**## Basic INNER JOIN (1–20)**

1. Get the names of all employees and their department names.

2. List all orders along with the names of the customers who placed them.

3. Show the product name and category name for all products.

4. List students and the courses they are enrolled in.

5. Retrieve book titles and author names.

6. Show all invoices along with the customer’s name and country.

7. Display all projects and their assigned manager names.

8. List all patients and their attending doctors.

9. Retrieve order details along with the corresponding product names.

10. Get all employees with their job titles.

11. Show all sales along with salesperson names.

12. Get all transactions and the bank branch they occurred in.

13. List all vehicles and their owners.

14. Show all classes with their teachers.

15. Get all movie names and their directors.

16. List all athletes with their respective teams.

17. Show products and their suppliers.

18. List blog posts with their respective authors.

19. Show employees and the location of their offices.

20. Display all items in a shopping cart with product descriptions.

---

**## LEFT JOIN (21–40)**

21. List all customers, even those who haven’t placed orders.

22. Get all products and their order quantities (even if never ordered).

23. Show all students and the grades they've received (if any).

24. List all employees and their assigned projects (if any).

25. Show all departments and their employees (if any).

26. Display all articles and any comments (if any).

27. List all books, even those without an author (null author\\_id).

28. Show all courses, including those with no enrollments.

29. Get all users and their last login time (if logged in).

30. List all patients, even if no appointment is scheduled.

31. Show all teachers, even if they are not teaching any class.

32. List all cars, even those not assigned to any customer.

33. Get all songs and their album names (even if not in any album).

34. Show all houses and the tenants living there (if any).

35. List all posts and the number of likes (even if zero).

36. Get all employees and their emergency contact details (if any).

37. Display all branches and their ATM machines (if any).

38. List all schools and their principals (even if no principal assigned).

39. Get all flights and the corresponding pilots (if any).

40. Show all resumes and job applications linked to them (if any).

---

**## RIGHT JOIN (41–55)**

41. Get all orders, including those without customer info.

42. Show all departments and employees assigned (even if none).

43. List all job postings and the applicants (even if no one applied).

44. Show all songs and artists (even if song is missing artist info).

45. Display all bank branches and employees (even if branch has no staff).

46. Get all flights and airports they depart from (even if flight data is missing).

47. List all stores and the inventory they hold (even if store is empty).

48. Show all sports teams and their players (even if team has no players).

49. List all invoices and the payments made (even if unpaid).

50. Show all orders and shipping details (even if not shipped).

51. Get all rental properties and tenants (even if vacant).

52. List all service centers and the vehicles serviced (even if none).

53. Show all products and the vendors supplying them (even if unsupplied).

54. Display all training sessions and attendees (even if session is empty).

55. Get all users and their transactions (even if they made none).

---

**## FULL OUTER JOIN (56–70)**

56. List all employees and their managers (even if no relation exists).

57. Get all books and authors (even if unmatched).

58. Display all customers and orders (even if either is missing).

59. Show all students and classes (even if not enrolled).

60. Get all products and their reviews (even if not reviewed).

61. List all applications and applicants (even if unlinked).

62. Show all users and their profile pictures (even if missing).

63. List all volunteers and events (even if not linked).

64. Display all assets and owners (even if unassigned).

65. Show all job titles and employees (even if no one holds the title).

66. Get all cities and their airports (even if no airport).

67. List all software and licenses (even if unused).

68. Show all classes and schedules (even if one is missing).

69. List all banks and customers (even if no link).

70. Display all employees and badges (even if missing).

---

**## SELF JOIN (71–80)**

71. Find all employees who report to someone (supervisor).

72. List employees and their managers' names.

73. Find all products that are substitutes of other products.

74. List students and their mentors (who are also students).

75. Find users who referred other users.

76. Display a list of employees with the same job title.

77. Find customers who live in the same city.

78. List products with the same price.

79. Find employees working in the same department.

80. Display employees with the same hire date.

---

**## Advanced Join Practice (81–100)**

81. Get the top 3 customers with most orders using JOIN.

82. List each department with the number of employees.

83. Show total sales by product category.

84. Find the most expensive product in each category.

85. Get all orders with product names and total price.

86. Display each student’s average grade.

87. List all authors and the number of books they've written.

88. Get customers who ordered more than 5 times.

89. List customers who never placed an order.

90. Show products not ordered in the last 6 months.

91. Find employees with the same manager.

92. List pairs of students enrolled in the same course.

93. Display products with no matching supplier or category.

94. Get the number of songs in each album.

95. Find stores with below-average number of products.

96. Show projects with no tasks assigned.

97. Get number of reviews per product.

98. List teachers who teach in more than one department.

99. Show the latest order for each customer.

100. Get all items with their category, vendor, and inventory status.

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

CustomerName VARCHAR(100),

ContactEmail VARCHAR(100),

City VARCHAR(50),

Country VARCHAR(50)

);

CREATE TABLE Orders (

OrderID INT PRIMARY KEY,

OrderDate DATE,

CustomerID INT,

TotalAmount DECIMAL(10, 2),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

CREATE TABLE Products (

ProductID INT PRIMARY KEY,

ProductName VARCHAR(100),

CategoryID INT,

SupplierID INT,

Price DECIMAL(10, 2)

);

CREATE TABLE OrderDetails (

OrderDetailID INT PRIMARY KEY,

OrderID INT,

ProductID INT,

Quantity INT,

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

CREATE TABLE Categories (

CategoryID INT PRIMARY KEY,

CategoryName VARCHAR(100)

);

CREATE TABLE Suppliers (

SupplierID INT PRIMARY KEY,

SupplierName VARCHAR(100),

ContactEmail VARCHAR(100)

);

CREATE TABLE Employees (

EmployeeID INT PRIMARY KEY,

EmployeeName VARCHAR(100),

ManagerID INT,

DepartmentID INT,

HireDate DATE,

JobTitle VARCHAR(100),

FOREIGN KEY (ManagerID) REFERENCES Employees(EmployeeID)

);

CREATE TABLE Departments (

DepartmentID INT PRIMARY KEY,

DepartmentName VARCHAR(100)

);

CREATE TABLE Students (

StudentID INT PRIMARY KEY,

StudentName VARCHAR(100),

MentorID INT,

FOREIGN KEY (MentorID) REFERENCES Students(StudentID)

);

CREATE TABLE Courses (

CourseID INT PRIMARY KEY,

CourseName VARCHAR(100)

);

CREATE TABLE Enrollments (

EnrollmentID INT PRIMARY KEY,

StudentID INT,

CourseID INT,

Grade CHAR(2),

FOREIGN KEY (StudentID) REFERENCES Students(StudentID),

FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)

);

CREATE TABLE Authors (

AuthorID INT PRIMARY KEY,

AuthorName VARCHAR(100)

);

CREATE TABLE Books (

BookID INT PRIMARY KEY,

Title VARCHAR(200),

AuthorID INT,

FOREIGN KEY (AuthorID) REFERENCES Authors(AuthorID)

);

CREATE TABLE Projects (

ProjectID INT PRIMARY KEY,

ProjectName VARCHAR(100),

ManagerID INT,

FOREIGN KEY (ManagerID) REFERENCES Employees(EmployeeID)

);

CREATE TABLE ProjectAssignments (

AssignmentID INT PRIMARY KEY,

ProjectID INT,

EmployeeID INT,

FOREIGN KEY (ProjectID) REFERENCES Projects(ProjectID),

FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)

);

-- Departments

INSERT INTO Departments VALUES (1, 'IT'), (2, 'HR');

-- Employees (with NULL for ManagerID initially to avoid circular FK)

INSERT INTO Employees VALUES

(1, 'Alice Johnson', NULL, 1, '2015-03-15', 'CTO'),

(2, 'Bob Smith', 1, 1, '2017-06-22', 'Developer'),

(3, 'Carol White', 1, 2, '2018-08-10', 'HR Manager');

-- Students

INSERT INTO Students VALUES

(1, 'John Doe', NULL),

(2, 'Jane Smith', 1),

(3, 'Mike Brown', 1);

-- Courses

INSERT INTO Courses VALUES

(1, 'Mathematics'),

(2, 'Physics');

-- Enrollments

INSERT INTO Enrollments VALUES

(1, 2, 1, 'A'),

(2, 3, 2, 'B');

-- Authors

INSERT INTO Authors VALUES

(1, 'J.K. Rowling'),

(2, 'George Orwell');

-- Books

INSERT INTO Books VALUES

(1, 'Harry Potter', 1),

(2, '1984', 2);

