# Physical Data Design for Licensing Self-Certification Portal

## Introduction

This document outlines the physical data design for the Licensing Self-Certification Portal (LSCP) project. It provides a blueprint for implementing the LSCP database, ensuring a robust and efficient data management system. This document details the physical data structure, process design, and relationships between key business entities within the LSCP.

## Objectives

The objectives of the LSCP are to:

1. Provide user-friendly and meaningful messages to users.
2. Store all electronic and paper submissions from applicant and authorized person (AP)/registered structural engineer (RSE) applications for safety certificates, non-purpose built schools, child care centers, and non-local higher and professional education courses.
3. Provide a Buildings Department (BD) departmental portal login for internal users, with User ID and password as an alternative.
4. Support the latest web browsers.
5. Comply with the standards of the Government System Architecture and government IT security policy.

## Physical Data Structure Specification

This section details the data model and associated descriptions.

### Physical Data Structure

The entity-relationship diagram (ERD) illustrates the relationships between entities.

[Insert ERD Diagram Here - The provided image links are not accessible, so a placeholder is used.]

**Entities Overview:**

There are 7 categories of entities in the data model design:

* (GCIS) Frontend - Application Forms submission
* (GCIS) Frontend - OTP login control
* Backend - Users
* Backend - Workflow of Application Forms submission

### (GCIS) Frontend - Application Forms Submission

[Insert Diagram 3.1-1 Here - The provided image links are not accessible, so a placeholder is used.]

### (GCIS) Frontend - OTP Login Control

[Insert Diagram 3.1-2 Here - The provided image links are not accessible, so a placeholder is used.]

### (BD) Backend - TBC

[Insert Diagram 3.1-3 Here - The provided image links are not accessible, so a placeholder is used.]

## Data Entity Description

This section outlines the conversion rules, assumptions, physical data table names, corresponding system entities, and key details stored in the database.

**Database Management System:** Microsoft SQL Server 2019 is selected.

**Conversion Rules and Assumptions:**

* Spatial and textual entities are stored in Microsoft SQL Server 2019.
* The following tables document the mapping of the Logical Data Model (LDM) to the physical data design.

**LSCP Frontend Tables:**

| Table ID | LSCP Name | LSCP Entity Description | Key Nature | Key Data Item | N/A |----------|---------------------|-----------------------------------------------------------------------------|------------|------------------------------| N/A |---|---|---|---|---| N/A | T-S-01 | ApplicationCases | Stores application numbers | PK | Id | N/A | | N/A | | N/A | ApplicationNo | N/A |---|---|---|---|---| N/A | T-S-02 | SchoolApp\_Infos | Stores the latest update of submitted application data as a single row | PK | Id | N/A | | N/A | | N/A | ApplicationNo | N/A |---|---|---|---|---| N/A | T-S-03 | SchoolApp\_Submissions | Stores submissions for each application | PK | Id | N/A | | N/A | | N/A | ApplicationNo | N/A | | N/A | | N/A | SubmissionId | N/A |---|---|---|---|---| N/A | T-S-04 | ApplicationFiles | Stores the file paths of applicant uploads | PK | Id | N/A | | N/A | | N/A | ApplicationNo | N/A | | N/A | | N/A | SubmissionId | N/A |---|---|---|---|---| N/A | T-S-05 | LSCPMasterTable | Stores meta-data or parameter data for the frontend | PK | Id | N/A | | N/A | | N/A | Code | N/A | | N/A | | N/A | Type + Code | N/A |---|---|---|---|---| N/A | T-S-06 | GenOtp | Stores generated OTP codes and their usage status | PK | Id | N/A | | N/A | | N/A | ApplicationNo + userId + Otp | N/A |---|---|---|---|---| N/A | T-S-07 | AdrBlk | Stores addresses imported from BCIS | PK | ADR\_BLK\_ID | N/A |---|---|---|---|---| N/A | T-S-08 | SYS\_META\_DATA | Stores meta data imported from BCIS | PK | SYS\_META\_DATA\_ID | N/A | | N/A | | N/A | REC\_TYPE | N/A | | N/A | | N/A | CODE | N/A |---|---|---|---|---| N/A | T-S-09 | Aprse | Stores AP / RSE information imported from MWMS 2.0 | PK | Id | N/A | | N/A | | N/A | Name + RegistrationNumber | N/A |---|---|---|---|---| **LSCP Backend Tables:**

