Fall 2006

Meeting the Challenges of Met Data with Mysql

Thomas S. Ciolek
Regis University

Meeting the Challenges of Met Data with MySQL

Thomas S. Ciolek

Regis University Master of Science in Computer Information Technology

Abstract

The objective of this project is to develop a relational database housing meteorological data used in direct support of emergency response activities for an atmospheric consulting firm, hereon referred to by the pseudonym WindInc. The intent is to demonstrate to WindInc how a relational database system would be superior to their present flat-file approach by providing the flexibility, performance, and reliability needed to meet their ever-increasing business demands, while simultaneously boosting the performance of their atmospheric dispersion modeling system.

Table of Contents

1. Emergency Response Background

1.1 Introduction

Hazardous materials are a reality of modern society. Because of the social climate of the world, people associate the phrase "hazardous materials" with terror and mass destruction. The mere phrase is enough to elicit anxiety and consternation amongst the general public. While it is true hazardous materials are key components in weapons and in acts of malevolence, they do have legitimate societal applications such as power-generation and in manufacturing. Their use is far too critical in key areas of industry to remove or replace them entirely. They will continue to exist and their ramifications must be taken seriously.

Hazardous materials emergencies are the release of hazardous contaminates outside designated containment boundaries. While they vary in nature, the most direct threat to public safety comes from atmospheric releases. Once airborne, a toxic cloud is subject to the whims of unpredictable winds and is beyond containment. The resulting health implications can be staggering, ranging from the minor irritation of bad odors to the extremes of permanent disability and even death. In a densely populated area, the devastation can be catastrophic.

The world has long been wary of the threat of hazardous atmospheric releases. High-profile events like the nuclear disaster at Chernobyl and the burning oil fields of Iraq showcased their severity on a world-wide scale. These examples also illustrated the two-sided threat: immediate death and the long-term side-effects not fully understood. The Iraqi mustard-gas attacks against the Kurds in the late 1980s and their present crisis with birth abnormalities are an unmistakable example of this. An arrest of a suspected terrorist plotting an attack on American soil exposed the public to the realities of “dirty bombs” and “biological agents”. But hazardous atmospheric releases are not limited to acts of aggression. The Environmental Protection Agency (EPA) had indicated that materials on the “hazardous materials” list are used directly by or transported through all major metropolitan areas within the United States. It is likely this applies to countless rural areas as well. Equipment malfunction, a truck accident, a train slipping off the tracks, or a fire at a facilities plant are all ways hazardous materials can non-deliberately be introduced into the atmosphere. Regardless of the origin, the proper authorities must have effective tools and techniques for addressing the problem.

1.2 ADM Introduction

The grim reality is that our level of preparedness for hazardous atmospheric releases is far from adequate. The available tools are insufficient; personnel are short in number and overworked; and crisis plans cannot be put in effect quickly enough. Fortunately, the government has branded hazardous atmospheric release preparedness a high-priority and is devoting considerable funds to address these shortcomings. One specific area of funding falls in the creation and expansion of atmospheric dispersion models.

An atmospheric dispersion model is a software model that generates toxic clouds, tracks their movements, anticipates their seriousness, and provides emergency response information. These models provide crisis managers with the critical data needed to answer the following questions: where the toxic cloud (plume) will travel, how large the plume is, when it will get there; how serious the impacts will be, and what protective actions should be taken.

One example of an operational atmospheric dispersion model is ADM, a pseudonym used from hereon. ADM was developed through the joint venture of the United States Department of Energy (DOE) and WindInc, a privately-owned United States-based atmospheric consulting firm. While the development was a joint venture, WindInc has assumed ownership of ADM and is solely responsible for its development, maintenance, installations, support, and general upkeep. ADM is an advanced modeling system designed for the following purposes: to predict the path and impacts of hazardous atmospheric releases; for direct application in hazards and risk assessment; for emergency preparedness; and for real-time emergency response. ADM is currently in use by the Oakridge National Laboratory in Tennessee and in support of the Sandia National Laboratory in New Mexico. It was also in operational use by and during the cleanup of the Rocky Flats Environmental Technology Site near Denver.

In addition to its atmospheric dispersion modeling capabilities, WindInc has recently upgraded ADM with the ability to facilitate smoke management forecasting (SMF). SMF is a specialized variation on atmospheric dispersion modeling to specifically address the problems associated with smoke plumes (fires). While smoke is a hazardous agent, the effects from it are more widely understood and tend to be less dramatic health-wise in open environments, excluding the discomfort it causes. Because of this, when ADM is operating in SMF mode, it will skip the health impact aspect of its modeling routines and will instead focus on the size of the plume and its projected direction. The main intention is to facilitate prescribed burns – fires the proper authorities are contemplating starting in an effort to "burn-out" and control active wildfires. Fire and government officials prefer to start prescribed fires during weather conditions that will minimize the spread of the fire while simultaneously reducing discomfort to the general public as much as possible. The ADM SMF mode can also be used to model existing fires, specifically wildfires. Generating plume maps during the duration of the fire can help officials study its growth rate, how weather conditions affected its growth, and the extent of time smoke plumes were present in populated areas.

2. Project Vision

2.1 Introduction

The key for ADM, or any atmospheric dispersion model, to be successful is a plentiful and continuous supply of recent meteorological (met) data. Met data is any individual data reading representing a specific characteristic of the atmosphere at a moment in time. There are scores of available met data applicable to atmospheric science with varying degrees of importance to ADM. At a bare minimum, ADM must have wind speed, wind direction, and ambient temperature to produce useful output.

Met data is accumulated and managed by meteorological stations (met stations). Met stations are operated by a variety of organizations ranging from government facilities to commercial organizations and private citizens. Which met data readings a met station will accumulate is at the discretion of those operating them and the capabilities of the technology. Some met stations record all available met readings; while others record only met data of interest. The frequency of recorded readings and the format is at the discretion of those running the stations, as is whether or not the data is shared and how. Some people believe met data is an open commodity and distribute it freely via web sites. Others see it as a profit opportunity and market it to the atmospheric community.

2.2 Existing Met Data Solution

To house this vast collection of meteorological data, WindInc developed the Generic Meteorological Storage and Meteorological Fields component, succinctly dubbed and hereon referred to as the Met Server. Acting as an independent sub-system, the Met Server has two primary functions: data acquisition and storage and a series of applications generating met-data specific derivative products. These products assist WindInc and its clients in assessing the state of the atmosphere and in emergency response recommendations.

2.2.1 Data Acquisition and Storage

The data storage component of the Met Server is a simulated database system. It is a flat-file system designed specifically to work with ADM on Linux systems. It is a highly-structured and complex system involving a specific directory structure, the existence of specialized files in specific directories, and the required environment variables. All pieces must be present and in the correct format for the system to operate properly.

Understanding the system begins with knowledge of its directory structure. All met data is stored within a single root directory called met_data. An associated environment variable called MET_DATA must be set to the path of this directory. Within met_data, a sub-directory exists for each met station. It is a stringent requirement that each met station have its own directory. Within each met station directory are two sub-directories. The first directory contains the met data. The approach taken is to have one Indexed Sequential Access Method (ISAM) ASCII file for each met station. All available met readings for that station are recorded in the format presented by that station. All data records are indexed by the timestamp of the incoming data. The acquired met data is stored on the same hard drive from which both ADM and the Met Server are run. With the turn of the year, a new ISAM file is created for each station. All data from the previous year is archived and removed. This is the only form of backup taken on the met data.

The second directory contains assorted files facilitating the extraction and usage of the met data. The first type of file is called a script file. The script file is a shell script that will extract the met data from its source, parse the met data of interest, and store it. Met data is typically extracted via web pages by downloading the entire HTML page or by obtaining a text file via FTP. The parsers are either UNIX commands for text files or Java-written HTML parsers. The script file produces the second type of file - the raw file. The raw file is a text file with a single line of comma delimited met data beginning with the met station ID and the timestamp of the reading. This file is overwritten with each successful met reading. In this context, successful means the script file was able to extract the last available met reading without error. At this point, an “extract and store” program is run to save the met data to its associated ISAM file. This program uses the raw file with the third and fourth types of files to store the met data. The third type of file is the raw format file. This is a text file that states what met data will be stored and its format. It works in conjunction with the fourth type of file – the descriptor file. The descriptor file is a text file containing information about the met station and the data it records. Met station information is compromised of data concerning the geographic coordinates of the met station, its elevation, any data constraints, and the frequency of met data readings. Met data information consists of the type of met reading and its format, both in terms of degree of precision and units. Each met station may have multiple levels and this information is broken down according to level.

