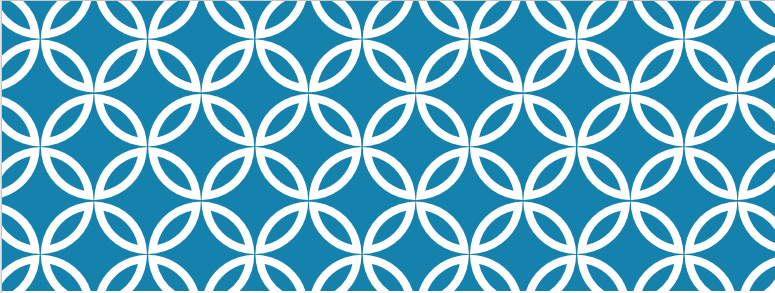
a

Data Architecture



MOSIP

Modular Open Source Identity Platform

Version 0.1 | 24 July 2018

Copyright Information

This document is the exclusive property of \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_; the recipient agrees that they may not copy, transmit, use or disclose the confidential and proprietary information in this document by any means without the expressed and written consent of Mindtree. By accepting a copy, the recipient agrees to adhere to these conditions to the confidentiality of \_\_\_\_\_\_\_\_\_\_\_\_\_ practices and procedures; and to use these documents solely for responding to \_\_\_\_\_\_\_\_\_\_\_\_\_\_ operations methodology.

Revision History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver** | **Change Description** | **Sections** | **Date** | **Author** | **Reviewer** |
| 0.1 | First Draft | All | 10-Aug-2018 | DM Team |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

References

|  |  |  |  |
| --- | --- | --- | --- |
| No | Document Name | Ver. | Location |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Glossary

|  |  |  |
| --- | --- | --- |
| **Terminology** | **Definition** | **Remarks** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Table of Contents

[Copyright Information 2](#_Toc530490110)

[Revision History 2](#_Toc530490111)

[References 2](#_Toc530490112)

[Glossary 2](#_Toc530490113)

[Table of Contents 3](#_Toc530490114)

[1 Introduction 6](#_Toc530490115)

[1.1 Context 6](#_Toc530490116)

[1.2 Purpose of this document 6](#_Toc530490117)

[1.3 Scope of this document 6](#_Toc530490118)

[2 High Level Data Architecture 7](#_Toc530490119)

[3 Data Architecture Principles 8](#_Toc530490120)

[3.1 Open Source and Vendor Neutral 8](#_Toc530490121)

[3.2 Adaptability 8](#_Toc530490122)

[3.3 Data Security and Governance 9](#_Toc530490123)

[3.4 Multi-Party 9](#_Toc530490124)

[3.5 Multi-Language 9](#_Toc530490125)

[3.6 Authorization 9](#_Toc530490126)

[3.7 Authentication 10](#_Toc530490127)

[3.8 High Performance 10](#_Toc530490128)

[3.9 Scalability 10](#_Toc530490129)

[3.10 Availability 10](#_Toc530490130)

[4 MOSIP Data Architecture 11](#_Toc530490131)

[4.1 <. . .> 11](#_Toc530490132)

[4.2 <. . .> 11](#_Toc530490133)

[5 MOSIP Data Model 11](#_Toc530490134)

[5.1 Data Model Standards 12](#_Toc530490135)

[5.2 Global Master Reference Data Model 12](#_Toc530490136)

[5.3 Conceptual ER Data Model 12](#_Toc530490137)

[5.4 Logical Data Model 12](#_Toc530490138)

[5.5 Physical Data Model 12](#_Toc530490139)

[6 Database Performance 12](#_Toc530490140)

[6.1 Database Server 12](#_Toc530490141)

[6.2 Table Partitioning Strategy 13](#_Toc530490142)

[6.3 Indexing Strategy 13](#_Toc530490143)

[7 Auditing, Event/s Logging, Alerts & Notification 13](#_Toc530490144)

[7.1 Auditing 13](#_Toc530490145)

[7.1.1 Application Auditing 14](#_Toc530490146)

[7.1.2 Database Auditing 15](#_Toc530490147)

[7.1.3 IT Infrastructure Auditing 16](#_Toc530490148)

[7.2 Event/s Logs 16](#_Toc530490149)

[7.3 Alerts & Notifications 16](#_Toc530490150)

