UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
SPRING 2008 – CS 411 – DATABASE SYSTEMS
GRADUATE PROJECT FINAL
Pacific Region Integrated Climate Information Portal (PRICIP)

Uday Kari (kari1)

Contents

Introduction

This project develops a methodology, design and implementation of a utilities suite that ingests vast and complex data with many possible redundancies and inconsistencies into an RDBMS. Geographic information in the data is then used to locate derived information generated by queries on a web map. End user can select from among a range of parameters within the data, submit a prepared query to generate a map that displays results of the query as a plotted set of interactive icons. Indeed, integration of the database-query result-set with a mapping web service is one the chief goals. Selecting a map icon reveals basic information about a point in a pop-up balloon. This balloon features links to specific climate related products and accompanying data sets e.g., annual rates of change; annual, seasonal, and monthly event magnitudes, and event recurrence intervals of strong winds, heavy rains, and high seas. By far the biggest prize is the conceptual methodology developed to model and store legacy climate data, untangling the numerous existing redundancies and preventing future inconsistencies through the use of a Relational Database Management System (RDBMS).

The products and data sets accessible via this utility represent the results of analyses by the theme-specific data integration and product development teams of NOAA's Integrated Surface Hourly (ISH) mean sea level pressure and wind speed data; the Global Historical Climate Network (GHCN) precipitation dataset; the National Water Level Observing Network (NWLON) tide gauge records; the National Data Buoy Center (NDBC) wave buoy records; the U.S. Army Corps of Engineers' Coastal Data Information (CDIP) buoy data, and other data.

Vision

The customers include Pacific Region Integrated Climate Information Portal (PRICIP) program managers and scientists, who need a superior way to capture and aggregate climate product "metadata", currently being compiled informally (in spreadsheets, text etc) by various PRICIP members. Using the proposed database driven web application for PRICIP, one will be able to retrieve product metadata and use web links featured within this metadata to drill down to more pertinent information such as "sea-level archives for past year at tsunami detection buoy number 222", maintained elsewhere in NOAA on various servers. Several such (diverse) links may be keyed into a single climate product, thereby demonstrating the power of an RDBMS backend as an integration platform. Indeed, one of the biggest benefits of this project would be formalization of a relational data to eliminate redundancies in raw data, introduce referential integrity while being faithful to the customer needs and supporting extensibility as seamlessly as feasible. In future, several more such hyperlinks may further extend this dataset to enhance the usefulness to more users. However, the initial evaluation plan is to get some 100 or so such "clean" product records into the database and to support exposition of this database through an intuitive web interface, featuring a map. As of April 30, we have ingested 1620 records.

For more and latest information on the NOAA PRICIP program visit http://www.pricip.org

Design

The project was kicked off by NOAA scientists at a PRICIP conference in October 2007. The latest version of semantic description of the problem domain is in Appendix A. In Figure 1 we have a consensus **high-level** Entity-Relationship (E-R) diagram articulating the design of climate data as a **station** (i.e. sensor systems and platforms such as meteorological station, tide gauge or wave buoy) that generates one or more climate science **products** comprising one or more **datasets** whose metadata is to be stored within this database.

Figure 1: High Level Entity Relationship Diagram of PRICIP schema

The complete E-R Diagram, derived from semantic description is in Appendix B. The design emphasizes faithfulness, avoiding redundancy, simplicity, right relationships and data independence. The design does not address any specific application (in fact, it is meant for several applications) albeit a web-based map application plotting layer being is being developed. However, the design does support seamless, polymorphic (in object-oriented sense), extensibility of the data through numerous "isa" relationships at every opportunity.

The relational model, derived from the above entity diagram, illustrates in Appendix C, is based on the following specific principles of deriving relations from E-R Diagrams:

1. Turn each entity set into a relation with same set of attributes. For example, the entity set "contacts" with attributes name, email, phone results in a relation (table) called "contacts" with columns name, email, phone.
2. Replace each many-to-many relationship with a relation whose attributes are the keys of the connecting entity sets. For example, PlatformContacts may be created associating various platforms onboard stations with various opertors (owners) of that platform (omitted for now).
3. Combine relations created from many-to-one relationships into a single relation set. For example, the relationship "generates" in Figure 1, gets rolled into Products and "comprising" gets rolled into Datasets. This is done by including the key for Stations, stationID in relation Products and transitively inserting productID in the relation Datasets.
4. To address various subclass structures (isa relatiohships), we follow the "E-R style" of conversion whereby each entity set in the hierarchy will have a corresponding table in the database but will include the key attribute of the root. In many instances, reflecting that project is still in its infancy, such relations are just having the key of the root entity. However, this isa structure makes the model almost infinitely extensible with respect to key selection parameters like Process and Timeframe. In other words, each entity under Process (like Strong Winds) and/or Timeframe (like Forecast) can be extended to feature several unique attributes of their own later on.

Application of the above design principles leads to the following relations (tables) within the PRICIP schema (the indent in the listing below reflects hierarchy within the E-R model).

- Stations(stationid, operatorid)
    - Remote_Sensing_Stations (stationid,[this is still under development])
    - Insitu_Stations(stationid,name,latitude,longitude,region,nation,status,metadata)
- Products(productid, stationid,contactid)
    - Targeted_Information_Products(productid, [this is still under development])
    - Derived_Data_Products(productid,param,type,name,source)
        - Hosts(productid,url,format,description)
        - Processes(productid)
            - HeavyRains(productid)
            - HighSeas(productid)
            - StrongWinds(productid)
        - Timeframes(productid)
            - Hindcasts(productid)
            - Nowcasts(productid)
            - Forecasts(productid)
            - Futurecasts(productid)
- DataSets(datasetid, productid, owner, filename, format, description, last updated, metadata)
    - RealTime(datasetid)
    - NearRealTime(datasetid)
    - Derived(datasetid)
    - Archived(datasetid)
- Contacts(contactid, name, phone, email)
- Operators(operatorid, name)
- Nations(ISO 3-letter Alpha Code, name)
- Regions(code, name)

The ***extensibility*** of the data model is perhaps more evident from the above the relational model. For instance, the relation/entity "Forecasts" is just a simple child of Products, featuring just the productid. However, it can extended to any degree of complexity at a later more enlightened future stage of PRICIP. In fact, it may have several attributes specific to climate forecasts while also preserving its place in the PRICIP data model. Conversely, any conceivable product (like various "casts" above) or types of products (like "processes", "timeframes"), may be integrated into the PRICIP data model in future.

In addition to the above, there are utility tables, attributes supporting various backend needs of the application. For example, there is a "master" table that reflects external raw data as-is, warts and all (redundancies, inconsistencies etc). If a spreadsheet is the raw source, the attributes of the master table are simply the columns of the spreadsheet table.

Extract Transform Load (ETL)

The process of getting the data from the raw form (spreadsheets) into the master tables and from there into the object-relational PRICIP schema is being called "Extract, Transform, Load (ETL)" and is discussed below. An interesting side "benefit" of ETL is that spreadsheets become, effectively, the temporary User Interface for this database. In the early stage of the project, this means we can directly address may such legacy datasets being independently maintained by various program participants.

There were two paths for an Extract Transform Load (ETL) process flow:

1. Relational
2. Object Oriented

Relational

In the relational method, we initially map the output from spreadsheet(s) directly into a single database table with all string columns within the database and then do normalization of that table within the database using SQL commands to fit the data into a more elegant schema determined by the E-R schema in Appendix B. I believe this best captures the various concepts and formal methods discussed in the CS 411 course requirements and has real world the technical advantage of scalability. This is because, once the data is captured in the database, we do not have to worry about, say, running out of memory when doing the transformation…just buy a big enough computer and sophisticated DBMS like Oracle and be assured of high level fidelity to SQL commands that will can take care of any ETL job declaratively. However, the drawback may be that it is very hard to do object-oriented analysis of the data. For instance, the (uniqueness of) object can be defined in very complex ways in object-oriented "comparators" whereas in relational models, we are limited to the atomic data in table cells. It may also be argued that data type conversion, such as string to integer or date, is more robust in programming languages than in general database conversion functions. However, the fact that raw data is already in a spreadsheet (as opposed to text) allows us some control over input format since spreadsheet support fairly sophisticated programming languages like Visual Basic (which has been used in this project). Disadvantages include the fact that long database scripts cannot be conveniently run from within a web application (so, evaluation becomes a bit challenging i.e. need a database that supports stored procedures and web interface to it).

Object Oriented

Some of the initial ETL code for this project was, in fact, written to read the export from spreadsheet(s) into a java program, ingested as XML. The basic strategy was to generate "lists" of unique objects from the raw data establish all the referential integrity between these object lists and to then simply insert (persist) them into the database using a framework/environment like Hibernate or even embedded SQL. In my opinion, the biggest problem in this case would be the scalability. Databases are generally more powerful in this respect. If the raw dataset is too large, we may be doing too much in random access

memory prior to the insert. The complexity of programming is also formidable and, arguably, redundant and error-prone. In essence we will be duplicating in java what can be (and should be) done within a RDBMS. This is because many functions naturally supported by the RDBMS like selection, projection and referential integrity will have to be conducted in the application layer. However, as mentioned above this method can useful if the data set is small and there are complex object based methods that need to be applied to convert the raw data for persistence within the database. It also has the advantage of being driven more conveniently through a web application.

ETL Implementation

A key "research finding" is to artfully blend the two approaches discussed above. First, ingest raw data into a master table using object oriented approach, doing as much of data conversions, e.g. text to date, decimal etc, using robust frameworks in the application layer and next, use the RDBMS to do what it does best, namely selection and projection, to populate the schema from the master table.

As illustrated in Figure 2, the raw data, typically, in spreadsheets, text files etc, is first converted into some sort of markup language such as XML. The key insight for choice of markup is that it should be something that can be easily digested (transformed) into Data Access Objects (DAO) just by specification i.e. not having to write any code from scratch. Once transformed into DAO, the insert operation is provided by several persistence framework that support Object/Relational mapping such as Hibernate and Spring Frameworks.

To enhance object oriented code reuse, we developed a two letter code for each type of dataset. For example the climate process code "HR" stands for Heavy Rains, "HS" stands for High Seas and "SW" means "Strong Winds".

The raw datasets (Microsoft Excel Spreadsheet) for Heavy Rains and High Seas are at:

- http://www.pricip.org/HR.xls
- http://www.pricip.org/HS.xls

Each of the above spreadsheets has an embedded macro that generates the following XML output:

- http://www.pricip.org/HR.xml
- http://www.pricip.org/HS.xml

The XML to DAO digest rules are specified by the following:

- http://www.pricip.org/digest-rules-HR.xml
- http://www.pricip.org/digest-rules-HS.xml

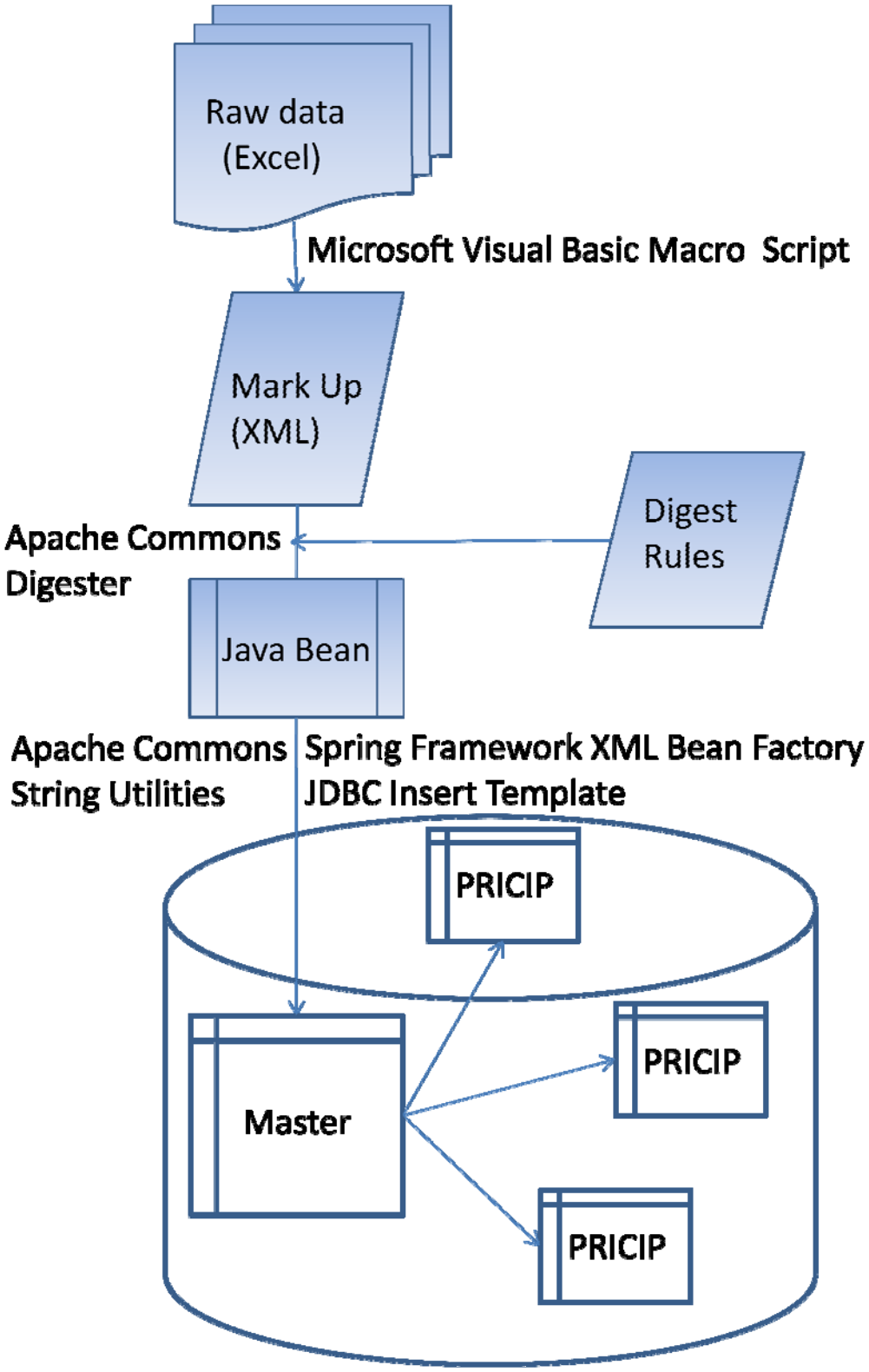


Figure 2: Extract Transform Load (ETL) Process Flow

In the initial stages of this project, the structure of raw datasets, XML output and digester rules are essentially identical, but the design, including E-R model and the above reusable, but polymorphic digesting process flow, allows for these process datasets to be very different from each other while sharing zero or more attributes. In other words, this ETL approach can be quickly adapted end-to-end for just about any type of dataset, without having to re-write a new parser.

The digest rules are applied to the XML using Apache Commons Digester framework:

- http://commons.apache.org/digester/

The application of digest rules to XML results in a "plain old java object" (POJO) that has a one-to-one mapping between its private member variables and the attributes of the master table within the RDBMS, which in turn, are the same as the column names in the raw data spreadsheet.