2.2.2 Data Access

Since the data storage component of the Met Server is not a relational database, it cannot utilize the benefits of SQL for data access in a straightforward manner. Instead, data access is relegated to using C-ISAM routines that are embedded in FORTRAN programs or by creating custom programs that will read and extract the ISAM files. Any processing, ordering and formatting required by the consuming application must be handled by the application.

ADM is the primary consumer of met data. C-ISAM routines are embedded directly in the ADM code; however, the exact nature of the queries is unknown because ADM source code is unavailable for this project. WindInc has indicated the queries consist of the following characteristics: extracting specific met readings from all available stations over a designated time frame; converting the data to a format understood by ADM; and performing some basic aggregation-type analysis on individual data fields. The remaining consumers, any application using met data, interact through two FORTRAN programs designed to monitor station activity and simplify data retrieval. The first program (get_latest_timestamp) is used to find the timestamp of the most recent met reading for a supplied station. It accepts a station ID and returns the timestamp of the most recent data reading. This is the main tool used by WindInc to determine station inactivity. Employees periodically run this program manually on the stations to ensure each is operating. The second program (get_met_data) is used for met data retrieval. Get_met_data accepts a station ID and a timestamp and will return the entire met reading for that station for that timestamp. This is used by all applications that need access to met data. It does not accept a range of timestamps or station IDs and will not return a range of values.

2.2.3 Wind Fields

The second primary feature of the Met Server is the Meteorological Fields Component. The Meteorological Fields Component is responsible for producing wind field maps. A wind field map is a three-dimensional representation of how winds are behaving in the atmosphere in terms of speed and direction at a particular time within a designated slice of the atmosphere. Wind fields are an extremely effective tool for atmospheric modeling because it is a visual representation of current atmospheric conditions, rather than a set of hard data values from which inferences must be drawn. The maps are generated based on all available met data for a designated timeframe covering a specified geographic location.

The wind field code was written in 1995 using FORTRAN, C, UNIX shell scripts, and a UNIX image-manipulation program. Wind field generation is a time-consuming process because of the nature of met data access and the highly inefficient wind field code. The process has three primary steps. The first step is the gathering of all current met readings for each met station within the supplied geographic domain and timeframe. Each met station is scanned to determine those readings that fall within the timeframe by making repeated calls of get_met_data. A temporary text file is created for each met station containing all the met data readings within the supplied timeframe in a comma-delimited format. The second step is a formatting step. Each temporary text file created in the proceeding step is read and formatted before being stored in a new text file (wind_field_data.txt). This file will contain all the met data of the supplied domain. It is a temporary file that will be overwritten with each subsequent wind field generation. The final step involves feeding wind_field_data.txt into a program (wind_field_generator) that will create a wind field based on analysis of the supplied met data.

2.3 Business Problem

ADM and the Met Server were designed and implemented on tight budgets by inexperienced software developers with little knowledge of database systems. As a result, both suffer from performance problems, inflexibility, maintenance difficulties, monitoring troubles, and scalability issues. Concerning the Met Server, these problems have resulted in the inability of WindInc to fully meet existing business requirements and the additional opportunities that have materialized over the course of its existence. This has translated into lost business and lost opportunity for growth. While the problems of the Met Server are numerous, WindInc has targeted five pressing issues in need of correction: poor performance, difficulty in using met data, inconsistent and unstructured met data storage, the difficulty in monitoring the Met Server, and the need to support five years of historical met data.

2.3.1 Performance

For ADM, or any atmospheric dispersion model, to be effective in emergency response, met data access must be near instantaneous. In an actual emergency, a crisis manager has only moments to establish the scenario, enter the necessary data for the model run, procure the results, and make decisions for evacuation and containment. Weather conditions, particularly wind speed and direction, can vary drastically from moment to moment. If current met readings cannot be obtained in a timely manner, crisis managers will either make decisions based on erroneous data or will be unable to make them at all. Either can spell disaster for those in the path of the plume.

While ADM has been applauded for the quality and accuracy of its results, it has been criticized for its lack of speed. The data storage component of the Met Server has been identified as a bottleneck targeted for correction. The problem is that WindInc chose not to utilize an existing database system, but rather implement their own. Because WindInc is not in the business of designing database systems, design flaws were inevitable. While accessing ISAM tables is fast, the routines to manipulate, format, sort, and extract the met data (most noticeably into ADM) are cumbersome and slow. Because of the decision to use one ISAM table for each met station, these costly routines must be applied to each station numerous times. Over the course of a 24-hour ADM model run or any attempt to access a wide amount of met data, this cost becomes overwhelming.

2.3.2 Inflexibility of Met Data

The met data storage component was designed to work specifically with ADM by developers unfamiliar with relational database systems. This narrow-scoped design decision limited their ability to utilize met data on a company-wide basis. When a new use or need arises, WindInc has three options to address the problem: using their two FORTRAN programs to slowly extract the data; writing new access programs using C-ISAM routines; or by developing a new way to extract the data by reading the ISAM files directly.

Any of these chosen paths are inherent with complexity and will be costly to develop. The existing FORTRAN routines are slow and process single rows. This means the number of minutes in a designated timeframe is how often the routines will be called for each station. Any formatting or aggregation will have to be performed on the application end and these are typically inefficient. The remaining options necessitates developers possess knowledge of antiquated C-ISAM routines and the structure of the ISAM tables, while having intricate knowledge of the format and structure of the stations in question. Since this is out-dated technology and convoluted, it is reasonable to assume the typical developer will have to learn both aspects from the ground-up. This adds additional development time, which translates to additional cost. This pattern will be repeated for maintenance work.

2.3.3 Inconsistent and Unstructured Data Storage

Another major problem is inconsistent and unstructured data storage. There are no standards within WindInc for how met stations collect, store, and present met data. Because of this, met readings often vary significantly in format, name, and units. WindInc extracts and stores met data in the format provided by the met station. Users attempting to query the data have to study the descriptor files to learn the format and perform necessary conversions as needed. It is not uncommon for users to forget this, or be completely unaware, and perform analysis on data in different formats under the assumption they are in the same format. A related problem is an inconsistent use of default values. At times, met stations will not record certain met readings. This results in a default value being listed. Because this value can vary from station to station, it can be difficult for WindInc to determine what is and is not a valid default value. This means WindInc can extract and make decisions on met data that is not actually there. The system also lacks any form of integrity constraints. Readings with clearly invalid values such as letters have slipped into the database, resulting in application crashes and misleading analytical results.

2.3.4 Difficultly Monitoring Met Server

Still another problem is the inability to ascertain the status of the Met Server. No logging or error detection was implemented. Data values are either recorded or they are not, and WindInc determines this by scanning individual ISAM files through their FORTRAN routines. Since this process was never automated, WindInc employees must run them manually. If they forget or are too busy to do so, they run the risk of having an emergency occur when no current met data is available. Being unaware of this fact can be extremely dangerous if they make protective action recommendations on outdated data they believe is valid.

There are several additional problems related to this shortcoming. WindInc must monitor disk space usage manually and on regular intervals. WindInc has no reasonable method of determining the performance of their queries or potential ways of optimizing them. There is no easy method of determining who is using the data and if those users are permitted to access it. Backup and recovery features must be performed manually and are less than straightforward in nature. Essentially, most database administration functions involve the use of cryptic command line programs or utilities that are not easy or pleasant to work with.

2.4 Project Need

2.4.1 Database

The primary need for WindInc is a relational database system that can serve as the foundation for meeting all of their business needs related to meteorological data. This database will occupy its own tier, thus operating independently of ADM, the Meteorological Fields component of the Met Server, and any other application dependent upon met data. In its inception, the database will only store met data from the state of Colorado, as the collective modeling domains within Colorado provide an adequate test bed. This database will not serve an acting role in emergency response until WindInc becomes satisfied with its performance and reliability, or if emergency forces its immediate use.

While implementing a relational database system and its associated technologies will satisfy the stated needs of the previous section, WindInc has explicitly stated the following requirements must be satisfied: speed, the ability to support 300 active met stations, consistent data storage, and ability to store five years of met data.

The first and most critical need is that the database must be fast. The importance of speed in emergency response cannot be stressed enough. The eventual function of this database is for use in emergency response. Split-second decisions must be made and the decision-makers must have the necessary information to make those decisions. If the database cannot provide this information, then it could cost lives, as well as existing contracts and potential lawsuits for WindInc. Unfortunately, the only performance metric provided by WindInc is there satisfaction based on comparison with the existing Met Server and the overall performance of the database.

The second requirement is the ability to support 300 met stations while maintaining performance expectations. This number is a compromise between WindInc's ambition and the reasonable expectations of the database given the hardware limitations that will be addressed shortly. With only met stations from Colorado in use, this figure is not expected to be reached. If the number of active stations does breach 300, then WindInc has agreed to devote the necessary resources to accommodate this volume of data.

