
Project Plan for an Improved CISN SCMC Station Information System

Version: 1.0

Revised: October 26, 2004

Prepared by: Vikki Appel

Prepared for: CISN Southern California Management Center

California Institute of Technology

United States Geological Survey

1.0 Introduction

This document outlines the project plan for developing an improved station information system for the CISON Southern California Management Center (SCMC). The goal of this project is to develop a system that can interact with a single database source to enter, update and retrieve station metadata easily and efficiently. The timeline for this project is to have the system functioning as the primary metadata source for the SCMC in one year, by September 1, 2005.

2.0 Current Methods

Problem Statement:

The current methods of keeping track of all station information at the SCMC are not providing the functionality required to meet the needs of all of the users who need complete, accurate station information from a single source.

Station information is currently found in the following places (not an exhaustive list):

1. Operator email list: changes to stations are mass-emailed to every member of the operators@iron list. These emails are archived at:
http://rift.gps.caltech.edu/station_updates/
2. Instrument shop database: Alberto Devora maintains an Access database of active station information.
3. Work area - **fix_dbase**: working area where changes are initially made (if changes are made directly to the production databases, they can be overwritten by contents of fix_dbase. Essentially, fix_dbase is the authoritative source of station information). Most changes to fix_dbase are manually-typed via SQL statements in sqlplus and due to denormalized database structure, a single change often necessitates changes to be made in multiple tables (e.g., if a station is discovered to be in the AZ network instead of the CI network, the update of CI to AZ must be made in each table that contains the NET attribute).
4. Production area - **trinetdb**: changes in fix_dbase are propagated to production databases (e.g., channel_data, simple_response) on SCEDC and RT databases. All RT databases get copies of active and historic channels.
5. Dataless SEED production area - **seed_feed**: the highly-denormalized NCEDC HT and IR schemas where data gets loaded via ASCII .tbl files. Station history is contained in the .tbl files and any change necessitates an update to the .tbl file and often, a wholesale regeneration of the database, making changes extremely difficult.
6. Chanloader scripts: required by RTS to parse out historic and SOH channels and generate channel lists of active channels that are appropriate for various RT processes.
7. CISON replicated tables: a test set of tables (e.g., channel_data) that have data replicated between NCEDC and SCEDC.
8. Hiroo Kanamori's PZ files: available in many forms, but searchable at:
http://www.data.scec.org/stations/hiroo_pz.php

3.0 Station Information System

The single most important determinant of a project's success is the identification of a clear and well-defined goal. The goal of this project is:

Build a simplified database-driven station information system that will provide station metadata that is as complete as possible and allow easy viewing and modification of data that is contained in the database.

A single person or a single functional group cannot accomplish this goal alone – it can only be accomplished by a team working together to gather information and understand the requirements of people and systems that use station information.

3.1. Business (Conceptual) Model Phase:

The SCEDC group has completed the analysis phase to determine the scope of the system and the key technical requirements for the system.

The scope of the system is to develop and implement a simplified metadata information system with the following capabilities:

- Provide accurate station/channel information for active stations to the SCSN real-time processing system.
- Provide accurate station/channel information for active and historic stations that have parametric data at the SCEDC e.g., for users retrieving data via STP from the SCEDC.
- Provide all necessary information to generate dataless SEED volumes for active and historic stations that have data at the SCEDC.
- Provide all necessary information to generate COSMOS V0 metadata information.
- Be updated through a graphical interface that is designed to minimize editing mistakes.
- Allow stations to be added to the system with a minimum, but incomplete set of information using predefined defaults that can be easily updated as more information becomes available. This aspect of the system becomes increasingly important with historic data when some aspects of the meta-data are simply not known
- Facilitate statewide metadata exchange for both real-time processing and provide a common approach to CISEN historic station metadata.

The system that will be developed has two fundamental design requirements:

- A well-designed database that is the single source of information, i.e., the authoritative information is what is contained in the database, not in flat-files or any other databases.
- Applications that interact with the data will interact directly with the database, i.e., they will not check information out of the database, manipulate it in another environment, and repopulate the database with the modified information.