Using Spring Framework, we can write DAO to map and persist the POJO into the RDBMS in numerous ways depending on the complexity of the database operation and features supported by the RDBMS. More information on using simple JDBC insert can be found in various Spring References such as:

- http://static.springframework.org/spring/docs/2.5.x/reference/jdbc.html#jdbc-simple-jdbc

We can leverage sophisticated (robust) data conversion utilities available in the Apache Commons Lang package to match the data types in the database in a vendor independent fashion:

- http://commons.apache.org/lang/index.html

This converts pure text received from XML into appropriate objects for non-character database fields like DATE and DECIMAL as appropriate for the underlying RDBMS.

The raw data first appears in the database as a "master" table. A set of ETL queries then projects this raw data into various PRICIP schema tables using the power of SQL to eliminate redundancies, enforce referential integrity and to potentially check for inconsistencies. This is where the RDBMS is much more useful than any application framework (like Spring or Hibernate). For example, we can populate the contacts table by simply doing (this is just sample, for actual ETL code, see Appendix E):

```
INSERT INTO Contacts (person,phone,email) (
 SELECT person, phone, email
 FROM Master
 GROUP BY person, phone, email
)
```

Appendix D contains the Excel macro that generates XML from the spreadsheet. The java code to digest XML and insert into database is in Appendix E, followed by the complete script that conducts the ETL function of moving data from the master table into various tables comprising the PRICIP database schema.

For this project, the RDBMS is PostGreSQL Version 8.3 (http://www.postgresql.org/) which supports the simple insert function along with an auto increment key within the master table. However, PostGreSQL does not support retrieval of the auto increment key after the insert operation. Therefore, although Spring provides for a method for this, the user gets a run time error (just an example of limitation RDBMS places on application development). We are using JDBC Version 4.0 driver running on top of Java Version 6.0 Virtual Machine (VM). The database can from any vendor as long as the appropriate JDBC driver is included in class path and configuration file is updated.

The entire ETL process flow, from XML to a fully populated PRICIP schema in the RDBMS, can be executed is a single console command within the project directory:

```
ant
```

This is because all of the above steps are coded as a batch process within a build.xml script file with default target called “db.build” that executes the all the java code in a forked VM and uses JDBC to run SQL scripts in the proper sequence. The only “manual” step is to output the XML from spreadsheet and to place it in the project working directory which by default is c:\pricip, but is also configurable. The said build script is in appendices.

For more information on ant see:

- http://ant.apache.org/

Exposition Considerations

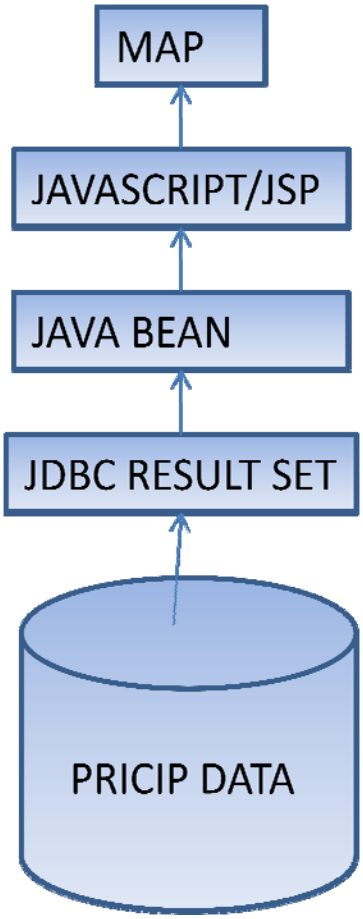


Figure 3: Exposition Process Flow

To view the data, a web form is used to submit a query to the database, which generates a JDBC result set. This result set is mapped into a java bean that is able to create the javascript variables within a server page. The javascript variables support map rendering in Google maps. This process flow ensures that data from within the PRICIP schema can be mapped.

The entry-point for the application is at:

http://www.pricip.org/ddp.php

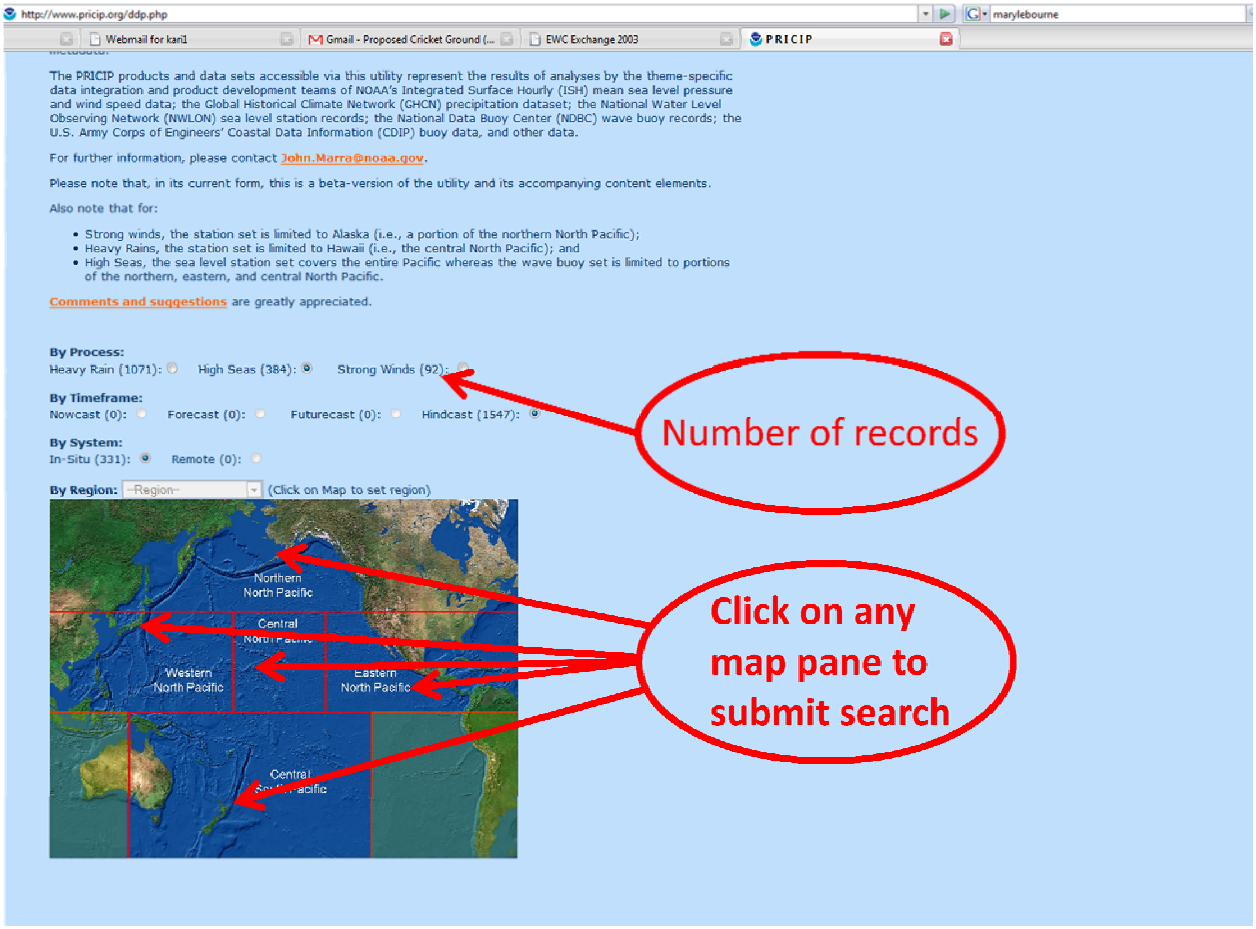


Figure 4: Application Launch Page (Entry Point: http://www.pricip.org/ddp.php)

The above application home page describes the utility in terms that the target audience will understand and presents a web form to navigate the database on a map. A high-level summary, showing the number of records for each category is using database connection by PHP.

```
$conn = pg_pconnect("host=localhost port=port dbname=pricip user=usr password=pwd");
if (!$conn)
{
  echo "Could not connect to database...";  exit;
}
Else
{
   $myresult = pg_exec($conn, "SELECT COUNT(*) FROM StrongWinds");
   $sw = pg_result($myresult, 0, 0); // get count for strong winds
   // and so on for all other data categories
```

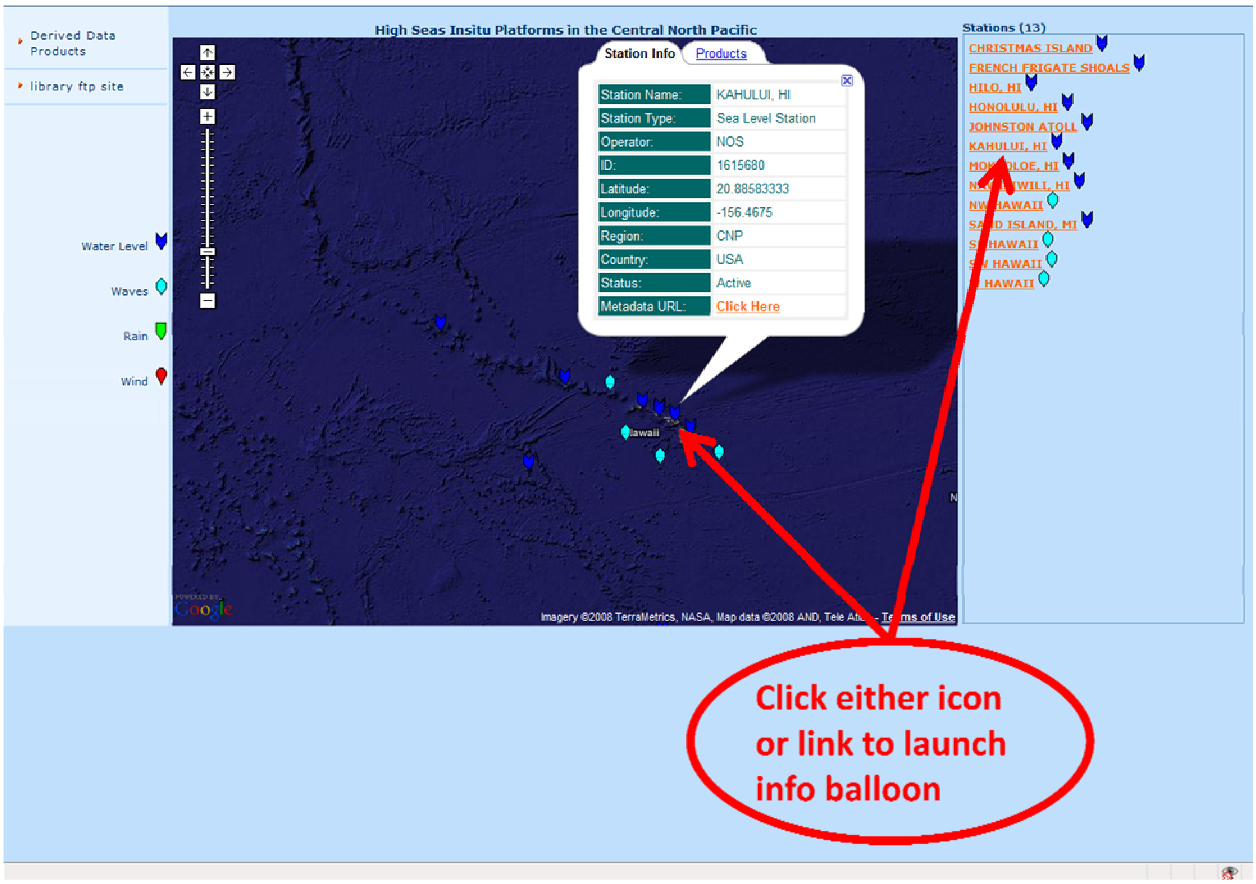


Figure 5: Google Map showing records extracted from database on a map.

The above page was mocked first mocked up with Google javascript API without any database backend. This can be done using XML (KML) or JSON (http://www.json.org). We are using JSON in this project.

```
var mapDataUrl = [{'latitude': 2.006846,'longitude': -157.487615,'IconImage':
'sls.png','markertab_info': '<table class="infotable"><tr><td class="infoLcolumn">Station
Name:</td><td class="infoRcolumn">CHRISTMAS ISLAND</td></tr><tr><td
class="infoLcolumn">Station Type:</td><td class="infoRcolumn">Sea Level
Station</td></tr><tr><td class="infoLcolumn">Operator:</td><td
class="infoRcolumn">UHSLC</td></tr><tr><td class="infoLcolumn">ID:</td><td
class="infoRcolumn">Unknown</td></tr><tr><td class="infoLcolumn">Latitude:</td><td
class="infoRcolumn">2.006846</td></tr><tr><td class="infoLcolumn">Longitude:</td><td
class="infoRcolumn">-157.487615</td></tr><tr><td class="infoLcolumn">Region:</td><td
class="infoRcolumn">CNP</td></tr><tr><td class="infoLcolumn">Country:</td><td
class="infoRcolumn">KIR</td></tr><tr><td class="infoLcolumn">Status:</td><td
class="infoRcolumn">Active</td></tr><tr><td class="infoLcolumn">Metadata URL:</td><td
class="infoRcolumn"><a href="http://ilikai.soest.hawaii.edu/uhslc/rqdssta.html"
target="_blank"> Click Here</a></td></tr></table>','markertab_data':'<table
class="infotable"><tr><td class="infoHeadcolumn" colspan="2">Patterns of
Variability</td></tr><tr><td colspan="2" align="center"><a
href="product.jsp?process=HS&region=CNP&product=VAR&id=1" target="_blank">Click
Here</a></td></tr></table>','name': 'CHRISTMAS ISLAND'},…
```

Once the balloon functionality is mocked up using JSON script (which can be reviewed by doing browser view source), the on application development challenge was to render the JSON script variable using a dynamic server page like PHP or JSP. We used JSP because java beans provide great deal of flexibility in generating the required output within the server page. So, basically, the application seamlessly jumps from the PHP "home-page" to JSP "map page" with the following (sample) HTTP GET

```
http://www.pricip.org/ingest/index.jsp?process=HS&region=CNP
```

The coding of java bean is perhaps too complicated to review, but suffice it to say that the above URL call to the "ingest" context of PRICIP with the parameters "process" and "region" will execute the following (sample) database read:

```
SELECT * FROM Insitu_Stations_HeavyRains_Hindcasts
WHERE region = 'CNP' ORDER BY stationname
```

Through use of pre-defined views in the ETL step and protecting this select operation with an if statement that executes only if the region is one of the five valid region codes, we prevent any sort of SQL injection (in addition to the fact that we are using java prepared statements).

The result set from the above SQL statement is mapped into objects that represent the station, products and Google map-markers. Each of these objects are java beans that are able to generated all the JSON, XML, KML or HTML needed in the JSP showing the map. For example, while Google maps uses JSON, the station links on the side bar use the exact same code as the icon. Therefore, we have the ability to "drive" the map from either the Google map or the stations list. Tool tips shown below facilitate navigation as well.