[8 Data Security and Governance 16](#_Toc530490151)

[8.1 Security Standards and Guidelines 16](#_Toc530490152)

[8.2 Data Authorization 16](#_Toc530490153)

[8.3 Data Authentication 17](#_Toc530490154)

[8.4 Data Access Governance 17](#_Toc530490155)

[9 Backup & Restore / Recovery / Archival 17](#_Toc530490156)

[9.1 Database Backup and Restore 17](#_Toc530490157)

[9.1.1 Backup interval 17](#_Toc530490158)

[9.1.2 Backup retention 17](#_Toc530490159)

[9.1.3 Performance impacts 17](#_Toc530490160)

[9.1.4 Restore Options 17](#_Toc530490161)

[9.1.5 Postgres Backup – Logical Backup 18](#_Toc530490162)

[9.1.6 Postgres Backup – Physical Backup 18](#_Toc530490163)

[9.2 Database Archival 18](#_Toc530490164)

[9.3 Database Recovery 18](#_Toc530490165)

[10 Naming Standards 18](#_Toc530490166)

[10.1 Naming Standards 18](#_Toc530490167)

[10.1.1 Common Naming Standards 19](#_Toc530490168)

[10.1.2 Database Server Naming Standards 19](#_Toc530490169)

[10.1.3 Database Naming Standards 19](#_Toc530490170)

[10.1.4 Schema Naming Standards 20](#_Toc530490171)

[10.1.5 Table Naming Standards 20](#_Toc530490172)

[10.1.6 Index Naming Standards 20](#_Toc530490173)

[10.1.7 Key Naming Standards (Primary, Unique, Foreign Keys) 20](#_Toc530490174)

[11 Assumptions / Constraints / Issues / Risks 21](#_Toc530490175)

[11.1 Assumptions 21](#_Toc530490176)

[11.2 Constraints 21](#_Toc530490177)

[11.3 Issues 21](#_Toc530490178)

[11.4 Risks 21](#_Toc530490179)

[References 21](#_Toc530490180)

[Contact 22](#_Toc530490181)

# Introduction

## Context

Modular Open Source Identity System (MOSIS) is a Digital Identity system implemented with a vision to enable any country to provision Unique Digital Identity Number (UDIN) for their resident and nonresident citizens. IIIT-B and the key stakeholders for this platform implementation would like to implement the system as an Open Source Project, with a community of users, contributors and developers.

As MOSIP is being developed as an open source identity platform project, Data Management and Data Architecture of MOSIP should comply with the Data Management Standards, Guidelines and Principles defined in the document that includes Database, Data Model and Database SQL programming.

## Purpose of this document

This document gives various Data management and Data model standards to be followed during the MOSIP development.

## Scope of this document

This document covers the data management standards, guidelines and principles to be followed by the MOSIP development and implementation teams, specifically PostgreSQL database, Data model, Database objects naming standards and best practices. Though the MOSIP is envisioned as open source, platform and technology independent, Postgres is the choice for initial release development and implementation.

# High Level Data Architecture

This section covers high-level data architecture of MOSIP.

<TBD - Architecture diagram >

# Data Architecture Principles

This section covers high-level data architecture principles of MOSIP.

* Open source and Vendor Neutral
* Adaptability
* Security
* Multi party
* Authorization
* Authentication
* Multi language support
* Performance and scalability
* High Availability
* Auditability

## Open Source and Vendor Neutral

To handle vendor neutrality and open source, the following consideration are followed while designing the data model and the database design.

* No business logic is applied at database level: Database will be used only to store and retrieve data. There is no business logic applied at database level other than Primary / Unique key, Not null and foreign keys. Foreign keys are applied within the same database, if a table is referenced in another database then no FK is applied.
* No specific database features will is used: All features of the database that are common across databases will be applied. If any feature is specific to a database then that will not be applied, instead, the feature will be built manually.
* All DDL, DML and DQL statements will follow ANSI standards
* Metadata approach to handle complex and flexible data structures
* Only following datatypes are being used
  + Character varying
  + Timestamp
  + Date
  + Integer
  + Number

## Adaptability

<Describe the requirement and elaborate in the respective sections as needed, provide links as needed if any>

