

Question 10: Control Flow Statements

Part A (5 marks):

Write a stored procedure **ProcessMonthlyFines** that:

1. Uses **WHILE** loop to process overdue loans (2 marks)
2. For each overdue loan, uses **IF-ELSE** to: (2 marks)
 - **IF** overdue ≤ 7 days: LateFee = 0.50 per day
 - **ELSE IF** overdue ≤ 14 days: LateFee = 1.00 per day
 - **ELSE IF** overdue ≤ 30 days: LateFee = 2.00 per day
 - **ELSE**: LateFee = 5.00 per day AND Status = 'Suspended'
3. Use **CONTINUE** for already processed loans (0.5 marks)
4. Use **BREAK** if processing more than 100 loans (0.5 marks)
5. Return summary: Total processed, Total fees, Suspended count

Part B (5 marks):

Create a stored procedure **CategorizeMembers** using **CASE**:

1. Use simple **CASE** to convert MemberType codes: (1.5 marks)
 - 'S' \rightarrow 'Student', 'F' \rightarrow 'Faculty', 'G' \rightarrow 'General', 'P' \rightarrow 'Premium'
 2. Use searched **CASE** to assign activity level: (2 marks)
 - 'Very Active': ≥ 10 loans in last 3 months
 - 'Active': 5-9 loans in last 3 months
 - 'Moderate': 1-4 loans in last 3 months
 - 'Inactive': 0 loans in last 3 months
 3. Use nested **CASE** for membership status: (1.5 marks)
 - Check IsActive, then check overdue loans, then check fines
 - Return: 'Active-Good', 'Active-Warning', 'Active-Suspended', 'Inactive'
 4. Display: Member Name, Type (full), Activity Level, Status, Total Loans
-

Question 8: Subqueries & Correlated Queries (10 Marks)

Part A (5 marks):

1. Simple Subqueries (2 marks):

- Members who borrowed more books than average
- Books priced higher than the average price in their genre
- Members from cities with more than 100 members

2. EXISTS/NOT EXISTS (2 marks):

- Books never borrowed (use NOT EXISTS)
- Members who have both loans AND fines (use EXISTS for both)
- Genres that have books but no active loans (use NOT EXISTS)

3. Correlated Subquery (1 mark):

- For each book, show how much more expensive it is than the average in its genre
- Members with above-average loan count compared to their city

Part B (5 marks):

Using Common Table Expressions (CTE):

1. Recursive CTE (3 marks): Create a recursive CTE that generates a date series for the last 12 months then:

- Join with Loans to show monthly loan counts
- Include months with zero loans
- Calculate running total of loans
- Show month-over-month growth percentage

2. Multiple CTEs (2 marks):

- CTE1: Calculate member statistics (total loans, fines, last activity)
 - CTE2: Calculate genre statistics (books count, avg price)
 - CTE3: Combine both to show members with their preferred genre (most borrowed)
 - Final query: Show comprehensive member profile with all statistics
-

Question 8: Subqueries

Part A (5 marks):

1. Simple Subqueries (2 marks):

- Members who borrowed more books than average
- Books priced higher than the average price in their genre
- Members from cities with more than 100 members

2. EXISTS/NOT EXISTS (2 marks):

- Books never borrowed (use NOT EXISTS)
- Members who have both loans AND fines (use EXISTS for both)
- Genres that have books but no active loans (use NOT EXISTS)

3. Correlated Subquery (1 mark):

- For each book, show how much more expensive it is than the average in its genre
- Members with above-average loan count compared to their city

Part B (5 marks):

Using Common Table Expressions (CTE):

1. Recursive CTE (3 marks): Create a recursive CTE that generates a date series for the last 12 months then:

- Join with Loans to show monthly loan counts
- Include months with zero loans
- Calculate running total of loans
- Show month-over-month growth percentage

2. Multiple CTEs (2 marks):

- CTE1: Calculate member statistics (total loans, fines, last activity)
 - CTE2: Calculate genre statistics (books count, avg price)
 - CTE3: Combine both to show members with their preferred genre (most borrowed)
 - Final query: Show comprehensive member profile with all statistics
-

