SIMS: Project Phase 3 Report

SIMS (Society Internal Management System) is implemented using the .NET Core architecture (UI → BLL → Data). The UI communicates only with a shared service interface (**InterfaceSimsService**). Using the Factory Design Pattern (**SimsServiceFactory**), our application selects one of two BLL implementations.

SQL Feature Implementation:

1) Stored Procedures

- We use stored procedures to implement operations such as event creation, member registration, expense insertion, and announcement broadcasting. These procedures enforce business rules such as event capacity limits and budget availability checks, and use transactions for atomicity and consistency. They are invoked directly by the application when running in Stored Procedure backend mode. **(sp_RegisterMemberForEvent, sp_AddExpenseWithBudgetCheck, sp_CreateEventWithBudget, sp_CreateAnnouncementAndNotify)**

2) User-Defined Functions

- User-defined scalar functions are used in our application for computing total event expenses, member attendance rates, and participation counts by status. These functions are used within views and reporting queries to provide calculations that are available in the UI. **(fn_GetEventTotalExpenses, fn_GetMemberAttendanceRate, fn_GetEventParticipationCount)**

3) Triggers (AFTER and INSTEAD OF)

- We use AFTER triggers to automatically update budget usage and maintain audit logs when expenses are inserted or updated. Similarly, The INSTEAD OF DELETE trigger logs event deletions before removal, guaranteeing auditability when the delete operation is initiated by the application. (**tr_Expense_AfterInsert, tr_Expense_AfterUpdate, tr_Event_InsteadOfDelete**)

4) Common Table Expressions (CTEs)

- CTEs are used within stored procedures to generate reports such as month-wise attendance trends and budget utilization summaries. These CTEs aggregate historical data and return result sets that the application then uses in its UI **(sp_GetAttendanceTrend, sp_GetBudgetUtilizationSummary)**

5) Views

- We use views in our application for commonly used reports, including upcoming events, per-event financial summaries, and member participation statistics. The application queries these views directly to display reporting information without having to reimplement complex joins in business logic. **(v_UpcomingEvents, v_EventFinancialSummary, v_MemberParticipationSummary)**

6) Indexes (various types)

- Indexes (Covering and Filtered Nonclustered): The system uses covering nonclustered indexes (IX_Participation_Event_Status, IX_Expense_Budget_Date) to optimize event participation queries and financial reporting by eliminating table lookups, and a filtered nonclustered index (IX_Notification_Unread) to efficiently retrieve unread notifications. These indexes are used by our application to improve performance for registration tracking, budget analysis, and notification retrieval.

7) Table Partitioning

- We apply Table partitioning to time-based tables using a date-range partition scheme. This allows the system to scale efficiently as historical expense and audit data grow (our largest tables), while having fast insert and query performance for recent records. **(Partitioned on Expense, AuditLog)**