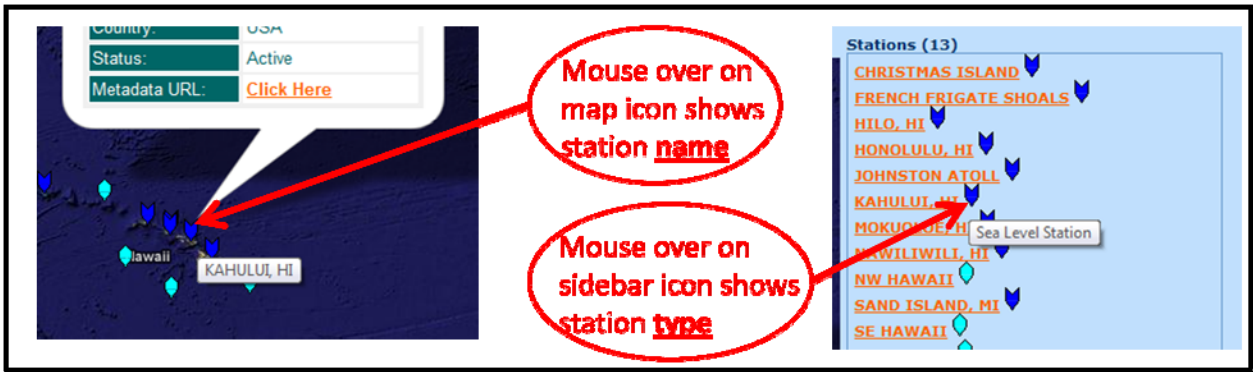


Figure 6: Tooltips

Clicking on the products tab, of the pop-up balloon leads to links that list all the products associated with a station. They are generated from the database using the following sample SQL:

```
SELECT * FROM ProductsView WHERE stationid =  27
```

The links from the Products Tab lead to a JSP that shows all the datasets (right now all datasets are represented by static plots) associated with the station-product combination. In this manner the material associated with a station can be infinitely nested to any level of data complexity since each of the links could be a web application with station, product and dataset parameters used to drill down to the required climatic information. The JSP is rendered with results from the following sample SQL:

```
SELECT * FROM ProductTypeView WHERE stationid = 27 AND producttype = 3
ORDER BY productid
```

Figure 7: Drill down from Products tab to individual datasets.

Evaluation

The simplest evaluation/testing is perhaps also the most likely use-case, which is hopefully self-explanatory (through help links within the web page):

1. http://www.pricip.org/ddp.php
   (Please carefully read the introduction and then scroll down to use the web-form below)
   (IMPORTANT: detailed help, with screen-shots, are available at "Help" link embedded in the web page)
2. Click on map to launch Google map page showing station markers (icons).
3. Click on any map icon (or sidebar link with station name). Mouse over icon shows station name.
4. Within balloon pop-up, please follows links in either "Station" tab" or "Products" tab.

This shows how practically an infinite numbers of climate products and associated datasets can be geo-coded (put on a map) according to the source station or platform that the data came in from. instead of just putting in a spreadsheet or a database, mapping was an optional nice to have feature which has been achieved in the project. Initially, we just wanted to accomplish a web-form based access but ended up exceeding our goals thanks to the Google Maps API.
From the administration perspective, basically, this model can be adapted to just about any spreadsheet (or equivalent "two-dimensional" dataset). By Stage 2, there were just two datasets, one for Heavy Rains and the other for High Seas. Now we already have a third, Strong Winds. To evaluate, we may add, delete or update the input spreadsheet(s) and this spreadsheet can be emailed to the administrator (me). I then load the data successfully (within minutes) by running the conversion into XML and from there into database scripts. If unsuccessful, I will discover the data inconsistency reply with error condition along the following lines "In your dataset these two tuples agree with w.r.t station name and product attributes but fail to agree on following attributes (functional dependency violation) or simple parse error etc". After these inconsistencies are fixed, we can prove that the process worked end-to-end by looking at the map in a development environment before applying the data patch to production. A sample interaction is discussed below (see next page).

After end-to-end development was complete, we actually added a new dataset, "Strong Winds" (SW), in addition to High Sea (HS) and Heavy Rains (HR), within a matter minutes. It was just a matter of creating and exporting the Strong Winds (SW) spreadsheet in XML, creating a digest file called http://www.pricip.org/digest-rules-SW.xml and running the ETL script. Then, all the string winds data could be mapped into its own master file as shown in Figure 2 (on Page 6) after which the ETL proceeds normally as for Heavy Rains and High Seas.

Further during the ETL, I was able to trap several data inconsistencies by creating and using strategic table views. For persistent, large scale errors (that could be fixed with a single SQL command), I introduced temporary scripts, instead of manually fixing the source file. Otherwise, we would have missed the deadline!

```
CREATE VIEW DifferentRegions AS
(
  SELECT a.stationname, a.latitude, a.longitude,
         a.region AS region1, b.region AS region2
   FROM  Insituview a, Insituview b
   WHERE a.stationname = b.stationname
   AND a.latitude = b.latitude
   AND a.longitude = b.longitude
   AND a.region <> b.region
)
;

UPDATE Master_Processes SET region = 'NNP'
WHERE station_name IN (SELECT DISTINCT stationname AS station_name
                       FROM DifferentRegions)
;
DROP VIEW IF EXISTS DifferentRegions
;
```

Figure 8: Sample SQL script dealing with "large-scale" logical inconsistency in raw data

For example, in Figure 8, a temporary (until raw data is fixed, and then dropped immediately) script will fix a situation where several hundred records had their regions messed up or missing in the hand-code spreadsheet. In all cases the policy was to add a default region NNP. Please note that this add on script will do absolutely nothing if used *after* the data anomaly is fixed since the DifferentRegions view will be empty.

Conclusions and Future

This project develops a methodology, design and implementation of a utilities suite that ingests vast and complex data with many possible redundancies and inconsistencies into an RDBMS. A simple visual inspection of the spreadsheets reveals the problem of redundancy in fields like contact person and potential problem of referential integrity posed by data fields like country, region etc. This is the fundamental problem being solved by putting this data into an RDBMS. The process of putting raw data into the RDBMS is called Extract Transform Load (ETL). This project also shows how various climate products and associated datasets can be put on a map. This shows how practically an infinite numbers of climate products and associated datasets can be geo-coded (put on a map) according to the source station or platform that the data came in from. Instead of just putting in a spreadsheet or a database, mapping was an optional nice to have feature which has been achieved in the project.

Possible future enhancements include:

- Initially, we just wanted to accomplish a web-form based access but ended up exceeding our goals thanks to the Google Maps API. However, now that mapping is possible, we are going to change the selection process to feature the map as the central navigation element of the tool with all other "layers" being on a sidebar menu. The web-form will most likely go away.
- Another challenge will be to extend the map paradigm to remote sensing stations. While we have a free product like Google Maps for terrestrial stations, perhaps need to look into some GUI from NASA to plot remote platforms.
- Control table field size declarations more tightly according to raw data. Right now, doing it by trial and error based on whether 1500 records are ingested successfully. Probably wasting a lot of space since the errors just say "failed on varchar(100)" in which case all the varchar(100) need to doubled to 200. Instead we could write some code to actually author the create table script based on actual maximum field lengths. Temporary measure could be to use unique lengths for all field during trial-and-error phase so at least we know which field is causing errors instead of having to expand all varchar(100).
- Build various application specific consistency checks , classified as fatal, warnings, etc (for common sense error checking) before, after and during the ETL process. An example, of "during" would be when inserting contacts to check (manually, if necessary) if one person has two emails (plausible, but most likely a typo in source spreadsheet)…this would be a warning in log files which should then be followed by up a manual check.
- User Interface to not just query data but to insert new data, possibly support various batch inserts. Also interfaces to conduct various SQL driven quality checks (to ensure data conforms to the basic assumption, no user errors etc) on the raw data before doing ETL to prevent garbage in, garbage out.
- A more fluent build process that addresses simultaneous builds for both development and production servers, externalizes all hard-coded references to main host and also conducts backups, archives.

- Introduce new tables as necessary for better data model (this is just a partial list): Station Types (types, description, icon), new master tables for each process, even if they are identical. Introduction of tables views that correspond to actual google map views, perhaps materialize them. Especially (and this can perhaps be accomplished in the application layers first/more conveniently), compute the extent and zoom of the google maps dynamically using the latitude, longitude on the markers extracted from the database…see if there is a out-of-the-box google function for that.
- Explore use of Hibernate to perhaps create a more useful ***bottoms-up*** application layer (using lessons learnt from Team Aloha team project). Then use Struts/MVC to build the application layer. Avoid scripting within the jsp. Also look into Faces, NetBeans.
- Improve Google Maps. Dynamically determine zoom and extent of google maps a request time based on the marker coordinates being extracted. Trying to get own map server to perhaps prevent occasional malfunction due to server at Google. Explore using the new version AJAX api to perhaps improve the performance. Achieve more dynamic data binding. Investigate Rails, Ruby etc.
- Improve performance, scalability. Looking into database indexing, query optimization etc. Install database on its own server and look into Oracle.
- If no data, do not go to maps page, simply pop-up a javascript that says no data. For every combination of data selection possible, do a select count(*) from the database and populate the launch page with the counts so as to prevent puzzled users from going where there is no data. Dynamically (using AJAX perhaps) tile the map in such a way that it shows only panes/area that are active.
- Security. Completely block any cross site scripting, ensure fromPage is from DDP home and that it is a human being doing a submission, not a bot (SQL Injection has already been addressed).
- Automate the ETL error checking end-to-end. Catch common mistakes. Do several more “normalization” checks, trap logical errors. Independently scrub the initial XML generated from spreadsheets before using the digester/parser. This eliminates the common problem of trailing or leading whitespace within character data tag (CDATA).
- Create a database model for a web site monitor that tracks usage of the application from apache and tomcat server logs.

Appendix A

Semantic Domain Description

PRICIP Portal Product Metadata Schema Template (V1a)

- **Product ID**
- **Product Class** *Tailored Information Product,* ***Derived Data Product***
- **Timeframe** ***Hindcast****, Futurecast, Forecast, Nowcast*
- **Region** *NNP, ENP, WNP, CNP, CSP*
- **Country** *ISO 3 digit code*
-
- **Theme** *SW, HR, HS*
- **Platform** ***In-situ****, Remote*
  - Insitu Station/System Information
    - Station Type *(met station, sea level station, wave buoy, etc.)*
    - Unique PRICIP ID/Name *###*
    - Station Name *Colombo-B*
    - Location (point)
      - Lat *N 6o 57' 49.08*
      - Long *E 79o 51' 27.49"*
    - Station Owner/Operator *UHSLC*
    - Station Operator ID (local ID) GLOSS02
    - Contact Information…(name, email, phone)
    - Active *Y/N*
    - Station Metadata *Y/N*
      - Y-URL ……..
- **Product Parameter-index** *annual maxima…*
  - Product Type *trend, variability, extremes…*
    - Product Name …
      - Product Source/Owner/Creator *UHSLC…*
      - Host Location/File Name *.url…*
      - Product Format *.jpg…*
      - About this product....Product Metadata*… text description of the data or product that explains the nature of the data or product and, where applicable, how it was derived*
    - Contact Information…(name, email, phone)

- **Data** *Real-Time or Near-Real Time (raw)*, ***Archived*** *(original qa/qc'd source), or* ***Derived*** *(modified)*

  - Derived (modified)
    - Source Name *COOPS annual maxima*
    - Location (see below- access the data) *url…*
    - Formats *ascii…*
    - Compliant Metadata *Yes or No*
      - Last Update *DD:MM:YY*
      - Location *url…*

  - Archived (original source, qa/qc'd)
    - Source Name *COOPS NWLON*
    - Location (see below- access the data) *url…*
    - Formats *ascii…*
    - Compliant Metadata *Yes or No*
      - Last Update *DD:MM:YY*
      - Location *url…*

Region	NNP – northern north pacific ENP – eastern north pacific WNP – western north pacific CNP– central north pacific CSP– central south pacific
Theme	SW– strong winds HR– heavy rains HS– high seas