-- Categories

INSERT INTO Categories VALUES

(1, 'Electronics'),

(2, 'Books');

-- Suppliers

INSERT INTO Suppliers VALUES

(1, 'Tech Supplies Co.', 'tech@supplies.com'),

(2, 'BookMart', 'books@mart.com');

-- Products

INSERT INTO Products VALUES

(1, 'Laptop', 1, 1, 1200.00),

(2, 'Smartphone', 1, 1, 800.00),

(3, 'Book - 1984', 2, 2, 15.00);

-- Customers

INSERT INTO Customers VALUES

(1, 'Alpha Corp', 'contact@alpha.com', 'New York', 'USA'),

(2, 'Beta LLC', 'info@beta.com', 'London', 'UK');

-- Orders

INSERT INTO Orders VALUES

(1, '2024-12-01', 1, 1215.00),

(2, '2024-12-05', 2, 815.00);

-- OrderDetails

INSERT INTO OrderDetails VALUES

(1, 1, 1, 1), -- 1 Laptop

(2, 1, 3, 1), -- 1 Book

(3, 2, 2, 1); -- 1 Smartphone

-- Projects

INSERT INTO Projects VALUES

(1, 'AI Development', 1),

(2, 'Website Redesign', 2);

-- ProjectAssignments

INSERT INTO ProjectAssignments VALUES

(1, 1, 2),

(2, 2, 3);

SELECT \* FROM Departments;

SELECT \* FROM Employees;

SELECT \* FROM Students;

SELECT \* FROM Courses;

SELECT \* FROM Enrollments;

SELECT \* FROM Authors;

SELECT \* FROM Books;

SELECT \* FROM Categories;

SELECT \* FROM Suppliers;

SELECT \* FROM Products;

SELECT \* FROM Customers;

SELECT \* FROM Orders;

SELECT \* FROM OrderDetails;

SELECT \* FROM Projects;

SELECT \* FROM ProjectAssignments;

– ## Basic INNER JOIN (1–20)

--1.Get the names of all employees and their department names.

SELECT \* FROM Employees;

SELECT \* FROM Departments;

SELECT E.EmployeeName, D.DepartmentName FROM Employees E

Join Departments D ON E.DepartmentID = D.DepartmentID;

--2. List all orders along with the names of the customers who placed them.

SELECT \* FROM Orders;

SELECT \* FROM Customers;

SELECT O.OrderID, C.CustomerName FROM Orders O

JOIN Customers C ON O.CustomerID = C.CustomerID;

--3. Show the product name and category name for all products.

SELECT \* FROM Products;

SELECT \* FROM Categories;

SELECT P.ProductName, C.CategoryName FROM Products P

JOIN Categories C ON P.CategoryID = C.CategoryID;

--4. List students and the courses they are enrolled in.

SELECT \* FROM Students;

SELECT \* FROM Courses;

SELECT \* FROM Enrollments;

SELECT S.StudentName, C.CourseName FROM Enrollments E

JOIN Students S ON E.StudentID = S.StudentID

JOIN Courses C ON E.CourseID = C.CourseID;

--5. Retrieve book titles and author names.

SELECT \* FROM Books;

SELECT \* FROM Authors;

SELECT B.Title, A.AuthorName FROM Books B

JOIN Authors A ON B.AuthorID = A.AuthorID;

--6. Show all invoices along with the customer’s name and country.

CREATE TABLE Customer (

customer\_id INT PRIMARY KEY,

name VARCHAR(100),

country VARCHAR(100));

CREATE TABLE Invoice (

invoice\_id INT PRIMARY KEY,

amount DECIMAL(10, 2),

date DATE,

customer\_id INT,

FOREIGN KEY (customer\_id) REFERENCES Customer(customer\_id));

INSERT INTO Customer VALUES

(1, 'Alice Johnson', 'USA'),

(2, 'Raj Patel', 'India');

INSERT INTO Invoice VALUES

(101, 299.99, '2025-03-01', 1),

(102, 150.00, '2025-03-05', 2);

SELECT \* FROM Customer;

SELECT \* FROM Invoice;

SELECT i.invoice\_id, i.amount, i.date, c.name AS customer\_name, c.country FROM Invoice i

JOIN Customer c ON i.customer\_id = c.customer\_id;

--7. Display all projects and their assigned manager names.

SELECT \* FROM Projects;

SELECT \* FROM ProjectAssignments;

SELECT \* FROM Employees;

SELECT P.ProjectName, E.EmployeeName AS ManagerName FROM Projects P

JOIN Employees E ON P.ManagerID = E.EmployeeID;

--8. List all patients and their attending doctors.

CREATE TABLE Doctor (

doctor\_id INT PRIMARY KEY,

name VARCHAR(100),

specialty VARCHAR(100));

CREATE TABLE Patient (

patient\_id INT PRIMARY KEY,

name VARCHAR(100),

doctor\_id INT,

FOREIGN KEY (doctor\_id) REFERENCES Doctor(doctor\_id));

INSERT INTO Doctor VALUES

(1, 'Dr. Smith', 'Cardiology'),

(2, 'Dr. Adams', 'Neurology');

INSERT INTO Patient VALUES

(101, 'John Doe', 1),

(102, 'Jane Roe', 2);

SELECT \* FROM Doctor;

SELECT \* FROM Patient;

SELECT p.name AS patient\_name, d.name AS doctor\_name, d.specialty FROM Patient p

JOIN Doctor d ON p.doctor\_id = d.doctor\_id;

--9. Retrieve order details along with the corresponding product names.

SELECT \* FROM OrderDetails;

SELECT \* FROM Products;

SELECT OD.OrderDetailID, P.ProductName FROM OrderDetails OD

JOIN Products P ON OD.ProductID = P.ProductID;

--10. Get all employees with their job titles.

SELECT \* FROM Employees;

SELECT EmployeeName, JobTitle FROM Employees;

--11. Show all sales along with salesperson names.

CREATE TABLE Salesperson (

salesperson\_id INT PRIMARY KEY,

name VARCHAR(100));

CREATE TABLE Sales (

sale\_id INT PRIMARY KEY,

amount DECIMAL(10, 2),

date DATE,

salesperson\_id INT,

FOREIGN KEY (salesperson\_id) REFERENCES Salesperson(salesperson\_id));

INSERT INTO Salesperson VALUES

(1, 'Alice'),

(2, 'Bob');

INSERT INTO Sales VALUES

(101, 500.00, '2025-01-01', 1),

(102, 700.00, '2025-01-02', 2);

SELECT \* FROM Salesperson;

SELECT \* FROM Sales;

SELECT s.sale\_id, s.amount, sp.name FROM Sales s

JOIN Salesperson sp ON s.salesperson\_id = sp.salesperson\_id;

--12. Get all transactions and the bank branch they occurred in.

CREATE TABLE Branch (

branch\_id INT PRIMARY KEY,

branch\_name VARCHAR(100));

CREATE TABLE Transactions (

transaction\_id INT PRIMARY KEY,

amount DECIMAL(10, 2),

branch\_id INT,

date DATE,

FOREIGN KEY (branch\_id) REFERENCES Branch(branch\_id));

INSERT INTO Branch VALUES

(1, 'Downtown'),

(2, 'Uptown');

INSERT INTO Transactions VALUES

(1001, 250.00, 1, '2025-04-01'),

(1002, 400.00, 2, '2025-04-02');

SELECT \* FROM Branch;

SELECT \* FROM Transactions;

SELECT t.transaction\_id, t.amount, b.branch\_name FROM Transactions t

JOIN Branch b ON t.branch\_id = b.branch\_id;

--13. List all vehicles and their owners.

CREATE TABLE Owners (

owners\_id INT PRIMARY KEY,

name VARCHAR(100));

CREATE TABLE Vehicle (

vehicle\_id INT PRIMARY KEY,

model VARCHAR(100),

owners\_id INT,

FOREIGN KEY (owners\_id) REFERENCES Owners(owners\_id));

INSERT INTO Owners VALUES

(1, 'David'),

(2, 'Emma');

INSERT INTO Vehicle VALUES

(501, 'Toyota Camry', 1),

(502, 'Honda Civic', 2);

SELECT \* FROM Owners;

SELECT \* FROM Vehicle;

SELECT v.vehicle\_id, v.model, o.name FROM Vehicle v

JOIN Owners o ON v.owners\_id = o.owners\_id;

--14. Show all classes with their teachers.