The third requirement is that the met data must be stored in a consistent format. This entails each met data reading being stored in one and only one unit. Functions will be written on the database level for conversions into the various units that may be needed by users. The responsibility of ensuring additional met stations conform to these standards must be borne by the developer, as the database will have no way of knowing what is being inserted. WindInc will provide the storage format for each met reading, as well as the available formats each met reading can be converted into.

The fourth requirement is that WindInc needs the database to be capable of storing the most recent five years of historical met data without sacrificing performance. Historical met data is of less use for emergency response activities, but the data is of great value to atmospheric and trend analysis. WindInc has atmospheric scientists and meteorologists on staff whose role could be expanded to include this type of analysis, should new opportunities become available. In addition, WindInc would also like to make this data readily available to the atmospheric research community for whatever their needs might be.

2.4.2 Graphical User Interface

WindInc expressed a desire for a graphical user interface to interact with the met data. Their current staff has basic familiarity with database systems and would prefer to avoid interacting with the database via command line tools or through database tools using direct SQL. However, the functionality requested demands a sophisticated application that is worthy of a thesis project in itself. Because this project is the database, a compromise was reached.

In addition to the database, this project will supply a functional prototype of the desired interface. This prototype will provide only a core set of basic functionality and the general layout of the application. Although the graphical elements for many features will be present, the functionality will not. Because it is a prototype, the code will not be optimized and any coding standards and policies of WindInc will not be strictly adhered to. Upon delivery of the GUI application, WindInc will assume control of the GUI and will tailor it to their needs as resources become available.

The basic functionality of the GUI centers on providing the ability for WindInc to view met data readings for a particular met station in a specified timeframe. Each met station will be grouped according to their associated modeling domain (or domains) and obtain all met data readings from a selected met station within the specified timeframe. Once displayed, the units of each met data reading can be adjusted according to user desires. In addition to displaying met data, each selection of a met station will display data about the station such as the type of station, the location of the station, the elevation of the station, and the frequency of readings.

The final feature provided is the ability to create user-defined met data. User-defined data is important for testing and for generating “what-if” scenarios. WindInc is an active emergency response team and is responsible for generating manuals and documents of steps to follow in an actual crisis. User-defined data is an effective way for WindInc to model specified scenarios under specific conditions. It allows them to see what would happen, how significant the result would be, and how to best respond to the crisis.

2.4.3 Meteorological Fields Component

The Meteorological Fields Component is responsible for generating the wind fields that provide WindInc with a crucial emergency response tool. While met data access is at the foundation of generating a wind field, the code performing the necessary analysis is complex and would require a significant software engineering undertaking. This is viewed as outside the scope of this project and because of this, will not be addressed.

3. Project Constraints

3.1 Financial Situation of WindInc

Before making design decisions, it is wise to have knowledge of what resources, in terms of money and technology, will be available. WindInc is a small company with limited financial capacity, surviving from contract to contract. While WindInc has procured a contract for this project, the amount of the contract is far below the amount necessary to realize their ambition. A realistic assessment of their immediate need versus their desired need was conducted to determine what this project will include and to what extent.

Idealistically this project would include not only the database, but the direct integration with ADM. Unfortunately, the state of ADM and the limited contract will not realize this vision. Instead, the project will focus on the development of a relational database and the prototype GUI. Of the two, the database is the most critical because it is the primary need. Despite not being active for emergency response, WindInc needs it to be production-ready, while simultaneously meeting the performance and volume requirements set forth by WindInc. WindInc is content with this approach because it will provide a low-cost demonstration, so they can evaluate if a relational database approach will provide the flexibility and robustness needed to support their daily operations.

3.2 Hardware Limitations

WindInc will be making sacrifices at every turn beginning with hardware. When the project was first discussed, it was believed the database would run on a devoted machine. A second corresponding machine of equal specification would act as a replicated server capable of assuming active duty upon failure of the primary server. Both machines would be new machines, albeit fairly low-end commodity machines. The machines were expected to have high-performance, dual hard-drives, and at least 2GB of RAM.

The reality is that not all of this vision will come to fruition, at least not in the beginning. No new machines will be purchased for the project. Instead, existing machines already serving roles in the WindInc environment will act as the hosts for the databases. The machines provided will be year-old Linux desktop machines running the Fedora operating system. The only specs supplied at the present time are:

- A single 2.8 GHz Intel ® Pentium ® 4 520 w/HT Tech processor.
- 1.0 GB DDR2 SDRAM, 533 MHz.
- A single 80 GB 5400 RPM hard drive.
- A DVD+/-RW/R CD-RW Combo W/DL support.

More in-depth details on brand and type are unknown. Since the machines are commodity machines, it is safe to assume those components are as well. It is possible second hard drives will be added, as well as additional RAM, but the project will be designed under the assumption these additions will not be made.

An additional complication with this scenario is that the machines hosting the databases will not be dedicated machines. The machines will each have an active installation of ADM running stand-by scenarios of specified domains every 15 minutes 24 hours a day. ADM produces four to five output maps and is fairly resource intensive during the duration of a run, which generally run in under two minutes for a 15-minute run. Additionally, these machines each run the wind fields code for their specified domain, the routines to extract and store met data readings, and the old Met Server code in continued support of ADM. All these programs run on regular intervals and will consume memory and processor overhead. This will cause the new database to compete for resources and this is expected to degrade performance, perhaps to levels below what WindInc deems acceptable. WindInc has acknowledged this possibility and because the database will not initially be active in emergency response activities, they are willing to accept it until dedication machines can be procured.

3.3 Software Limitations

3.3.1 Operating System

ADM is currently operating on Fedora version 4.0. Installed over a year ago, the operating system has never been patched or updated in any way by WindInc. Version 4.0 has a reputation for being buggy and WindInc shares this assessment. WindInc has accepted a recommendation to upgrade to version 5.0. As a result, all development and testing will occur on the Fedora 5.0 platform. Both computers will run this platform and WindInc will be responsible for patches and upgrades.

3.3.2 Database Software

While it is expected a low-cost relational database system will be selected, WindInc wanted an assessment of available alternatives for future consideration should suitable financial sources become available and this database growth surpasses their expectations. Three categories of database management systems were assessed for this project. Commercial database systems were the first. Database systems such as Microsoft SQL Server, Oracle, and IBM's DB2 Server are feature-rich products that have dominated the market. They can store an almost unlimited amount of data while maintaining performance requirements. The available features can meet almost any imaginable business need. Sophisticated backup and recovery, monitoring, and diagnostic tools are mature and ever-increasingly easy to use tools. However, these systems are expensive.

"Express Editions" of commercial RDBMS systems comprised the second category. SQL Server 2005 Express Edition, Oracle Express, and DB2 Express Edition are light-weight versions of their commercial counterparts. These are functioning database servers that are free in both cost and of licensing restrictions. These systems are designed for learning, low-volume applications, for being embedded in distributed applications, and for prototyping. Because they are developed on the same code-base, the databases can be easily ported into the commercial versions, which would be an excellent benefit for WindInc should they move to a commercial system. However, these systems tend to be burdened by capacity and usage constrictions.

Open-source RDBMS make up the final category. Developed by the open-source community, these systems are low-cost alternatives quickly gaining a competitive edge in the database world. Products such as MySQL and PostgreSQL are becoming increasingly feature-rich products than can run on a variety of operating systems and low-cost hardware. Their performance, reliability, and confidence of the user community have been increasing with each release. However, being open-source, they lack the sophisticated feature sets of the commercial products and obtaining help for problematic issues can be difficult.

After careful deliberation, an open-source option will be used. The financial investment required for a commercial database server is unrealistic at this point. A commercial license can run upwards of $10,000, perhaps more; whereas an open-source license can run upwards of a few hundred dollars. With a limited contract and amount of excess cash, WindInc can only devote resources to mission-critical endeavors. The new database will not be branded mission-critical until its performance and reliability has been proven acceptable through an operational trial of at least six months. During this period, it will function as an autonomous system and minimal resources (both financial and hardware) will be allocated in support of its operations.

The size restrictions were the primary factor in not using express editions. Requirements demand the database is capable of supporting 300 active met stations. If all stations operate with 90% reliability, they would conservatively generate over 35 million rows of met data a year. This estimates to approximately 7 or 8 GB a year. With the requirement for five years of met data, this translates to 40GB of data. The express editions are limited by size restrictions. SQL Server 2005 Express Edition has a maximum database size of 4 GB (MSDN), as does Oracle Express (Oracle). While DB2 Express is not restricted by database size limitations, a lack of developer familiarity with the product removed it from consideration (IBM).