* Data driven with Configuration parameters.
* Configuration is considered throughout the MOSIP Modules
* Customization if required to adapt any requirement
* Wide variety of native data types are supported including JSON
* Support several interfaces and programming language libraries
* Flexibility to customize for any new changes to be minimal
* Data Migration needs to be considered with minimal changes. Migration should not impact the current data

## Data Security and Governance

<Describe the requirement and elaborate in the respective sections as needed, provide links as needed if any>

## Multi-Party

<Describe the requirement and elaborate in the respective sections as needed, provide links as needed if any>

* All different actors like Human, Computers, Devices…etc. will be registered and managed in data store.
* System accommodated all the parties like individuals, officers, Admins, government bodies, Agencies, TSP…etc.
* Each party will get the privilege and access specific modules
* RBAC (Role Based Access Control) to be applied
* UBAC (User Based Access control) also to be applied.
* Dates/timestamp stored in MOSIP will be in UTC

## Multi-Language

As MOSIP system will be used by multiple countries, there is a need to support multiple languages. So as per the requirements, MOSIP will support 3 languages as configured by the country level administrator.

The following datasets will be captured and stored in multiple languages

* Master Data
* ID data of an individual
* Transaction comments
* Labels used in UI
* Messages and notifications

From a database side the data will support **UTF-8 Unicode character set** to store data entered in multiple languages.

There will not be any in-built support to translate data at database level. Any translation or transliteration will be handled at API or UI layer.

## Authorization

<Describe the requirement and elaborate in the respective sections as needed, provide links as needed if any>

## Authentication

<Describe the requirement and elaborate in the respective sections as needed, provide links as needed if any>

## High Performance

<Describe the requirement and elaborate in the respective sections as needed, provide links as needed if any>

To support high performance, following database features are applied

1. Database sharding is applied on uin dataset. By default, base sharding algorithm will be applied in MOSIP system. SI can define the sharding algorithm based on the deployment setup
2. History tables are defined to store updated information. These tables will be used to get point-in-time details
3. All tables will have a primary key index on the primary key field. This will help in faster retrievals and joins
4. All foreign keys will have indexes defined so that it will help in faster joins
5. No referential integrity is applied on tables across databases
6. Partitioning????

## Scalability

<Describe the requirement and elaborate in the respective sections as needed, provide links as needed if any

* MOSIP Data system can be scaled based on the loads.
* Specific to country implementations MOSIP Data System can be hosted on scale-up or scale-down infra based on the requirement.
* High performance and can handle hundreds of terabytes data
* Robust, feature-rich, fully ACID compliant database
* DB sharding – to be manually implemented for individual data, Sharding can be done using first few digits of UIN number??? Need to finalize
* Support for horizontal scalability – need to discuss this with DBA
* Performance SLAs to be taken care during development phase. Classifications to be done

## Availability

<Describe the requirement and elaborate in the respective sections as needed, provide links as needed if any>