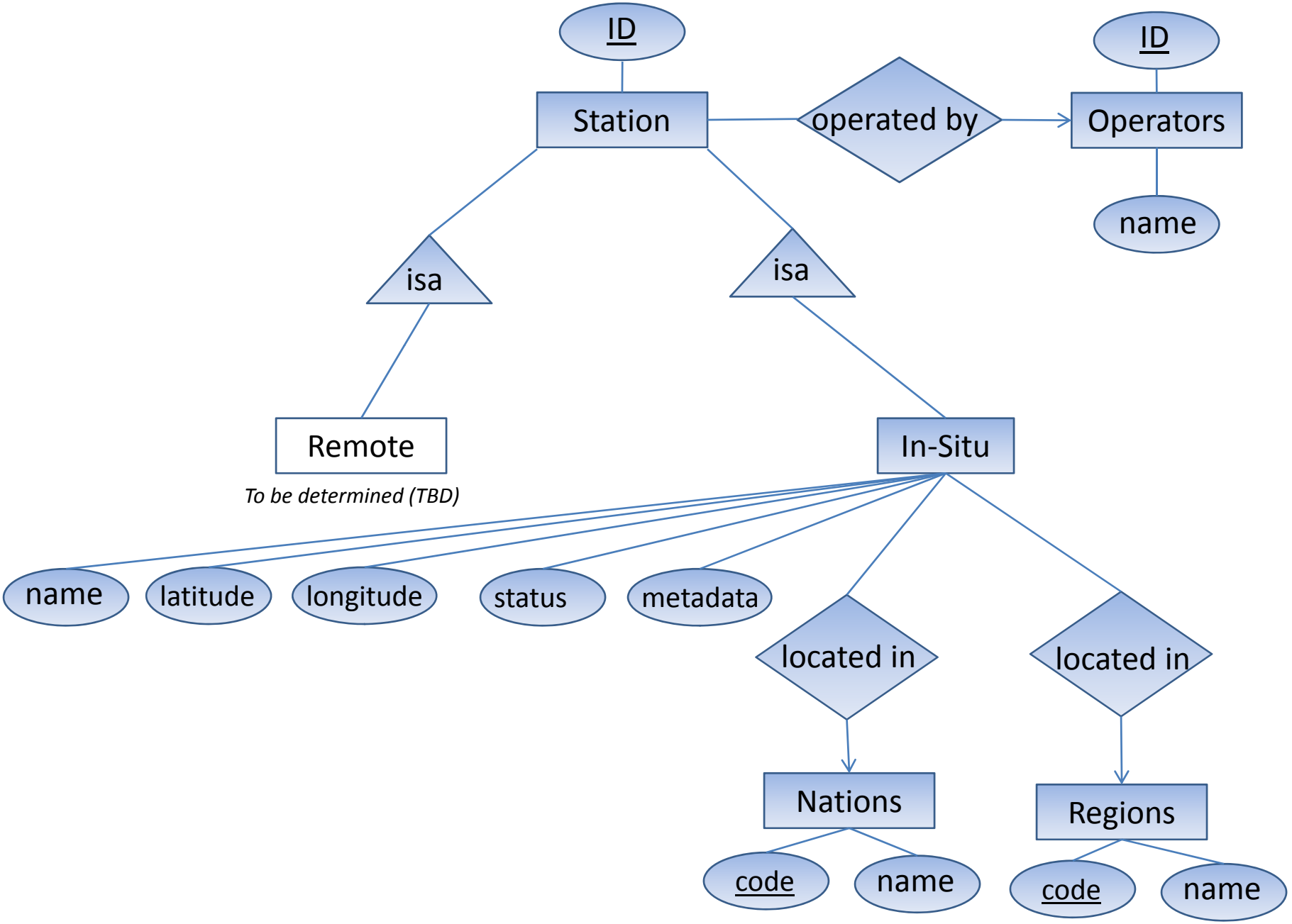
Appendix B

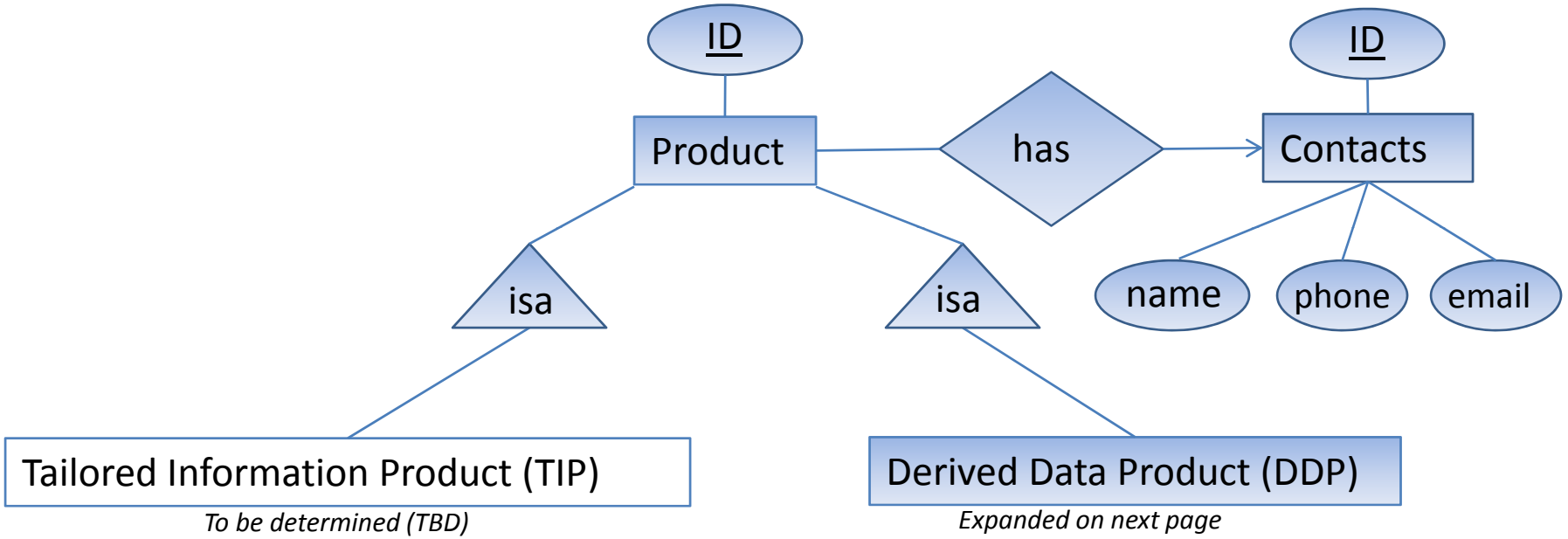
Full E-R Diagram

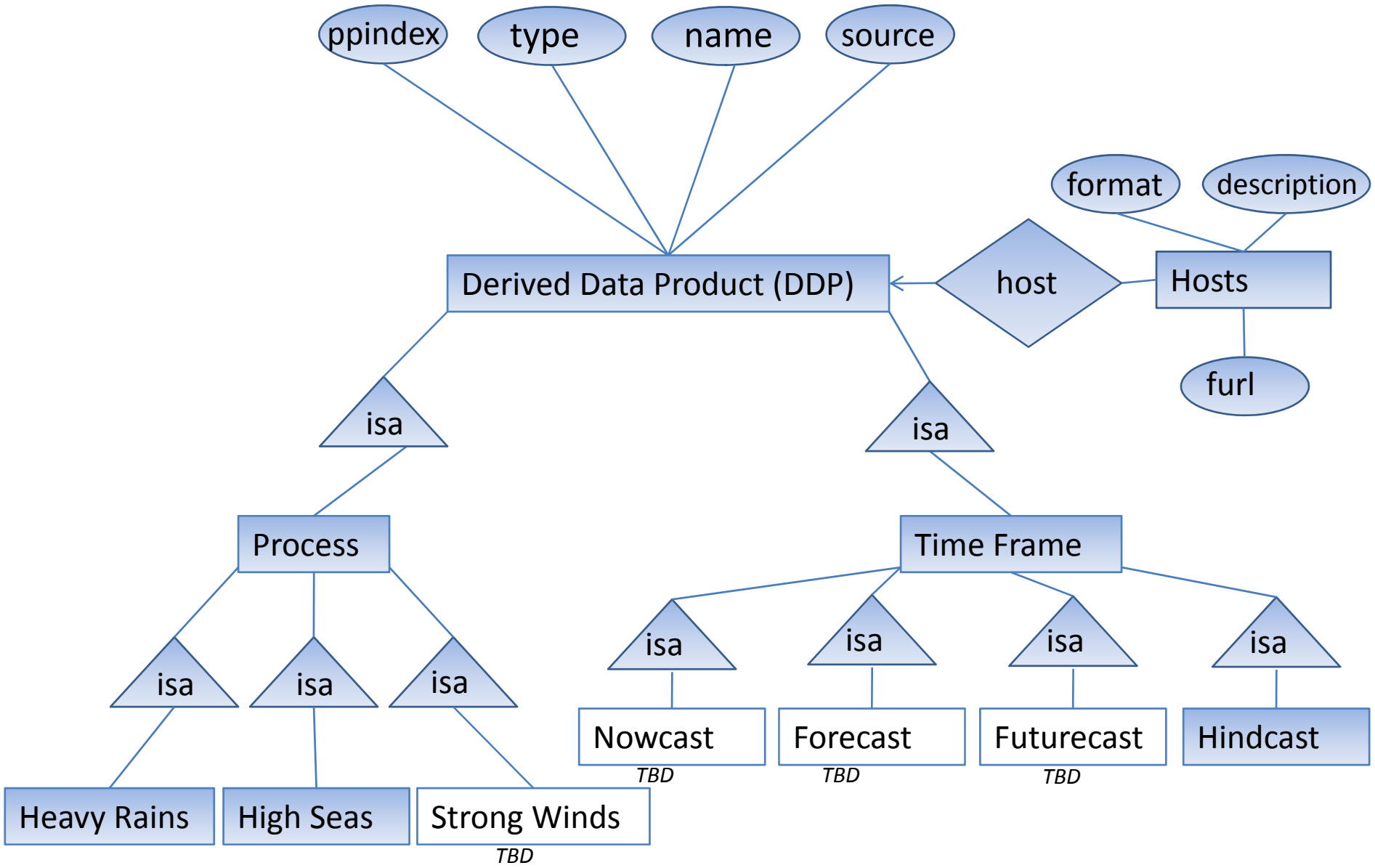
Please note that not all entities are fully fleshed out yet in terms of attributes, just the ones for which the team has data at this stage. Such entity sets are just placeholders for possible future extensions of the database (and depicted as empty boxes) and are not included in the implementation.

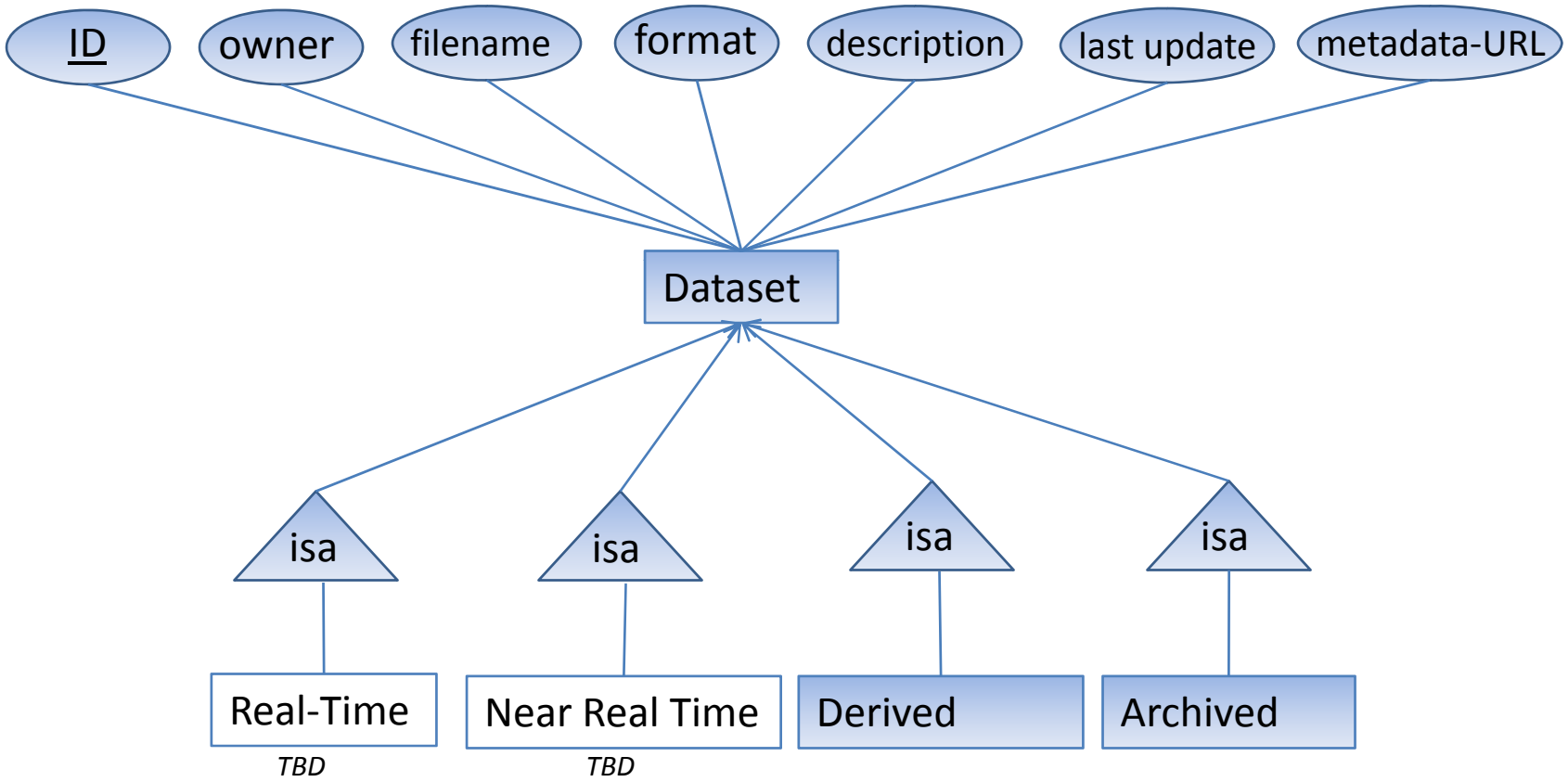
HIGH LEVEL ENTITY RELATIONSHIP MODEL

A sensor **station**(such as meteorological station, tide gauge or wave buoy) generates one or more climatic **products** that comprise one or more **datasets** whose metadata is stored within this database. Attributes are omitted for clarity of high-level ER diagram. Each of the above entities are further modeled in detail in the following pages of this appendix.


ID
Station
operated by
ID
Operators
name
isa
isa
Remote
To be determined (TBD)
In-Situ
name
latitude
longitude
status
metadata
located in
located in
Nations
Regions
code
name
code
name


ID
Product
has
ID
Contacts
isa
isa
name
phone
email
Tailored Information Product (TIP)
To be determined (TBD)
Derived Data Product (DDP)
Expanded on next page

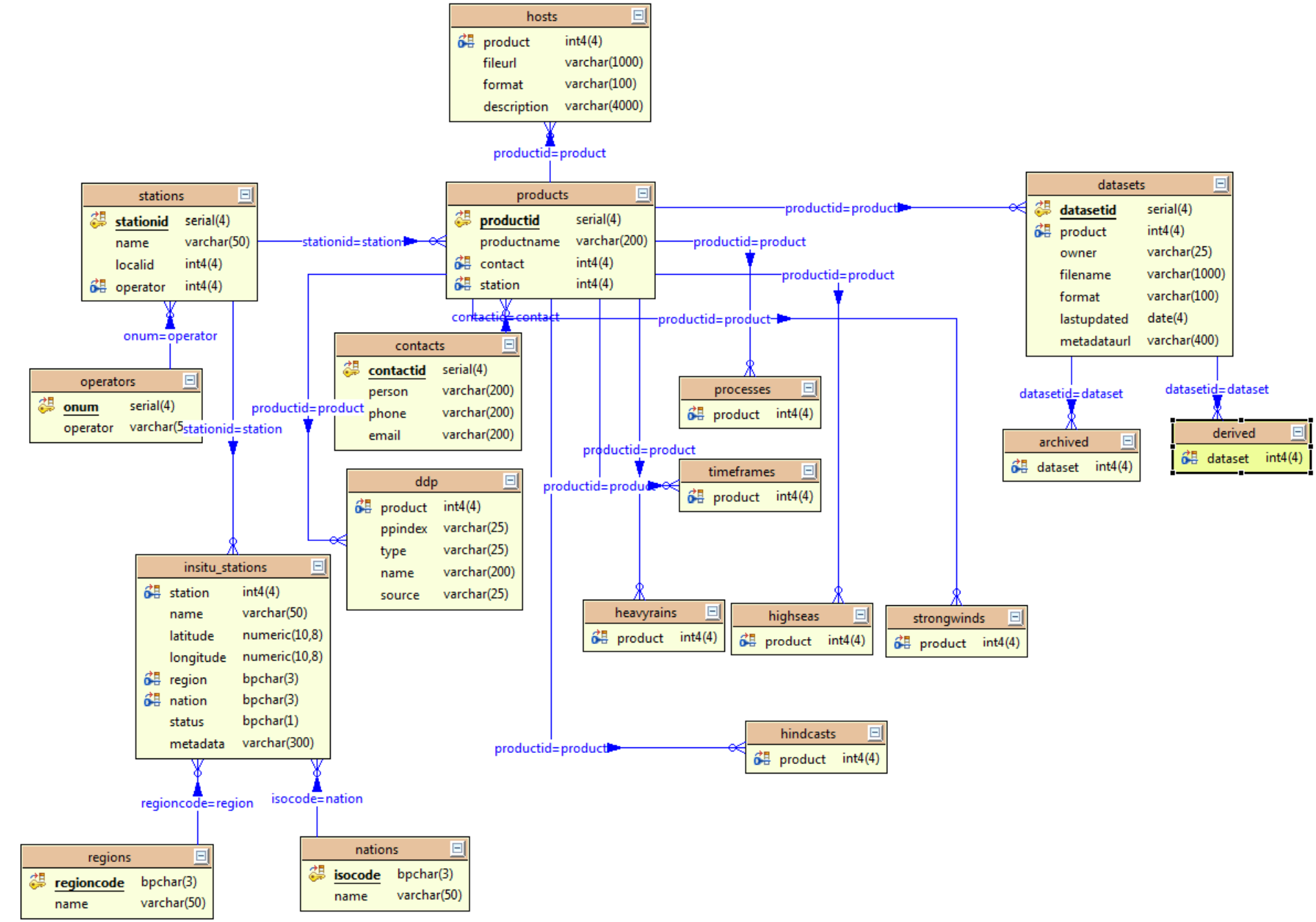

ppindex
type
name
source
format
description
Derived Data Product (DDP)
host
Hosts
furl
isa
isa
Process
Time Frame
isa
isa
isa
isa
isa
isa
isa
Heavy Rains
High Seas
Strong Winds
TBD
Nowcast
TBD
Forecast
TBD
Futurecast
TBD
Hindcast