Because WindInc has no vendor or brand loyalty, the question of which open-source database system to use was left to the discretion of the project designer. Three open-source database systems were considered: Firebird, MySQL, and PostgreSQL.

Firebird is a newer contender in the open-source RDBMS market. Firebird offers a feature set adequate for most database needs, a mature code base, and has a proven performance/reliability record, as it is the open-source code behind Borland InterBase 6.0. For this project, the following features were attractive: Firebird is free, it offers replication and hot backups, and it offers foreign key support.

The most attractive feature to WindInc is that Firebird offers a royalty-free distribution license. Essentially, this means Firebird can be packaged, distributed, and modified according to user needs without needing permission or providing compensation. Second, Firebird offers replication and hot backups. Replication is a key tool in ensuring 24/7 availability, while easing the burden of taking backups. Should replication not be implemented, hot backups will become necessary. Third, Firebird offers foreign key support. Data integrity is a high priority for this project and measures will ideally be taken at the database level, rather than at the application level. Bulk inserts of existing data may become a reality and foreign key support will play an integral role in ensuring the data is valid.

Firebird has drawbacks as well. The first is that newer products in the open-source world have fewer resources for support. These critical resources include: general documentation of Firebird; sources on areas such as optimization, administration, and development; and sources for frequently asked questions and other tricks that might be useful. The second shortcoming is the lack of sophisticated GUI tools. Modern DBAs are becoming increasingly reliant on GUI tools for administration, optimization, and development, largely in part to simplicity and the time-saving benefits. Not having these forces intuitive knowledge of command-line functionality, which coupled with a lack of documentation and general database knowledge, could be a difficult and time-consuming process (Firebird).

The second system assessed was MySQL, which is by far the most popular open-source RDBMS. With over 6 million downloads and powering countless web sites including Yahoo!, Citysearch, and Ticketmaster, MySQL has become the name associated with the open-source RDBMS market. MySQL has numerous benefits attractive to this project. The first is the availability of storage engines (table types) and the ability to select which storage engine or combination of storage engine is best suited for the database. This is a powerful feature in that it allows applications to be matched with the storage engine best suited for its purpose. Not every application requires transactions and by using the MyISAM engine, data retrievals will be faster. If the application requires transaction support, the InnoDB engine can service those needs. If tables need to be partitioned over multiple computers, the NDB Cluster engine should be considered. Intelligent mapping of storage engine to application type is a large reason why MySQL has deservedly earned its reputation for speed. A second advantage is the plethora of available GUI tools and their quality. As mentioned, the modern DBA is becoming increasingly reliant on GUI tools to perform the mundane tasks and help monitor the health of the system. MySQL Administrator can be used to perform most imaginable administration tasks. A DBA can configure the system parameters, perform backups and recovery, create users and set their privileges, perform certain monitoring tasks, and perform numerous other administrative tasks. MySQL Query Browser is used for creating, executing, and optimizing queries in a graphical environment. This tool significantly eases the burden of creating and monitoring queries, stored procedures, and functions, as well as providing useful debugging features. The MySQL Migration Toolkit provides a graphical utility for migrating scheme and data from various database systems to MySQL. Existing met data databases are in use by the atmospheric science community and some are willing to share that data. This feature could significantly ease the process.

MySQL has significant drawbacks as well. The first is that the two transactional tables (InnoDB and BDB) are developed by outside companies. These companies were both recently acquired by Oracle (Enterprise Users Calm). This has created considerable anxiety in the MySQL community as to how Oracle will proceed, if at all. By cutting off their transactional capabilities, Oracle could cripple the attempts by MySQL to enter the enterprise market, as MySQL is years away from developing their own transactional storage engine. The fear of having to re-implement a database in a different database system has some developers leaning towards alternatives, rather than MySQL. WindInc is among those sharing this concern.

A second drawback is a less-flexible licensing scheme. MySQL is released under the GNU GPL (General Public License). An application using MySQL has two options. The first requires derivative works (ADM) to be similarly licensed. The second is to purchase a commercial license for those applications and companies unable or unwilling to abide by the stipulations of the GNU GPL. The price of this license generally ranges from between $300 and $600. WindInc software, ADM specifically, does not operate under the GNU GPL license and once coupled with ADM, this database will not either. A commercial license must be purchased and the expected cost will be somewhere between $300 and $600. A final shortcoming is that MySQL lacks a free product that can perform hot backups on Windows systems. While the system will be Linux-based, there is a movement within WindInc to standardize the majority of their machines to a single operating system – Windows. Should this happen, this database would be among the applications ported over. A commercial product called Hot Copy can be purchased, but with the financial constraints of WindInc, this is not likely. Also, Hot Copy is limited in use in that it only works with the InnoDB storage engine. An additional product, mysqlhotcopy, is available for MyISAM tables, but this product only works on Linux systems (MySQL Network).

The final open-source RDBMS considered was PostgreSQL. Despite its roots tracing back to 1987, PostgreSQL has not been as enthusiastically embraced as MySQL. However, given the recent acquisitions by Oracle, developers and DBAs are beginning to give PostgreSQL a serious look.

PostgreSQL is the most feature-rich of the open-source RDBMS product line and its use is advantageous in many ways. One benefit is the aforementioned features line. It supports stored procedures, assorted procedural languages in a variety of popular programming languages, views, tablespaces, cursors, subqueries, and many other features. While version 5.0 of MySQL does contain many of these features, PostgreSQL is still ahead in terms of reliability. A second benefit is that PostgreSQL does offer hot backups on all operating systems. The need for hot backups for this database has already been addressed. A final advantage is that PostgreSQL is distributed under the BSD license. This fundamentally allows free use of the code and database as long as the credits are maintained.

PostgreSQL has numerous drawbacks as well. The first and often most frequently cited is that it lacks the speed of MySQL. If this database is to be successfully used in emergency response, it must be fast. PostgreSQL uses transactional tables, which are inherently slower, and countless tests of equally well-designed databases have proven MySQL is the faster system. PostgreSQL has made considerable improvements in speed, but it has yet to catch up. A second drawback is that PostgreSQL lacks the help resources of MySQL. The available material is significantly better than Firebird; however, the available sources on optimization and developing with the variety of procedural languages have proven substandard and not particularly helpful (PostgreSQL).

After an analysis of the options, the decision was made to use MySQL. Firebird was eliminated from consideration because the development team has no familiarity with the product. Of the remaining two, the primary factors of consideration were cost versus performance. While PostgreSQL offers the lowest cost, it is believed MySQL will offer the better performance, which is the top priority of this project. Given the minimal amount of users and infrequency of updates, it is believed the database can be designed to the point performance standards can be met.

3.3.3 Development Software

All development software and development environments must be freeware. Like their choice of database server, WindInc has no development language loyalties. In its present state, ADM is written in six programming and scripting languages. Because of this, the choice of language or languages has also been left to the discretion of the designer. C# has been chosen language for the development of the GUI. Visual C# Express is available for free download from the Microsoft web site. While this suite lacks the feature-set of Visual Studio, it does come free of licensing restrictions and a WYSIWYG editor that will ease the development of a GUI. It also comes with intellisense code completion capability, which is an enormous time-saver during development (MSDN). The problem is Visual C# Express is available only on the Windows platform. The solution comes by utilizing the Mono platform on Linux. Mono is a means of developing C# applications on Linux. However, Linux lacks a development environment that can compare to Visual C# Express. The solution is to develop on a Windows machine and port the code to the Linux development machine. The infrastructure of .NET should allow this transition with minimal difficulty, assuming the Mono platform has been successfully configured on Linux (Mono Project).

4. System Analysis and Design

Designing this database for WindInc will be challenging. The challenge is to design a fast system with minimal financial backing, inadequate and antiquated hardware, and dependent upon met data supplied from outside organizations.

4.1 Type of System

4.1.1 OLTP or OLAP

The first step in designing the database is to understand the type of system it is. The initial requirements indicate it will either be an OLTP (Online Transaction Processing) system, an OLAP (Online Analytical Processing) system, or a hybrid of the two. The first type is the Online Transaction Processing (OLTP) system. OLTP systems are real-time systems that support the mission-critical, day-to-day operations that sustain a business. These databases have higher degrees of normalization with many tables to eliminate data redundancy and backups occur frequently because any business cannot afford to lose operational data.