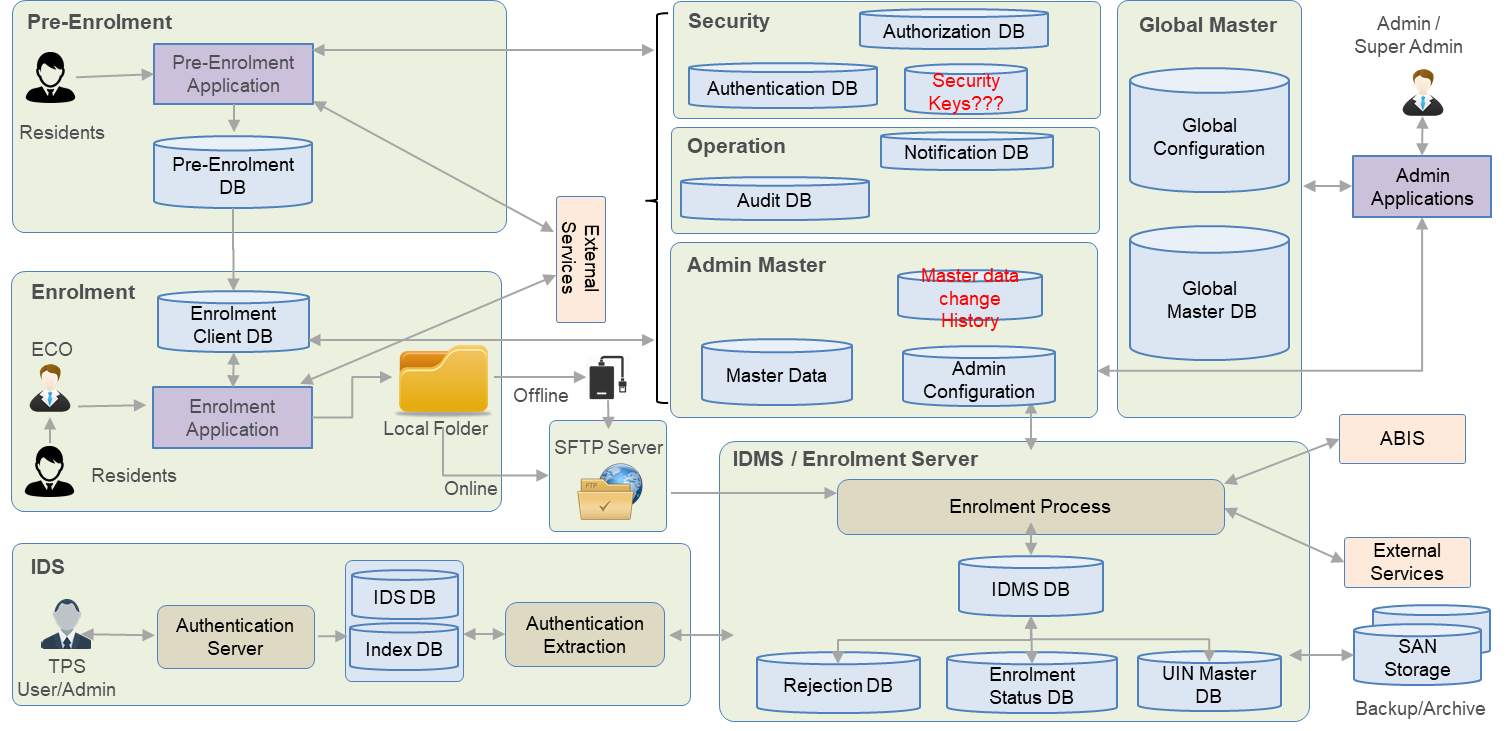
* Check what HA options available in PostgreSQL
* Backup and DR: Should be defined and implemented by SIs
* RDBMS chosen supports high availability, capable of hot backup for high availability.
* Data replication:

# MOSIP Data Architecture

This section covers high-level data architecture standards of MOSIP.

* MOSIP is developed on a open source data platform.
* Chosen data platform is free and open source.
* Chosen data platform can be replaced with another equivalent component with help of abstraction components
* Software and technology stack doesn’t mandate any specific vendor

Below diagram provides the data architecture of MOSIP system



## <. . .>

<TODO>

## <. . .>

<TODO>

# MOSIP Data Model

This section contains detailed information for data model design standards, guidelines and principles for global master reference, conceptual, logical and physical model design standards been followed in MOSIP project.

## Data Model Standards

* Meaningful Naming: All the objects that are being created will have a meaningful naming.
* PK, UK, FK, Not Null constraints
* Flexible model
* Most of the data is encrypted
* No business logic, no defaults, no use of sequences, identify fields at database level
* No DB triggers, avoid DB functions, procedures. If needed, It should be ANSI compliant
* SQL used in application should be ANSI Compliant
* The data is stored in multiple languages as configured by administrator of the system
* Referential integrity is applied on the tables belonging to same database. If the objects are referred across DB then this will be handled at the application layer.
* Only ID information is encrypted
* System related data will be stored in system default language (mostly English).

## Global Master Reference Data Model

<TODO>

## Conceptual ER Data Model

<https://github.com/mosip/mosip/tree/DEV_database_sprint5/database-scripts/data_model>

## Logical Data Model

<TODO>

## Physical Data Model

<https://github.com/mosip/mosip/tree/DEV_database_sprint5/database-scripts/data_model>

# Database Performance

## Database Server

The database is designed in a way that each module can be deployed on same or separate database servers. Even the id repository database is designed to store the data into different database server using sharding methodology. The sharding logic will be implemented at the API level.

