University Data Warehouse Design Issues: A Case Study

Melissa C. Lin

Chief Information Office, University of Florida

Abstract

A discussion of the design and modeling issues associated with a data warehouse for the University of Florida, as developed by the office of the Chief Information Officer (CIO). The data warehouse is designed and implemented on a mainframe system using a highly de-normalized DB2 repository for detailed transaction data and for feeding data to heterogeneous data models owned by different administrative units. The details of technology selection, business requirements, tools building, cultural challenges, architecture modes, models, and hardware information will be described. The data warehouse analysis, logical and physical design, application server, and implementation issues will also be explained.

I. Introduction

The computing and data service environment at the University of Florida is large and diverse. It was formed within the numerous political and funding boundaries of the past several decades. The advancement of new technologies and the need for quick access to up-to-date student and employee data have put great pressure on the university to develop and to maintain a central database for administrative use. The data warehouse project had to utilize existing computing facilities and databases, bringing them together and using their strengths in new ways. The Office of the CIO had to create a data warehouse that supported all administrative units and provided easy, timely, accurate access to the information maintained by key administrative offices across campus. These offices include the Office of Information Systems, the Office of the University Registrar, the Dean of Students Office, the International Student Center, Student Financial Affairs, the Health Science Center, Academic Advising Center, University Libraries, the Graduate School and others.

Our data warehouse is a central, logical site that stores many data models, supports central management's priorities, and complements the university's business needs. It facilitates a broad scope of tasks, such as:

- Extracting data from legacy systems and other data sources,
- Cleansing, scrubbing, and preparing data for decision supports,

- Maintaining consistent data in appropriate data storage,
- Ensuring and protecting information assets at minimum cost,
- Accessing and analyzing data using a variety of end user tools,
- Mining data for significant relationships, and
- Providing both summarized data as well as extremely fine-grained data.

II. Academic Data Warehouse Analysis

UF Data Warehouse Requirements

After the concept and the budget for the data warehouse was approved, many administrative units volunteered to become campus test sites and were involved in the data warehouse project from the beginning of the design phase. Our customers helped us to ensure that the data modeling and system design precisely fit their business requirements, which included protection of the source data from legacy systems, consistency of transaction and warehouse data, intense security, naming standards, and a variety of reports needs.

Frequently asked questions during our requirement analysis:

Q: What level of security will the data warehouse provide to ensure and protect data assets?

A: We will follow the Buckley Amendment to provide system and data security down to the individual record level. The owner of the data source will authorize usage.

Note: The Buckley Amendment is State University System rules, state statutes, and the Family Educational Rights and Privacy Act of 1974.

Q: How do we maintain the data increment and recover from potential system failure?

A: We will provide two parallel interface systems using EAGLE (Enhanced Application Generation Language for the Enterprise) that is a locally written and developed application for Web access, and Java applications to access the data warehouse. Each serving is as the other's substitute. In addition, we also will back up the system daily.

Q: What is the basic knowledge required of end users?

A: We build many choices of canned queries for end users. So they click the buttons to select the options that they need. We also provide training for end users that need to run their own queries.

The Architecture of the UF Data Warehouse

We decided to build our data warehouse on our existing UF systems that have already offered certain functions we need. The NERDC (Northeast Regional Data Center is located on campus) provides hardware and software infrastructures and IBM mainframe 9672-R55 with DB2/OS390 support. After acquiring a better understanding of our hardware systems, network and operating systems, software tools, and business requirements, we began to design a UF data warehouse taking data source, technical infrastructure, customer expectations, and budget into consideration.

We assessed our software needs during initial design of the architecture review. The software needs to accommodate our data warehouse are to build data models, to extract, integrate, transform, and load data to DB2 tables, and to deliver data via application servers to end users. The architecture of the UF data warehouse, as shown in Figure 1, supports an integrated, single, enterprise-wide collection of data with object-oriented, nonvolatile, and time-variant characteristics.