Question 11: UNION & Set Operations [10 Marks]

Part A (4 marks):

Explain the difference between:

1. UNION vs UNION ALL (with performance implications) (1 mark)
2. INTERSECT and EXCEPT operations (with examples) (1 mark)
3. Write queries demonstrating all four operations using Library tables (2 marks)

Part B (6 marks):

Create a **Unified Communication List** using UNION:

1. Combine data from: (3 marks)
 - Active Members: Name, Email, Phone, 'Member' as ContactType, JoinDate as AssociatedDate, City
 - Staff: Name, Email, Phone, 'Staff' as ContactType, HireDate as AssociatedDate, Department as City
 - Publishers: PublisherName, ContactEmail, NULL as Phone, 'Publisher' as ContactType, NULL as Date, Country as City
 2. Requirements: (3 marks)
 - Remove duplicates based on Email
 - Sort by ContactType, then Name
 - Only contacts from last 3 years
 - Add a calculated column showing "days since association"
 - Group by ContactType and show count
 - Format output with proper column aliases
-

Question 12: Joins - Comprehensive [10 Marks]

Part A (6 marks):

1. INNER JOIN (2 marks): Create a detailed borrowing report:
 - Member: Name, Email, Type, City
 - Book: Title, Author, Genre, Price
 - Loan: LoanDate, DueDate, ReturnDate, Status
 - Calculate: Days borrowed, Late fee if any, Status (On Time/Late)

- Filter: Last 6 months only
- Sort: Latest loans first

2. **LEFT JOIN (2 marks):** List ALL books with borrowing statistics:

- Book: Title, Author, Genre, Price, Quantity
- Statistics: Times borrowed (0 if never), Last borrow date
- Category: 'Hot' (>20 borrows), 'Popular' (10-20), 'Average' (5-9), 'Slow' (1-4), 'Never' (0)
- Show inventory status: In Stock / Out of Stock

3. **CROSS JOIN (2 marks):** Generate a calendar report matrix:

- All genres crossed with all months of current year
- Show actual loan count for each combination
- Use LEFT JOIN after CROSS JOIN
- Display as: Genre | Jan | Feb | Mar | ... | Dec | Total

Part B (4 marks):

Create a **Master Library Dashboard** using multiple joins:

- Member information with activity summary
- Currently borrowed books (comma-separated list)
- Reservation details (if any)
- Fine summary (paid/unpaid amounts)
- Staff assigned to member (if premium member)
- Last 3 activities (loan dates or fine payments)
- Use appropriate joins (INNER, LEFT, RIGHT as needed)
- Include members even with no activity
- Show top 50 most active members

Question 13: Stored Procedures & Parameters [10 Marks]

Part A (5 marks):

Create a stored procedure **BorrowBook** with:

Parameters:

- (@)MemberID INT

BookID INT

@LoanDays INT = 14 (default)

@Status VARCHAR(50) OUTPUT

• @LoanID INT OUTPUT

Logic:

1. Validate member exists and is active (1 mark)
2. Validate book exists and is available (1 mark)
3. Check if member has overdue books (if yes, reject) (1 mark)
4. Create loan record with calculated due date (1 mark)
5. Update book AvailableQuantity (1 mark)
6. Return status message and new LoanID
7. Use proper error handling

Part B (5 marks):

Create a stored procedure **SearchBooks** with flexible parameters:

Parameters:

- @Title VARCHAR(200) = NULL
- @Author VARCHAR(100) = NULL
- @Genre VARCHAR(50) = NULL
- @MinPrice DECIMAL(10,2) = NULL
- @MaxPrice DECIMAL(10,2) = NULL
- @MinYear INT = NULL
- @MaxYear INT = NULL
- @SortBy VARCHAR(20) = 'Title' -- Title, Author, Price, Year
- @SortOrder VARCHAR(4) = 'ASC' -- ASC or DESC
- @PageNumber INT = 1
- @PageSize INT = 20

Requirements:

1. Build dynamic WHERE clause based on provided parameters (2 marks)

- Use comments for complex logic
- Assume proper database permissions

Database Schema

Library Management System with the following tables:

SQL

Members (MemberID, Name, Email, PhoneNumber, MemberType, JoinDate, City, IsActive)