## Table Partitioning Strategy

Table partitioning is part of an advanced database design for better SQL performance. Generally partitioning strategy is advisable and need to be followed when the data set is very high over millions of rows. It is dividing the table using set of columns as key that

It logically divides the table using a column or columns of data as an index which demarcates the beginning and ending of a segment of data into individual partitions. These act like tables within a table and will improve querying performance. The key is to select the correct data to use as the partition index field and to include that as an element in the SQL where or having clause.

The way partitions are defined in PostgreSQL is different when compared to other commercial databases. The partitions use techniques like inheritance, constraints and db triggers or rules. This will make things complex so we might not use partition in MOSIP.

In MOSIP, we have the following types of tables where we will have huge datasets, they are listed below

* **Transaction**

Transaction tables are huge and will have lots of data. Most of the queries will be on open or in process transactions. So, the main transaction table can contain only open transaction and all closed transactions can be moved to a separate table.

* **Audit**

Only 6 month of audit data can be maintained in audit table and rest of the data can be moved to audit history table.

* **UIN**

UIN data will be sharded and stored in different databases, which will provide horizontal scalability.

## Indexing Strategy

All the tables will have a primary key and a primary index created. Also, the foreign keys that are created will also have an index. Apart from that, based on the need necessary indexes would be created.

# Auditing, Event/s Logging, Alerts & Notification

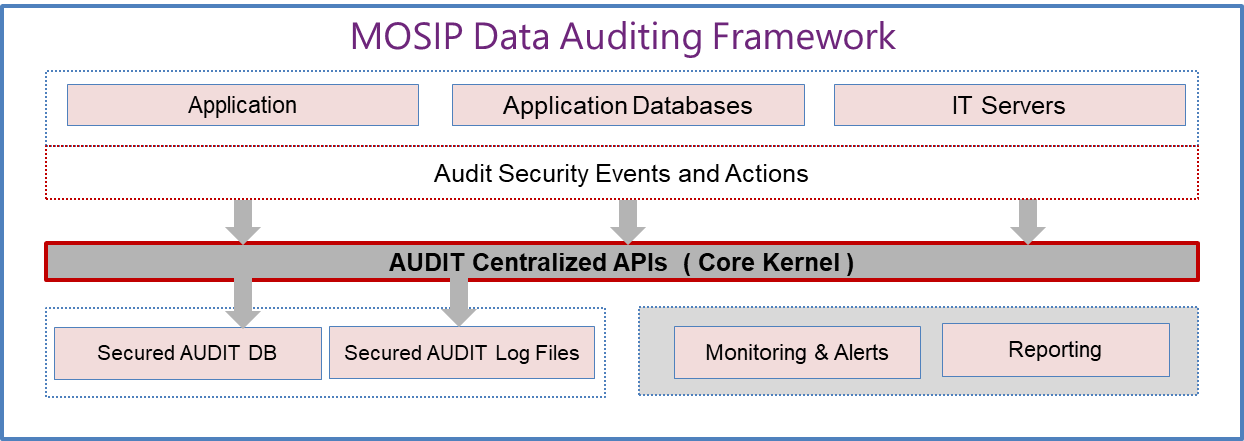
## Auditing

Auditing is an examining process to ensure set of policies and practices are followed and respected by IT infrastructure owners and application users. In MOSIP, the following types of audits (not limited to), are followed and may be extended as per the requirements set by the implementer and/or MOSIP authority.

* Auditing on specific business activities.
* Auditing on a limited subset of data, like PII and classified secured data.
* Auditing on database objects, roles, security.
* Auditing on the database servers, access, connections.
* Auditing on systems/operating servers including system security.
* Auditing on client application level events and activities.
* Auditing on users activities.

Generally, MOSIP auditing consists of three layers and may be extended with more detailed auditing features.

* IT Infrastructure (Servers)
* Database
* Application



The scope of audit details covered in this document are database, classified/secured data, and application level auditing as per business requirements.

