**1. Banking – Fund Transfer Procedure**

* Write a **procedure** transfer\_funds(p\_from\_acc, p\_to\_acc, p\_amount)
* Deduct amount from sender, add to receiver.
* If balance is insufficient then all transactions should be rollbacked.
* Insert a log entry into transaction\_history.

**Table Structure**

1. CREATE TABLE accounts (acc\_no INT PRIMARY KEY, cust\_name VARCHAR(50), balance INT);
2. CREATE TABLE transaction\_history (txn\_id INT AUTO\_INCREMENT PRIMARY KEY, from\_acc INT, to\_acc INT, amount INT, txn\_date DATE DEFAULT CURDATE);

**2. E-commerce – Order Placement**

* Create a **procedure** place\_order(p\_customer\_id, p\_product\_id, p\_qty)
* Check if stock is available.
* If yes, reduce stock, insert into orders, and calculate bill with 18% GST.
* If not, raise an exception.

**Table Structure**

1. CREATE TABLE products (product\_id INT PRIMARY KEY, product\_name VARCHAR(50), price INT, stock INT);
2. CREATE TABLE orders (order\_id INT AUTO\_INCREMENT PRIMARY KEY, customer\_id INT, product\_id INT, qty INT, total\_price INT, order\_date DATE DEFAULT curdate());

**2.1. E-commerce**

* Assign random discounts between 5% and 30% whenever a product is added.
* Whenever a new order is placed, assign a random cashback (0 to ₹100).

**3. Healthcare – Patient Billing Function**

* Write a **function** calc\_bill(p\_patient\_id)
* Fetches **doctor fee** and **medicine fee** from the treatment table.
* Checks **insurance** status from the patients table.
* Applies an **insurance discount** (say 20% if insurance = 'YES').
* Returns the **final payable amount**.

**Table Structure**

1. CREATE TABLE patients (patient\_id INT PRIMARY KEY, name VARCHAR(50), insurance VARCHAR(3) CHECK (insurance IN ('YES','NO')));
2. CREATE TABLE treatment (treatment\_id INT PRIMARY KEY, patient\_id INT, doctor\_fee INT, medicine\_fee INT);

**3.1. Returns the sum of doctor and medicine fees for a patient**

* Write a **function** total\_treatment\_cost(p\_patient\_id INT)

**3.2. Returns the insurance coverage amount (20% of total bill if insured, else 0).**

* Write a **function** insurance\_coverage(p\_patient\_id)

**3.3. Returns a status message: "Insured" or "Not Insured".**

* Write a **function** patient\_status(p\_patient\_id INT)

**3.4. Returns the patient\_id of the patient with highest bill.**

* Write a **function** highest\_bill\_patient()

**3.5. Returns total medicine expense for a patient.**

* Write a **function** medicine\_expense(p\_patient\_id INT)

**3.6. Returns total doctor fee expense for a patient.**

* Write a **function** doctor\_expense(p\_patient\_id INT)

**3.7. Returns final bill including tax.**

* Write a **function** net\_payable\_with\_tax(p\_patient\_id INT, tax\_rate DECIMAL(5,2))

**4. HR – Payroll Processing**

* Write a **procedure** process\_salary(p\_emp\_id)
* Fetch basic, allowances, deductions.
* Calculate net salary.
* Insert salary slip record in payroll\_history.

**Table Structure**

1. CREATE TABLE employees (emp\_id INT PRIMARY KEY, name VARCHAR2(50), basic INT, Allowance INT, deduction INT);
2. CREATE TABLE payroll\_history (payroll\_id INT AUTO\_INCREMENT PRIMARY KEY, emp\_id INT, net\_salary INT, payroll\_date DATE DEFAULT CURDATE());

**5. Travel – Ticket Booking System**

* Create a **procedure** book\_ticket(p\_passenger, p\_train\_id)
* Check seat availability.
* If available, assign seat and confirm booking.
* If not, put passenger in a waiting list.

**Table Structure**

1. CREATE TABLE trains (train\_id INT PRIMARY KEY, train\_name VARCHAR(50), total\_seats INT, booked\_seats INT DEFAULT 0);
2. CREATE TABLE tickets (ticket\_id INT AUTO\_INCREMENT PRIMARY KEY, passenger\_name VARCHAR(50), train\_id INT, seat\_no INT, booking\_date DATE DEFAULT CURDATE(), status VARCHAR(20) DEFAULT 'CONFIRMED');
3. CREATE TABLE waiting\_list (wait\_id INT AUTO\_INCREMENT PRIMARY KEY, passenger\_name VARCHAR(50), train\_id INT, request\_date DATE DEFAULT CURDATE());

**6. Inventory – Apply Bulk Discount**

* Procedure apply\_discount(p\_category, p\_percent)
* Apply 5% discount to all products in a given category.
* Log the update into discount\_history.

**Table Structure**

1. CREATE TABLE product\_catalog (product\_id INT PRIMARY KEY, category VARCHAR(30), price INT);
2. CREATE TABLE discount\_history (hist\_id INT AUTO\_INCREMENT PRIMARY KEY, category VARCHAR(30), percent INT, updated\_on DATE DEFAULT CURDATE());

**7. Banking – Fixed Deposit Interest**

* Procedure calculate\_fd\_interest(p\_fd\_id)
* Compute maturity amount = principal + (principal \* rate \* years).
* Insert into fd\_history.