| Table ID | LSCP Name | LSCP Entity Description | Key Nature | Key Data Item | N/A |----------|---------------------|-----------------------------------------------------------------------------|------------|--------------------| N/A |---|---|---|---|---| N/A | T-S-01 | ApplicationCases | Stores application numbers | PK | Id | N/A | | N/A | | N/A | ApplicationNo | N/A |---|---|---|---|---| N/A | T-S-02 | SchoolApp\_Infos | Stores the latest update of submitted application data as a single row | PK | Id | N/A | | N/A | | N/A | ApplicationNo | N/A |---|---|---|---|---| N/A | T-S-03 | SchoolApp\_Submissions | Stores submissions for each application | PK | Id | N/A | | N/A | | N/A | ApplicationNo | N/A | | N/A | | N/A | SubmissionId | N/A |---|---|---|---|---| N/A | T-S-04 | ApplicationFiles | Stores the file paths of applicant upload files | PK | Id | N/A | | N/A | | N/A | ApplicationNo | N/A | | N/A | | N/A | SubmissionId | N/A |---|---|---|---|---| N/A | T-S-05 | LSCPMasterTable | Stores meta-data or parameter data for the frontend | PK | Id | N/A | | N/A | | N/A | Code | N/A | | N/A | | N/A | Type + Code | N/A |---|---|---|---|---| N/A | T-S-06 | SYS\_META\_DATA | Stores meta data imported from BCIS | PK | SYS\_META\_DATA\_ID | N/A | | N/A | | N/A | REC\_TYPE | N/A | | N/A | | N/A | CODE | N/A |---|---|---|---|---| N/A | T-S-07 | AdrBlkFileRef | Stores the relationship between ADR\_BLK and SYS\_FILE\_REF | PK | ADR\_BLK\_FILEREF\_ID | N/A | | N/A | | N/A | ADR\_BLK\_ID | N/A | | N/A | | N/A | SYS\_FILE\_REF\_ID | N/A |---|---|---|---|---|

## Database Schema Analysis and Optimization Recommendations

Based on the provided database schema and code, here are some observations and recommendations for physical data design:

**1. Collection Sizes and Indexing:**

* **Large Collections:** sysfilerefs (601808 documents, 204.70 MB) and adrblkfilerefs (566948 documents, 154.89 MB) are significantly larger than other collections. This suggests that queries against these collections could benefit significantly from proper indexing.
* **Indexing Recommendations:**
  + sysfilerefs: Consider indexing sysFileRefId, frefPref, frefSeq, frefYr, frefSuf. The choice of which fields to include in a compound index depends on the most common query patterns.
  + adrblkfilerefs: Index adrBlkFileRefId, adrBlkId, and sysFileRefId. Again, compound indexes based on common query patterns would be beneficial.
* **Collection Growth:** Monitor the growth of these large collections. If they continue to grow rapidly, consider data archiving or partitioning strategies.

**2. Data Types and Field Usage:**

* **Mixed Data Types:** The tasks and eminutes collections have fields with mixed data types (e.g., user in tasks is sometimes a string and sometimes an objectId). This can lead to inefficient queries and potential data integrity issues. Standardize the data types for these fields. Using objectId is generally preferred for relationships.
* **String vs. ObjectId:** In the tasks collection, the user field is sometimes a string and sometimes an objectId. This inconsistency should be resolved. If user represents a relationship to the users collection, it should consistently be an objectId.
* **Unused Collections:** The submissions collection is empty. If it's not being used, consider removing it. If it's intended for future use, ensure it's properly designed.
* **Redundant Data:** The Application schema stores address information in multiple ways (e.g., AddressOfPremiseEN, AddressOfPremiseCN, address object). Consolidate this information into a single, well-structured address object to avoid redundancy and improve data consistency.
* **String vs. Date:** The Case schema has fields like ReceivedDate, SubstantialReplyDate, and ActualReplyDate which are sometimes date and sometimes null. Ensure consistency.
* **Boolean Representation:** The Case schema uses ViaSCS as a boolean. Ensure that all boolean fields are consistently represented as booleans (true/false) and not as strings or other types.