OLTP systems have numerous characteristics, although not all are required for a system to be considered OLTP. Among the most important is the high-level of performance. Because these systems are often mission-critical, they have to be exceptionally fast and reliable. End users, particularly customers, have limited amounts of patience and will quickly lose confidence in inadequately performing systems. OLTP systems are transaction-oriented. Being real-time systems, it a reasonable assumption these systems will be subjected to repeated INSERT, UPDATE, and DELETE commands, perhaps in large quantities. These commands must be executed quickly using limited amounts of resources because it is unacceptable for these queries to be detrimental to selection queries. OLTP systems represent the current state of an entity. Real-time means the data is an accurate reflection of the organization the current moment in time. Decisions are made from this data and inaccurate or out-dated data can lead to catastrophic decisions that can cripple an organization. OLTP systems serve well-defined processes. This means the database is subjected to more predefined queries and less ad hoc querying, although it does happen. The database is defined for a specific purpose and the queries are written to cater that purpose. These queries are run repeatedly and rarely change.

The second type of database application is the Online Analytical Processing (OLAP) system. OLAP systems are historical systems that function on historical data. They are aggregate systems designed for the reporting and statistical analysis that is used to identify trends and patterns that will help decision-makers make informed and intelligent decisions. These systems are highly de-normalized because data redundancy is not problematic. In fact, data redundancy is often purposely implemented in an effort to increase retrieval speed. Backups are usually nonexistent because the data can simply be reloaded from OLTP backups and they often grow to such enormous sizes that taking regular backups can become a costly and time-consuming endeavor (OLTP vs. OLAP).

There is no standard definition of what an OLAP system is. Performance is not as critical as in OLTP systems. Queries are typically aggregate in nature and can involve millions of rows of data. Obviously this will slow performance, particularly in full-table scans. Since these systems are not operation-critical, this is acceptable. INSERTS are larger and slower because of the large number of indexes. However, INSERTS usually occur in off-hour periods, thus the speed is not a problem. OLAP systems are analysis-oriented. This refers to the use of aggregate queries for reporting and was described above. OLAP systems represent the historical view of an organization. OLAP systems are fed from data of OLTP systems and this data is fed in after the fact, meaning they represent the past of a business and not its present state. On the other hand, OLAP systems serve the management community. The analysis capabilities of OLAP databases are designed to provide managers with an idea of what is happening in the business, what may happen in the future, and how the business should respond. OLAP systems serve undefined ad hoc processes. Data can be assessed in many different ways. The motivation for data assessment changes with each business need. Because of this, the design has to be flexible enough to accommodate many needs and not a select few, as it is nearly impossible to anticipate the ever-changing nature of the business world and market trends (Introduction to OLAP).

Part of drawing the distinction between designing an OLTP or OLAP system is to understand the intended use of the database (OLTP vs. OLAP). In a generalized sense, WindInc wants the database to support two distinct areas of functionality. The first function is to support their existing (and future) real-time emergency response applications, the eventual integration with ADM and the Meteorological Fields Component, and be the backbone of a planned web site to provide real-time met data access to the public. This is what must provide instant answers to the questions concerning the current state of the atmosphere. The second area is for needed reporting and analysis functionality on historical met data. Nothing falling into this realm will be directly used in emergency response activities. These features are for analysis after the fact. For example, WindInc might need to find the percentage the wind was blowing each direction the past year. A second example might be to find the average relative humidity by year for each of the last five summers when the temperature is above 80 degrees Fahrenheit. While these are speculative guesses, it is assured their queries will be more aggregate and inquisitive in nature than the type required to support emergency response activities.

Unfortunately these two intended uses conflict with one another in terms of design. Supporting WindInc's emergency response capabilities would be best met with an OLTP system while their reporting/analysis needs would be best met with an OLAP solution. Developing a hybrid solution to accommodate both needs is a possibility. However, it is feared this attempt, coupled with hardware limitations, might degrade performance. WindInc will need to make a decision on whether they want an OLTP system to support their emergency response activities on an OLAP system supporting their reporting/analysis needs; whether they want a hybrid system to support both; or whether new options should be explored to address the problem.

4.1.2 Decision

WindInc was addressed with the concerns presented by each solution and possible options for meeting both needs. After deliberation, it was agreed upon to implement a single system facilitating both options with the intention of adding a second system in the future to segregate OLAP functionality. The primary function of this database is to serve emergency response activities. To best achieve these results, the database will be designed as an OLTP system. Existing historical met data will not be ported from the Met Server into this system. This means the database will not reach a size where OLAP activity will become a problem for some time. Within six months of implementation, WindInc hopes to have procured a system to act as host for the second database. This second database will act as a data warehouse. It is from this system all OLAP activity will occur. This segregation will allow the active emergency response system to remain small of size and if designed properly, will ensure optimal performance. Because the OLAP is targeted for the future, implementation details will not be addressed.

4.2 Requirements

The following sections address only the expectations of the OLTP system. WindInc was vague in specific requirements, which allowed a great deal of latitude in design. Their stated requirements will be assessed below, as well as any explanation on concessions that need to be made to accommodate the requirement without impacting the database negatively.

The first, and by far most critical, requirement is speed. If the database cannot perform acceptably, it can be of no use to active emergency response solutions in any form. Unfortunately the only supplied performance metric is that the database must be fast and its performance must exceed that of the existing Met Server. No performance figures have been provided. This means the success, or failure, is subject to the satisfaction of WindInc. In an effort to meet this requirement, the database will be designed for speed.

The second requirement is the creation of an application that will extract met data from its source, format it to meet the internal standards, and then store the formatted data. This application is meant to be the primary means by which new met data is stored. The application is to run continuously at periodic intervals, determined by the frequency of data readings per station. They range from every minute to once an hour. For each met station, the application must know how to extract the data from the source, what met readings are of interest, how to properly convert the readings into the internal formats, and how to store them.

The third requirement is the implementation of data integrity measures. A problem plaguing the old Met Server was the large-scale duplication of met data readings and the inability to prevent it. Met stations are notoriously unreliable. They go down frequently and can be unavailable for weeks at a time. When WindInc requests met data, it is requesting the most recent met data readings. In the case of a down station, this is the last data reading before the application ceased operation. The Met Server will store this reading as a new reading. This leads to high levels of duplicated data, which wastes space, skews analysis results, and slows down data extraction routines.

The fourth requirement is the ability to add new stations quickly. WindInc is on a never-ending quest for new and quality sources of met data. When a new station is found, they have indicated the need for the ability to quickly interrogate the station for the necessary data, a method to extract it properly, and the ability to store it. This requirement has been downgraded in strictness and has been removed from the scope of this project. The problem is that the majority of met data is extracted by scraping HTML pages. With the lack of discipline and structure in HTML, writing an HTML parser is a precise endeavor that requires detailed knowledge of the page structure. Writing a code generator for a HTML parser would be an incredibly difficult and expensive undertaking. WindInc has agreed this feature is beyond the scope of this project and unlikely of pursuit.

The fifth requirement is to implement a means of high-availability. A database operating in an emergency response environment must be available at all times. Because WindInc is supplying two machines for this project, replication will be the chosen path. Utilizing replication also has the added benefit of satisfying a second requirement – an easier means of backup and recovery. The specific details will be addressed shortly.

The sixth requirement is to provide the ability to extract the met data in various formats, specifically XML and the comma-delimited format. A problem in the atmospheric community is a lack of standards on how met data is presented to the community. Each met station determines the format of the data, and even a common source has been known to have different formats amongst their stations. This makes data extraction a difficult and time-consuming process. WindInc believes met data should be an open, easily-used commodity and they want to begin spearheading this campaign on the foundation of this database.

The seventh requirement is to have five-years of met data online. There are numerous reasons for this requirement. One reason is that WindInc needs the ability to extract interesting weather conditions for model runs. This is an excellent way for WindInc to study accident scenarios under actual weather conditions and document the results for the preparedness documentation they distribute to clients. A second reason is for the purpose of research and meeting business opportunities through the analysis of historical data. The Met Server renders this nearly impossible, especially when analysis spans multiple years or multiple stations. Still another reason is to provide historical data for use by the atmospheric community.

4.3 Database Design

4.3.1 Storage Engine Selection

One unique benefit of MySQL is that it provides the designer the ability to select the storage engine type (table type) for their database. Each storage engine offers specific features designed to cater to specific types of applications. Each storage engine also has drawbacks that may create shortcomings in the database. A designer can even mix and match table types within a single database, although this creates additional levels of complexity. While many storage engines are available, the decision for this database has been limited to two types: MyISAM and InnoDB.