3.2. Design Phase:

The project is currently in the Design Phase, where the investment of time and effort will have significant positive impact on the success of the end product. The main purpose of this phase is to bring all these pieces together to form a logical database model

containing all entities with their attributes, domains and relationships, together with a complete function model with its hierarchy, as well as domain constraints (on attributes), business rules (constraints), and events that trigger functions. The output of the Design Phase stage is a complete logical database model that will be implemented as a physical database model.

Database model: identifies and organizes the required data logically and physically. A database model tells you what information is to be contained in a database, how the information will be used, and how the items in the database will be related to each other.

Task: Develop a new station information logical model (schema) to represent the station data necessary to provide the capabilities identified in the conceptual model (e.g., the real-time operation of SCSN; to production of dataless SEED volumes, etc.).

Who: Vikki, Rob, Ellen, Nora

Duration: 1-2 months

Dependencies: none

3.3. Implementation Phase

After the data modeling is complete and the logical model has been established and implemented as a physical database schema, programs that access the database and insert, update and access the database can be developed.

Task: Write procedures, packages, triggers and constraints to ensure database integrity and access methods.

Who: Ellen, Vikki

Duration: ongoing

Dependencies: Completed data model

Task: Design an interactive interface (probably using PHP) to edit to modify and add data to the database.

Who: Nora, Ellen

Duration: 2-4 months

Dependencies: Completed data model

Task: Design and implement code to produce dataless SEED volumes and RESP files from new schema.

Who: Marie-Odile

Duration: 5-6 months

Dependencies: Completed and implemented data model; populated (can use data from .tbl file or seed_feed) database.

In this system, historic stations will be added to the database according to the best available information at the time of data-entry. In most cases, older station information will not need to be altered, so the metadata is complete (based on our as-complete-as-possible model) for that station. If additional information is discovered about an historic station, this system will allow the parameter to be updated easily, without jeopardizing any currently existing metadata. In this model, the maintenance of historic stations will

eventually become an extremely infrequent task. This structure is in contrast to many of the current approaches to metadata where databases are deleted and historic stations are continually re-loaded, placing the data in jeopardy of modification during these transactions.

3.4. Team Structure

This effort will be more successful with input from members of the CISN community who are committed to achieving the project's objectives. Currently, the development project team is composed of SCEDC staff and the following individuals will do the bulk of the work:

Rob Clayton: PI

Vikki Appel: will lead the development of this project. She will coordinate the team, direct the work and report to Rob Clayton.

Ellen Yu: responsible for database implementation, writing procedures, packages, triggers and constraints to ensure database integrity and access methods.

Marie-Odile Stotzer: 0.6 FTE programmer (start date November 22nd) whose focus is the station information system project. Her primary tasks will be to write code to generate and ingest dataless SEED volumes from/to the newly-designed schema and provide backwards compatibility with NCEDC system i.e., ingest and generate .tbl files.

Nora Mullaney: programming data-entry interface and providing SEED expertise.

Advisory Group: The SCMC team will do most of the core work, but will consult with a team of members from Northern California, IRIS and the ANSS for input.

3.5. The CISN SCMC Post-SIS Implementation

1. Operator email list: Will continue to exist, but may be driven from the GUI.
2. Instrument shop database: continue because of the kind of information they track (e.g., battery swaps)
3. **fix_dbase:** No longer necessary to maintain.
4. **trinetdb:** e.g., channel_data, simple_response on SCEDC and RT database will continue to exist as views that draw data from SIS; as time passes, codes will eventually access SIS.
5. **seed_feed:** No longer necessary to maintain, dataless SEED volumes will come from the SIS
6. Chanloader scripts: will draw data from views generated from the SIS.
7. CISN replicated tables: will draw data from views generated from the SIS.
8. Hiroo Kanamori's PZ files: Will continue to exist.

4.0 Appendix A – Database Details

4.1. Data Model

The **data model** is the product of the database design process which aims to identify and organize the required data logically and physically. A database model identifies what information is to be contained in a database, how the information will be used, and how the items in the database will be related to each other.

A well-normalized logical data model is implicitly designed for performance. If it's not well modeled, it becomes clear to the applications and the users. Unfortunately, at this point, it is extremely difficult and often impossible to change the physical model, so most organizations will try to tune the software for performance, which has very little chance of success – software tuning cannot overcome errors in the underlying data model. A well thought-out data model reduces the need for such changes and helps application maintainability.