* All the events at MOSIP System are stored in secured AUDIT DB for audit analysis, fraud management, alerts and notifications.
* Event information like event, what, when, who, why, where, which etc. are captured for auditing
* Events are used to track the changes and for the fraud management

### Application Auditing

The MOSIP application is expected to capture and maintain its audit logs, captured during various events and actions including

* Application configurations and customizations
* application login users
* user actions
* classified data access
* attribute level data capture/store
* errors
* exceptions
* etc

**Audit\_Log** table is created in a secured separate database, which consist of following attributes to track actions/events. Core Kernel API to be used to log / insert audit log entries. There is no direct access rights/permission will be granted for application users.

|  |  |  |  |
| --- | --- | --- | --- |
| ***Sl No*** | ***AUDIT Attributes*** | ***Description*** |  |
| *1* | *APP Event Type* | *Login, DataChange, Edit, Saved, etc* |  |
| *2* | *App Object Name* | *table/object name* |  |
| *3* | *App Object Type* |  |  |
| *4* | *Schema Name* |  |  |
| *5* | *DB Name* |  |  |
| *6* | *Table Name* |  |  |
| *7* | *Client System IP* |  |  |
| *8* | *Application Name/ID* |  |  |
| *9* | *Session User Name/Id* |  |  |
| *10* | *Action Timestamp* |  |  |
| *11* | *Audit LogSaved Timestamp* |  |  |
| *12* | *Audit Log Text* |  |  |
| *13* | *Statement Text* |  |  |
| *14* | *Fields Impacted (json)* |  |  |
| *15* | *OLD Values* |  |  |
| *16* | *NEW Values* |  |  |
| *17* | *JSON\_Store* |  |  |
|  | *(Others …)* |  |  |
|  |  |  |  |

The **Audit\_Log** table is a generic structure with nullable attributes with an additional JSONstore which is a custom data store for unstructured, customized data logs. Application team can use the Core Kernal API to populate the audit log information as needed, for Database, for Application (event, exception etc) as per audit requirements.

### Database Auditing

Database level auditing is a one of the critical feature implemented in MOSIP to ensure tracking of all data accesses and data related events and process. Database auditing includes all CRUD (create, read, update, delete) operations.

* someone tries to access data directly on the database bypassing the app logging system
* data access by privileged user or a DBA.
* database objects related activities ( create/drop etc)

The options available in PostgreSQL regarding database audit logging are:

* extensive logging using log\_statement = all database option
* this option can be enabled for database server login
* it has a performance impact and can be enabled as required with caution.
* Implementing custom trigger functions on database tables and objects as needed.
* Using tools provided by postgres community
  + audit-trigger 91plus (<https://github.com/2ndQuadrant/audit-trigger>)
  + pgaudit extension (<https://github.com/pgaudit/pgaudit>)

### IT Infrastructure Auditing

<Infrastructure / server security and auditing to be covered in detail on IT Architecture documents. The scope of this Data Management document is limited to data, database and application data.>

## Event/s Logs

<describe application, database event logs (errors, exceptions, etc) process and process for capturing, storing, accessing them for analysis, reporting etc>

## Alerts & Notifications

<Describe Application and Database alerts to be setup, configured, implemented etc >

# Data Security and Governance

## Security Standards and Guidelines

* Individuals data is secured at every stage of the system.
* Necessary data will be encrypted during transport and persistent store. Encryption and decryption at cell level (row level)
* All users will be authenticated with Multi Factor Authentication
* User access permissions can be granted or revoked on any object at the database, schema or table levels
* DBA should not see and perform any action on the data
* No Foreign Key reference on IDs like UIN, Registration ID…etc. User surrogate keys for reference
* Data classification should be done for encryption, access grants…etc.
* No select query is accepted from any type of request or users other than application, No Select \* query at all.
* No export and import operation of database is allowed
* Audit details at DB, Table, Data level – Check feasibility
* Passwords will be stored in DB - hashed
* Identify how we can restrict/control data access for backend users at DB level. Also test how we can log the actions performed at backend level

## Data Authorization

* MOSIP Kernel uses externalized authorization component to control accesses to the kernel operations
* RBAC to be implemented at row level
* CRUD operation
* DB users / schemas to be defined for each application

## Data Authentication

* All the MOSIP resources are secured. The requests are allowed only after authentication
* Authentication details and authentication methods of users will stored at data system
* DBA admin policies to be defined
* Password change policies
* Forgot password
* Physical access
* OTP
* Application level users and passwords to be maintained

## Data Access Governance

<TODO>

# Backup & Restore / Recovery / Archival

<

>

## Database Backup and Restore

<TO DO>

### Backup interval

<to do>

### Backup retention

< to do >

### Performance impacts

< to do >

### Restore Options

<to do>

### Postgres Backup – Logical Backup

* Postgres pg\_dump, sql etc.
* SQL script dump of schema + data
* Restored through SQL commands
* Great flexibility
* Not the greatest for performance

### Postgres Backup – Physical Backup

* Filesystem snapshots
* pg\_basebackup
* Manual” base backups

https://www.hagander.net/talks/Backup%20strategies.pdf

## Database Archival

This will be implemented as part of the SI’s strategy.

