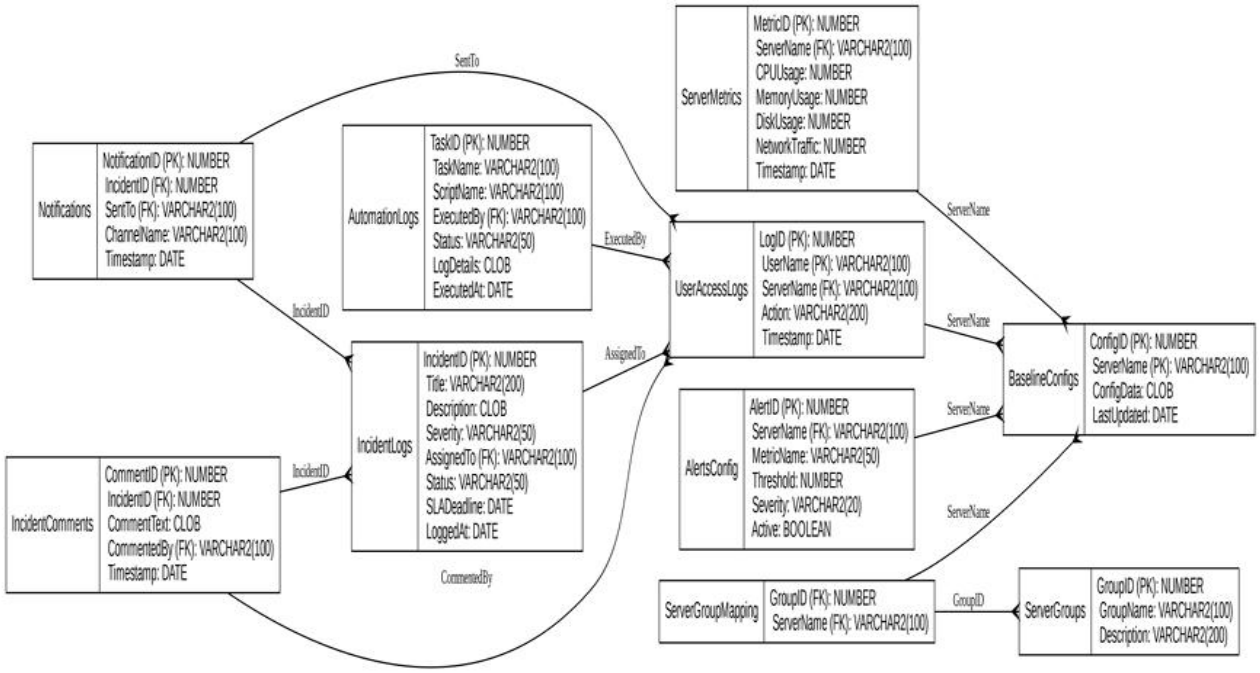
Database Normalization: InfraVision

1. Introduction

The baseline schema is derived from the ER Diagram based on SRS, which is suboptimal and could cause issues for data integrity and system performance as it is scaled.

Initial ER Diagram:



- Remove data redundancies through normalization techniques
- Build appropriate entity relationships with integrity constraints
- Make an improved security framework and access control.
- Hence, we changed the initial schema into a normalized relational database.

2. Initial Schema Assessment

The database relied on server names as its primary key than on unique identifiers which were repeating across multiple tables. Primary keys were not consistently applied with some tables not having entity integrity constraints.

Intitial:

- ServerMetrics
- BaselineConfigs
- ServerGroups
- ServerGroupMapping
- AlertsConfig
- UserAccessLogs

- AutomationLogs
- IncidentLogs
- IncidentComments

Notifications

Tables referenced servers by text names instead of IDs, creating maintenance issues when server names changed. The initial design consisted of disconnected tables with redundant data and limited relationships.

- ServerMetrics tracked performance statistics but lacked proper server references.
- BaselineConfigs stored configuration data without version control
- ServerGroups defined logical groupings without structured relationships.
- ServerGroupMapping attempted to create associations using text identifiers
- UserAccessLogs tracked activity without proper user references
- AutomationLogs recorded tasks using text-based identifiers
- IncidentLogs documented problems without categorization or relationship structure
- IncidentComments stored communications with minimal referential constraints.

3. Normalization Process

3.1 Core Entity Establishment

We made a stand-alone Servers table with ServerID as the primary key, replacing previous scattered server name references. Then, implemention of Users table with UserID was done. We added a Locations table to track physical server placement.Then we implemented a Roles table to manage access control hierarchies.

We established relationships between the tables by adding foreign key constraints throughout the schema and creating required tables due to many-to-many relations.

3.2 Server Management Enhancement

Changed ServerMetrics to reference servers by ID rather than name, eliminating oncoming inconsistencies. We expanded metric collection by proper timestamp tracking. The server org improved through revised ServerGroups and ServerGroupMapping tables with apt constraints.

3.3 Configuration Management Restructuring

Made a ConfigurationProfiles table to establish standardized baseline definitions across multiple servers. Version tracking is for configuration history. The ConfigurationHistory table is to allow for auditing.

3.4 Security Model Implementation

The Users table has implementation of password hashing and secure access controls.

Roles table establishes permission levels.

UserRoles is for role assignment management.

UserAccessLogs is to capture activity patterns.

SuspiciousActivity monitoring system flags potential security concerns.

3.5 Operational Functionality Enhancement

Task automation has TaskTypes and AutomationScripts

AutomationLogs provides execution tracking. Alert management has separates definitions from configurations through AlertDefinitions and AlertsConfig tables. The new Alerts table tracks actual alert instances with acknowledgment capabilities. Incident management is there with categorization improvements along withIncidentTypes table and enhanced notification delivery via dedicated tables.

3.6 Load Balancing

LoadBalancingEvents is there for implemention of balancing operations across various server groups.

LoadBalancingDetails table records before/after metrics for data analytics.

4. Final Schema

4.1 Core Server

- Servers
- ServerMetrics
- Locations
- ServerGroups
- ServerGroupMapping

4.2 Managing Configurations

- ConfigurationProfiles
- ServerConfigurations links servers to active configurations
- ConfigurationHistory maintains a complete audit trail.

4.3 User Management

- Users store account information.
- Roles assign permission levels depending on seniority.
- UserRoles assigns permissions to users.
- UserAccessLogs tracks activity.
- SuspiciousActivity is for threat monitoring.

4.4 AutomationOfTasks

3 tables are present:

- TaskTypes
- AutomationScripts
- AutomationLogs

4.5 Alert Management

3 Tables are present :

- AlertDefinitions
- AlertsConfig
- Alerts tracks

4.6 Incident Management

5 tables are present

- IncidentTypes for issues
- IncidentLogs logs specific incidents
- IncidentComments tracks communication,
- NotificationChannels defines how notification is sent
- Notifications records alert delivery

4.7 Load Balancing

Two tables support resource optimization:

- LoadBalancingEvents tracks balancing operations
- LoadBalancingDetails records server state changes.

5. Benefits

Foreign key relationships is to enforce referential integrity in the system. We've eliminated redundant storage. There is improved query performance. The design scales efficiently as we add servers, users, and functionality.

The security model now provides user permissions with complete activity tracking. Audit capabilities now have a complete history of configuration changes and user actions. Incident management follows a structured approach to tracking and resolution. Reporting capabilities have expanded substantially, supporting complex analytical queries.