Books (BookID, Title, Author, Genre, Price, PublicationYear, ISBN, Quantity, AvailableQuantity)

Loans (LoanID, MemberID, BookID, LoanDate, DueDate, ReturnDate, LateFee, Status)

Fines (FineID, LoanID, MemberID, FineAmount, FineDate, IsPaid, PaymentDate)

Staff (StaffID, FirstName, LastName, Email, Department, Salary, HireDate, IsActive)

Reservations (ReservationID, MemberID, BookID, ReservationDate, ExpiryDate, Status, Priority)

Publishers (PublisherID, PublisherName, Country, ContactEmail, EstablishedYear)

BookPublishers (BookID, PublisherID, PublishDate)

- Each question carries 10 marks
 - Write SQL queries clearly with proper formatting
 - Provide explanations where asked
 - Manage your time, approximately 12 minutes per question
 - Use comments for complex logic
 - Assume proper database permissions
-

Database Schema

Library Management System with the following tables:

sql

Members (MemberID, Name, Email, PhoneNumber, MemberType, JoinDate, City, IsActive)

Books (BookID, Title, Author, Genre, Price, PublicationYear, ISBN, Quantity, AvailableQuantity)

Loans (LoanID, MemberID, BookID, LoanDate, DueDate, ReturnDate, LateFee, Status)

Fines (FineID, LoanID, MemberID, FineAmount, FineDate, IsPaid, PaymentDate)

Staff (StaffID, FirstName, LastName, Email, Department, Salary, HireDate, IsActive)

Reservations (ReservationID, MemberID, BookID, ReservationDate, ExpiryDate, Status, Priority)

Publishers (PublisherID, PublisherName, Country, ContactEmail, EstablishedYear)

BookPublishers (BookID, PublisherID, PublishDate)

Question 10: Control Flow Statements [10 Marks]

Part A (5 marks):

Write a stored procedure **ProcessMonthlyFines** that:

1. Uses WHILE loop to process overdue loans (2 marks)
2. For each overdue loan, uses IF-ELSE to: (2 marks)
 - IF overdue ≤ 7 days: LateFee = 0.50 per day
 - ELSE IF overdue ≤ 14 days: LateFee = 1.00 per day
 - ELSE IF overdue ≤ 30 days: LateFee = 2.00 per day
 - ELSE: LateFee = 5.00 per day AND Status = 'Suspended'
3. Use CONTINUE for already processed loans (0.5 marks)
4. Use BREAK if processing more than 100 loans (0.5 marks)
5. Return summary: Total processed, Total fees, Suspended count

Part B (5 marks):

Create a stored procedure **CategorizeMembers** using CASE:

1. Use simple CASE to convert MemberType codes: (1.5 marks)
 - 'S' \rightarrow 'Student', 'F' \rightarrow 'Faculty', 'G' \rightarrow 'General', 'P' \rightarrow 'Premium'
2. Use searched CASE to assign activity level: (2 marks)
 - 'Very Active': ≥ 10 loans in last 3 months
 - 'Active': 5-9 loans in last 3 months
 - 'Moderate': 1-4 loans in last 3 months
 - 'Inactive': 0 loans in last 3 months
3. Use nested CASE for membership status: (1.5 marks)
 - Check IsActive, then check overdue loans, then check fines
 - Return: 'Active-Good', 'Active-Warning', 'Active-Suspended', 'Inactive'
4. Display: Member Name, Type (full), Activity Level, Status, Total Loans

5. Handle NULL parameters appropriately (1 mark)

Question 14: Cursors & Advanced Loops [10 Marks]

Part A (5 marks):

Explain:

1. What is a cursor and when should you use it? (1 mark)
2. Types of cursors and their differences (1 mark)
3. Performance implications of cursors vs set-based operations (1 mark)
4. Provide an example where cursor is necessary (2 marks)

Part B (5 marks):

Write a stored procedure **CalculateMonthlyStatistics** that:

1. Declares a cursor to loop through all active members (1 mark)
2. For each member, calculates: (2 marks)
 - Total books borrowed this month
 - Total fines paid this month
 - New/Renewed status
 - Sends email if overdue books exist
3. Updates a summary table **MemberMonthlyStats** (1 mark)
4. Uses proper cursor lifecycle: (1 mark)
 - DECLARE, OPEN, FETCH, CLOSE, DEALLOCATE
5. Includes error handling and cursor cleanup

Question 15: Error Handling, Views & Triggers [10 Marks]

Part A (4 marks):

Error Handling with TRY-CATCH:

Write a stored procedure **TransferBook** that:

Uses TRY-CATCH with proper error handling (2 marks)

3. Captures: ERROR_NUMBER, ERROR_MESSAGE, ERROR_LINE (1 mark)

4. Logs errors to an ErrorLog table (1 mark)

5. Uses THROW to re-raise errors

6. Returns appropriate status codes

Part B (3 marks):

Create Views:

1. **ActiveLoansView** (1 mark):

- Shows current active loans with member and book details
- Include calculated "days remaining" column
- Filter only active (not returned) loans

2. **MemberDashboardView** (1 mark):

- Member summary with stats (total loans, active loans, fines)
- Include computed columns for member category
- Encrypt view definition using WITH ENCRYPTION

3. **PopularBooksView** (1 mark):

- Top 100 most borrowed books
- Include ranking and borrowing statistics
- Use indexed view for performance (WITH SCHEMABINDING)

Part C (3 marks):

Create Triggers:

1. **Trigger on Loans table** (1.5 marks):

- AFTER INSERT trigger
- Automatically decrements AvailableQuantity in Books
- Creates audit entry in LoanHistory
- Validates business rules (member not suspended, book available)

2. **Trigger on Books table** (1.5 marks):

- INSTEAD OF DELETE trigger

- Write queries for Books table showing total price with and without handling nulls.

Part B (5 marks):

1. **Date Conversions (2 marks):** Convert current datetime to:

- YYYY-MM-DD HH:MI:SS
- Month DD, YYYY
- DD/MM/YYYY HH:MI AM/PM
- Weekday, Month DD, YYYY

2. **Safe Type Conversions (2 marks):**

- Convert Price to INT safely, show 'Invalid' for conversion failures
- Convert PublicationYear to DATE (assume Jan 1st), handle invalid year
- Use TRY_CAST and TRY_CONVERT appropriately

3. **COLLATE Operations (1 mark):**

- Case-sensitive search: Books with exact title "SQL Server"
- Case-insensitive search: Members with name containing "john"

2. **Advanced Sorting (2 marks):**

- Top 10 most expensive books with RANK, DENSE_RANK, and ROW_NUMBER
- Show difference between these ranking functions
- Books sorted by Genre, then Price DESC, then Title ASC

3. **Pattern Matching (2 marks):**

- Books with titles starting with 'The' or ending with 'Programming'
- Emails with exactly 3 characters before @ symbol
- Phone numbers in format XXX-XXX-XXXX or (XXX) XXX-XXXX

Part B (4 marks):

Create a comprehensive query with pagination:

- Show books with detailed information
 - Filter: Genre in ('Programming', 'Database', 'Science'), Price > 400
 - Sort by Price DESC, Title ASC
 - Implement pagination: Page size = 10, show page 2
 - Include total count of matching records
 - Use OFFSET-FETCH for pagination
-

1. **Basic Aggregation (2 marks):**

- Genre-wise: Count of books, Average price, Min/Max price, Total value (Price * Quantity)
- City-wise member distribution: Count, Percentage of total
- Monthly loan statistics for current year: Month, Total loans, Unique members

2. **Advanced Grouping with GROUPING SETS (2 marks):**

- Write a query using GROUPING SETS to show book statistics for:
 - Books grouped by Genre only
 - Books grouped by Author only
 - Books grouped by both Genre AND Author together
 - Grand total (all books)
- Display: Grouping columns, COUNT of books, AVG price, SUM of total value
- Use GROUPING() function to identify which rows are subtotals

3. **HAVING Clause (2 marks):**

- Genres with more than 10 books and average price > 500
- Members who borrowed more than 5 books and have paid fines > 100
- Authors with at least 3 books where cheapest book > 300

Part B (4 marks):