## Database Recovery

This will be implemented as part of the SI’s strategy.

# Naming Standards

## Naming Standards

A simple and consistent naming convention for database objects, when followed rigorously, can help database application developers greatly. This is because developers, once they get used to the convention, can quickly identify objects belonging to their application and are less likely to make mistakes regarding the contents of columns. In fact, an inadequate or improperly followed convention can actually increase the development effort by unnecessarily tying the application code to intricacies of the database physical design and by making application developers overly dependent on the DBA’s.

The intended audience for this document is data modelers, DBA’s and database application developers.

The purpose of this document is to propose a simple and consistent naming mechanism for all the objects in a database schema. The naming convention is aimed at reducing the dependence of the application developers on the database administrator by naming objects in a way that unambiguously defines their contents. Where applicable, there is also a mention of how not to name the object and the reasons for this.

Of course, there can be no one absolute convention that will solve all naming problems and have universal appeal. The conventions offered in this document are merely one way of naming things that appears to work. The reader is free to modify these to suit individual project needs.

Since most of the objects of MOSIP is created on their own databases, we will not have any prefix or suffix of the application / module name or abbreviation to the table or object name. If needed, we can have multiple schemas screated

### Common Naming Standards

In the naming of the objects within the DB, the following common standards are applied.

* Singular Names to the entities
* Object name length to be less than 30 chars
* Lower case object names separated by underscore (\_)
* Only defined abbreviations are used
* No prefix or suffix to the table names
* Each table is defined an alias, this alias is used in referenced tables, constraints and index names.