**3. Relationships and Foreign Keys:**

* **Explicit Relationships:** Use explicit relationships (foreign keys) between collections to enforce data integrity and improve query performance. For example, the application field in the cases collection should be a Schema.Types.ObjectId referencing the Application collection.
* **Populate vs. Manual Joins:** The code frequently uses .populate() to retrieve related data. While convenient, this can be inefficient for large datasets. Consider using aggregation pipelines for more complex queries that require joining data from multiple collections.

**4. Code Review Findings:**

* **Inconsistent Naming Conventions:** The codebase uses a mix of camelCase and snake\_case for variable and property names. Establish a consistent naming convention and apply it throughout the codebase.
* **Hardcoded Values:** The code contains hardcoded values (e.g., client IDs, API endpoints, file paths). Move these values to environment variables or configuration files.
* **Error Handling:** The error handling in the route handlers is generally good, but consider adding more specific error messages and logging to aid in debugging.
* **Duplicated Code:** The syncFrontendSubmissions.js script contains a lot of logic that is similar to the applications.js route. Refactor this code to reduce duplication.
* **Security:** The auth.js route directly compares the password with the hashed password. Ensure that you are using bcryptjs.compareSync to compare the password.

**5. Specific Collection Recommendations:**

* **tasks:** Consider indexing the application and submissionCase fields for faster lookups.
* **eminutes:** Index the submissionCase field.
* **applications:** Index ApplicationNo, NameOfSchoolCN, NameOfSchoolEN, assignedBS, assignedGR. Consider a geospatial index on the address field if location-based queries are needed.
* **notifications:** Index user, task, and eminute.
* **bsblocks:** Index blockId and bdgis.
* **cases:** Index application, assignedBS, assignedGR. Consider indexing Category, Nature, Region, District, Area if these are frequently used in queries.
* **oauthtokens:** Index accessToken, refreshToken, and user.
* **attachments:** Index application, submissionCase, and sysFileRefId.
* **users:** Index osdpLoginId, team, and role.
* **adrblkfilerefs:** Index adrBlkId and sysFileRefId.

**6. Codebase Structure:**

* **Configuration:** The configuration files (config/\*.js) contain various constants and mappings. Consider organizing these into a more structured configuration management system.
* **Utilities:** The utils/ directory contains a mix of utility functions. Organize these into more specific modules based on their functionality.
* **Models:** The models (models/\*.js) define the data structures. Ensure that these models are consistent with the database schema.
* **Routes:** The routes (routes/\*.js) handle the API endpoints. Consider using a more structured routing system to improve maintainability.

**7. Security Considerations:**

* **Environment Variables:** Ensure that all sensitive information (e.g., database credentials, API keys, client secrets) are stored in environment variables and not hardcoded in the code.
* **Input Validation:** Implement robust input validation to prevent SQL injection and other security vulnerabilities.
* **Authentication and Authorization:** The requireUser middleware provides basic authentication. Consider implementing more granular authorization controls to restrict access to specific resources based on user roles.

**8. Performance Considerations:**

* **Query Optimization:** Analyze the most common queries and optimize them using appropriate indexing and query techniques.
* **Caching:** Implement caching to reduce the load on the database for frequently accessed data.
* **Connection Pooling:** Ensure that the database connection pool is properly configured to handle the expected load.

**9. Scalability Considerations:**

* **Database Sharding:** If the database continues to grow rapidly, consider sharding the database to distribute the data across multiple servers.
* **Microservices Architecture:** Consider breaking the application into smaller, independent microservices to improve scalability and maintainability.

By addressing these points, the LSCP can be designed to be a robust, efficient, and scalable system that meets the needs of the Buildings Department.