ID
owner
filename
format
description
last update
metadata-URL
Dataset
isa
isa
isa
isa
Real-Time
TBD
Near Real Time
TBD
Derived
Archived

Appendix C

Relational Model

RELATIONAL MODEL FOR PRICIP DERIVED DATA PRODUCTS



Appendix D

Excel-to-XML VBA Script

Module2 - 1

```
Public Function ExportToXML(FullPath As String, RowName _
  As String) As Boolean

'PURPOSE: EXPORTS AN EXCEL SPREADSHEET TO XML
'PARAMETERS: FullPath: Full Path of File to Export Sheet to
'             RowName: XML Attribute Name to give to each row

'RETURNS: True if Successful, false otherwise

'EXAMPLE: ExportToXML "C:\mysheet.xml", "Employee"

'NOTES:
'This function has the following quirks and limitations.
'If you find that they are not consistent with the behavior
'you desire for your solution, you should be able to
'modify the code without too much difficulty

'       1) Designed to be used inside Excel as a macro
'        not with VB.  If you want to use from VB
'        Add code to use Excel Object model
'
'       2) This snippet works with the
'          the first worksheet in the workbook.
'          If you want to make this a variable,
'          You can change the code to add the worksheet
'          Number as a parameter.
'
'       3) This code uses the worksheet name as the top-level
'          XML attribute.
'
'       4) The first row of the sheet is assumed to contain the
'          attribute (column) names, while the following rows
'          are assumed to contained the data values
'
'       5) No data for blank cells are written to the
'          XML file.
'
'       6) The CDATA attribute is included with each value
'
'       7) The function assumes that the first column of
'          each row in the sheet has a value.  If it finds a
'          blank first column it exits.  This is in order
'          to prevent it from printing blank row
'*************************************************

On Error GoTo ErrorHandler


Dim colIndex As Integer
Dim rwIndex As Integer
Dim asCols() As String
Dim oWorkSheet As Worksheet
Dim sName As String
Dim lCols As Long, lRows As Long
Dim iFileNum As Integer


Set oWorkSheet = ThisWorkbook.Worksheets(1)
sName = oWorkSheet.Name
lCols = oWorkSheet.Columns.Count
lRows = oWorkSheet.Rows.Count
lRows = 1090


ReDim asCols(lCols) As String

iFileNum = FreeFile
Open FullPath For Output As #iFileNum

For i = 0 To lCols - 1
    'Assumes no blank column names
```

```
    If Trim(Cells(2, i + 1).Value) = "" Then Exit For
    asCols(i) = Cells(2, i + 1).Value
Next i

If i = 0 Then GoTo ErrorHandler
lCols = i

Print #iFileNum, "<?xml version=""1.0""?>"
Print #iFileNum, "<" & sName & ">"
For i = 11 To lRows
'If Trim(Cells(i, 1).Value) = "" Then Exit For
Print #iFileNum, "<" & RowName & ">"

    For j = 1 To lCols

        If Trim(Cells(i, j).Value) <> "" Then
           Print #iFileNum, "  <" & asCols(j - 1) & "><![CDATA[";
           Print #iFileNum, Trim(Cells(i, j).Value);
           Print #iFileNum, "]]></" & asCols(j - 1) & ">"
           DoEvents 'OPTIONAL
        Else
           Print #iFileNum, "  <" & asCols(j - 1) & "/>"
        End If
    Next j
    Print #iFileNum, " </" & RowName & ">"
Next i

Print #iFileNum, "</" & sName & ">"
ExportToXML = True
ErrorHandler:
If iFileNum > 0 Then Close #iFileNum
Exit Function
End Function

Sub generateXML()
  ExportToXML "C:\mysheet-HR.xml", "Product"
End Sub
```