CREATE TABLE Teacher (

teacher\_id INT PRIMARY KEY,

name VARCHAR(100));

CREATE TABLE Class (

class\_id INT PRIMARY KEY,

class\_name VARCHAR(100),

teacher\_id INT,

FOREIGN KEY (teacher\_id) REFERENCES Teacher(teacher\_id));

INSERT INTO Teacher VALUES

(1, 'Mrs. Smith'),

(2, 'Mr. Johnson');

INSERT INTO Class VALUES

(101, 'Math', 1),

(102, 'Science', 2);

SELECT \* FROM Teacher;

SELECT \* FROM Class;

SELECT c.class\_name, t.name FROM Class c

JOIN Teacher t ON c.teacher\_id = t.teacher\_id;

--15. Get all movie names and their directors.

CREATE TABLE Director (

director\_id INT PRIMARY KEY,

name VARCHAR(100));

CREATE TABLE Movie (

movie\_id INT PRIMARY KEY,

title VARCHAR(100),

director\_id INT,

FOREIGN KEY (director\_id) REFERENCES Director(director\_id));

INSERT INTO Director VALUES

(1, 'Christopher Nolan'),

(2, 'Steven Spielberg');

INSERT INTO Movie VALUES

(201, 'Inception', 1),

(202, 'Jurassic Park', 2);

SELECT \* FROM Director;

SELECT \* FROM Movie;

SELECT m.title, d.name FROM Movie m

JOIN Director d ON m.director\_id = d.director\_id;

--16. List all athletes with their respective teams.

CREATE TABLE Team (

team\_id INT PRIMARY KEY,

team\_name VARCHAR(100));

CREATE TABLE Athlete (

athlete\_id INT PRIMARY KEY,

name VARCHAR(100),

team\_id INT,

FOREIGN KEY (team\_id) REFERENCES Team(team\_id));

INSERT INTO Team VALUES

(1, 'Team A'),

(2, 'Team B');

INSERT INTO Athlete VALUES

(301, 'Usain Bolt', 1),

(302, 'Michael Phelps', 2);

SELECT \* FROM Team;

SELECT \* FROM Athlete;

SELECT a.name, t.team\_name FROM Athlete a

JOIN Team t ON a.team\_id = t.team\_id;

--17. Show products and their suppliers.

SELECT \* FROM Products;

SELECT \* FROM Suppliers;

SELECT P.ProductName, S.SupplierName

FROM Products P

JOIN Suppliers S ON P.SupplierID = S.SupplierID;

--18. List blog posts with their respective authors.

CREATE TABLE Author (

author\_id INT PRIMARY KEY,

name VARCHAR(100));

CREATE TABLE BlogPost (

post\_id INT PRIMARY KEY,

title VARCHAR(100),

content TEXT,

author\_id INT,

FOREIGN KEY (author\_id) REFERENCES Author(author\_id));

INSERT INTO Author VALUES

(1, 'John Doe'),

(2, 'Jane Smith');

INSERT INTO BlogPost VALUES

(101, 'Intro to SQL', 'This is a SQL tutorial.', 1),

(102, 'Advanced Python', 'Decorators and generators explained.', 2)

SELECT \* FROM BlogPost;

SELECT \* FROM Author;

SELECT b.title, a.name FROM BlogPost b

JOIN Author a ON b.author\_id = a.author\_id;

--19. Show employees and the location of their offices.

CREATE TABLE Office (

office\_id INT PRIMARY KEY,

location VARCHAR(100));

CREATE TABLE Employee (

emp\_id INT PRIMARY KEY,

name VARCHAR(100),

office\_id INT,

FOREIGN KEY (office\_id) REFERENCES Office(office\_id));

INSERT INTO Office VALUES

(1, 'New York'),

(2, 'San Francisco');

INSERT INTO Employee VALUES

(1, 'Alice', 1),

(2, 'Bob', 2);

SELECT \* FROM Office;

SELECT \* FROM Employee;

SELECT e.name, o.location FROM Employee e

JOIN Office o ON e.office\_id = o.office\_id;

--20. Display all items in a shopping cart with product descriptions.

CREATE TABLE Product (

product\_id INT PRIMARY KEY,

name VARCHAR(100),

description TEXT);

CREATE TABLE ShoppingCart (

cart\_item\_id INT PRIMARY KEY,

product\_id INT,

quantity INT,

FOREIGN KEY (product\_id) REFERENCES Product(product\_id));

INSERT INTO Product VALUES

(1, 'Laptop', '14-inch screen, 8GB RAM'),

(2, 'Mouse', 'Wireless optical mouse');

INSERT INTO ShoppingCart VALUES

(1, 1, 1),

(2, 2, 2);

SELECT \* FROM ShoppingCart;

SELECT \* FROM Product;

SELECT p.name, p.description, s.quantity FROM ShoppingCart s

JOIN Product p ON s.product\_id = p.product\_id;

– ## LEFT JOIN (21–40)

--21. List all customers, even those who haven’t placed orders.

SELECT \* FROM Customers;

SELECT \* FROM Orders;

SELECT c.CustomerName, o.OrderID, o.TotalAmount FROM Customers c

LEFT JOIN Orders o ON c.CustomerID = o.CustomerID;

--22. Get all products and their order quantities (even if never ordered).

SELECT \* FROM Products;

SELECT \* FROM OrderDetails;

SELECT p.ProductName, oi.Quantity FROM Products p

LEFT JOIN OrderDetails oi ON p.ProductID = oi.ProductID;

--23. Show all students and the grades they've received (if any).

SELECT \* FROM students;

SELECT \* FROM Student;

SELECT Name, Grade FROM Student;

--24. List all employees and their assigned projects (if any).

SELECT \* FROM Employees;

--SELECT \* FROM Employee;

SELECT \* FROM Projects;

--SELECT \* FROM ProjectAssignments;

SELECT e.EmployeeName, p.ProjectName FROM Employees e

LEFT JOIN Projects p ON e.ManagerID = p.ManagerID;

--25. Show all departments and their employees (if any).

SELECT \* FROM Employees;

SELECT \* FROM Departments;

SELECT D.DepartmentName, E.EmployeeName FROM Departments D

LEFT JOIN Employees E ON D.DepartmentID = E.DepartmentID;

--26. Display all articles and any comments (if any).

CREATE TABLE articles (

article\_id INT PRIMARY KEY,

title VARCHAR(255));

CREATE TABLE comments (

comment\_id INT PRIMARY KEY,

article\_id INT,

comment\_text VARCHAR(255),

FOREIGN KEY (article\_id) REFERENCES articles(article\_id));

INSERT INTO articles (article\_id, title) VALUES

(1, 'How to Learn SQL'),

(2, 'Benefits of Exercise'),

(3, 'Top Travel Destinations'),

(4, 'Understanding AI');

INSERT INTO comments (comment\_id, article\_id, comment\_text) VALUES

(1, 1, 'Very helpful article!'),

(2, 1, 'Thanks, this clarified a lot.'),

(3, 2, 'Exercise really helps me focus.'),

(4, 3, 'I want to visit these places!');

SELECT \* FROM articles;

SELECT \* FROM comments;

SELECT A.title, C.comment\_text FROM articles A

LEFT JOIN comments C ON A.article\_id = C.article\_id;

--27. List all books, even those without an author (null author\\_id).

SELECT \* FROM Authors;

SELECT \* FROM Books;

SELECT b.BookID, b.Title, a.AuthorName FROM Books b

LEFT JOIN Authors a ON b.AuthorID = a.AuthorID;

--28. Show all courses, including those with no enrollments.

SELECT \* FROM Courses;

SELECT \* FROM Enrollments;

SELECT C.CourseName, E.EnrollmentID FROM Courses C

LEFT JOIN Enrollments E ON C.CourseID = E.CourseID;

--29. Get all users and their last login time (if logged in).

CREATE TABLE logins (

login\_id INT PRIMARY KEY,

UserID INT,

login\_time DATETIME,

FOREIGN KEY (UserID) REFERENCES Users(UserID));

INSERT INTO logins (login\_id, UserID, login\_time) VALUES

(1, 1, '2025-05-01 09:00:00'),

(2, 1, '2025-05-06 10:30:00'),

(3, 2, '2025-05-02 14:15:00'),

(4, 3, '2025-05-03 08:45:00'),

(5, 2, '2025-05-06 16:00:00');

SELECT \* FROM Users;

SELECT \* FROM logins;

SELECT U.UserID, U.UserName, MAX(l.login\_time) AS last\_login\_time FROM Users U

LEFT JOIN logins L ON U.UserID = L.UserID

GROUP BY

U.UserName, U.UserID;

--30. List all patients, even if no appointment is scheduled.