**Table Structure**

1. CREATE TABLE fixed\_deposit (fd\_id INT PRIMARY KEY, customer\_id INT, principal INT, rate INT, years INT);
2. CREATE TABLE fd\_history (hist\_id INT AUTO\_INCREMENT PRIMARY KEY, fd\_id INT, maturity\_amount INT, calc\_date DATE DEFAULT CURDATE());

**8.0.1. Online Shopping – Cancel Order**

* Write a **procedure** cancel\_order(p\_order\_id)
* Update status to “CANCELLED”.
* Add cancelled quantity back to stock and
* Insert into order\_log

**8.0.2. Place a New Order**

* Write a **procedure** place\_order(IN p\_order\_id INT, IN p\_product\_id INT, IN p\_qty INT)
* Update status to 'PLACED'.
* Reduce stock quantity and
* Insert into order\_log

**8.0.3. Update Order Quantity**

* Write a **procedure** update\_order\_qty(IN p\_order\_id INT, IN p\_new\_qty INT)
* Update the stock quantity and
* Update into order\_log

**8.0.4. Add New Product**

* Write a **procedure** add\_product(IN p\_product\_id INT, IN p\_name VARCHAR(50), IN p\_stock INT)

**8.0.5. Get All Orders for a Product**

* Write a **procedure** get\_orders\_for\_product(IN p\_product\_id INT)

**Table Structure**

1. CREATE TABLE shop\_orders (order\_id INT PRIMARY KEY, product\_id INT, qty INT, status VARCHAR(20) DEFAULT 'PLACED');
2. CREATE TABLE shop\_products (product\_id INT PRIMARY KEY, product\_name VARCHAR(50), stock INT);
3. CREATE TABLE order\_log (log\_id INT AUTO\_INCREMENT PRIMARY KEY, order\_id INT, action VARCHAR(20), log\_date DATE DEFAULT CURDATE());

**8.1. UDF to Check Status Before Cancelling (use the above tables)**

* Write a **function** check\_order\_status(p\_order\_id INT)
* status = 'CANCELLED' RETURN 'Order already cancelled'.
* status = 'PLACED' THEN RETURN 'Order can be cancelled'.

**8.2. UDF to Return Total Cancelled Orders for a Product**

* Write a **function** product\_cancel\_count(p\_product\_id INT)

**8.3. UDF to Get Product Stock After Cancellation**

* Write a **function** get\_stock(p\_product\_id INT)

**8.4. UDF to Return Total Cancelled Quantity of a Product**

* Write a **function** product\_cancelled\_qty(p\_product\_id INT)

**8.5.a. UDF to Return First Action of an Order**

* Write a **function** first\_order\_action(p\_order\_id INT)

**8.5.b. UDF to Return Last Action of an Order**

* Write a **function** last\_order\_action(p\_order\_id INT)

**8.6. UDF to Check if Product is Out of Stock**

* Write a **function** is\_out\_of\_stock(p\_product\_id INT)

**8.7. UDF to Get Current Order Status**

* Write a **function** get\_order\_status(p\_order\_id INT)

**8.8. UDF to Check if Order Can Be Cancelled, if order status is ‘Placed’**

* Write a **function** can\_cancel(p\_order\_id INT)

**8.9. UDF to Return Days Since Order Placement**

* Write a **function** days\_since\_order(p\_order\_id INT)

**9. University – Student Promotion**

* Write a p**rocedure** promote\_students(p\_course\_id)
* If marks ≥ 50 → promote to next semester.
* Update student\_academic table.

**Table Structure**

CREATE TABLE student\_academic (student\_id INT PRIMARY KEY, course\_id INT, semester INT, marks INT);

**9.1 University – Student**

* A stored procedure that assigns students to random batches (A/B/C)

**10. Hotel Management – Check-in**

* Write a p**rocedure** check\_in(p\_guest\_id, p\_room\_no)
* Update room status to “Occupied”.
* Insert record in checkin\_log.

**Table Structure**

1. CREATE TABLE rooms (room\_no INT PRIMARY KEY, status VARCHAR(20) DEFAULT 'Available');
2. CREATE TABLE checkin\_log (log\_id INT AUTO\_INCREMENT PRIMARY KEY, guest\_id INT, room\_no INT, checkin\_date DATE DEFAULT CURDATE());

**11. Inventory Stock Alert System**

* Scan all items.
* Check if stock < reorder\_level.
* Automatically insert a reorder request.

**Table Structure**

1. CREATE TABLE Inventory (inventory\_id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY, product\_id INT, quantity INT DEFAULT 0, reorder\_level INT DEFAULT 10, last\_updated DATE DEFAULT (CURRENT\_DATE), FOREIGN KEY (product\_id) REFERENCES Products(product\_id));
2. CREATE TABLE ReorderRequests (reorder\_id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY, inventory\_id INT, request\_date DATE DEFAULT CURRENT\_DATE, quantity\_requested INT, status VARCHAR(20) DEFAULT 'pending', FOREIGN KEY (inventory\_id) REFERENCES Inventory(inventory\_id));