The MyISAM storage engine is the default engine for MySQL. MyISAM tables are noted for their excellent performance and useful set of features. MyISAM tables are attractive for this database for several reasons. The first is that reads (SELECT statements) are extremely fast. MyISAM tables do not support transactions or referential integrity. This translates to limited overhead on the tables with relatively unhindered data extraction. A second benefit (related to the first) is that the limited overhead and nature of storage means less disk space is required for storage. MyISAM has an efficient storage method in which tables are represented by two separate files: a data file and an index file. A final attractive feature is that the format is platform neutral and can be ported to MySQL installations on any platform. If WindInc does standardize to a single platform, this feature will become extremely important (Zawodny & Balling, 2004).

InnoDB tables are the choice for databases requiring transactions. The obvious benefit here is that they support transactions. InnoDB transactions respect ACID principles and offer rollbacks for unwanted transactions that slip through. A second benefit is that InnoDB offers greater flexibility in concurrency than MyISAM tables. MyISAM tables use shared locks for reads, but exclusive table locks on writes. InnoDB tables use Multi-Version Concurrency Control (MVCC) to achieve the higher level of concurrency. A final benefit is that InnoDB tables offer referential integrity in the form of foreign key constraints. With the WindInc requirement of data integrity, this is an enormous benefit that will simplify database design by implementing referential integrity at the table level. Implementing referential integrity in MyISAM will be far more complicated, as it will have to be implemented through stored procedures or on the application level (Zawodny et al., 2004).

After careful consideration, the decision was made to use the MyISAM storage engine. The driving factor behind this decision was the speed offered by MyISAM and the nature of this database. First, this database does not require transaction support. Each met station is an individual unit and must be treated accordingly. Even if coming from the same geographical met station, the success or failure of a met reading entry should not be contingent on the success or failure of proceeding stations. The second reason is this database will consist of mainly SELECT and INSERT/UPDATE queries, upwards of nearly 100% of the activity. This type of application is best suited to reap the speed benefits offered by MyISAM tables. The third reason is this will not be an overwhelmingly large database in terms of table size. The efforts to maintain database integrity through stored procedures will not be substantial, nor will the performance cost they might incur.

4.3.2 Database Tables

After the analysis of the requirements and design considerations, the end result was a fairly simple database. What follows are the tables to be included and a brief description of their contents.

- Units - The units for various meteorological readings. Examples – mph, m/s, degrees C. Preference in reading format varies from user to application and met stations record readings in various formats. It is important to know the format of each reading so it can be converted to the proper internal storage unit, as well as provide users with the ability to select a desired format and convert between them.
- MetReadings - A unique meteorological reading. Examples – wind direction, wind speed, ambient temperature. The met stations lack a naming standard; therefore, a reading such as “wind direction” might be observed under the following names: “wind direction from north”, “wind direction from n”, and “wind direction from true north”. It is imperative to know what each station is actually recording to limit confusion, avoid data redundancy, and so assessments are not based on mistaken met readings.
- StationMetReadings - A linking table that stores the met readings for each station.
- Timezones - List of time zones. Met stations will store data with a timestamp, but there is a lack of consistency on the timestamp used. Some

use the time zone where station is located; others use more "universal" time zones; while others use something different. Time conversions must be accounted for when storing met data. There are a finite amount of time zones and this table is expected to change infrequently.

- StationType – The type of met station. Examples include: balloons, profilers, and surface stations. There are limited types of stations and this table is expected to change infrequently.
- Stations - A station is a met station at a particular height. Real-world met stations may be comprised of numerous levels and each level may record different met readings. Analytic distinctions can be made on readings according to the level they are recorded from, largely because of the height distinctions. For example, Station A has 3 levels – 1, 2, and 3. Therefore, station A is actually three distinct stations – Station A, level 1; Station A, level 2; and Station A, level 3.
- Domains - A domain is a geographically-bounded area that ADM has sufficient met data coverage and geographic data to perform model runs. For example, Denver County may be considered the "Denver domain". Any met station found within the boundary may belong to the "Denver domain" and if it does, it will be used in modeling of that area. Domains can overlap and met stations can belong to multiple domains.
- DomainStations – A linking table that stores domains and the met stations that belong to them.

- StationData – Pertinent data (mostly geographical) about each met station location. This is a separate table to account for the level distinction. Each level of each met station is considered a distinct station, but there will only be one station data entry. Met stations can be real stations or user-created for the purpose of user-defined data.
- MetData - The actual met data recorded. Initially, there will be 35 met readings available for collection. There is a possibility new met fields will be added in the future, but once a met reading belongs to this table, it will not be removed.
- UserDefinedMetData – This is met data that will be created by users. This will contain the minimum met readings required to run ADM and a few others useful to the model. This data is expected to be altered and deleted frequently; therefore, a decision was made to include it as an individual table, rather than causing unnecessary activity on the MetData table.
- Scenarios - A scenario is a collection of met data used for an individual modeling run on ADM. WindInc wants "interesting" model runs stored for later study and the creation of preparedness documentation. The same applies to user-defined runs.
- ScenarioData – A linking table to store the met data associated with each scenario.

Of the database tables, MetData is by far the most active and most critical to the functionality of anything requiring met data. In a full production environment, the table row size will grow into the tens of millions quickly. An index strategy will be developed and enhanced as time progresses. Initially, the table will be indexed according to station id and the timestamp. Initial usage patterns suggest a near 50/50 split between accessing the data based on the station id and the timestamp. Analysis and experimentation will indicate the best solution. Additionally, the choice of using MyISAM tables will allow the benefit of index compression on timestamp indexes. Because the timestamp is a string column in the database, index compression will enable common string prefixes to be factored out. In a large table, this can be a considerable space-saving benefit.

The remaining tables row size will be small. While MetData will climb into the millions, only StationMetReadings has the potential to surpass a thousand rows, unless the database happens to expand to coverage of the entire United States. Because of their size, only primary keys will be implemented. This is for the sole purpose of maintaining data integrity. A more intelligent index strategy will not be necessary because they will be ignored, as sequential scans will be the faster option.

4.3.3 Database Objects

Due to the selection of MyISAM tables, database security, and in following good principles of database design, all database operations will be restricted to stored procedures, functions, and views. Using stored procedures will allow the database to satisfy the requirement of data integrity at the expense of more complicated procedures and slower INSERT operations. WindInc has indicated a willingness to accept slower INSERT operations as long as retrieval performance meets their expectations. Stored procedures will be created to handle all INSERT, UPDATE, and DELETE operations for each table. Database users will have their access restricted to these procedures and will not be permitted to add to or alter the contents of the database manually.

Database functions will be created to handle the conversions from one unit to another. WindInc has supplied a list of almost sixty desired conversions for implementation. It is expected more will be added. Database views make up the final objects of importance. The MetData table will contain 35 data fields. This means every entry into the table will include 35 data fields. Default values will be provided for absent fields. The idea of a view for each station is to provide the user a simple way to query met fields of interest available from that specific station (Harrison, 2006).

4.3.4 Backup and Recovery and Replication

WindInc is supplying two computers for the project. Because of this, backup and recovery procedures will be facilitated through replication. Replication is a process by which an exact duplicate of a database (a slave) can be maintained on a separate server by periodically querying specific logs of a second server (the master). By utilizing replication, WindInc's requirement of high-availability can simultaneously be achieved.

MySQL offers asynchronous replication. Asynchronous replication means the changes to the slave are made after the changes have been made to the master. This means there will be a slight delay between the two systems. In MySQL, all changes on the master are recorded in the binary logic once replication has been implemented. The slave server, or servers, will query this log at periodic intervals to acquire the changes. In MySQL, the querying of the binary log occurs so quickly the two systems will appear to be in-sync at all times. This will allow the slave serve to act as a failover machine in case the master goes down. It should be noted here that any programming and DNS configuration to implement failover support will be handled solely by WindInc.

Backups will be performed by taking down the slave system and taking a complete backup of the database. The backup will then be compressed and stored on disk, along with the necessary replication files – the master.info file, relay logs, relay index, and so forth. These files are critical to successfully restoring a failed slave. Once a week, the above mentioned files will be burned to DVD to provide a recent off-computer copy of the database. This backup policy is in accordance with the desires of WindInc to have a simple solution they can perform with minimal effort. When the addition of the data warehouse becomes a reality, this backup procedure will perform effectively, given the small size of the database (Zawodny et al., 2004)

Recovery will be handled based on what has failed. In the event of total slave failure, a snapshot of the master server will be taken and ported to the slave server. At this point, the slave will be reconfigured as a slave and replication will acquire the data changes since the snapshot was taken. In the event of individual table failure within a slave, online backups can be taken from the master and rebuilt in this manner. Any changes can be observed from the log files and copied accordingly, assuming replication missed them. In the event of a master failure, the slave will become the new master and the former master will assume the role of slave. Recovery procedures will be the same at this point. In the event of both master and slave failure, both servers will be rebuilt from the copy of the latest complete database. In the event of erroneous data being inserted or mistaken rows being deleted, the corrections will be found by querying the binary log, which records a list of all changes made to the database. Corrective measures will then be taken through queries, as rollover capabilities will not be available.

