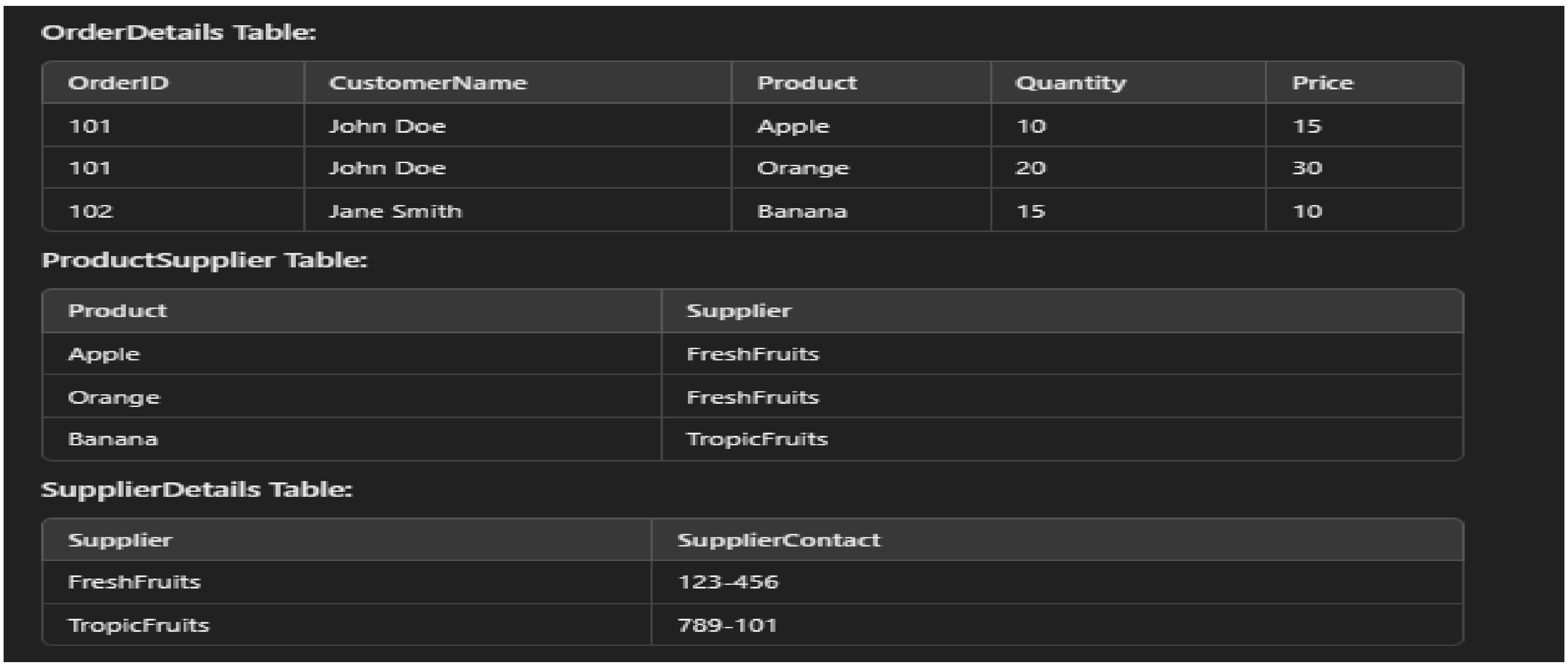
OrderDetails Table:

OrderID	CustomerName	Product	Quantity	Price
101	John Doe	Apple	10	15
101	John Doe	Orange	20	30
102	Jane Smith	Banana	15	10

ProductSupplier Table:

Product	Supplier	SupplierContact
Apple	FreshFruits	123-456
Orange	FreshFruits	123-456
Banana	TropicFruits	789-101

3NF



OrderDetails Table:

OrderID	CustomerName	Product	Quantity	Price
101	John Doe	Apple	10	15
101	John Doe	Orange	20	30
102	Jane Smith	Banana	15	10

ProductSupplier Table:

Product	Supplier
Apple	FreshFruits
Orange	FreshFruits
Banana	TropicFruits

SupplierDetails Table:

Supplier	SupplierContact
FreshFruits	123-456
TropicFruits	789-101

ANS:

1NF:

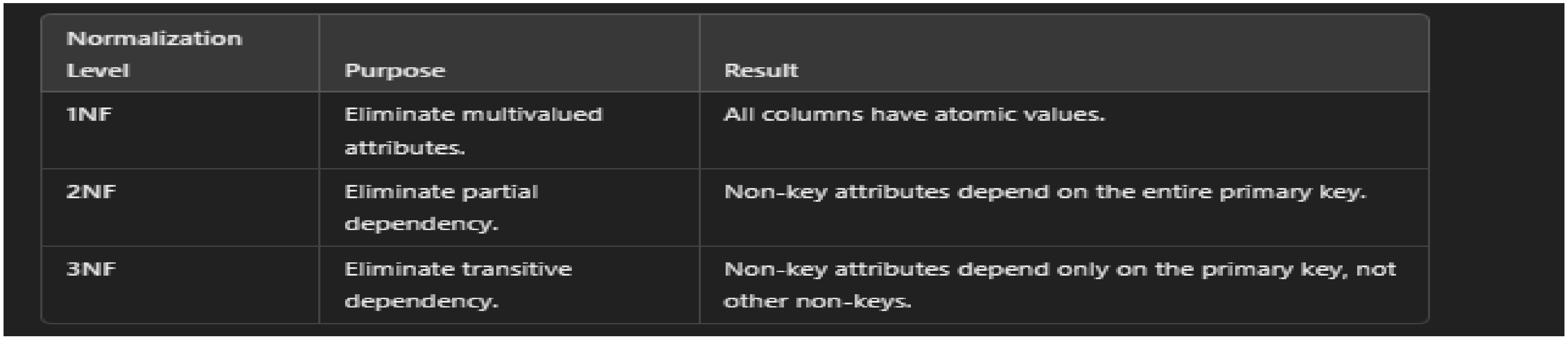
```
CREATE TABLE OrderDetails_1NF (
    OrderID INT,
    CustomerName VARCHAR(50),
    Product VARCHAR(50),
    Quantity INT,
    Price DECIMAL(10, 2),
    Supplier VARCHAR(50),
    SupplierContact VARCHAR(20)
);

INSERT INTO OrderDetails_1NF
VALUES
(101, 'John Doe', 'Apple', 10, 15, 'FreshFruits', '123-456'),
(101, 'John Doe', 'Orange', 20, 30, 'FreshFruits', '123-456'),
(102, 'Jane Smith', 'Banana', 15, 10, 'TropicFruits', '789-101');
```

2NF:

```
CREATE TABLE OrderDetails_2NF (
    OrderID INT,
    CustomerName VARCHAR(50),
    Product VARCHAR(50),
    Quantity INT,
    Price DECIMAL(10, 2),
    PRIMARY KEY (OrderID, Product)
);

CREATE TABLE ProductSupplier (
    Product VARCHAR(50) PRIMARY KEY,
    Supplier VARCHAR(50),
    SupplierContact VARCHAR(20)
);

INSERT INTO OrderDetails_2NF
VALUES
(101, 'John Doe', 'Apple', 10, 15),
(101, 'John Doe', 'Orange', 20, 30),
(102, 'Jane Smith', 'Banana', 15, 10);

INSERT INTO ProductSupplier
VALUES
('Apple', 'FreshFruits', '123-456'),
('Orange', 'FreshFruits', '123-456'),
('Banana', 'TropicFruits', '789-101');
```

3NF

```
CREATE TABLE SupplierDetails (
    Supplier VARCHAR(50) PRIMARY KEY,
    SupplierContact VARCHAR(20)
);

INSERT INTO SupplierDetails
VALUES
('FreshFruits', '123-456'),
('TropicFruits', '789-101');

ALTER TABLE ProductSupplier
DROP COLUMN SupplierContact;

INSERT INTO ProductSupplier
VALUES
('Apple', 'FreshFruits'),
('Orange', 'FreshFruits'),
('Banana', 'TropicFruits');
```



Normalization Level	Purpose	Result
1NF	Eliminate multivalued attributes.	All columns have atomic values.
2NF	Eliminate partial dependency.	Non-key attributes depend on the entire primary key.
3NF	Eliminate transitive dependency.	Non-key attributes depend only on the primary key, not other non-keys.