Appendix E

ETL Scripts

```
-- This create the master table required for
-- capturing the raw data from external source
DROP VIEW IF EXISTS ContactsView
;
DROP VIEW IF EXISTS DDPView
;
DROP VIEW IF EXISTS HostView
;
DROP VIEW IF EXISTS InsituView
;
DROP VIEW IF EXISTS OperatorsView
;
DROP VIEW IF EXISTS ArchivedView
;
DROP VIEW IF EXISTS Insitu_Stations_HighSeas_Hindcasts
;
DROP VIEW IF EXISTS Insitu_Stations_HeavyRains_Hindcasts
;
DROP TABLE IF EXISTS MASTER_PROCESSES
;
CREATE TABLE MASTER_PROCESSES
       (RECORD_NUMBER      SERIAL PRIMARY KEY,
        PROCESS_CODE       VARCHAR(25),
        PRODUCT_CLASS      VARCHAR(25),
        TIMEFRAME          VARCHAR(25),
        THEME              VARCHAR(25),
        REGION             VARCHAR(25),
        COUNTRY            VARCHAR(25),
        PUBLISH_DATE       DATE,
        PLATFORM_TYPE      VARCHAR(25),
        STATION_ID         VARCHAR(20),
        STATION_TYPE       VARCHAR(20),
        STATION_NAME       VARCHAR(30),
        LATITUDE           DECIMAL(12,8),
        LONGITUDE          DECIMAL(12,8),
        OPERATOR           VARCHAR(25),
        OPERATORID         VARCHAR(25),
        ACTIVE             VARCHAR(25),
        METADATA           VARCHAR(25),
        METADATA_URL       VARCHAR(200),
        PRODUCT_ID         VARCHAR(20),
        PPINDEX            VARCHAR(20),
        PRODUCT_TYPE       VARCHAR(25),
        PRODUCT_NAME       VARCHAR(200),
        PRODUCT_SOURCE     VARCHAR(20),
        HOST_LOCATION      VARCHAR(200),
        PRODUCT_FORMAT     VARCHAR(10),
        ABOUT              VARCHAR(200),
        PERSON             VARCHAR(200),
        PHONE              VARCHAR(200),
        EMAIL              VARCHAR(200),
        SOURCETYPE3        VARCHAR(25),
        DATAID3            VARCHAR(25),
        SOURCEID3          VARCHAR(25),
        LOCATION3          VARCHAR(200),
        FORMAT3            VARCHAR(10),
        METADATACOMPLIANT3 VARCHAR(25),
        METALASTUPDATE3    DATE,
        METADATAURL3       VARCHAR(200),
        SOURCETYPE2        VARCHAR(25),
        DATAID2            VARCHAR(25),
        SOURCEID2          VARCHAR(25),
```

```
        LOCATION2           VARCHAR(200),
        FORMAT2             VARCHAR(10),
        METADATACOMPLIANT2 VARCHAR(25),
        METALASTUPDATE2     DATE,
        METADATAURL2        VARCHAR(200))
;
```

```
-- ////////////////////////////////////////////////////
-- //  P R I C I P   S C H E M A   C R E A T E   S C R I P T
-- //
--     Relation names are slightly different from data model.
--     Some extra attributes, convenience tables are
--     included to facilitate ETL.  Wherever the relational
--     model differs from formal E-R diagram there is a comment
--
-- First, drop all tables in the reverse order of
-- them being created to ensure no dangling
-- references
--
--
DROP TABLE IF EXISTS DERIVED
;
DROP TABLE IF EXISTS ARCHIVED
;
DROP TABLE IF EXISTS DATASETS
;
DROP TABLE IF EXISTS HEAVYRAINS
;
DROP TABLE IF EXISTS HIGHSEAS
;
DROP TABLE IF EXISTS STRONGWINDS
;
DROP TABLE IF EXISTS HINDCASTS
;
DROP TABLE IF EXISTS PROCESSES
;
DROP TABLE IF EXISTS TIMEFRAMES
;
DROP TABLE IF EXISTS DDP
;
DROP TABLE IF EXISTS HOSTS
;
DROP TABLE IF EXISTS PRODUCTS
;
DROP TABLE IF EXISTS INSITU_STATIONS
;
DROP TABLE IF EXISTS NATIONS
;
DROP TABLE IF EXISTS REGIONS
;
DROP TABLE IF EXISTS STATIONS
;
DROP TABLE IF EXISTS OPERATORS
;
DROP TABLE IF EXISTS CONTACTS
;
--
--  BEGIN CREATING TABLES
--  TO DO: OPTIMIZE FIELD DECLARATIONS
--         BASED ON CONTENTS OF MASTER TABLES
--         WHATEVER HAS TO FAIL WILL DO SO
--         WHEN IMPORTING RAW DATA.
--         THIS SCHEMA CAN BE SAFELY MINIMAL
--         THEREBY MAKING APPLICATIONS EFFICIENT.
--
CREATE TABLE CONTACTS
      (CONTACTID      SERIAL PRIMARY KEY,
       PERSON         VARCHAR(200),
       PHONE          VARCHAR(200),
```

```
       EMAIL          VARCHAR(200))
;
CREATE TABLE OPERATORS
      (ONUM           SERIAL PRIMARY KEY,
       OPERATOR       VARCHAR(50))
;
CREATE TABLE STATIONS
      (STATIONID      SERIAL PRIMARY KEY,
--
--     NAME enables putting key stationid into this tables "isa" child INSITU_STATIONS
--
       STATIONNAME    VARCHAR(50),
--
--     LOCALID is just an optional organization specific ID given by operator.
--             Has no significance in the data model, just preserving information
provided.
--
       LOCALID        VARCHAR(25),
       ONUM           INTEGER REFERENCES OPERATORS(ONUM))
;
CREATE TABLE REGIONS
      (REGIONCODE     CHAR(3) NOT NULL PRIMARY KEY,
       REGIONNAME     VARCHAR(50))
;
CREATE TABLE NATIONS
      (ISOCODE        CHAR(3) NOT NULL PRIMARY KEY,
       COUNTRYNAME    VARCHAR(50))
;
CREATE TABLE INSITU_STATIONS
      (STATIONID      INTEGER REFERENCES STATIONS(STATIONID),
       STATIONNAME     VARCHAR(50),
       LATITUDE       DECIMAL(12,8),
       LONGITUDE      DECIMAL(12,8),
       REGION         CHAR(3) REFERENCES REGIONS(REGIONCODE),
       NATION         CHAR(3) REFERENCES NATIONS(ISOCODE),
--
--     STATUS A=Active, I=Inactive, U=Unknown
--
       STATUS         CHAR(1) CHECK (STATUS IN ('A', 'I', 'U')),
       METADATA       VARCHAR(300))
;
CREATE TABLE PRODUCTS
      (PRODUCTID      SERIAL PRIMARY KEY,
       PRODUCTNAME    VARCHAR(200),
       CONTACTID      INTEGER REFERENCES CONTACTS(CONTACTID),
       STATIONID      INTEGER REFERENCES STATIONS(STATIONID))
;
CREATE TABLE HOSTS
      (PRODUCTID      INTEGER REFERENCES PRODUCTS(PRODUCTID),
       FILEURL        VARCHAR(1000),
       FORMAT         VARCHAR(100),
       DESCRIPTION    VARCHAR(4000))
;
CREATE TABLE DDP
      (PRODUCTID      INTEGER REFERENCES PRODUCTS(PRODUCTID),
       PPINDEX        VARCHAR(25),
       DDPTYPE        VARCHAR(25),
       DDPNAME        VARCHAR(200),
       DDPSOURCE      VARCHAR(25))
;
CREATE TABLE TIMEFRAMES
      (PRODUCTID      INTEGER REFERENCES PRODUCTS(PRODUCTID))
```

```
;
CREATE TABLE PROCESSES
      (PRODUCTID      INTEGER REFERENCES PRODUCTS(PRODUCTID))
;
CREATE TABLE HINDCASTS
      (PRODUCTID      INTEGER REFERENCES PRODUCTS(PRODUCTID))
;
CREATE TABLE HIGHSEAS
      (PRODUCTID      INTEGER REFERENCES PRODUCTS(PRODUCTID))
;
CREATE TABLE HEAVYRAINS
      (PRODUCTID      INTEGER REFERENCES PRODUCTS(PRODUCTID))
;
CREATE TABLE STRONGWINDS
      (PRODUCTID      INTEGER REFERENCES PRODUCTS(PRODUCTID))
;
CREATE TABLE DATASETS
      (DATASETID      SERIAL PRIMARY KEY,
       PRODUCTID      INTEGER REFERENCES PRODUCTS(PRODUCTID),
       OWNER          VARCHAR(25),
       FILENAME       VARCHAR(1000),
       FORMAT         VARCHAR(100),
       LASTUPDATED    DATE,
       METADATAURL    VARCHAR(400))
;
CREATE TABLE ARCHIVED
      (DATASETID      INTEGER REFERENCES DATASETS(DATASETID))
;
CREATE TABLE DERIVED
      (DATASETID      INTEGER REFERENCES DATASETS(DATASETID))
;
```

```
INSERT INTO Regions (regioncode, regionname) VALUES ('NNP','Northern North Pacific');
INSERT INTO Regions (regioncode, regionname) VALUES ('ENP','Eastern North Pacific');
INSERT INTO Regions (regioncode, regionname) VALUES ('WNP','Western North Pacific');
INSERT INTO Regions (regioncode, regionname) VALUES ('CNP','Central North Pacific');
INSERT INTO Regions (regioncode, regionname) VALUES ('CSP','Central South Pacific');
```

```
INSERT INTO Nations (isocode, countryname) VALUES ('ABW','Aruba');
INSERT INTO Nations (isocode, countryname) VALUES ('AFG','Afghanistan');
INSERT INTO Nations (isocode, countryname) VALUES ('AGO','Angola');
INSERT INTO Nations (isocode, countryname) VALUES ('AIA','Anguilla');
INSERT INTO Nations (isocode, countryname) VALUES ('ALA','Åland Islands');
INSERT INTO Nations (isocode, countryname) VALUES ('ALB','Albania');
INSERT INTO Nations (isocode, countryname) VALUES ('AND','Andorra');
INSERT INTO Nations (isocode, countryname) VALUES ('ANT','Netherlands Antilles');
INSERT INTO Nations (isocode, countryname) VALUES ('ARE','United Arab Emirates');
INSERT INTO Nations (isocode, countryname) VALUES ('ARG','Argentina');
INSERT INTO Nations (isocode, countryname) VALUES ('ARM','Armenia');
INSERT INTO Nations (isocode, countryname) VALUES ('ASM','American Samoa');
INSERT INTO Nations (isocode, countryname) VALUES ('ATA','Antarctica');
INSERT INTO Nations (isocode, countryname) VALUES ('ATF','French Southern Territories');
INSERT INTO Nations (isocode, countryname) VALUES ('ATG','Antigua and Barbuda');
INSERT INTO Nations (isocode, countryname) VALUES ('AUS','Australia');
INSERT INTO Nations (isocode, countryname) VALUES ('AUT','Austria');
INSERT INTO Nations (isocode, countryname) VALUES ('AZE','Azerbaijan');
INSERT INTO Nations (isocode, countryname) VALUES ('BDI','Burundi');
INSERT INTO Nations (isocode, countryname) VALUES ('BEL','Belgium');
INSERT INTO Nations (isocode, countryname) VALUES ('BEN','Benin');
INSERT INTO Nations (isocode, countryname) VALUES ('BFA','Burkina Faso');
INSERT INTO Nations (isocode, countryname) VALUES ('BGD','Bangladesh');
INSERT INTO Nations (isocode, countryname) VALUES ('BGR','Bulgaria');
INSERT INTO Nations (isocode, countryname) VALUES ('BHR','Bahrain');
INSERT INTO Nations (isocode, countryname) VALUES ('BHS','Bahamas');
INSERT INTO Nations (isocode, countryname) VALUES ('BIH','Bosnia and Herzegovina');
INSERT INTO Nations (isocode, countryname) VALUES ('BLM','Saint Barthélemy');
INSERT INTO Nations (isocode, countryname) VALUES ('BLR','Belarus');
INSERT INTO Nations (isocode, countryname) VALUES ('BLZ','Belize');
INSERT INTO Nations (isocode, countryname) VALUES ('BMU','Bermuda');
INSERT INTO Nations (isocode, countryname) VALUES ('BOL','Bolivia');
INSERT INTO Nations (isocode, countryname) VALUES ('BRA','Brazil');
INSERT INTO Nations (isocode, countryname) VALUES ('BRB','Barbados');
INSERT INTO Nations (isocode, countryname) VALUES ('BRN','Brunei Darussalam');
INSERT INTO Nations (isocode, countryname) VALUES ('BTN','Bhutan');
INSERT INTO Nations (isocode, countryname) VALUES ('BVT','Bouvet Island');
INSERT INTO Nations (isocode, countryname) VALUES ('BWA','Botswana');
INSERT INTO Nations (isocode, countryname) VALUES ('CAF','Central African Republic');
INSERT INTO Nations (isocode, countryname) VALUES ('CAN','Canada');
INSERT INTO Nations (isocode, countryname) VALUES ('CCK','Cocos (Keeling) Islands');
INSERT INTO Nations (isocode, countryname) VALUES ('CHE','Switzerland');
INSERT INTO Nations (isocode, countryname) VALUES ('CHL','Chile');
INSERT INTO Nations (isocode, countryname) VALUES ('CHN','China');
INSERT INTO Nations (isocode, countryname) VALUES ('CIV','Côte d''Ivoire');
INSERT INTO Nations (isocode, countryname) VALUES ('CMR','Cameroon');
INSERT INTO Nations (isocode, countryname) VALUES ('COG','Congo');
INSERT INTO Nations (isocode, countryname) VALUES ('COK','Cook Islands');
INSERT INTO Nations (isocode, countryname) VALUES ('COL','Colombia');
INSERT INTO Nations (isocode, countryname) VALUES ('COM','Comoros');
INSERT INTO Nations (isocode, countryname) VALUES ('CPV','Cape Verde');
INSERT INTO Nations (isocode, countryname) VALUES ('CRI','Costa Rica');
INSERT INTO Nations (isocode, countryname) VALUES ('CUB','Cuba');
INSERT INTO Nations (isocode, countryname) VALUES ('CXR','Christmas Island');
INSERT INTO Nations (isocode, countryname) VALUES ('CYM','Cayman Islands');
INSERT INTO Nations (isocode, countryname) VALUES ('CYP','Cyprus');
INSERT INTO Nations (isocode, countryname) VALUES ('CZE','Czech Republic');
INSERT INTO Nations (isocode, countryname) VALUES ('DEU','Germany');
INSERT INTO Nations (isocode, countryname) VALUES ('DJI','Djibouti');
INSERT INTO Nations (isocode, countryname) VALUES ('DMA','Dominica');
INSERT INTO Nations (isocode, countryname) VALUES ('DNK','Denmark');
INSERT INTO Nations (isocode, countryname) VALUES ('DOM','Dominican Republic');
```

```
INSERT INTO Nations (isocode, countryname) VALUES ('DZA','Algeria');
INSERT INTO Nations (isocode, countryname) VALUES ('ECU','Ecuador');
INSERT INTO Nations (isocode, countryname) VALUES ('EGY','Egypt');
INSERT INTO Nations (isocode, countryname) VALUES ('ERI','Eritrea');
INSERT INTO Nations (isocode, countryname) VALUES ('ESH','Western Sahara');
INSERT INTO Nations (isocode, countryname) VALUES ('ESP','Spain');
INSERT INTO Nations (isocode, countryname) VALUES ('EST','Estonia');
INSERT INTO Nations (isocode, countryname) VALUES ('ETH','Ethiopia');
INSERT INTO Nations (isocode, countryname) VALUES ('FIN','Finland');
INSERT INTO Nations (isocode, countryname) VALUES ('FJI','Fiji');
INSERT INTO Nations (isocode, countryname) VALUES ('FLK','Falkland Islands (Malvinas)');
INSERT INTO Nations (isocode, countryname) VALUES ('FRA','France');
INSERT INTO Nations (isocode, countryname) VALUES ('FRO','Faroe Islands');
INSERT INTO Nations (isocode, countryname) VALUES ('FSM','Federarated States of
Micronesia');
INSERT INTO Nations (isocode, countryname) VALUES ('GAB','Gabon');
INSERT INTO Nations (isocode, countryname) VALUES ('GBR','United Kingdom');
INSERT INTO Nations (isocode, countryname) VALUES ('GEO','Georgia');
INSERT INTO Nations (isocode, countryname) VALUES ('GGY','Guernsey');
INSERT INTO Nations (isocode, countryname) VALUES ('GHA','Ghana');
INSERT INTO Nations (isocode, countryname) VALUES ('GIN','Guinea');
INSERT INTO Nations (isocode, countryname) VALUES ('GIB','Gibraltar');
INSERT INTO Nations (isocode, countryname) VALUES ('GLP','Guadeloupe');
INSERT INTO Nations (isocode, countryname) VALUES ('GMB','Gambia');
INSERT INTO Nations (isocode, countryname) VALUES ('GNB','Guinea-Bissau');
INSERT INTO Nations (isocode, countryname) VALUES ('GNQ','Equatorial Guinea');
INSERT INTO Nations (isocode, countryname) VALUES ('GRC','Greece');
INSERT INTO Nations (isocode, countryname) VALUES ('GRD','Grenada');
INSERT INTO Nations (isocode, countryname) VALUES ('GRL','Greenland');
INSERT INTO Nations (isocode, countryname) VALUES ('GTM','Guatemala');
INSERT INTO Nations (isocode, countryname) VALUES ('GUF','French Guiana');
INSERT INTO Nations (isocode, countryname) VALUES ('GUM','Guam');
INSERT INTO Nations (isocode, countryname) VALUES ('GUY','Guyana');
INSERT INTO Nations (isocode, countryname) VALUES ('HKG','Hong Kong');
INSERT INTO Nations (isocode, countryname) VALUES ('HMD','Heard Island and McDonald
Islands');
INSERT INTO Nations (isocode, countryname) VALUES ('HND','Honduras');
INSERT INTO Nations (isocode, countryname) VALUES ('HRV','Croatia');
INSERT INTO Nations (isocode, countryname) VALUES ('HTI','Haiti');
INSERT INTO Nations (isocode, countryname) VALUES ('HUN','Hungary');
INSERT INTO Nations (isocode, countryname) VALUES ('IDN','Indonesia');
INSERT INTO Nations (isocode, countryname) VALUES ('IMN','Isle of Man');
INSERT INTO Nations (isocode, countryname) VALUES ('IND','India');
INSERT INTO Nations (isocode, countryname) VALUES ('IOT','British Indian Ocean
Territory');
INSERT INTO Nations (isocode, countryname) VALUES ('IRL','Ireland');
INSERT INTO Nations (isocode, countryname) VALUES ('IRQ','Iraq');
INSERT INTO Nations (isocode, countryname) VALUES ('ISL','Iceland');
INSERT INTO Nations (isocode, countryname) VALUES ('ISR','Israel');
INSERT INTO Nations (isocode, countryname) VALUES ('ITA','Italy');
INSERT INTO Nations (isocode, countryname) VALUES ('JAM','Jamaica');
INSERT INTO Nations (isocode, countryname) VALUES ('JEY','Jersey');
INSERT INTO Nations (isocode, countryname) VALUES ('JOR','Jordan');
INSERT INTO Nations (isocode, countryname) VALUES ('JPN','Japan');
INSERT INTO Nations (isocode, countryname) VALUES ('KAZ','Kazakhstan');
INSERT INTO Nations (isocode, countryname) VALUES ('KEN','Kenya');
INSERT INTO Nations (isocode, countryname) VALUES ('KGZ','Kyrgyzstan');
INSERT INTO Nations (isocode, countryname) VALUES ('KHM','Cambodia');
INSERT INTO Nations (isocode, countryname) VALUES ('KIR','Kiribati');
INSERT INTO Nations (isocode, countryname) VALUES ('KNA','Saint Kitts and Nevis');
INSERT INTO Nations (isocode, countryname) VALUES ('KWT','Kuwait');
INSERT INTO Nations (isocode, countryname) VALUES ('LAO','Lao People''s Democratic
```

```
Republic');
INSERT INTO Nations (isocode, countryname) VALUES ('LBN','Lebanon');
INSERT INTO Nations (isocode, countryname) VALUES ('LBR','Liberia');
INSERT INTO Nations (isocode, countryname) VALUES ('LBY','Libyan Arab Jamahiriya');
INSERT INTO Nations (isocode, countryname) VALUES ('LCA','Saint Lucia');
INSERT INTO Nations (isocode, countryname) VALUES ('LIE','Liechtenstein');
INSERT INTO Nations (isocode, countryname) VALUES ('LKA','Sri Lanka');
INSERT INTO Nations (isocode, countryname) VALUES ('LSO','Lesotho');
INSERT INTO Nations (isocode, countryname) VALUES ('LTU','Lithuania');
INSERT INTO Nations (isocode, countryname) VALUES ('LUX','Luxembourg');
INSERT INTO Nations (isocode, countryname) VALUES ('LVA','Latvia');
INSERT INTO Nations (isocode, countryname) VALUES ('MAC','Macao');
INSERT INTO Nations (isocode, countryname) VALUES ('MAF','Saint Martin (French part)');
INSERT INTO Nations (isocode, countryname) VALUES ('MAR','Morocco');
INSERT INTO Nations (isocode, countryname) VALUES ('MCO','Monaco');
INSERT INTO Nations (isocode, countryname) VALUES ('MDG','Madagascar');
INSERT INTO Nations (isocode, countryname) VALUES ('MDV','Maldives');
INSERT INTO Nations (isocode, countryname) VALUES ('MEX','Mexico');
INSERT INTO Nations (isocode, countryname) VALUES ('MHL','Marshall Islands');
INSERT INTO Nations (isocode, countryname) VALUES ('MLI','Mali');
INSERT INTO Nations (isocode, countryname) VALUES ('MLT','Malta');
INSERT INTO Nations (isocode, countryname) VALUES ('MMR','Myanmar');
INSERT INTO Nations (isocode, countryname) VALUES ('MNE','Montenegro');
INSERT INTO Nations (isocode, countryname) VALUES ('MNG','Mongolia');
INSERT INTO Nations (isocode, countryname) VALUES ('MNP','Northern Mariana Islands');
INSERT INTO Nations (isocode, countryname) VALUES ('MOZ','Mozambique');
INSERT INTO Nations (isocode, countryname) VALUES ('MRT','Mauritania');
INSERT INTO Nations (isocode, countryname) VALUES ('MSR','Montserrat');
INSERT INTO Nations (isocode, countryname) VALUES ('MTQ','Martinique');
INSERT INTO Nations (isocode, countryname) VALUES ('MUS','Mauritius');
INSERT INTO Nations (isocode, countryname) VALUES ('MWI','Malawi');
INSERT INTO Nations (isocode, countryname) VALUES ('MYS','Malaysia');
INSERT INTO Nations (isocode, countryname) VALUES ('MYT','Mayotte');
INSERT INTO Nations (isocode, countryname) VALUES ('NAM','Namibia');
INSERT INTO Nations (isocode, countryname) VALUES ('NCL','New Caledonia');
INSERT INTO Nations (isocode, countryname) VALUES ('NER','Niger');
INSERT INTO Nations (isocode, countryname) VALUES ('NFK','Norfolk Island');
INSERT INTO Nations (isocode, countryname) VALUES ('NGA','Nigeria');
INSERT INTO Nations (isocode, countryname) VALUES ('NIC','Nicaragua');
INSERT INTO Nations (isocode, countryname) VALUES ('NOR','Norway');
INSERT INTO Nations (isocode, countryname) VALUES ('NIU','Niue');
INSERT INTO Nations (isocode, countryname) VALUES ('NLD','Netherlands');
INSERT INTO Nations (isocode, countryname) VALUES ('NPL','Nepal');
INSERT INTO Nations (isocode, countryname) VALUES ('NRU','Nauru');
INSERT INTO Nations (isocode, countryname) VALUES ('NZL','New Zealand');
INSERT INTO Nations (isocode, countryname) VALUES ('OMN','Oman');
INSERT INTO Nations (isocode, countryname) VALUES ('PAK','Pakistan');
INSERT INTO Nations (isocode, countryname) VALUES ('PAN','Panama');
INSERT INTO Nations (isocode, countryname) VALUES ('PCN','Pitcairn');
INSERT INTO Nations (isocode, countryname) VALUES ('PER','Peru');
INSERT INTO Nations (isocode, countryname) VALUES ('PHL','Philippines');
INSERT INTO Nations (isocode, countryname) VALUES ('PLW','Palau');
INSERT INTO Nations (isocode, countryname) VALUES ('PNG','Papua New Guinea');
INSERT INTO Nations (isocode, countryname) VALUES ('POL','Poland');
INSERT INTO Nations (isocode, countryname) VALUES ('PRI','Puerto Rico');
INSERT INTO Nations (isocode, countryname) VALUES ('PRT','Portugal');
INSERT INTO Nations (isocode, countryname) VALUES ('PRY','Paraguay');
INSERT INTO Nations (isocode, countryname) VALUES ('PYF','French Polynesia');
INSERT INTO Nations (isocode, countryname) VALUES ('QAT','Qatar');
INSERT INTO Nations (isocode, countryname) VALUES ('REU','Réunion');
INSERT INTO Nations (isocode, countryname) VALUES ('ROU','Romania');
INSERT INTO Nations (isocode, countryname) VALUES ('RUS','Russian Federation');
```

```
INSERT INTO Nations (isocode, countryname) VALUES ('RWA','Rwanda');
INSERT INTO Nations (isocode, countryname) VALUES ('SAU','Saudi Arabia');
INSERT INTO Nations (isocode, countryname) VALUES ('SDN','Sudan');
INSERT INTO Nations (isocode, countryname) VALUES ('SEN','Senegal');
INSERT INTO Nations (isocode, countryname) VALUES ('SGP','Singapore');
INSERT INTO Nations (isocode, countryname) VALUES ('SGS','South Georgia and the South
Sandwich Islands');
INSERT INTO Nations (isocode, countryname) VALUES ('SHN','Saint Helena');
INSERT INTO Nations (isocode, countryname) VALUES ('SJM','Svalbard and Jan Mayen');
INSERT INTO Nations (isocode, countryname) VALUES ('SLB','Solomon Islands');
INSERT INTO Nations (isocode, countryname) VALUES ('SLE','Sierra Leone');
INSERT INTO Nations (isocode, countryname) VALUES ('SLV','El Salvador');
INSERT INTO Nations (isocode, countryname) VALUES ('SMR','San Marino');
INSERT INTO Nations (isocode, countryname) VALUES ('SOM','Somalia');
INSERT INTO Nations (isocode, countryname) VALUES ('SPM','Saint Pierre and Miquelon');
INSERT INTO Nations (isocode, countryname) VALUES ('SRB','Serbia');
INSERT INTO Nations (isocode, countryname) VALUES ('STP','Sao Tome and Principe');
INSERT INTO Nations (isocode, countryname) VALUES ('SUR','Suriname');
INSERT INTO Nations (isocode, countryname) VALUES ('SVK','Slovakia');
INSERT INTO Nations (isocode, countryname) VALUES ('SVN','Slovenia');
INSERT INTO Nations (isocode, countryname) VALUES ('SWE','Sweden');
INSERT INTO Nations (isocode, countryname) VALUES ('SWZ','Swaziland');
INSERT INTO Nations (isocode, countryname) VALUES ('SYC','Seychelles');
INSERT INTO Nations (isocode, countryname) VALUES ('SYR','Syrian Arab Republic');
INSERT INTO Nations (isocode, countryname) VALUES ('TCA','Turks and Caicos Islands');
INSERT INTO Nations (isocode, countryname) VALUES ('TCD','Chad');
INSERT INTO Nations (isocode, countryname) VALUES ('TGO','Togo');
INSERT INTO Nations (isocode, countryname) VALUES ('THA','Thailand');
INSERT INTO Nations (isocode, countryname) VALUES ('TJK','Tajikistan');
INSERT INTO Nations (isocode, countryname) VALUES ('TKL','Tokelau');
INSERT INTO Nations (isocode, countryname) VALUES ('TKM','Turkmenistan');
INSERT INTO Nations (isocode, countryname) VALUES ('TLS','Timor-Leste');
INSERT INTO Nations (isocode, countryname) VALUES ('TON','Tonga');
INSERT INTO Nations (isocode, countryname) VALUES ('TTO','Trinidad and Tobago');
INSERT INTO Nations (isocode, countryname) VALUES ('TUN','Tunisia');
INSERT INTO Nations (isocode, countryname) VALUES ('TUR','Turkey');
INSERT INTO Nations (isocode, countryname) VALUES ('TUV','Tuvalu');
INSERT INTO Nations (isocode, countryname) VALUES ('UGA','Uganda');
INSERT INTO Nations (isocode, countryname) VALUES ('UKR','Ukraine');
INSERT INTO Nations (isocode, countryname) VALUES ('UMI','United States Minor Outlying
Islands');
INSERT INTO Nations (isocode, countryname) VALUES ('URY','Uruguay');
INSERT INTO Nations (isocode, countryname) VALUES ('USA','United States');
INSERT INTO Nations (isocode, countryname) VALUES ('UZB','Uzbekistan');
INSERT INTO Nations (isocode, countryname) VALUES ('VAT','Holy See (Vatican City State)');
INSERT INTO Nations (isocode, countryname) VALUES ('VCT','Saint Vincent and the
Grenadines');
INSERT INTO Nations (isocode, countryname) VALUES ('VEN','Venezuela');
INSERT INTO Nations (isocode, countryname) VALUES ('VNM','Viet Nam');
INSERT INTO Nations (isocode, countryname) VALUES ('VUT','Vanuatu');
INSERT INTO Nations (isocode, countryname) VALUES ('WLF','Wallis and Futuna');
INSERT INTO Nations (isocode, countryname) VALUES ('WSM','Samoa');
INSERT INTO Nations (isocode, countryname) VALUES ('YEM','Yemen');
INSERT INTO Nations (isocode, countryname) VALUES ('ZAF','South Africa');
INSERT INTO Nations (isocode, countryname) VALUES ('ZMB','Zambia');
INSERT INTO Nations (isocode, countryname) VALUES ('ZWE','Zimbabwe');
INSERT INTO Nations (isocode, countryname) VALUES ('ASC','Ascension Island');
INSERT INTO Nations (isocode, countryname) VALUES ('CPT','Clipperton Island');
INSERT INTO Nations (isocode, countryname) VALUES ('DGA','Diego Garcia');
INSERT INTO Nations (isocode, countryname) VALUES ('TAA','Tristan da Cunha');
```

```java
package org.pricip.ingest;

import org.apache.commons.dbcp.BasicDataSource;

import java.util.Date;
import java.text.SimpleDateFormat;
import java.util.List;
import java.util.Map;
import java.util.HashMap;

import java.util.logging.Logger;

import org.apache.commons.lang.StringUtils;
import org.apache.commons.lang.math.NumberUtils;

import org.springframework.core.io.FileSystemResource;
import org.springframework.beans.factory.BeanFactory;
import org.springframework.beans.factory.xml.XmlBeanFactory;
import org.springframework.jdbc.core.simple.SimpleJdbcInsert;

import org.apache.commons.configuration.Configuration;
import org.apache.commons.configuration.PropertiesConfiguration;


public class InsertDao
{
  private Logger myLogger = Logger.getLogger(this.getClass().getCanonicalName());

  private SimpleJdbcInsert insertRecord;

  private String process = "";

  private int count = 1;

  public InsertDao(String p)
  {
    process = p;
  }


  public void setDataSource()
  {
    String homedir = "c:/pricip";
    try
    {
      //  specifying without absolute path, the file will be searched automatically in
the following locations, in the current directory,
      //  in the user home directory, in the classpath
      Configuration config = new PropertiesConfiguration("environment.properties");
      homedir = config.getString("home.dir");
    }
    catch (Exception e)
    {
      myLogger.warning("Could not load configurtion " + e);
      myLogger.warning("Assuming home dir is " + homedir);
    }

    FileSystemResource fsr = new FileSystemResource(homedir + "/db-config.xml");
    BeanFactory factory = new XmlBeanFactory(fsr);
    BasicDataSource dataSource = (BasicDataSource) factory.getBean("dataSource");
    //  create the insert record for table master_processes with the attribute
record_number defined as auto incrementing (serial in Postgres)
```

```
    this.insertRecord =  new
 SimpleJdbcInsert(dataSource).withTableName("master_processes").usingGeneratedKeyColumns("
ecord_number");
  }

  public void add(Excelproduct xp) {
      Map<String, Object> parameters = new HashMap<String, Object>();

      // no need to do this since usingGeneratedKeyColumns
      // parameters.put("record_number", record_number);
      putString(parameters,"process_code", process);
      putString(parameters,"product_class", xp.getProductclass());
      putString(parameters,"timeframe", xp.getTimeframe());
      putString(parameters,"theme", xp.getTheme());
      putString(parameters,"region", xp.getRegion());
      putString(parameters,"country", xp.getCountry());
      // caution this needs to be cleaned up later when we actually have publish
dates...just putting nulls for now
      putDate(parameters,"publish_date", xp.getPublishdate());
      putString(parameters,"platform_type", xp.getPlatformtype());
      // ignore station id parameter...must be generated in ETL process, not here
      putString(parameters,"station_type", xp.getStationtype());
      putString(parameters,"station_name", xp.getStationname());

      putDouble(parameters,"latitude", xp.getLatitude());
      putDouble(parameters,"longitude", xp.getLongitude());

      putString(parameters,"operator", xp.getOperator());
      putString(parameters,"operatorid", xp.getOperatorid());
      putStatus(parameters,"active", xp.getActive());
      putString(parameters,"metadata", xp.getStationmetadata());
      putString(parameters,"metadata_url", xp.getStationmetadataurl());
      // ignore product_id parameter
      putString(parameters,"ppindex", xp.getPpindex());
      putString(parameters,"product_type", xp.getProducttype());
      putString(parameters,"product_name", xp.getProductname());
      putString(parameters,"product_source", xp.getProductsource());
      putString(parameters,"host_location", xp.getHostlocation());
      putString(parameters,"product_format", xp.getProductformat());
      putString(parameters,"about", xp.getAbout());
      putString(parameters,"person", xp.getPerson());
      putString(parameters,"phone", xp.getPhone());
      putString(parameters,"email", xp.getEmail());

      putString(parameters,"sourcetype3", xp.getSourcetype3());
      // ignore data id
      putString(parameters,"sourceid3", xp.getSourceid3());
      putString(parameters,"location3", xp.getLocation3());
      putString(parameters,"format3", xp.getFormat3());
      putString(parameters,"metadatacompliant3", xp.getMetadatacompliant3());

      putDate(parameters,"metalastupdate3", xp.getMetalastupdate3());

      putString(parameters,"metadataurl3", xp.getMetadataurl3());

      putString(parameters,"sourcetype2", xp.getSourcetype2());
      // ignore data id
      putString(parameters,"sourceid2", xp.getSourceid2());
      putString(parameters,"location2", xp.getLocation2());
      putString(parameters,"format2", xp.getFormat2());
      putString(parameters,"metadatacompliant2", xp.getMetadatacompliant2());
```

```
      putDate(parameters,"metalastupdate2", xp.getMetalastupdate2());

      putString(parameters,"metadataurl2", xp.getMetadataurl2());

      insertRecord.execute(parameters);

      // myLogger.info("Inserted Record " + count);
      count++;
  }

  public void add(List<Excelproduct> products)
  {
    setDataSource();
    for (java.util.Iterator<Excelproduct>iter = products.iterator(); iter.hasNext();)
    {
      Excelproduct xp = iter.next();
      add(xp);
    }
    myLogger.info("Inserted " + count + " " + process + " records into " +
insertRecord.getTableName());
  }

  // put a clean string into the map value
  private void putString (Map<String, Object>  params, String key, String val)
  {
    String value = StringUtils.trimToEmpty(val);
    if (value.length() == 0)
    {
      // even a one char field can take this in
      params.put(key, "?");
    }
    else
    {
      params.put(key, value);
    }
  }

  // put a clean double (or zero) into the map value
  private void putDouble (Map<String, Object>  params, String key, String val)
  {
    double d = NumberUtils.toDouble(val, 0.0);
    Double value = new Double(d);
    params.put(key, value);
  }

  // put a clean double (or zero) into the map value
  private void putInteger (Map<String, Object>  params, String key, String val)
  {
    int i = NumberUtils.toInt(val, 0);
    Integer value = new Integer(i);
    params.put(key, value);
  }

  // put a clean string into the map value
  private void putStatus (Map<String, Object>  params, String key, String val)
  {
    String value = StringUtils.trimToEmpty(val);

    if (value.equalsIgnoreCase("Y"))
    {
      value = "A"; // active
    }
```

```
      else if (value.equalsIgnoreCase("N"))
      {
        value = "I"; // inactive
      }
      else
      {
        value ="U"; // unknown
      }

      if (value.length() > 0)
      {
        params.put(key, value);
      }
    }

    // put a clean date into the map
    private void putDate (Map<String, Object>  params, String key, String val)
    {
      // this the format that should be coming out of Excel by default
      SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy");
      Date value = null;
      try
      {
        value = sdf.parse(val);
      }
      catch (Exception e)
      {
        // ignore
      }
      if (value == null)
      {
        return;
      }
      else
      {
        params.put(key, value);
      }
    }
}
```

```
-- ////////////////////////////////////
-- // C O N T A C T S
INSERT INTO Contacts (person,phone,email) (
 SELECT person, phone, email
 FROM Master_Processes
 GROUP BY person, phone, email)
;
-- ////////////////////////////////////
-- // O P E R A T O R S
INSERT INTO Operators (operator) (
 SELECT operator
 FROM Master_Processes
 GROUP BY operator)
;
-- ////////////////////////////////////
-- // S T A T I O N S
--
-- (Normalization)
-- First, create useful views of the type StationPlusAttributes
-- capturing Functional Dependency (FD).
--
-- For all stations, the following FD holds
-- station_name --> operator, operatorid
--
-- For InSitu stations, the following FD holds
-- station_name, insitue --> latitude, longitude, region, nation, status
--
-- When Remote sensing (satellite) relation is created,
-- we will manufacure a similar view.
--
-- The strategy is to use these views to first populate the Root
-- relation, Stations, of both Remote as well as Insitu.
--
-- VERY IMPORTANT ! ! ! ! ! ! ! ! ! ! ! ! ! !
--
-- Verify beforehand whether the FD holds.  That is, for example, if any
-- two tuples of master (raw) file agree on Station_name, they must agree on all
-- other attributes on the right hand side of the assumed FD.  Otherwise
-- you will have all sort of the porblems.
--
-- Excellent way to check is if GROUP BY station_name has same
-- records as GROUP BY station_name (and other attributes)...otherwise
-- as you add attributes to GROUP BY you will get more records...have to
-- manually fix the errors before doing this ETL.
--
CREATE OR REPLACE VIEW OperatorsView AS (
 SELECT station_name AS stationname, operator, operatorid AS localid
 FROM Master_Processes
 GROUP BY station_name, operator, operatorid)
;
CREATE OR REPLACE VIEW InSituView AS (
 SELECT station_name AS stationname, latitude, longitude, region,
        country AS nation, active AS status, metadata_url AS metadata
 FROM Master_Processes
 GROUP BY station_name, latitude, longitude, region, country, active, metadata_url)
;
--
-- CAUTION:  Proceed only if the above two have same number of records
--           which in turn must be same as just doing GROUP BY Station
--           SELECT COUNT(*) FROM OperatorsView
--           SELECT COUNT(*) FROM InsituView
--
```

```
--           The natutal join below happens on attribute OPERATOR in
--           OperatorsView and Operators
--
INSERT INTO Stations (stationname, localid, onum) (
 SELECT stationname, localid, onum
 FROM OperatorsView NATURAL JOIN Operators)
;
--
--           The natutal join below happens on attribute NAME in
--           InSituView NATURAL JOIN Stations, thereby linking
--           Insitu stations with the parent entity, station
--
INSERT INTO Insitu_Stations (stationid, stationname, latitude, longitude, region,
                             nation, status, metadata) (
 SELECT stationid, stationname, latitude, longitude, region, nation, status, metadata
 FROM InSituView NATURAL JOIN Stations)
;
-- ////////////////////////////////////
-- // P R O D U C T S
-- //
--
-- (Normalization)
-- First, create views of the type ProductPlusAttributes
-- capturing Functional Dependency (FD).
--
-- For all products, the following FD holds
-- station_name, product_name --> person, phone, email
-- station_name, product_name --> ppindex, type, source
--
-- When Targeted Information Products (TIP) are created, we will form a similar
-- view.
--
-- The strategy is to use these views to first populate the Root
-- relation, Products, of both TIP as well as DDP.
--
-- VERY IMPORTANT ! ! ! ! ! ! ! ! ! ! ! ! ! !
--
-- Verify beforehand whether the FD holds.  That is, for example, if any
-- two tuples of master (raw) file agree on Station_name, they must agree on all
-- other attributes on the right hand side of the assumed FD.  Otherwise
-- you will have all sort of the porblems.
--
-- Excellent way to check is if GROUP BY station_name, product_name has same
-- records as GROUP BY station_name (and other attributes)...otherwise
-- as you add attributes to GROUP BY you will get more records...have to
-- manually fix the errors before doing this ETL.
--
-- For above FDs to hold...
--
-- SELECT COUNT(*) FROM SPView should equal
-- SELECT COUNT(*) FROM ContactsView which should equal
-- SELECT COUNT(*) FROM DDPView
--
CREATE OR REPLACE VIEW ContactsView AS (
 SELECT station_name AS stationname, product_name AS productname, person, phone, email
 FROM Master_Processes
 GROUP BY station_name, product_name, person, phone, email)
;
CREATE OR REPLACE VIEW DDPView AS (
 SELECT station_name AS stationname, product_name AS productname,
        product_type AS ddptype, product_source AS ddpsource, ppindex,
        process_code AS process, timeframe
```