4.3.5 Data Warehousing

The implementation of a data warehouse would have enormous benefits for WindInc. First, by having a warehousing option, the amount of online data in the emergency response system (OLTP system) could be kept to a minimum, possibly no more than a month of data. Historical data is irrelevant in an actual emergency response crisis. This would ensure the performance demands while allowing for the expansion into modeling domains outside of Colorado in spite of the hardware limitations. A second benefit is the assurance WindInc could continue to rely on the simplified backup and recovery procedure outlined above. WindInc has expressed little interest in becoming database experts and would like for necessary procedures to remain as simple as possible. A third benefit is the unlimited use of historical met data for analysis and reporting without the concern of bogging down their emergency response capabilities. This would be enormously valuable for research and certain business opportunities presently unable to be fulfilled.

4.3.6 Database Monitoring

To simplify monitoring the databases, the MySQL Administrator will be installed on all machines. MySQL Administrator is a graphical administration tool that even a novice can use relatively effectively. It can be used for user administration, keeping tabs on the size of tables, the status of replication, can aid in backup and recovery procedures, can be used to create new database objects, can allow simplified access to important log files, and can be used to monitor the overall health of the database. At some point, WindInc will assume administrative responsibility of the database. Basic documentation and training will be provided to interested parties.

4.3.7 Miscellaneous

WindInc needs the ability to output text in numerous formats. The two most pressing are in XML and the comma-delimited format. At some point in time, this capability will be present in their GUI application. For the time being, it will have to come via built-in command tools supplied by MySQL. MySQL can easily output data in both of these formats with simple commands. However, this means the WindInc staff will need to receive additional documentation and training. Although it is not ideal, they will accept this solution.

4.4 Met Data Storage Application

4.4.1 Met Data Sources

WindInc has no met stations of their own, nor will they in the foreseeable future. Because of this, they are forced to rely on outside sources. The two most common methods of obtaining met data are by scraping HTML pages and by parsing text files obtained via FTP.

Parsing HTML is extremely difficult because it lacks strict standards and tiny changes to the layout of the page can render a parser useless. Also, with a lack of standards amongst the atmospheric community, each organization has different layouts for their pages, thus each station generally requires its own parser. This is both time-consuming and costly. Parsing HTML can also be a slow process due to the intelligence required by the parser because of the layout of a page and its size.

FTP can also be problematic. WindInc has one primary source of data, accounting for nearly 75% of their met stations. The problem is that this organization presents met data in a single file that is appended daily, rather than replaced, on subsequent readings. This means a typical file can reach up to 30MB by the end of the day. This can be an extremely slow process if you account for the time it takes to download the file, parse out the necessary information, and store it. Data extraction from this source also demands the developer download a complex FORTRAN application to perform the extractions. This application is highly complicated, poorly documented, and not easy to learn.

4.4.2 Met Data Extraction Application Design

The met data gathering application (presently unnamed) will be a console-based C# application. This means existing parsers and data extraction methods created by WindInc will not be used directly. However, source code will be provided to help ease the burden of creating new parsers.

The main program will be an endless loop that calls all known parsers at specified time intervals, as each station acquires data in different time sets. The main loop will first cycle through a list of HTML parsers, calling a parse method on each in turn. If the method succeeds, the met readings will be recorded. If the method fails, the station in question and timeframe will be noted in an error log file. The sophistication of this error system will increase with the passing of time. Upon completion of the HTML parsers, the loop will then cycle through a list of FTP parsers, repeating the procedure above. This will ensure the much slower FTP parsers are called last. This loop will repeat every minute. This should provide ample time for all parsing and index rebuilds required by the database to accommodate the new INSERT operations.

Three types of C# objects will be created to run within this main application. The first is an interface called IParser. IParser will describe the behavior of all parser objects. This will simplify how the loop looks for and calls parser methods. The second is a base parser class called Parser. This class will contain all members and methods common amongst parsers. This is a meant to avoid buggy code redundancy. The third type is all the parsers that need to be created. They will follow the naming format of ParserStationName. As an example, a met station called NOAA will be called ParserNOAA. The parser classes will inherit the base class Parser, as well as the IParser interface. Unique methods will be created as needed, as well as the overriding of any Parser methods.

The addition of a new station will require a temporary shutdown of the met data gathering application, but WindInc is not concerned by that, assuming the new class has been well-tested and downtime is minimal. This code will rest on all machines linked by the replication scheme; however, it will only run on the master. Should a slave become a master, it is important that the met data gathering application be started on the new master immediately.

5. Aftermath

5.1 Implementation Results

The database was implemented without incident. The database was created using MySQL 5.0.22, which at the time of creation was the most recent version. The database was created on a development Windows machine running the XP operating system. Database tables were created using the graphical features of MySQL Administrator and all database objects were created with MySQL Query Browser. Users were created using MySQL Administrator, but privileges had to be configured using straight SQL because the MySQL Administrator proved buggy in this area. The complete SQL structure of the finished database was dumped to a text file and successfully ported to the Linux-based production machine running the Fedora 5.0 operating system. Presently, the database is absorbing met data from forty stations.

Both the GUI prototype and met data gathering application were developed using Visual C # 2005 Express Editions on a Windows XP machine. The reason behind this was to harness the development benefits offered by Visual Studio. Linux lacks an IDE that can rival Visual Studio in terms of C# development. The Mono platform is included in a standard installation of Fedora 5.0, thus alleviating any installation burdens presented by Mono. With the environment properly configured, both applications were ported with minimal effort. The met data gathering application is a console-based application configured to run in the background when the machine boots. The GUI must be started manually either by a direct command or via a button created specifically for this purpose. The met data gathering application also has a button in case it needs to be restarted.

5.2 Performance

Despite not having a dedicated machine, the database has been performing above expectations. The database has remained stable and data retrieval has been near instantaneous. Any noticeable performance concern is the on the GUI side, which was expected because the GUI application code was not optimized. The actual retrievals from the database have been averaging less than a second. This is from a combination of the ability of MySQL and the minimal data volume. Ten of the forty stations pull new met data every minute. This computes to 1,440 daily readings per station for a total of 14,440 new readings daily. The remaining thirty stations pull new met data every five minutes. This computes to 288 daily readings per station for a total of 8,640 new readings daily. The combined total equals 23,080 daily readings, 161,560 weekly readings, and approximately 646,240 monthly readings (Appendix B). These figures translate to approximately 203 MB for the MetData table data file and another 12.3 MB for the index file. With the met data gathering application checking for new readings every minute, this gives MySQL ample time to perform its function without consuming excessive resources needed for other tasks.

5.3 The Future

The priority addition planned for the future is the addition of the OLAP server for data warehousing purposes. This machine is expected to be added within the next six months. It is unknown at this time the specs of this machine or whether or not it will be a new machine or one pulled from the existing network. Regardless, the machine will run the same operating system as the OLTP and replicated machine. The design of the database has not been discussed, although it is not expected to be much different from the OLTP database due to small number of tables. It might be feasible to leave the design as is and implement the warehouse as a second replicated server. However, this would create an additional level of complexity that WindInc might want to avoid.

A second discussed addition is a feature on the GUI that can harness the database to monitor station status at all times. A met station going down is a regular occurrence and WindInc has no indication of this other than querying the station to see if it returns recent data. This puts WindInc in danger of an emergency occurring without recent met data. A method of providing WindInc with notification of a downed station would be an important tool in ensuring met data interruptions are held to a minimum. This would also be used to determine the statistical reliability of each station.

A third future addition is a web page providing access to the met data. There are two reasons for this vision. The first reason stems from the desire to provide met data to the atmospheric community in common formats, specifically XML and the comma-delimited format. The vision is to allow users the ability to select stations of interest, individual met readings, and the format of those met readings. Providing this type of data access would be a key step in eliminating the problems faced in acquiring met data from multiple sources with disjoint standards. Of course, this would be subject to the permissions granted by those operating the met stations from which met data is drawn. The second reason is to offer the ability to perform tasks handled by the GUI and the eventual OLAP server via the web. The intent is to extend the use of met data outside of the office, because employees travel and work from home. This would expand the capabilities of all emergency response employees considerably, as emergencies are not restricted to standard business hours.

5.4 Project Reflections

This project was, and is continuing to be, a series of lessons and realizations. Because this project began with requirements gathering and continued on through design, development, implementation, and administration, these lessons are far-reaching and considerable in number. Rather than completely rehash, the three most important realizations will be addressed: the importance of strict requirements, the effectiveness of MySQL, and the potential importance of this project for WindInc.