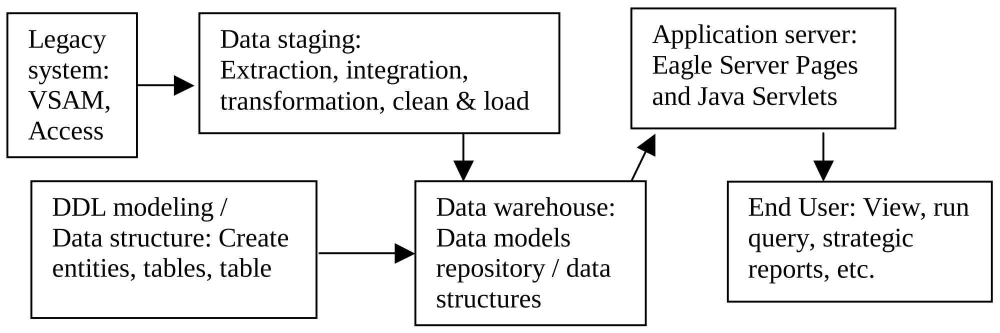

Legacy system: VSAM, Access
Data staging: Extraction, integration, transformation, clean & load
Application server: Eagle Server Pages and Java Servlets
DDL modeling / Data structure: Create entities, tables, table
Data warehouse: Data models repository / data structures
End User: View, run query, strategic reports, etc.

Figure 1. The Architecture of UF Data Warehouse

Data Flow from Source System to End User Desktop
To drive the business requirements effectively, we analyzed the key factors of each source data file to determine and translate the data into design considerations. We discussed data flow of the input and output with each administrative unit, taking into consideration ***who*** has the information, ***what*** is the information, **to *which*** end user and ***what*** end-results are expected. We then defined demand, verified the business analysis and priority, and reviewed the design of the data structure with our customers.

By now, we understood how raw data is extracted from various source systems, and how that data is driven to the warehouse. The raw data is combined and aligned in a data staging area. The same set of data staging services are used to select, aggregate, and restructures the data into the data set, as defined in the data models. These data sets are loaded into DB2. End users view the data through dynamic Web pages that access predefined canned queries produced by Eagle Server Pages (ESP).

The UF data warehouse is designed for continuous change; tasks include adding and changing the attributes of entities such as students and courses. The data warehouse is built to collect data from several source systems. Data will be cleaned, integrated, and stored de-normalized in a central repository. As shown in Figure 2, the data is extracted from the legacy systems and transformed in the data staging area, then loaded to the warehouse for user view.

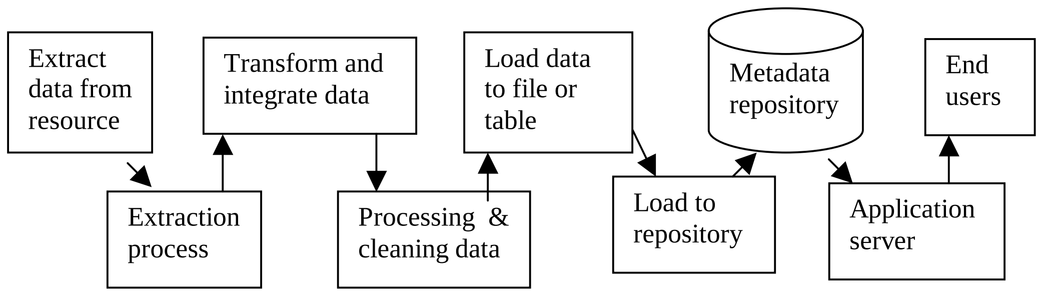

Extract data from resource
Transform and integrate data
Load data to file or table
Metadata repository
End users
Extraction process
Processing & cleaning data
Load to repository
Application server

Figure 2. Data Flow of the UF Data Warehouse

III. Logical and Physical Design of the UF Data Warehouse

Logical Design of Data Models

After designing the architecture, we designed the logical data structures and data models. The logical data model is built to define entities, to add attributes, data type and index key group, and to determine primary key and foreign key relationships. Based on the business requirements, each data model consists of the related entities and data definitions. Modeling activities include consideration of:

a. The Entities Relationships

The relationship between entities is based on our business requirements. Some entities may stand alone in a data model. There are four basic entity relationships:

One-to-One: Relationship is a single value in both directions.

Example: Each student has one SSN and each SSN represents one student.

One-to-Many or **Many-to-One**: One and only one instance of the first entity is related to many instances of the second entity.

Example: A professor teaches more than one subject. More than one section is opened for a subject, so more than one professor teaches the same subject.

Many-to-Many: Relationships are multi-valued in both directions.

Example: Students take more than one course and courses have more than one student.

