Minutes of session Day 13

SQL Fundamentals	SQL commands
	Constraints
	Primary key
	Foreign key
	Types of constraints - Not null, Check, Unique.
	DDL Commands
	DCL & TCL Commands
	Grant & Revoke, Commit & Rollback

Case study based on above topics : College Course enrollment System(CES)

Following entities are there :

Students:

Courses:

Instructors:

Enrollments :

Scenario: ABC college wants to digitize their enrollment system so that :

1.Which students are enrolled in which courses.

2.Who is teaching each course ?

3.There should be rules like minimum student age, unique email IDs.

4. Allow only authorized users to make changes or view specific data.

5. Procedure to get student info by ID

Note: There should be a rollback option for a transaction if incorrect data is inserted.

A temporary read access to specific users like auditors.

Maintain a log table for all major operations.

Step 1: Create Database CollegeDB; and Use same database

Step 2: Create tables with constraints based on the above entities.

Step 3: Insert values in above tables

Step 4: Perform DCL(Grant & Revoke)

Step 5: Use TCL(Transaction control)

Step 6: Create a Logging table

Case Study 1: Hospital Patient Management System

Objective: Optimize patient data processing using advanced SQL features.

Step 1: Data Analysis Requirements

- Aggregate Functions: Calculate average treatment costs per department.
- Scalar Functions: Format patient phone numbers consistently (e.g., +1-XXX-XXX-XXXX).

Step 2: Data Relationships

- Inner Join: Link Patients ↔ Treatments to identify active cases.
- Left Join: List all doctors (even those without assigned patients).

Step 3: Automation

- Stored Procedure: Generate monthly reports with parameters for date ranges.
- Transaction: Ensure atomicity when transferring patients between wards.

Step 4: Error Handling

- TRY-CATCH: Rollback if a patient is assigned to a non-existent ward.

Key Hints:

- Use `SUM()` and `GROUP BY` for cost analysis.
- Create a scalar function to reformat contact details.
- Implement a transaction when updating multiple tables (e.g., patient transfer).

Case Study 2: E-Commerce Order Processing

Objective: Enhance order fulfillment with SQL automation.

Step 1: Business Rules

- Check Constraints: Validate order dates (no future dates).
- User-Defined Function: Calculate dynamic discounts (e.g., 10% for bulk orders).

Step 2: Data Consolidation

- Self-Join: Find customers who bought the same product twice.
- Union: Combine active/promotional product listings.

Step 3: Transaction Flow

1. Begin transaction on order placement.
2. Deduct inventory (with stock validation).
3. Commit only if payment succeeds; else rollback.

Step 4: Audit Trail

- Trigger: Log all order changes to an audit table.

Key Hints:

- Use `AVG()` and `COUNT()` for sales analytics.
- Design a table-valued function to fetch customer order history.
- Test isolation levels for concurrent order processing.

---

1. Topic Coverage:
    - Functions: Scalar (formatting), Aggregate (analytics), UDFs (business logic).
    - Joins: Inner (data matching), Left (optional relationships), Self (hierarchies).
    - Transactions: ACID compliance in critical operations.
2. VIVA Prep:
    - *Q: When to use a scalar vs. table-valued function?*
      A: Scalar for single-value transforms; table-valued for dataset returns.
    - *Q: How does isolation level impact inventory updates?*
      A: Higher isolation prevents overselling but may reduce concurrency.

Case Study 1: Hospital Patient Management System

Objective: Optimize patient data processing using SQL functions, joins, and transactions.

Table Structure Hints:

1. Patients Table
    - `patient_id` (PK)
    - `phone` (VARCHAR, use scalar function to format)
    - `ward_id` (FK to Wards table)
2. Treatments Table
    - `treatment_id` (PK)
    - `patient_id` (FK)
    - `cost` (DECIMAL, use `SUM()` for department totals)
3. Doctors Table
    - `doctor_id` (PK)
    - `department_id` (FK)

Constraints:

- `CHECK (cost > 0)` (Prevent negative values)
- `FOREIGN KEY (ward_id) REFERENCES Wards(ward_id)`

Key Steps:

1. Scalar Function: Format `phone` to `+1-XXX-XXX-XXXX`.
2. Aggregate Query: `AVG(cost) GROUP BY department_id`.
3. Self-Join: Find patients with multiple treatments.

FAQ:

Q: Why use a scalar function for phone formatting?

A: Ensures consistency across apps/reports without duplicating logic.

Q: Difference between `DELETE` and `TRUNCATE`?

Feature	`DELETE`	`TRUNCATE`
Speed	Slower (logs each row)	Faster (deallocates pages)
Where Clause	Supports `WHERE`	No filtering

Triggers	Fires triggers	No triggers

Case Study 2: E-Commerce Order Processing

Objective: Streamline orders with transactions and error handling.

Table Structure Hints:

1. Orders Table
    - `order_id` (PK)
    - `order_date` (`CHECK (order_date <= GETDATE())`)
2. OrderItems Table
    - `order_id` (FK, cascade delete)
    - `product_id` (FK)
    - `quantity` (`CHECK (quantity > 0)`)
3. Inventory Table
    - `product_id` (PK)
    - `stock` (Use `BEGIN TRANSACTION` for updates)

Constraints:

- `UNIQUE (product_id, order_id)` (Prevent duplicate items).

Key Steps:

1. Stored Procedure: Place orders with `TRY-CATCH` rollback.
2. Table-Valued Function: Fetch order history by customer.
3. Cross-Join: Generate promo codes for all products.

FAQ:

Q: When to use a stored procedure vs. function?

Feature	**Stored Procedure**	**Function**
Transactions	Supports `BEGIN/COMMIT`	Not allowed
Return	Optional	Mandatory

Usage	`EXEC Proc`	`SELECT dbo.Func()`

Q: Difference between `INNER JOIN` and `LEFT JOIN`?

- `INNER JOIN`: Only matching rows.
- `LEFT JOIN`: All left rows + matched right rows (NULLs if no match).

Summary Table: Key Differences

Topic	**Comparison Points**
Joins	`INNER` (matches) vs `LEFT` (all left + matches)
Indexes	Clustered (physical order) vs Non-clustered (logical order)
ACID	Atomicity (all-or-nothing) vs Isolation (concurrency control)

Action Items:

1. Implement hospital DB with phone-formatting function.
2. Simulate e-commerce order failure/rollback scenarios.

Case Study: Inventory Management System with Advanced SQL Features

1. Introduction

This case study explores the application of Advanced Queries, Functions, Joins, Stored Procedures, and Transactions in an Inventory Management System for an e-commerce company, *TechGadgets Inc*.

Business Problem

TechGadgets Inc. needs an efficient database system to:

- Track product inventory levels.
- Calculate discounts dynamically.
- Generate sales reports.
- Ensure data consistency during bulk updates.

---

2. User Stories

1. As a Sales Manager, I want to view total sales per product category to analyze performance.
2. As a Warehouse Supervisor, I need to check low-stock items to initiate restocking.
3. As a Customer Support Agent, I need to fetch order details with customer information for issue resolution.
4. As a Database Administrator, I need to ensure atomicity when updating stock levels during bulk orders.

---

3. Database Schema

Tables:

- Products (`ProductID`, `ProductName`, `CategoryID`, `Price`, `StockQuantity`)
- Categories (`CategoryID`, `CategoryName`)
- Orders (`OrderID`, `CustomerID`, `OrderDate`, `TotalAmount`)
- OrderDetails (`OrderDetailID`, `OrderID`, `ProductID`, `Quantity`, `UnitPrice`)
- Customers (`CustomerID`, `CustomerName`, `Email`, `Phone`)