Got it — I'll give you:

1. A minimal MySQL database schema and data so you can run and test it.
2. Scenarios where each type of SQL join is relevant.

---

1. Minimal RDBMS Schema (MySQL)

```sql
-- Create tables
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(50),
    city VARCHAR(50)
);

CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    product VARCHAR(50),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);

-- Insert minimal data
INSERT INTO Customers (customer_id, name, city) VALUES
(1, 'Alice', 'New York'),
(2, 'Bob', 'Los Angeles'),
(3, 'Charlie', 'Chicago');

INSERT INTO Orders (order_id, customer_id, product) VALUES
(101, 1, 'Laptop'),
(102, 1, 'Mouse'),
(103, 2, 'Keyboard');
```

---

2. Join Types & Scenarios

a) INNER JOIN

Scenario: Find customers who have placed at least one order.

```sql
SELECT c.name, o.product
FROM Customers c
INNER JOIN Orders o ON c.customer_id = o.customer_id;
```

Why use it: Only returns records where there's a match in both tables. Result:

name	product
Alice	Laptop
Alice	Mouse
Bob	Keyboard

---

b) LEFT JOIN (LEFT OUTER JOIN)

Scenario: List all customers and their orders, even if they haven't ordered yet.

```
SELECT c.name, o.product
FROM Customers c
LEFT JOIN Orders o ON c.customer_id = o.customer_id;
```

Why use it: Shows all records from the left table (Customers) even if there's no match in Orders. Result:

name	product
Alice	Laptop
Alice	Mouse
Bob	Keyboard
Charlie	NULL

---

c) RIGHT JOIN (RIGHT OUTER JOIN)

Scenario: List all orders and include customer names if available.

```
SELECT c.name, o.product
FROM Customers c
RIGHT JOIN Orders o ON c.customer_id = o.customer_id;
```

Why use it: Shows all records from the right table (Orders) even if there's no match in Customers (useful if some orders are from deleted/unregistered customers). Result: Same as INNER JOIN here because all orders have customers, but in real scenarios it may show NULL for missing customers.

---

d) FULL OUTER JOIN (Not directly in MySQL — simulate with UNION)

Scenario: Show all customers and all orders, matching where possible.

```sql
SELECT c.name, o.product
FROM Customers c
LEFT JOIN Orders o ON c.customer_id = o.customer_id
UNION
SELECT c.name, o.product
FROM Customers c
RIGHT JOIN Orders o ON c.customer_id = o.customer_id;
```

Why use it: Combines LEFT and RIGHT JOIN results to get everything.

---

e) CROSS JOIN

Scenario: Generate all possible combinations of customers and orders (like a marketing "who could buy what" scenario).

```sql
SELECT c.name, o.product
FROM Customers c
CROSS JOIN Orders o;
```

Why use it: Cartesian product — no matching condition.

---

f) SELF JOIN

Scenario: Show customers in the same city.

```sql
SELECT a.name AS customer1, b.name AS customer2, a.city
FROM Customers a
JOIN Customers b ON a.city = b.city AND a.customer_id < b.customer_id;
```

Why use it: Comparing a table to itself for relationships.