Figure 3 shows one-to-one, one-to-many, many-to-one, and many-to-many relationships.

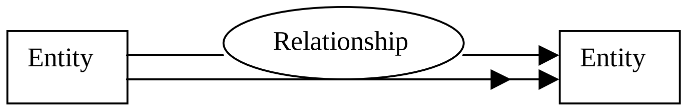


Figure 3. Entity relationship

b. Identifying the Attributes and Data Type

Each entity has many attributes. Each attribute is defined with a related data type, valid values, and keys (primary key, foreign key, alternate key, etc.). We defined the data attributes and data columns that customers must collect in the database, and made sure that each data column corresponds to an attribute of an entity.

c. Dimensional Model

The dimensional model is a logical design technique that seeks to make data available to end users in an intuitive framework to facilitate querying. It identifies and classifies the important business components in a subject area. Each dimensional model contains one table with multi-part keys from a fact table and a set of dimension tables.

Each dimension table is assigned a primary key with a set of attributes that are not related to one another. The primary key is a unique key for the dimension table, which is replicated in a fact table where it is referred to as a foreign key. The fact table contains many foreign keys that relate to the appropriate rows in each of the dimension tables. The purpose of a foreign key is to establish the uniqueness of each fact table record. Figure 4 shows the Dimension Table and Fact Table with Primary Key and Foreign Key.

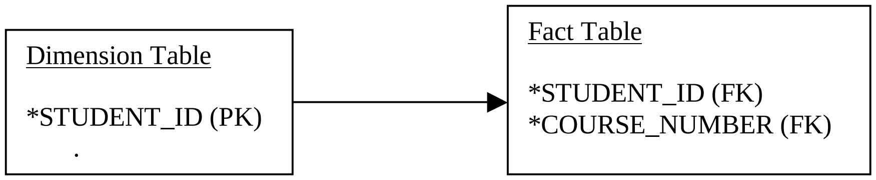


Figure 4. Dimension Table and Fact Table with Primary Key and Foreign Key

d. Normalization Design and De-Normalization Design

Based on the business requirements, our tables are either normalized or de-normalized. In a normalization design, one fact is stored in one place in the system with related facts of the single entity. A normalized design avoids redundancy and inconsistency. It also optimizes data access at the expense of data retrievals.

In a de-normalized design, one fact is stored in many places. A de-normalized design has much redundant data. It optimizes data access at the expense of data modification. A de-normalized relational design is preferred when browsing data and producing reports.

Frequently asked questions to help the designer determine a table design:

- Can the system achieve acceptable performance without de-normalizing?
- Will specialized expertise be required to code ad-hoc queries against the de-normalized data?
- Will system performance be acceptable or unacceptable after de-normalizing?

Physical Design of Data Models

After designing the logical data models, we designed the physical data models to implement the entities and relationships of the logical data model with DB2. In the physical design, we defined data naming and data type, reviewed the table plan, and created DB2 tables, table space and indexes to maximize database access performance within a DB2 data structure.