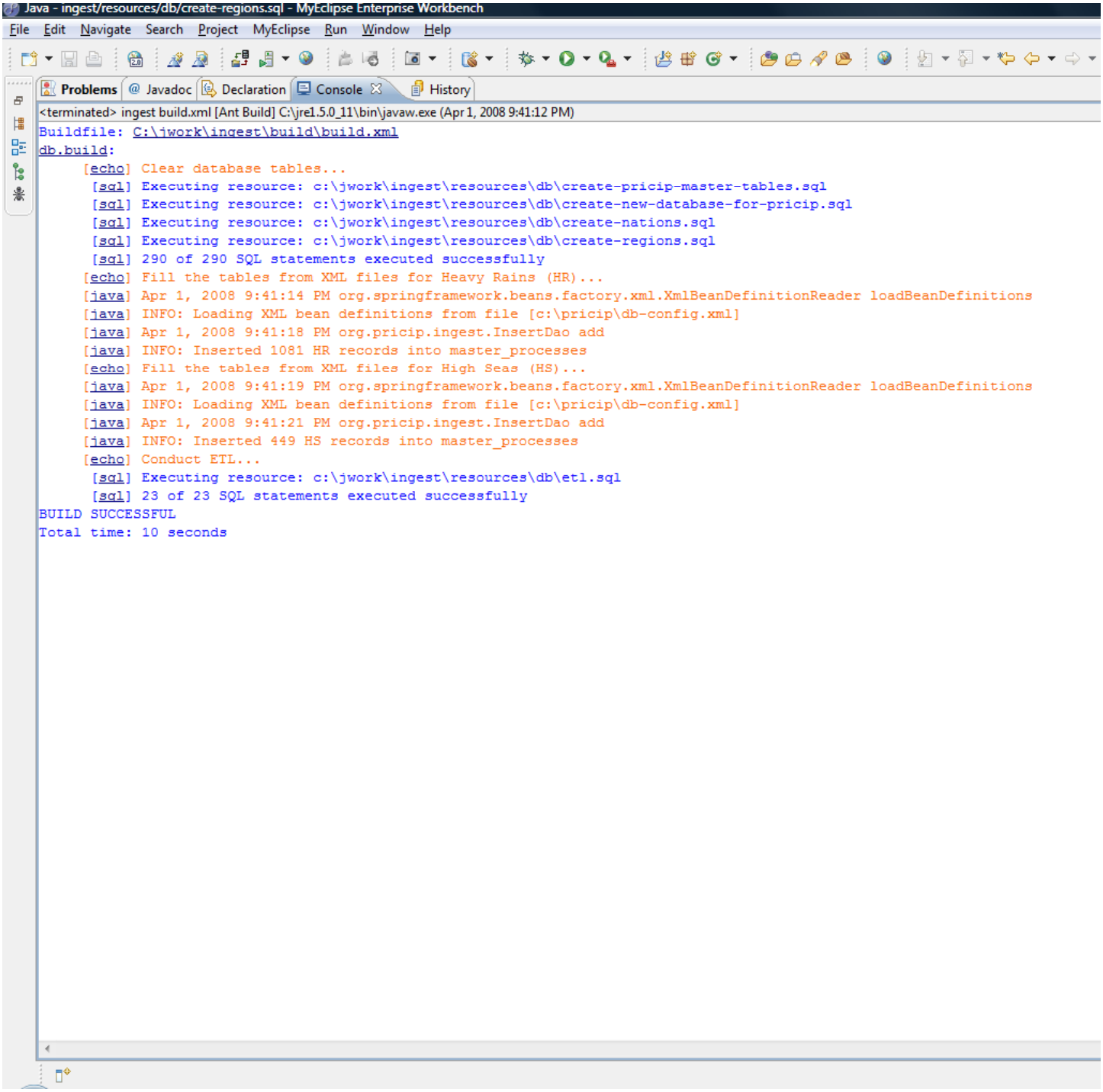
```
 FROM Master_Processes
 GROUP BY station_name, product_name, product_type, product_source,
          ppindex, process_code, timeframe)
;
CREATE OR REPLACE VIEW HostView AS (
 SELECT station_name AS stationname, product_name AS productname,
        host_location AS fileurl, product_format AS format, about AS description
 FROM Master_Processes
 GROUP BY station_name, product_name,host_location, product_format, about)
;
--
--  Begin Product Inserts
--
INSERT INTO Products (contactid, stationid, productname) (
 SELECT contactid, stationid, productname
 FROM ContactsView NATURAL JOIN Stations NATURAL JOIN Contacts)
;
INSERT INTO DDP (productid, ppindex, ddptype, ddpname, ddpsource) (
 SELECT productid, ppindex, ddptype, productname, ddpsource
 FROM DDPView NATURAL JOIN Stations NATURAL JOIN Products)
;
INSERT INTO Hosts (productid, fileurl, format, description) (
 SELECT productid, fileurl, format, description
 FROM Hostview NATURAL JOIN Stations NATURAL JOIN Products)
;
INSERT INTO Timeframes (productid) (
 SELECT productid FROM DDP)
;
INSERT INTO Processes (productid) (
 SELECT productid FROM DDP)
;
INSERT INTO HighSeas (productid) (
 SELECT productid
 FROM DDPView NATURAL JOIN Stations NATURAL JOIN Products
 WHERE process='HS')
;
INSERT INTO HeavyRains (productid) (
 SELECT productid
 FROM DDPView NATURAL JOIN Stations NATURAL JOIN Products
 WHERE process='HR')
;
INSERT INTO StrongWinds (productid) (
 SELECT productid
 FROM DDPView NATURAL JOIN Stations NATURAL JOIN Products
 WHERE process='SW')
;
INSERT INTO HindCasts (productid) (
 SELECT productid
 FROM DDPView NATURAL JOIN Stations NATURAL JOIN Products
 WHERE timeframe='H')
;
--
-- And so on for other timeframes
--
-- ////////////////////////////////////
-- // D A T A S E T S
-- //
--
-- (Normalization)
-- Investigate whether station_name, product_name determine
-- data2 (archived)
CREATE OR REPLACE VIEW ArchivedView AS (
```

```
 SELECT station_name AS stationname, product_name AS productname,
        process_code AS process, timeframe,
        sourceid2 AS owner, location2 AS filename, format2 AS format,
        metalastupdate2 AS lastupdated, metadataurl2 AS metadataurl
 FROM Master_Processes
 GROUP BY station_name, product_name, process_code, timeframe,
          sourceid2, location2, format2, metalastupdate2,
          metadataurl2)
;
INSERT INTO Datasets (productid, owner, filename, format, lastupdated, metadataurl) (
 SELECT productid, owner, filename, format, lastupdated, metadataurl
 FROM ArchivedView NATURAL JOIN Stations NATURAL JOIN Products)
;
INSERT INTO Archived (datasetid) (
 SELECT datasetid FROM Datasets)
;
--
--  Couple of useful views
--
CREATE OR REPLACE VIEW Insitu_Stations_HighSeas_Hindcasts AS
 (SELECT stationname, operator, localid, latitude, longitude,
         region, nation AS country, status
  FROM   Stations NATURAL JOIN Insitu_Stations NATURAL JOIN Operators
  WHERE  stationid IN (SELECT stationid
                       FROM Products NATURAL JOIN HighSeas NATURAL JOIN Hindcasts))
;
CREATE OR REPLACE VIEW Insitu_Stations_HeavyRains_Hindcasts AS
 (SELECT stationname, operator, localid, latitude, longitude,
         region, nation AS country, status
  FROM   Stations NATURAL JOIN Insitu_Stations NATURAL JOIN Operators
  WHERE  stationid IN (SELECT stationid
                       FROM Products NATURAL JOIN HeavyRains NATURAL JOIN Hindcasts))
;
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
  +========================================================================+
  |  FILENAME: build.xml                                                   |
  +========================================================================+
  |  DESCRIPTION:                                                          |
  |                                                                        |
  |  ANT script for building Ingest executable jar, builds the database and  |
  |  does other batch processes involved in the PRICIP database ingest     |
  +========================================================================+
  |  HISTORY                                                               |
  |                                                                        |
  |   [26-MAR-2008] Uday Kari, created jarfile task with all dependencies    |
  |                            to create ingest.jar that uploads XML into db |
  |   [26-MAR-2008] Uday Kari, add db.build target as default which runs all |
  |                            the scripts, including ingest.jar           |
  |   [29-MAR-2008] Uday Kari, (minor) add zip target to archive project     |
  +========================================================================+
-->

<project name="ingest" default="db.build">

<property file="build.properties" />
  <path id="libraries">
    <pathelement path="${work.dir}/pdclib/lib/commons-beanutils.jar" />
    <pathelement path="${work.dir}/pdclib/lib/commons-collections-3.2.jar" />
    <pathelement path="${work.dir}/pdclib/lib/commons-configuration-1.3.jar" />
    <pathelement path="${work.dir}/pdclib/lib/commons-dbcp-1.2.1.jar" />
    <pathelement path="${work.dir}/pdclib/lib/commons-digester.jar" />
   <pathelement path="${work.dir}/lib/commons-lang-2.3.jar" />
   <pathelement path="${work.dir}/pdclib/lib/commons-logging.jar" />
   <pathelement path="${work.dir}/pdclib/lib/commons-pool-1.3.jar" />
   <pathelement path="${work.dir}/lib/spring-2.5.1.jar" />
 </path>

 <property name="classpath" refid="libraries" />

 <target name="clean">
  <echo message="DELETING old ${context.name}.jar..." />
  <delete file="${target.dir}/${context.name}.jar" />
 </target>

 <target name="compile" depends="clean">
  <echo message="Compiling ${src.dir} to ${classes.dir} " />
   <javac srcdir="${src.dir}" destdir="${classes.dir}"
      classpath="${classpath}" debug="true" deprecation="true" optimize="false" />
 </target>

 <target name="jarfile" depends="compile">
  <echo message="Building jar file ${jar.name} in ${target.dir} from base directory
${classes.dir} " />
  <jar destfile="${target.dir}/${jar.name}" basedir="${classes.dir}"

includes="**/ProductRecordsLists.class,**/InsertDao.class,**/Excelproduct.class">
    <manifest>
      <attribute name="Main-Class" value="${main.class}"/>
      <attribute name="Class-Path" value="${class.path}"/>
     </manifest>
  </jar>

  <echo message="copying  ${jar.name} in ${target.dir} to ${ftproot}" />
  <copy todir="${ftproot}">
```

```
      <fileset dir="${target.dir}" >
       <include name="${jar.name}" />
      </fileset>
    </copy>

 </target>

 <target name="db.build">

    <echo message="Clear database tables..." />

    <sql driver="org.postgresql.Driver"
         url="jdbc:postgresql:pricip"
         userid="postgres"
         password="admin123">

      <classpath>
       <pathelement path="${work.dir}/lib/postgresql-8.1-405.jdbc3.jar"/>
      </classpath>

      <transaction
src="${work.dir}/${context.name}/resources/db/create-pricip-master-tables.sql"/>
      <transaction
src="${work.dir}/${context.name}/resources/db/create-new-database-for-pricip.sql"/>
      <transaction  src="${work.dir}/${context.name}/resources/db/create-nations.sql"/>
      <transaction  src="${work.dir}/${context.name}/resources/db/create-regions.sql"/>

    </sql>

    <echo message="Fill the tables from XML files for Heavy Rains (HR)..." />
    <java dir="${target.dir}"
          jar="${target.dir}/${context.name}.jar"
          fork="true"
          failonerror="true">
       <arg value="HR"/>
    </java>

    <echo message="Fill the tables from XML files for High Seas (HS)..." />
    <java dir="${target.dir}"
          jar="${target.dir}/${context.name}.jar"
          fork="true"
          failonerror="true">
       <arg value="HS"/>
    </java>


    <echo message="Conduct ETL..." />

    <sql driver="org.postgresql.Driver"
         url="jdbc:postgresql:pricip"
         userid="postgres"
         password="admin123">

      <classpath>
       <pathelement path="${work.dir}/lib/postgresql-8.1-405.jdbc3.jar"/>
      </classpath>

      <transaction  src="${work.dir}/${context.name}/resources/db/etl.sql"/>

    </sql>

  </target>
```

```
 <target name="zipitup">
    <zip destfile="${ftproot}/${context.name}.zip"   basedir="${work.dir}/${context.name}"
  />
  </target>


</project>
```