### Database Server Naming Standards

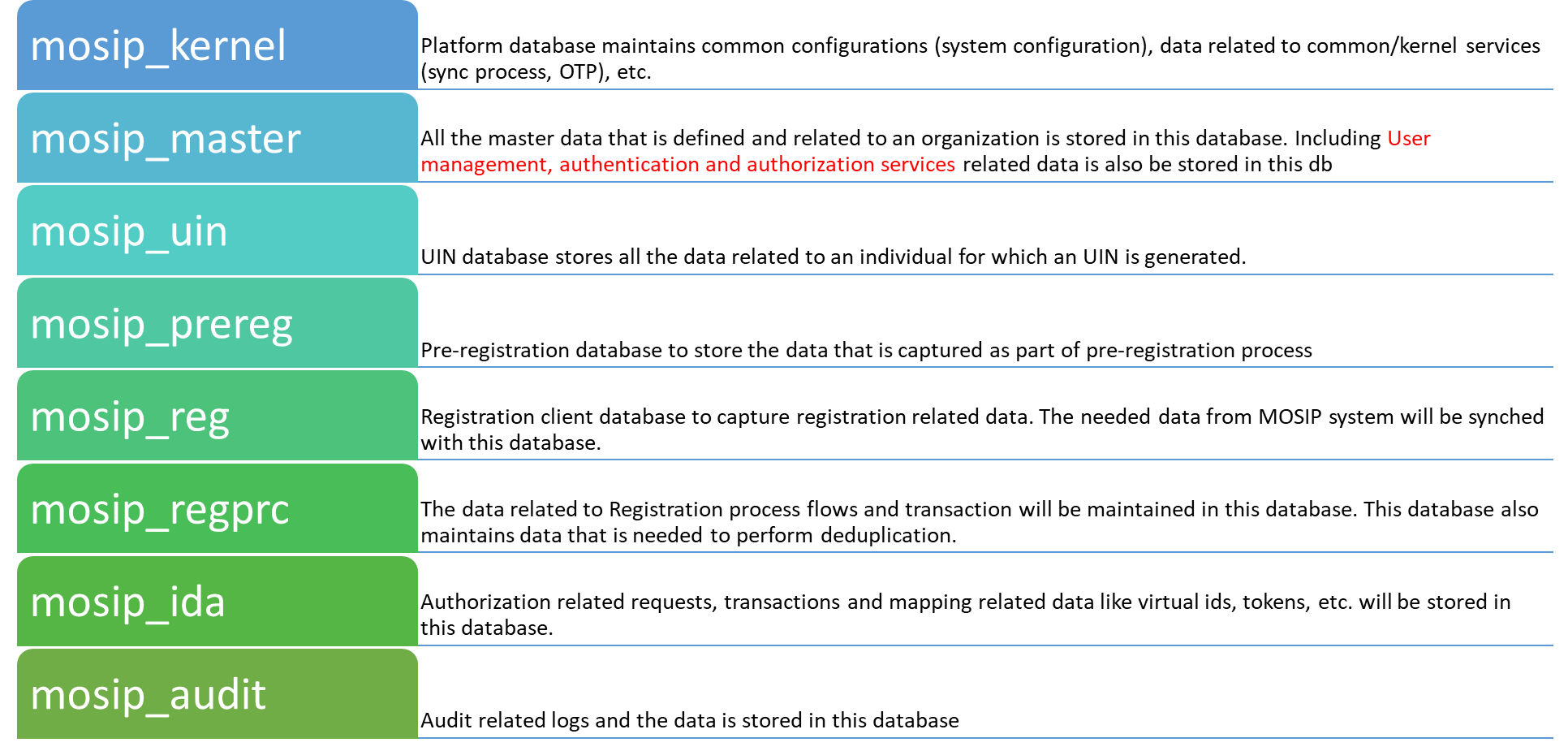
<TBD>

### Database Naming Standards

The database names will follow the below naming convention

mosip\_<abbreviated value of the application/module name>

In MOSIP system, we will have the following databases



### Schema Naming Standards

Schema name is named after the DB name, by default, without mosip\_. If there are more than one schemas in a DB, then a proper single word name is assigned, either full word or an abbreviated word.

### Table Naming Standards

The table name can have one or two words that describe the contents of the table separated by underscore (\_). If there are more words then those can be abbreviated based on the standards. Making sure the table name length is less than 30.

The description should always be in singular (for example, REGISTRATION, REG\_TRANSACTION) since they are easier to use and are shorter. Storing the name in plural could be cumbersome, especially in the case of tables used to resolve many-to-many relationships as these could have two plurals in the name.

Table names should NOT denote whether the underlying object is a table or a view because this could change during the application cycle (for example, a join view may be converted to a pre-populated table for performance reasons or a table may be converted to a view to show some extra computed columns).

An alias for each table is defined, this alias can be used in various other places like reference keys, indexes, constraints, etc.

### Index Naming Standards

Indexes are named as <table\_name alias>\_<col abbreviation>\_idx\_<n>

Here n is a number of 2 digits like 01, 02, etc.

### Key Naming Standards (Primary, Unique, Foreign Keys)

**Primary Key:** Each table should have a primary key, the key should be named as **“pk\_<table\_alias>\_<column\_name>”**. If it is a composite key, then in place of column name any meaning full name can be provided. PK should be defined on business key, in case for some reason a business key fields cannot be used to define a PK then add a surrogate key to the table.

**Unique Key:** If a surrogate is used as PK then create a unique key, on fields that uniquely defines a business key. The naming of the unique key should be **“uk\_<table\_alias>\_<column\_names>”**.

**Foreign Key:** Any references from a table with the master / other tables, the creating a foreign key is mandatory. This helps maintain referential integrity. Foreign key can be named as **FK\_<referring table alias>\_<referred table alias>\_<column description>**

# Assumptions / Constraints / Issues / Risks

This section covers risk and issues of MOSIP program.

## Assumptions

MOSIP data architecture has been designed for open source database technologies.

## Constraints

<Describe constraints regarding database design, limitations that impacts design etc.>

## Issues

<Describe issues related to database design, availability of support, plug in, drivers etc.>

## Risks

<Describe Risks on database design, open source, support, availability, new updates etc >

# References

**<document reference links >**

# Contact

Sadanandegowda Dm [Sadanandegowda.Dm@mindtree.com](mailto:Sadanandegowda.Dm@mindtree.com) , Architect.

Shanmugam Muthusamy [Shanmugam.Muthusamy@mindtree.com](mailto:Shanmugam.Muthusamy@mindtree.com), Senior Technical Manager.

Nasir Khan [Nasir.Khan@mindtree.com](mailto:Nasir.Khan@mindtree.com), Program Director.