One realization that stands out is the importance of demanding the client provide a clear, tangible vision of their desire with solid requirements. The development team was formerly employed by WindInc and the idea for this project began four years ago during their tenure. Because of this, WindInc was often lethargic in providing requirements and approving those that did exist because they felt the development team understood the problem well enough to drive it to fruition. While the development team enjoyed the freedom this provided, it only led to disagreements on features, misunderstandings on functionality, and last minute changes that forced redesign. On two separate occasions the project was redesigned from scratch. It was finally agreed upon by both parties that WindInc alone would provide the requirements and were prompt and clear with them from that point on. Once this occurred, the project proceeded smoothly and to WindInc's satisfaction.

A second realization is the effectiveness of MySQL for this project and as a legitimate RDBMS. The decision to use MySQL was primarily driven by finances. It was unknown how MySQL would respond in WindInc's production environment on non-devoted machines running proprietary hardware, as its performance history is not as considerable as those of the commercial RDBMS genre and this did create a degree of apprehension. That apprehension has dissipated since its implementation. Its performance has exceeded the expectations of both the development team and WindInc. The test environment was configured to absorb met data from 300 met stations, the maximum capacity on the supplied hardware. Row counts quickly grew into the millions and its performance still met WindInc's expectations. Once the OLAP server assumes an active role, the emergency response servers should never reach this row count, thus furthering its expected performance. In addition, not once has the reliability of MySQL been an issue. The test server and production servers have been running for months and not a single crash has occurred. There is no doubt that MySQL will be able to serve effectively in an emergency response environment.

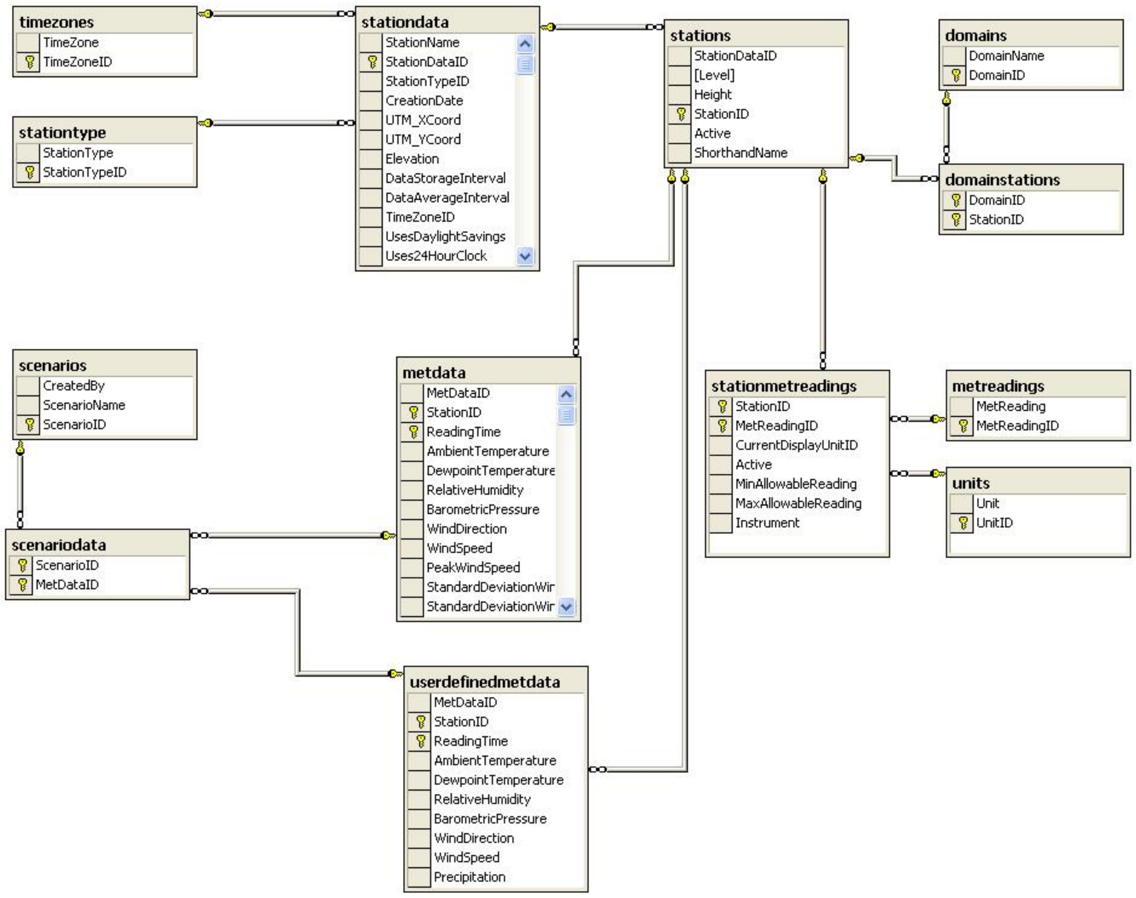
A third realization is how useful this project could be to WindInc and the atmospheric community. WindInc, and many in the atmospheric community, want easy access to met data. Met data sources are of a wide variety and of differing formats. While it is easy for a person to see the data via a Web page, it is far more difficult for computers to extract and process the data. A continuing problem is the providers changing the formats of their Web pages without notification. This is breaking the HTML parsers and causing lost data. Once WindInc implements a Web page running off their OLAP server, met data from a potentially unlimited number of sources will be provided in a common format like XML, which is easily understood by computer applications. If met data providers follow this lead, the proliferation of met data could be unbounded.

5.5 Conclusion

The intent of this project is to demonstrate the effectiveness of a relational database solution within the WindInc environment. Factoring in the problems this solution rectified with the opportunities it created, the project was successful. What WindInc had was a vast repository of valuable data they were unable to harness to its fullest potential. Now they have a single repository of easily-accessed and structured data with a limitless potential for use. Once completely integrated in their environment, WindInc will have an important tool by which they can dynamically embrace the opportunities of the future.

Appendix A

ER-Diagram



Appendix B
Growth Rate

Frequency of Readings (Minutes)	Daily Readings	Weekly Readings	Monthly Readings	Yearly Readings
1	1440	10108	43800	525600
5	288	2022	8760	105120
10	144	1011	4380	52560
15	96	674	2920	35040
20	72	505	2190	26280
30	48	337	1460	17520
60	24	168	730	8760

Table 1 - Number of new met data readings

Frequency of Readings (Minutes)	Daily Readings	Weekly Readings	Monthly Readings	Yearly Readings
1	1.859	13.047	56.534	678.404
5	0.372	2.610	11.307	135.681
10	0.186	1.305	5.653	67.840
15	0.124	0.870	3.769	45.227
20	0.093	0.652	2.827	33.920
30	0.062	0.435	1.884	22.613
60	0.031	0.217	0.942	11.307

Table 2 - Database Size (data files + indexes)

Annotated Bibliography

Enterprise Users Calm as MySQL Community Freaks (2006). Retrieved from http://www.eweek.com/article2/0,1895,1930360,00.asp

Firebird - Firebird Knowledgebase (2006). Retrieved from http://www.firebirdsql.org/index.php?op=doc&id=userdoc

Harrison, Guy (2006). *MySQL Stored Procedure Programming.* O'Reilly Media Inc

Highly Available Databases (2004). Retrieved from http://www.firstsql.com/highavailability.html

IBM - DB2 Express-C (2006). Retrieved from http://www-306.ibm.com/software/data/db2/express/

Introduction to OLAP Data Warehousing Review (2006). Retrieved from http://www.dwreview.com/OLAP/Introduction_OLAP.html

Mono Project (2006). Retrieved from http://www.mono-project.com/Main_Page

MSDN - SQL Server 2005 Express Edition (2006). Retrieved from http://msdn.microsoft.com/vstudio/express/sql/

MySQL Network (2006). Retrieved from http://www.mysql.com/

OLAP vs OLTP (2006). Retrieved from http://blogs.netindonesia.net/kiki/archive/2006/02/28/8822.aspx

Oracle Technology Network - Oracle Database 10g Express Edition (2006). Retrieved from http://www.oracle.com/technology/products/database/xe/index.html

PostgreSQL - PostgreSQL 8.1.4 Documentation (2006). Retrieved from http://www.postgresql.org/docs/8.1/static/index.html

Why Use OLAP? (2006) Retrieved from http://www.mitsonline.com/news-why-use-olap.htm

Zawodny, Jeremy D., & Balling Derek J. (2004). *High Performance MySQL*. O'Reilly Media Inc.