4.2. Database Normalization

Normalization is a series of steps followed to obtain a database design that allows for efficient access and storage of data in a relational database. The purpose of the normalization process is to **reduce redundancy** (same information stored more than once), and **secure data integrity** (ensure that the database contains valid information). This is achieved by reducing large entities (large meaning a large number of attributes) into several other, lesser entities which together contains the same information without repeating it.

Example: Normalized vs. Denormalized Databases:

Case A

CHANNEL_INFO

#Channel_id

Net

Sta

Seedchan

Location

Ondate

Offdate

SENSITIVITY_INFO

#Sensitivity_id

#Channel_id (FK)

Stage_seq

Sensitivity

Frequency

[...]

Case B (current schema)

CHANNEL_DATA

#Net

#Sta

#Seedchan

#Location

#Ondate

Offdate

Channel

Channelsrc

SENSITIVITY

#Net

#Sta

#Seedchan

#Location

#Ondate

#stage_seq

Offdate

Offdate

Channel

Channelsrc

Sensitivity

Frequency

[...]

Performance vs. total performance

Denormalizing will not have a noticeable impact on performance unless there are 100,000 + transactions for each row in the table -- these are indexed searches. A good DBA will put indexes on all of the columns that make up the compound key, so regardless of the volume, the Oracle search algorithms will go straight on to the correct rows in the database in a normalized or denormalized database.

A common reason for the kind of denormalization shown in case B is when there are very few updates, and a higher likelihood of reporting sensitivity – this case is typical in a data warehouse situation, not an actively-updated database. However if there are very few transactions, then joining according to case A should not be resource intensive, should it? Is a denormalization really necessary?

Flexibility

Case A is more flexible than case B. A database is typically designed for a given application and they usually work very well together. However, others will find the database useful and they will wish to work on it, as well and this system needs to be designed to work with applications that haven't been developed yet.

Updating synchronization

Case B requires much more work each time a channel is inserted, updated or deleted. An insert/update/delete mechanism that is absolutely fool-proof must be written or a person must be responsible for ensuring that the EIGHT fields that are the same in channel_data and sensitivity were the same before the change are synchronized. If not, data is orphaned and we lose data integrity. In a denormalized database, changing redundant attributes in one table, but not in the other tables that contain that information results in a loss of data integrity.

In normalizing a database, you try to do several things:

- arrange data into logical groupings or subschemas so that each group describes some small part of the whole story.
- ensure that each column of each table clearly defines and describes its data content and that you leave no ambiguity about the data stored in that column.
- minimize the amount of duplicate data that could potentially be stored in a database.
- organizing the data so that when you need to modify it, you make the change in only one place.

Every time a decision to denormalize the database is made, a price is paid. The cost is lost flexibility, future scalability, performance, and data integrity. By denormalizing, there is data redundancy in the database, which needs to be managed through program code, either at the user interface (UI) or by using triggers, but is currently the responsibility of the individual modifying the data.

Summary:

Denormalization may solve one part of dealing with performance, but it creates possible performance problems in several other areas. Total performance impact must be evaluated. Furthermore, data integrity is at (high) risk. A clean, normalized database should always give a good performance, as well as preserve data integrity.

4.3. Primary Keys:

There are three fundamental demands on the candidate keys that we must never deviate from, if it is to become the subject for a primary key:

1. The candidate key must be unique within its domain.
2. The candidate key cannot hold NULL values.
3. The candidate key can never change. It must hold the same value for a given occurrence of an entity for the lifetime of that entity.

4.4. Use of Database Packages, Procedures and Functions

Embedded SQL statements should NOT be allowed in applications. All SQL that is routinely executed should be placed in stored procedures and functions, contained in database packages. When all SQL programming is left to individual programmers, individual modules result which have individual performance characteristics and without constraints at the database level, it is up to each and every programmer to do the job (or NOT to do the job).

What are the benefits?

1. The programmer does not need worry about the database structure
2. Tuning of SQL is done independently of how many times (or rather places) this access path is in use
3. Changes in the database structure/access paths are not influencing the application logic
4. It will easily outperform any programmer – our DBA can write the statements to utilize indexes and other performance improving hints that programmers aren't aware of.