CREATE TABLE appointments (

id INT IDENTITY(1,1) PRIMARY KEY,

patient\_id INT,

appointment\_date DATE,

doctor\_name VARCHAR(255),

FOREIGN KEY (patient\_id) REFERENCES Patient(patient\_id));

INSERT INTO appointments VALUES

(101, '2025-05-06', 'Dr. Smith'),

(102, '2025-05-07', 'Dr. Johnson');

SELECT \* FROM Patient;

SELECT \* FROM appointments;

SELECT p.name, a.appointment\_date FROM Patient p

LEFT JOIN appointments a ON p.patient\_id = a.patient\_id;

--31. Show all teachers, even if they are not teaching any class.

SELECT \* FROM Teacher;

SELECT \* FROM Class;

SELECT T.name, C.class\_name FROM Teacher T

LEFT JOIN Class C ON T.teacher\_id = C.teacher\_id;

--32. List all cars, even those not assigned to any customer.

CREATE TABLE cars (

car\_id INT IDENTITY(1,1) PRIMARY KEY,

car\_name VARCHAR(255),

customer\_id INT,

FOREIGN KEY (customer\_id) REFERENCES customers(CustomerID) );

INSERT INTO cars VALUES

('Toyota Corolla', 1),

('Honda Civic', 2),

('Ford Mustang', NULL);

SELECT \* FROM Customers;

SELECT \* FROM cars;

SELECT C.car\_name, CU.CustomerName FROM cars C

LEFT JOIN Customers CU ON C.customer\_id = CU.CustomerID;

--33. Get all songs and their album names (even if not in any album).

CREATE TABLE albums (

album\_id INT IDENTITY(1,1) PRIMARY KEY,

album\_name VARCHAR(255));

CREATE TABLE songs (

song\_id INT IDENTITY(1,1) PRIMARY KEY,

song\_name VARCHAR(255),

album\_id INT NULL,

FOREIGN KEY (album\_id) REFERENCES albums(album\_id));

INSERT INTO albums (album\_name) VALUES

('Album A'),

('Album B');

INSERT INTO songs (song\_name, album\_id) VALUES

('Song 1', 1),

('Song 2', 1),

('Song 3', 2),

('Song 4', NULL);

SELECT \* FROM albums;

SELECT \* FROM songs;

SELECT s.song\_id, s.song\_name, a.album\_name FROM songs s

LEFT JOIN albums a ON s.album\_id = a.album\_id;

--34. Show all houses and the tenants living there (if any).

CREATE TABLE houses (

house\_id INT IDENTITY(1,1) PRIMARY KEY,

address VARCHAR(255)

);

CREATE TABLE tenants (

tenant\_id INT IDENTITY(1,1) PRIMARY KEY,

name VARCHAR(255),

house\_id INT NULL,

FOREIGN KEY (house\_id) REFERENCES houses(house\_id)

);

INSERT INTO houses (address) VALUES

('123 Maple St'),

('456 Oak Ave'),

('789 Pine Rd');

INSERT INTO tenants (name, house\_id) VALUES

('Alice', 1),

('Bob', 1),

('Charlie', 2);

SELECT \* FROM houses;

SELECT \* FROM tenants;

SELECT h.house\_id, h.address, t.tenant\_id, t.name AS tenant\_name FROM houses h

LEFT JOIN tenants t ON h.house\_id = t.house\_id;

--35. List all posts and the number of likes (even if zero).

CREATE TABLE posts (

post\_id INT IDENTITY(1,1) PRIMARY KEY,

post\_title VARCHAR(255)

);

CREATE TABLE likes (

like\_id INT IDENTITY(1,1) PRIMARY KEY,

post\_id INT NOT NULL,

FOREIGN KEY (post\_id) REFERENCES posts(post\_id)

);

INSERT INTO posts (post\_title) VALUES

('First Post'),

('Second Post'),

('Third Post');

INSERT INTO likes (post\_id) VALUES

(1),

(1),

(2);

SELECT \* FROM posts;

SELECT \* FROM likes;

SELECT p.post\_id, p.post\_title, COUNT(l.like\_id) AS like\_count FROM posts p

LEFT JOIN likes l ON p.post\_id = l.post\_id

GROUP BY p.post\_id, p.post\_title;

--36. Get all employees and their emergency contact details (if any).

CREATE TABLE EmergencyContacts (

ContactID INT PRIMARY KEY,

EmployeeID INT,

ContactName VARCHAR(100),

Relationship VARCHAR(50),

PhoneNumber VARCHAR(20),

FOREIGN KEY (EmployeeID) REFERENCES Employees(EmployeeID)

);

INSERT INTO EmergencyContacts (ContactID, EmployeeID, ContactName, Relationship, PhoneNumber) VALUES

(1, 1, 'Eve', 'Spouse', '1234567890'),

(2, 2, 'Frank', 'Brother', '9876543210'),

(3, 3, 'Grace', 'Friend', '5551234567');

SELECT \* FROM Employees;

SELECT \* FROM EmergencyContacts;

SELECT e.EmployeeID,e.EmployeeName, ec.ContactName,ec.PhoneNumber FROM Employees e

LEFT JOIN EmergencyContacts ec ON e.EmployeeID = ec.EmployeeID;

--37. Display all branches and their ATM machines (if any).

CREATE TABLE Branches (

BranchID INT PRIMARY KEY,

BranchName VARCHAR(100),

Location VARCHAR(100)

);

CREATE TABLE ATMs (

ATMID INT PRIMARY KEY,

BranchID INT,

ATMType VARCHAR(50),

FOREIGN KEY (BranchID) REFERENCES Branches(BranchID)

);

INSERT INTO Branches VALUES

(1, 'Main Branch', 'Downtown'),

(2, 'West Branch', 'West City'),

(3, 'North Branch', 'North Avenue');

INSERT INTO ATMs VALUES

(101, 1, 'Cash Deposit'),

(102, 1, 'Cash Withdrawal'),

(103, 2, 'Multi-function');

SELECT \* FROM Branches;

SELECT \* FROM ATMs;

SELECT b.BranchID, b.BranchName, a.ATMID,a.ATMType FROM Branches b

LEFT JOIN ATMs a ON b.BranchID = a.BranchID;

--38. List all schools and their principals (even if no principal assigned).

CREATE TABLE Schools (

SchoolID INT PRIMARY KEY,

SchoolName VARCHAR(100),

City VARCHAR(100)

);

CREATE TABLE Principals (

PrincipalID INT PRIMARY KEY,

SchoolID INT,

PrincipalName VARCHAR(100),

FOREIGN KEY (SchoolID) REFERENCES Schools(SchoolID)

);

INSERT INTO Schools VALUES

(1, 'Green Valley High', 'New York'),

(2, 'Sunrise Academy', 'Los Angeles'),

(3, 'Maple Leaf School', 'Chicago');

INSERT INTO Principals VALUES

(1, 1, 'Mr. John Smith'),

(2, 2, 'Mrs. Linda Green');

SELECT \* FROM Schools;

SELECT \* FROM Principals;

SELECT S.SchoolName, P.PrincipalName FROM Schools S

LEFT JOIN Principals P ON S.SchoolID = P.SchoolID;

--39. Get all flights and the corresponding pilots (if any).

CREATE TABLE Flights (

FlightID INT PRIMARY KEY,

FlightNumber VARCHAR(50),

Destination VARCHAR(100)

);

CREATE TABLE Pilots (

PilotID INT PRIMARY KEY,

FlightID INT,

PilotName VARCHAR(100),

FOREIGN KEY (FlightID) REFERENCES Flights(FlightID)

);

INSERT INTO Flights VALUES

(1, 'AI101', 'New Delhi'),

(2, 'BA202', 'London'),

(3, 'DL303', 'New York');

INSERT INTO Pilots VALUES

(1, 1, 'Captain Raj'),

(2, 2, 'Captain Smith');

SELECT \* FROM Flights;

SELECT \* FROM Pilots;

SELECT F.FlightNumber, P.PilotName FROM Flights F

LEFT JOIN Pilots P ON F.FlightID = P.FlightID;

--40. Show all resumes and job applications linked to them (if any).