Create a comprehensive analytics report:

- Show MemberType statistics: Total members, Active percentage, Avg loans per member, Total fines collected
 - Include members with at least 1 loan
 - Having total fines > 50 OR active loans > 3
 - Sort by total loans descending
 - Include grand total row using ROLLUP
 - Format currency values with \$ symbol
-

BookID INT

@LoanDays INT = 14 (default)

@Status VARCHAR(50) OUTPUT

- @LoanID INT OUTPUT

Logic:

1. Validate member exists and is active (1 mark)
2. Validate book exists and is available (1 mark)
3. Check if member has overdue books (if yes, reject) (1 mark)
4. Create loan record with calculated due date (1 mark)
5. Update book AvailableQuantity (1 mark)
6. Return status message and new LoanID
7. Use proper error handling

Part B (5 marks):

Create a stored procedure **SearchBooks** with flexible parameters:

Parameters:

- @Title VARCHAR(200) = NULL
- @Author VARCHAR(100) = NULL
- @Genre VARCHAR(50) = NULL
- @MinPrice DECIMAL(10,2) = NULL
- @MaxPrice DECIMAL(10,2) = NULL
- @MinYear INT = NULL
- @MaxYear INT = NULL
- @SortBy VARCHAR(20) = 'Title' -- Title, Author, Price, Year
- @SortOrder VARCHAR(4) = 'ASC' -- ASC or DESC
- @PageNumber INT = 1
- @PageSize INT = 20

Requirements:

1. Build dynamic WHERE clause based on provided parameters (2 marks)
2. Implement dynamic sorting (1 mark)

Question 4: String, Date & Numeric Functions [10 Marks]

Part A (6 marks):

Write SQL queries using appropriate functions:

1. Display member names with first letter of first and last name capitalized, email in lower case.
 - Example: "john doe" → "John Doe"
2. Extract and display: (1.5 marks)
 - First 3 characters of ISBN
 - Last 4 characters of phone numbers as masked: "XXX-XXX-1234"
 - Length of each book title
3. Calculate for each loan: (1.5 marks)
 - Days between loan date and due date (loan period)
 - Days between loan date and return date (actual days borrowed)
 - If not returned, days between loan date and today
4. For all books, show: (1.5 marks)
 - Price rounded to nearest 10
 - Price with 2 decimal places
 - Absolute difference between original and rounded price

Part B (4 marks):

Create a stored procedure **FormatMemberReport** that:

- Takes @MemberID as parameter
- Returns formatted output showing:
 - Full name in title case

- Write queries for Books table showing total

Part B (5 marks):

1. Date Conversions (2 marks): Convert current datetime to:

- YYYY-MM-DD HH:MI:SS
- Month DD, YYYY
- DD/MM/YYYY HH:MI AM/PM
- Weekday, Month DD, YYYY

2. Safe Type Conversions (2 marks):

- Convert Price to INT safely, show 'Invalid' for conversion failures
- Convert PublicationYear to DATE (assume Jan 1st), handle invalid y
- Use TRY_CAST and TRY_CONVERT appropriately

3. COLLATE Operations (1 mark):

- Case-sensitive search: Books with exact title "SQL Server"
- Case-insensitive search: Members with name containing "john"

Feature	Temporary Table	Table Variable
Storage Location	?	?
Scope	?	?
Indexing	?	?
Statistics	?	?
Transaction Rollback	?	?
Performance (small data)	?	?
Performance (large data)	?	?
When to Use	?	?

Provide 2 examples of when each is most appropriate.

Part B (6 marks):

Write a stored procedure **GenerateAnnualReport** that:

- Creates a temp table #QuarterlyStats with: (2 marks)
 - Quarter (Q1, Q2, Q3, Q4)
 - TotalLoans, UniqueMembers, Revenue, NewMembers
 - AvgBooksPerMember, PopularGenre
- Populates it with data for current year (2 marks)
- Uses a table variable @Comparison to compare with last year (1 mark)
- Adds calculated columns: (1 mark)
 - YoY Growth Percentage
 - Quarter Rank by Revenue
 - Running total of loans
- Returns results with formatted output and cleans up (1 mark)