Java - ingest/resources/db/create-regions.sql - MyEclipse Enterprise Workbench
File Edit Navigate Search Project MyEclipse Run Window Help
Problems Javadoc Declaration Console History
<terminated> ingest build.xml [Ant Build] C:\jre1.5.0_11\bin\javaw.exe (Apr 1, 2008 9:41:12 PM)
Buildfile: C:\jwork\ingest\build\build.xml
db.build:
[echo] Clear database tables...
[sql] Executing resource: c:\jwork\ingest\resources\db\create-pricip-master-tables.sql
[sql] Executing resource: c:\jwork\ingest\resources\db\create-new-database-for-pricip.sql
[sql] Executing resource: c:\jwork\ingest\resources\db\create-nations.sql
[sql] Executing resource: c:\jwork\ingest\resources\db\create-regions.sql
[sql] 290 of 290 SQL statements executed successfully
[echo] Fill the tables from XML files for Heavy Rains (HR)...
[java] Apr 1, 2008 9:41:14 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
[java] INFO: Loading XML bean definitions from file [c:\pricip\db-config.xml]
[java] Apr 1, 2008 9:41:18 PM org.pricip.ingest.InsertDao add
[java] INFO: Inserted 1081 HR records into master_processes
[echo] Fill the tables from XML files for High Seas (HS)...
[java] Apr 1, 2008 9:41:19 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
[java] INFO: Loading XML bean definitions from file [c:\pricip\db-config.xml]
[java] Apr 1, 2008 9:41:21 PM org.pricip.ingest.InsertDao add
[java] INFO: Inserted 449 HS records into master_processes
[echo] Conduct ETL...
[sql] Executing resource: c:\jwork\ingest\resources\db\etl.sql
[sql] 23 of 23 SQL statements executed successfully
BUILD SUCCESSFUL
Total time: 10 seconds