CREATE TABLE Resumes (

ResumeID INT PRIMARY KEY,

CandidateName VARCHAR(100),

Skills VARCHAR(255)

);

CREATE TABLE JobApplications (

ApplicationID INT PRIMARY KEY,

ResumeID INT,

JobTitle VARCHAR(100),

CompanyName VARCHAR(100),

FOREIGN KEY (ResumeID) REFERENCES Resumes(ResumeID)

);

INSERT INTO Resumes VALUES

(1, 'Amit Sharma', 'Python, SQL, ML'),

(2, 'Neha Patel', 'Java, Spring'),

(3, 'John Doe', 'React, Node.js');

INSERT INTO JobApplications VALUES

(1, 1, 'Data Scientist', 'TechCorp'),

(2, 1, 'ML Engineer', 'InnoSoft'),

(3, 3, 'Frontend Dev', 'WebSolutions');

SELECT \* FROM Resumes;

SELECT \* FROM JobApplications;

SELECT R.ResumeID, R.CandidateName, J.JobTitle FROM Resumes R

LEFT JOIN JobApplications J ON R.ResumeID = J.ResumeID;

**– ## RIGHT JOIN (41–55)**

--41. Get all orders, including those without customer info.

SELECT \* FROM Customers;

SELECT \* FROM Orders;

SELECT o.OrderID, o.OrderDate,c.CustomerName FROM Customers c

RIGHT JOIN Orders o ON c.CustomerID = o.CustomerID;

--42. Show all departments and employees assigned (even if none).

SELECT \* FROM Departments;

SELECT \* FROM Employees;

SELECT d.DepartmentID,d.DepartmentName,e.EmployeeName FROM Employees e

RIGHT JOIN Departments d ON e.DepartmentID = d.DepartmentID;

--43. List all job postings and the applicants (even if no one applied).

CREATE TABLE Jobs (

JobID INT PRIMARY KEY,

JobTitle VARCHAR(100)

);

CREATE TABLE Applications (

ApplicationID INT PRIMARY KEY,

JobID INT,

ApplicantName VARCHAR(100),

FOREIGN KEY (JobID) REFERENCES Jobs(JobID)

);

INSERT INTO Jobs VALUES

(1, 'Software Engineer'), (2, 'Data Analyst'), (3, 'UI Designer');

INSERT INTO Applications VALUES

(201, 1, 'Anita'), (202, 2, 'Ravi');

SELECT \* FROM Jobs;

SELECT \* FROM Applications;

SELECT j.JobTitle, a.ApplicantName FROM Applications a

RIGHT JOIN Jobs j ON a.JobID = j.JobID;

--44. Show all songs and artists (even if song is missing artist info).

CREATE TABLE Artists (

ArtistID INT PRIMARY KEY,

ArtistName VARCHAR(100)

);

CREATE TABLE music (

SongID INT PRIMARY KEY,

Title VARCHAR(100),

ArtistID INT,

FOREIGN KEY (ArtistID) REFERENCES Artists(ArtistID)

);

INSERT INTO Artists VALUES

(1, 'Taylor Swift'), (2, 'Ed Sheeran');

INSERT INTO music VALUES

(301, 'Love Story', 1),

(302, 'Shape of You', 2),

(303, 'Unknown Song', NULL);

SELECT \* FROM Artists;

SELECT \* FROM music;

SELECT m.Title,a.ArtistName FROM Artists a

RIGHT JOIN music m ON a.ArtistID = m.ArtistID;

--45. Display all bank branches and employees (even if branch has no staff).

ALTER TABLE Employees

ADD BranchID INT;

UPDATE Employees

SET BranchID = 2

WHERE EmployeeID = 1;

UPDATE Employees

SET BranchID = 1

WHERE EmployeeID = 2;

UPDATE Employees

SET BranchID = 3

WHERE EmployeeID = 3;

SELECT \* FROM Branches;

SELECT \* FROM Employees;

SELECT B.BranchName, E.EmployeeName FROM Employees E

RIGHT JOIN Branches B ON E.BranchID = B.BranchID ;

--46. Get all flights and airports they depart from (even if flight data is missing).

CREATE TABLE Airports (

AirportID INT PRIMARY KEY,

AirportName VARCHAR(100)

);

INSERT INTO Airports VALUES

(1, 'Heathrow'), (2, 'JFK'), (3, 'Dubai Intl');

ALTER TABLE Flights

ADD AirportID INT;

UPDATE Flights

SET AirportID = 1

WHERE FlightID = 3;

UPDATE Flights

SET AirportID = 2

WHERE FlightID = 1;

UPDATE Flights

SET AirportID = 3

WHERE FlightID = 2;

SELECT \* FROM Airports;

SELECT \* FROM Flights;

SELECT f.FlightNumber,a.AirportName FROM Airports a

RIGHT JOIN Flights f ON a.AirportID = f.AirportID;

--47. List all stores and the inventory they hold (even if store is empty).

CREATE TABLE Stores (

StoreID INT PRIMARY KEY,

StoreName VARCHAR(100)

);

CREATE TABLE Inventory (

ItemID INT PRIMARY KEY,

StoreID INT,

ItemName VARCHAR(100),

FOREIGN KEY (StoreID) REFERENCES Stores(StoreID)

);

INSERT INTO Stores VALUES

(1, 'ElectroMart'), (2, 'GroceryWorld');

INSERT INTO Inventory VALUES

(601, 1, 'Laptop'),

(602, 1, 'Smartphone');

SELECT \* FROM Stores;

SELECT \* FROM Inventory;

SELECT s.StoreName, i.ItemName FROM Inventory i

RIGHT JOIN Stores s ON i.StoreID = s.StoreID;

--48. Show all sports teams and their players (even if team has no players).

CREATE TABLE Players (

PlayerID INT PRIMARY KEY,

PlayerName VARCHAR(100),

team\_id INT,

FOREIGN KEY (team\_id) REFERENCES Team(team\_id)

);

INSERT INTO Players VALUES

(701, 'Leo', 1),

(702, 'Max', 2);

SELECT \* FROM Team;

SELECT \* FROM Players;

SELECT t.team\_name,p.PlayerName FROM Players p

RIGHT JOIN Team t ON p.team\_id = t.team\_id;

--49. List all invoices and the payments made (even if unpaid).

CREATE TABLE Payments (

PaymentID INT PRIMARY KEY,

InvoiceID INT,

PaymentAmount DECIMAL(10,2),

FOREIGN KEY (InvoiceID) REFERENCES Invoice(invoice\_id)

);

INSERT INTO Payments VALUES

(801, 101, 500.00),

(802, 102, 750.00),

(803, 103, 850.00);

;

INSERT INTO Invoice VALUES

(103, 599.99, '2025-03-06', 2);

SELECT \* FROM Invoice;

SELECT \* FROM Payments;

SELECT i.invoice\_id, p.PaymentAmount FROM Payments p

RIGHT JOIN Invoice i ON p.InvoiceID = i.invoice\_id;

--50. Show all orders and shipping details (even if not shipped).

CREATE TABLE Shipping (

ShippingID INT PRIMARY KEY,

OrderID INT,

ShippingDate DATE,

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)

);

INSERT INTO Shipping VALUES

(901, 1, '2025-05-04'),

(902, 2, '2025-05-05');

SELECT \* FROM Orders;

SELECT \* FROM Shipping;

SELECT o.OrderID, o.OrderDate, s.ShippingDate FROM Shipping s

RIGHT JOIN Orders o ON s.OrderID = o.OrderID;

--51. Get all rental properties and tenants (even if vacant).

CREATE TABLE Properties (

PropertyID INT PRIMARY KEY,

Address VARCHAR(255),

tenant\_id INT,

FOREIGN KEY (tenant\_id) REFERENCES Tenants(tenant\_id)

);

INSERT INTO Properties VALUES

(101, '12 Oak Street', 1),

(102, '45 Pine Avenue', NULL),

(103, '78 Maple Road', 2);

SELECT \* FROM Tenants;

SELECT \* FROM Properties;

SELECT p.PropertyID, p.Address, [t.name](http://t.name) FROM Tenants t

RIGHT JOIN Properties p ON t.tenant\_id = p.tenant\_id;

--52. List all service centers and the vehicles serviced (even if none).

CREATE TABLE ServiceCenters (

CenterID INT PRIMARY KEY,

CenterName VARCHAR(100)

);

INSERT INTO ServiceCenters VALUES

(1, 'QuickFix'),

(2, 'AutoCare'),

(3, 'SpeedyServ');

ALTER TABLE Vehicle

ADD CenterID INT;

UPDATE Vehicle

SET CenterID = 1

WHERE vehicle\_id = 501;

UPDATE Vehicle

SET CenterID = 2

WHERE vehicle\_id = 502;

SELECT \* FROM ServiceCenters;

SELECT \* FROM Vehicle;

SELECT v.vehicle\_id, v.model, s.CenterName

FROM Vehicle v

RIGHT JOIN ServiceCenters s ON v.CenterID = s.CenterID;

--53. Show all products and the vendors supplying them (even if unsupplied).

CREATE TABLE Vendors (

VendorID INT PRIMARY KEY,

VendorName VARCHAR(100)

);

INSERT INTO Vendors VALUES

(1, 'Global Supplies'),

(2, 'TechSource'),

(3, 'OfficeMart');

ALTER TABLE Products

ADD VendorID INT;

UPDATE Products

SET VendorID = 1

WHERE ProductID = 2;

UPDATE Products

SET VendorID = 2

WHERE ProductID = 3;

UPDATE Products

SET VendorID = 3

WHERE ProductID = 1;

SELECT \* FROM Vendors;

SELECT \* FROM Products;

SELECT v.VendorName, p.ProductName FROM Vendors v

RIGHT JOIN Products p ON v.VendorID = p.VendorID;

--54. Display all training sessions and attendees (even if session is empty).

CREATE TABLE Attendees (

AttendeeID INT PRIMARY KEY,

AttendeeName VARCHAR(100)

);

CREATE TABLE Sessions (

SessionID INT PRIMARY KEY,

Topic VARCHAR(100),

AttendeeID INT,

FOREIGN KEY (AttendeeID) REFERENCES Attendees(AttendeeID)

);

INSERT INTO Attendees VALUES

(1, 'John'),

(2, 'Emma'),

(3, 'Liam');

INSERT INTO Sessions VALUES

(501, 'SQL Basics', 1),

(502, 'Advanced Python', NULL),

(503, 'Data Analysis', 2);

SELECT \* FROM Attendees;

SELECT \* FROM Sessions;

SELECT a.AttendeeName, s.Topic FROM Attendees a

RIGHT JOIN Sessions s ON a.AttendeeID = s.AttendeeID;

--55. Get all users and their transactions (even if they made none).

CREATE TABLE Users (

UserID INT PRIMARY KEY,

UserName VARCHAR(100)

);

INSERT INTO Users VALUES

(1, 'David'),

(2, 'Sophia'),

(3, 'Chris');

ALTER TABLE Transactions

ADD UserID INT;

UPDATE Transactions

SET UserID = 1

WHERE transaction\_id = 1001;

UPDATE Transactions

SET UserID = 2

WHERE transaction\_id = 1002;

SELECT \* FROM Users;

SELECT \* FROM Transactions;

SELECT t.transaction\_id, t.Amount, u.UserName

FROM Transactions t

RIGHT JOIN Users u ON t.UserID = u.UserID;

**– ## FULL OUTER JOIN (56–70)**

--56. List all employees and their managers (even if no relation exists).

SELECT \* FROM Employees;

SELECT e.EmployeeID AS EmployeeID,e.EmployeeName AS EmployeeName,m.EmployeeName AS ManagerName FROM Employees e

FULL OUTER JOIN Employees m ON e.ManagerID = m.EmployeeID;

--57. Get all books and authors (even if unmatched).

SELECT \* FROM Authors;

SELECT \* FROM Books;

SELECT b.Title, a.AuthorName FROM Books b

FULL OUTER JOIN Authors a ON b.AuthorID = a.AuthorID;

--58. Display all customers and orders (even if either is missing).

SELECT \* FROM Customers;

SELECT \* FROM Orders;

SELECT c.CustomerName,o.OrderID,o.TotalAmount FROM Customers c

FULL OUTER JOIN Orders o ON c.CustomerID = o.CustomerID;

--59. Show all students and classes (even if not enrolled).

ALTER TABLE Class

ADD StudentID INT;

UPDATE Class

SET StudentID = 1

WHERE class\_id = 101;

UPDATE Class

SET StudentID = 3

WHERE class\_id = 102;

SELECT \* FROM Students;

SELECT \* FROM Class;

SELECT

s.StudentName,

c.class\_name

FROM Students s

FULL OUTER JOIN Class c ON s.StudentID = c.StudentID;

--60. Get all products and their reviews (even if not reviewed).

CREATE TABLE Reviews (

ReviewID INT PRIMARY KEY,

ProductID INT,

Rating INT

);

INSERT INTO Reviews VALUES

(101, 1, 5),

(102, 1, 4),

(103, 4, 3);

SELECT \* FROM Products;

SELECT \* FROM Reviews;

SELECT p.ProductID, p.ProductName, r.Rating FROM Products p

FULL OUTER JOIN Reviews r ON p.ProductID = r.ProductID;

--61. List all applications and applicants (even if unlinked).

CREATE TABLE Applicants (

applicant\_id INT PRIMARY KEY,

applicant\_name VARCHAR(100)

);

INSERT INTO Applicants VALUES

(1, 'Alice'),

(2, 'Bob'),

(3, 'Charlie');

ALTER TABLE Applications

ADD applicant\_id INT;

UPDATE Applications

SET applicant\_id = 1

WHERE ApplicationID = 201;

UPDATE Applications

SET applicant\_id = 2

WHERE ApplicationID = 202;

SELECT \* FROM Applications;

SELECT \* FROM Applicants;

SELECT a.ApplicationID, ap.applicant\_name

FROM Applications a

FULL OUTER JOIN Applicants ap

ON a.applicant\_id = ap.applicant\_id;

--62. Show all users and their profile pictures (even if missing).

CREATE TABLE ProfilePictures (

picture\_id INT PRIMARY KEY,

user\_id INT,

picture\_url VARCHAR(200),

FOREIGN KEY (user\_id) REFERENCES Users(UserID)

);

INSERT INTO ProfilePictures VALUES

(201, 1, 'pic1.jpg'),

(202, 2, 'pic2.jpg');

SELECT \* FROM Users;

SELECT \* FROM ProfilePictures;

SELECT u.username, p.picture\_url FROM Users u

FULL OUTER JOIN ProfilePictures p ON u.UserID = p.user\_id;

--63. List all volunteers and events (even if not linked).

CREATE TABLE Volunteers (

volunteer\_id INT PRIMARY KEY,

name VARCHAR(100)

);

CREATE TABLE Events (

event\_id INT PRIMARY KEY,

volunteer\_id INT,

event\_name VARCHAR(100),

FOREIGN KEY (volunteer\_id) REFERENCES Volunteers(volunteer\_id)

);

INSERT INTO Volunteers VALUES

(1, 'John'),

(2, 'Sara');

INSERT INTO Events VALUES

(301, 1, 'Charity Run'),

(302, NULL, 'Beach Cleanup');

SELECT \* FROM Events;

SELECT \* FROM Volunteers;

SELECT v.name AS volunteer\_name, e.event\_name FROM Volunteers v

FULL OUTER JOIN Events e ON v.volunteer\_id = e.volunteer\_id;

--64. Display all assets and owners (even if unassigned).

CREATE TABLE Assets (

asset\_id INT PRIMARY KEY,

asset\_name VARCHAR(100),

owners\_id INT,

FOREIGN KEY (owners\_id) REFERENCES Owners(owners\_id)

);

INSERT INTO Assets VALUES

(401, 'Laptop', 1),

(402, 'Printer', NULL);

SELECT \* FROM Owners;

SELECT \* FROM Assets;

SELECT a.asset\_name, o.name FROM Assets a

FULL OUTER JOIN Owners o ON a.owners\_id = o.owners\_id;

--65. Show all job titles and employees (even if no one holds the title).

ALTER TABLE Employees

ADD JobID INT;

UPDATE Employees

SET JobID = 1

WHERE EmployeeID = 1;

UPDATE Employees

SET JobID = 2

WHERE EmployeeID = 2;

UPDATE Employees

SET JobID = 3

WHERE EmployeeID = 3;

SELECT \* FROM Jobs;

SELECT \* FROM Employees;

SELECT jt.JobTitle, e.EmployeeName FROM Jobs jt

FULL OUTER JOIN Employees e ON jt.JobID = e.JobID;

--66. Get all cities and their airports (even if no airport).

CREATE TABLE cities (

city\_id INT PRIMARY KEY,

city\_name VARCHAR(100)

);

INSERT INTO cities VALUES

(1, 'New York'),

(2, 'Los Angeles'),

(3, 'Chicago'),

(4, 'Houston');

ALTER TABLE Airports

ADD city\_id INT;

UPDATE Airports

SET city\_id = 2

WHERE AirportID = 1;

UPDATE Airports

SET city\_id = 1

WHERE AirportID = 2;

UPDATE Airports

SET city\_id = 3

WHERE AirportID = 3;

SELECT \* FROM Airports;

SELECT \* FROM cities;

SELECT c.city\_name, a.AirportName FROM cities c

FULL OUTER JOIN airports a ON c.city\_id = a.city\_id;

--67. List all software and licenses (even if unused).

CREATE TABLE software (

software\_id INT PRIMARY KEY,

software\_name VARCHAR(100)

);

CREATE TABLE licenses (

license\_id INT PRIMARY KEY,

license\_key VARCHAR(100),

software\_id INT,

FOREIGN KEY (software\_id) REFERENCES software(software\_id)

);

INSERT INTO software VALUES

(1, 'Microsoft Office'),

(2, 'Adobe Photoshop'),

(3, 'Visual Studio');

INSERT INTO licenses VALUES

(201, 'XYZ-123-AAA', 1),

(202, 'ABC-456-BBB', 1),

(203, 'DEF-789-CCC', 2);

SELECT \* FROM software;

SELECT \* FROM licenses;

SELECT s.software\_name,l.license\_key FROM software s

FULL OUTER JOIN licenses l ON s.software\_id = l.software\_id;

--68. Show all classes and schedules (even if one is missing).

CREATE TABLE schedules (

schedule\_id INT PRIMARY KEY,

class\_id INT,

schedule\_time TIME,

FOREIGN KEY (class\_id) REFERENCES Class(class\_id)

);

INSERT INTO schedules VALUES

(1, 101, '09:00:00'),

(2, 102, '11:00:00');

SELECT \* FROM Class;

SELECT \* FROM schedules;

SELECT c.class\_name, s.schedule\_time FROM class c

FULL OUTER JOIN schedules s ON c.class\_id = s.class\_id;

--69. List all banks and customers (even if no link).

CREATE TABLE banks (

bank\_id INT PRIMARY KEY,

bank\_name VARCHAR(100)

);

INSERT INTO banks VALUES

(1, 'Bank of America'),

(2, 'Chase'),

(3, 'Wells Fargo');

ALTER TABLE Customer

ADD bank\_id INT;

UPDATE Customer

SET bank\_id = 1

WHERE customer\_id = 1;

UPDATE Customer

SET bank\_id = 2

WHERE customer\_id = 2;

SELECT \* FROM Customer;

SELECT \* FROM banks;

SELECT b.bank\_name, c.name FROM banks b

FULL OUTER JOIN customer c ON b.bank\_id = c.bank\_id;

--70. Display all employees and badges (even if missing).

CREATE TABLE badges (

badge\_id INT PRIMARY KEY,

badge\_number VARCHAR(100),

emp\_id INT,

FOREIGN KEY (emp\_id) REFERENCES employee(emp\_id)

);

INSERT INTO badges VALUES

(301, 'B123', 1),

(302, 'B456', 2);

SELECT \* FROM Employee;

SELECT \* FROM badges;

SELECT e.name, b.badge\_number FROM employee e

FULL OUTER JOIN badges b ON e.emp\_id = b.emp\_id;

**– ## SELF JOIN (71–80)**

--71. Find all employees who report to someone (supervisor).

SELECT \* FROM Employees;

SELECT

e.EmployeeName AS Employee,

m.EmployeeName AS Manager

FROM

Employees e

JOIN

Employees m ON e.DepartmentID = m.EmployeeID;

--72. List employees and their managers' names.

SELECT \* FROM Employees;

SELECT

e1.EmployeeName AS employee,

e2.EmployeeName AS Manager

FROM

Employees e1

JOIN

Employees e2

ON

e1.DepartmentID = e2.EmployeeID ;

--73. Find all products that are substitutes of other products.

SELECT \* FROM Products;

SELECT

p.ProductName AS Product,

s.ProductName AS Substitute

FROM

products p

JOIN

products s ON p.SupplierID = s.ProductID;

--74. List students and their mentors (who are also students).

SELECT \* FROM Students;

SELECT

s1.StudentName AS student,

s2.StudentName AS mentor

FROM

Students s1

JOIN

Students s2

ON

s1.MentorID = s2.StudentID ;

--75. Find users who referred other users.

ALTER TABLE Users

ADD referred\_by INT;

UPDATE Users

SET referred\_by = 1

WHERE USERID = 1;

UPDATE Users

SET referred\_by = 3

WHERE USERID = 2;

UPDATE Users

SET referred\_by = 2

WHERE USERID = 3;

SELECT \* FROM Users;

SELECT

u1.UserName AS Referrer

FROM

users u1

JOIN

users u2 ON u1.UserID = u2.referred\_by;

--76. Display a list of employees with the same job title.

SELECT \* FROM Employees;

SELECT

e1.EmployeeName AS emp1,

e2.EmployeeName AS emp2,

e1.JobTitle

FROM

Employees e1

JOIN

Employees e2

ON

e1.JobTitle = e2.JobTitle AND e1.EmployeeID < e2.EmployeeID;

--77. Find customers who live in the same city.

SELECT \* FROM Customers;

SELECT

c1.CustomerName AS cus1,

c2.CustomerName AS cus2,

c1.City

FROM

Customers c1

JOIN

Customers c2

ON

c1.City = c2.City AND c1.CustomerID < c2.CustomerID;

--78. List products with the same price.

SELECT \* FROM Products;

SELECT

p1.ProductName AS product1,

p2.ProductName AS product2,

p1.price

FROM

Products p1

JOIN

Products p2

ON

p1.price = p2.price AND p1.ProductID < p2.ProductID;

--79. Find employees working in the same department.

SELECT \* FROM Employees;

SELECT

e1.EmployeeName AS Employee1,

e2.EmployeeName AS Employee2,

e1.DepartmentID

FROM

Employees e1

JOIN

Employees e2

ON

e1.DepartmentID = e2.DepartmentID AND e1.EmployeeID < e2.EmployeeID;

--80. Display employees with the same hire date.

SELECT \* FROM Employees;

SELECT

e1.EmployeeName AS Employee1,

e2.EmployeeName AS Employee2,

e1.HireDate

FROM

Employees e1

JOIN

Employees e2

ON

e1.HireDate = e2.HireDate AND e1.EmployeeID < e2.EmployeeID;

**– ## Advanced Join Practice (81–100)**

--81. Get the top 3 customers with most orders using JOIN.

SELECT \* FROM Orders;

SELECT \* FROM Customer;

SELECT TOP 3

c.customer\_id,

c.name,

COUNT(o.OrderID) AS total\_orders

FROM

Customer c

JOIN

Orders o ON c.customer\_id = o.CustomerID

GROUP BY

c.customer\_id, [c.name](http://c.name);

--82. List each department with the number of employees.

SELECT \* FROM Employees;

SELECT DepartmentID, COUNT(\*) AS employee\_count FROM Employees

GROUP BY DepartmentID;

--83. Show total sales by product category.

ALTER TABLE Sales

ADD ProductID INT;

ALTER TABLE Sales

ADD Quantity INT;

UPDATE Sales

SET ProductID = 1 , Quantity = 2

WHERE sale\_id = 101;

UPDATE Sales

SET ProductID = 2, Quantity = 1

WHERE sale\_id = 102;

SELECT \* FROM Sales;

SELECT \* FROM Products;

SELECT \* FROM Categories;

SELECT

C.CategoryName,

SUM (P.Price \* S.Quantity) AS total\_sales

FROM

Sales S

JOIN

Products P ON S.ProductID = P.ProductID

JOIN

Categories C ON P.CategoryID = C.CategoryID

GROUP BY

C.CategoryName;

--84. Find the most expensive product in each category.

SELECT \* FROM Categories;

SELECT \* FROM Products;

SELECT

C.CategoryName,C.CategoryID, P.Price,

P.ProductName AS expensive\_product

FROM

Products P

JOIN

Categories C ON P.CategoryID = C.CategoryID

WHERE

P.Price = (

SELECT MAX(P2.Price)

FROM Products P2

WHERE P2.CategoryID = P.CategoryID );

--85. Get all orders with product names and total price.

SELECT \* FROM OrderDetails;

SELECT \* FROM Products;

SELECT

o.OrderID, P.ProductName, o.Quantity,

(o.Quantity \* p.Price) AS TotalPrice

FROM

OrderDetails o

JOIN

Products P ON o.ProductID = p.ProductID;

--86. Display each student’s average grade.

CREATE TABLE Grades (

GradeID INT PRIMARY KEY,

StudentID INT,

Subject VARCHAR(50),

Grade DECIMAL(5,2),

FOREIGN KEY (StudentID) REFERENCES Students(StudentID)

);

INSERT INTO Grades VALUES

(1, 1, 'Math', 85.5),

(2, 1, 'Science', 90.0),

(3, 2, 'Math', 70.0),

(4, 2, 'Science', 75.5),

(5, 3, 'Math', 92.0);

SELECT \* FROM Students;

SELECT \* FROM Grades;

SELECT

s.StudentID,

s.StudentName,

AVG(g.Grade) AS AverageGrade

FROM

Students s

JOIN

Grades g ON s.StudentID = g.StudentID

GROUP BY

s.StudentID, s.StudentName;

--87. List all authors and the number of books they've written.

SELECT \* FROM Author;

SELECT \* FROM Books;

SELECT

a.author\_id,

a.name,

COUNT(b.BookID) AS BookCount

FROM

Author a

LEFT JOIN

Books b ON a.author\_id = b.AuthorID

GROUP BY

a.author\_id, [a.name](http://a.name);

--88. Get customers who ordered more than 5 times.

SELECT \* FROM Customers;

SELECT \* FROM Orders;

SELECT

c.CustomerID,

c.CustomerName,

COUNT(o.OrderID) AS OrderCount

FROM

Customers c

JOIN

Orders o ON c.CustomerID = o.CustomerID

GROUP BY

c.CustomerID, c.CustomerName;

--89. List customers who never placed an order.

SELECT \* FROM Customers;

SELECT \* FROM Orders;

SELECT

c.CustomerID,

c.CustomerName

FROM

Customers c

LEFT JOIN

Orders o ON c.CustomerID = o.CustomerID

WHERE

o.OrderID IS NULL;

--90. Show products not ordered in the last 6 months.

SELECT \* FROM Products;

SELECT \* FROM Orders;

SELECT \* FROM OrderDetails;

SELECT

p.ProductID,

p.ProductName

FROM

Products p

JOIN (

SELECT DISTINCT od.ProductID

FROM OrderDetails od

JOIN Orders o ON od.OrderID = o.OrderID

WHERE o.OrderDate >= DATEADD(MONTH, -6, GETDATE())

) recent\_orders

ON p.ProductID = recent\_orders.ProductID

WHERE recent\_orders.ProductID IS NULL;

--91. Find employees with the same manager.

SELECT \* FROM Employees;

SELECT

e1.EmployeeName AS Employee1,

e2.EmployeeName AS Employee2,

e1.ManagerID

FROM

Employees e1

JOIN

Employees e2 ON e1.ManagerID = e2.ManagerID

WHERE

e1.EmployeeID < e2.EmployeeID;

--92. List pairs of students enrolled in the same course.

INSERT INTO Enrollments (EnrollmentID, StudentID, CourseID, Grade)

VALUES

(3, 1, 1, 'A'),

(4, 2, 2, 'B');

SELECT \* FROM Enrollments;

SELECT

e.StudentID AS student1,

e2.StudentID AS student2,

e.CourseID

FROM Enrollments e

join Enrollments e2 ON e.CourseID = e2.CourseID

WHERE

e.StudentID < e2.StudentID;

--93. Display products with no matching supplier or category.

SELECT \* FROM Products;

SELECT \* FROM Categories;

SELECT P.ProductName, C.CategoryName

FROM Products P

JOIN Categories C ON P.CategoryID = C.CategoryID

WHERE

C.CategoryID IS NULL;

--94. Get the number of songs in each album.

SELECT \* FROM songs;

SELECT \* FROM albums;

SELECT

a.album\_name,

COUNT(s.song\_id) AS song\_count

FROM

albums a

JOIN

songs s ON a.album\_id = s.album\_id

GROUP BY

A.album\_name;

--95. Find stores with below-average number of products.

ALTER TABLE Products

ADD StoreID INT;

UPDATE Products

SET StoreID = 1

WHERE ProductID = 1;

UPDATE Products

SET StoreID = 2

WHERE ProductID = 2;

UPDATE Products

SET StoreID = 3

WHERE ProductID = 3;

SELECT \* FROM Stores;

SELECT \* FROM Products;

WITH product\_count AS (

SELECT StoreID, COUNT(\*) AS total\_products

FROM products

GROUP BY StoreID

),

average\_count AS (

SELECT AVG(total\_products) AS avg\_products

FROM product\_count

)

SELECT s.StoreID, s.StoreName, pc.total\_products

FROM stores s

JOIN product\_count pc ON s.StoreID = pc.StoreID,

average\_count ac

WHERE pc.total\_products < ac.avg\_products;

--96. Show projects with no tasks assigned.

CREATE TABLE tasks (

task\_id INT PRIMARY KEY,

task\_name VARCHAR(100),

ProjectID INT,

FOREIGN KEY (ProjectID) REFERENCES projects(ProjectID)

);

INSERT INTO tasks (task\_id, task\_name, ProjectID) VALUES

(1, 'Design UI', 1),

(2, 'Backend API', 1),

(3, 'Create Database', 2),

(4, 'User Login', 2);

SELECT \* FROM Projects;

SELECT \* FROM tasks;

SELECT

P.ProjectName, T.task\_name

FROM Projects P

JOIN tasks T ON P.ProjectID = T.ProjectID

WHERE T.task\_name IS NULL;

--97. Get number of reviews per product.

SELECT \* FROM Reviews;

SELECT \* FROM Products;

SELECT

p.ProductID,

p.ProductName,

COUNT(r.ReviewID) AS review\_count

FROM

products p

LEFT JOIN

reviews r ON p.ProductID = r.ProductID

GROUP BY

p.ProductID, p.ProductName;

--98. List teachers who teach in more than one department.

CREATE TABLE teaches (

teacher\_id INT,

department\_id INT,

FOREIGN KEY (teacher\_id) REFERENCES teacher(teacher\_id),

);

CREATE TABLE depts (

dept\_id INT PRIMARY KEY,

dept\_name VARCHAR(100)

);

INSERT INTO teaches (teacher\_id, department\_id) VALUES

(1, 101),

(1, 102),

(2, 101),

(2, 103);

INSERT INTO depts (dept\_id, dept\_name) VALUES

(101, 'Computer Science'),

(102, 'Mathematics'),

(103, 'Physics');

SELECT \* FROM depts;

SELECT \* FROM Teacher;

SELECT \* FROM teaches;

SELECT

T.name, count(DISTINCT te.department\_id) AS NUM

FROM Teacher T

JOIN teaches te ON T.teacher\_id = te.teacher\_id

JOIN depts D ON D.dept\_id = te.department\_id

GROUP BY

T.name

HAVING count(te.department\_id) > 1;

--99. Show the latest order for each customer.

SELECT \* FROM Orders;

SELECT \* FROM Customers;

SELECT

c.CustomerID,

c.CustomerName,

o.OrderID,

o.OrderDate

FROM

customers c

JOIN

orders o ON c.CustomerID = o.CustomerID

WHERE

o.OrderDate = (

SELECT MAX(OrderDate)

FROM orders

WHERE CustomerID = c.CustomerID);

--100. Get all items with their category, vendor, and inventory status.

CREATE TABLE items (

item\_id INT PRIMARY KEY,

item\_name VARCHAR(100),

category\_id INT,

vendor\_id INT

);

INSERT INTO items (item\_id, item\_name, category\_id, vendor\_id) VALUES

(1, 'Laptop', 1, 1),

(2, 'Chair', 2, 2),

(3, 'T-Shirt', 3, 3);

ALTER TABLE inventory

ADD inventory\_status VARCHAR(50);

UPDATE Inventory

SET inventory\_status = 'In Stock'

WHERE ItemID = 601;

UPDATE Inventory

SET inventory\_status = 'Out of Stock'

WHERE ItemID = 602;

UPDATE Inventory

SET ItemID = 1

WHERE ItemID = 601;

UPDATE Inventory

SET ItemID = 2

WHERE ItemID = 602;

SELECT \* FROM items;

SELECT \* FROM Categories;

SELECT \* FROM Vendors;

SELECT \* FROM Inventory;

SELECT

i.item\_id, i.item\_name, C.CategoryName,V.VendorName, inv.inventory\_status

FROM items i

FULL OUTER JOIN Categories C ON i.category\_id = C.CategoryID

FULL OUTER JOIN Vendors V ON i.vendor\_id = V.VendorID

FULL OUTER JOIN Inventory inv ON i.item\_id = inv.ItemID;