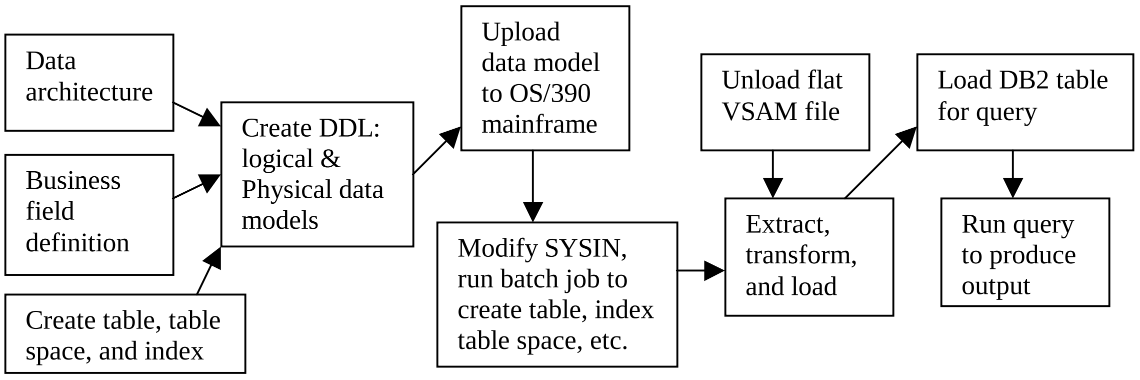
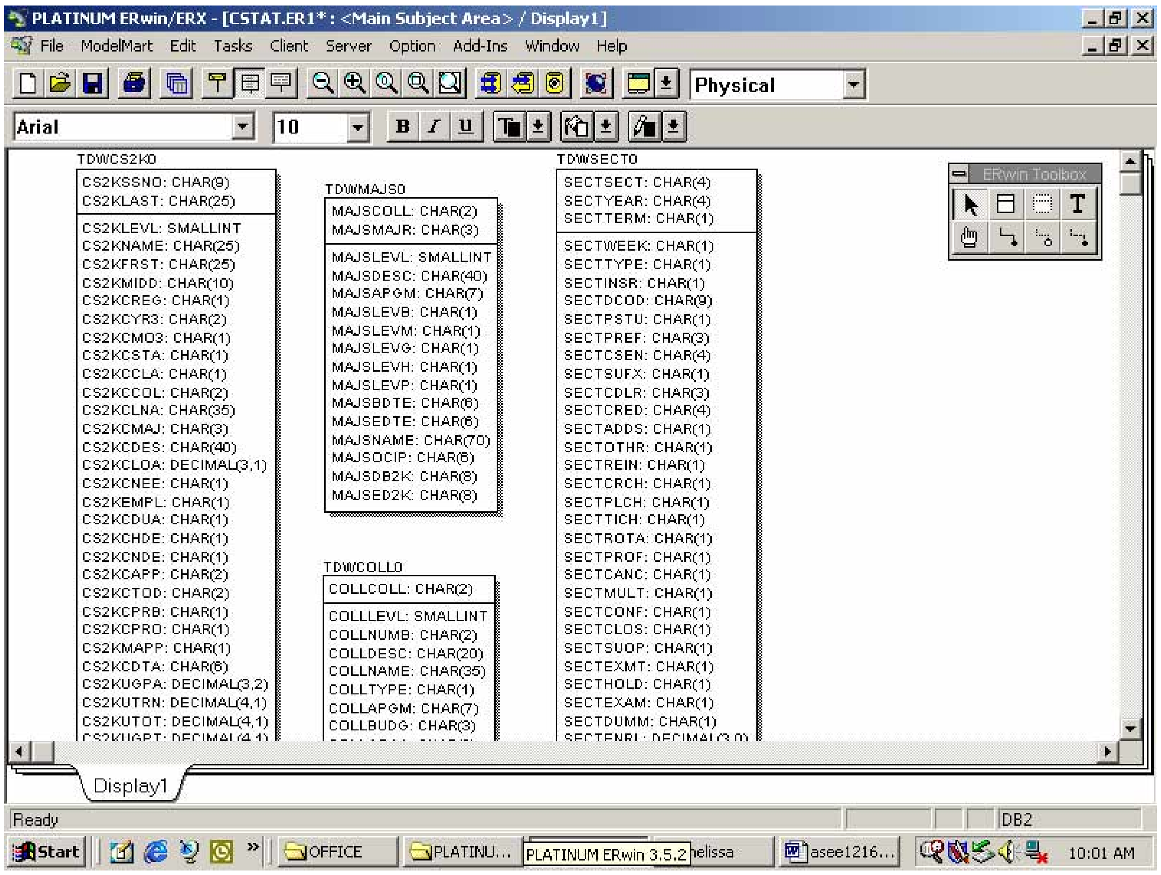

Data architecture
Business field definition
Create table, table space, and index
Create DDL: logical & Physical data models
Upload data model to OS/390 mainframe
Modify SYSIN, run batch job to create table, index table space, etc.
Unload flat VSAM file
Extract, transform, and load
Load DB2 table for query
Run query to produce output

Figure 5. Data Flow and Processing


PLATINUM ERwin/ERX - [CSTAT.ER1* : <Main Subject Area> / Display1]
File ModelMart Edit Tasks Client Server Option Add-Ins Window Help
Physical
Arial
10
ERwin Toolbox
Display1
Ready
DB2
Start
OFFICE
PLATINU...
PLATINUM ERwin 3.5.2
helissa
asee1216...
10:01 AM

```
TDWCS2K0
CS2KSSNO: CHAR(9)
CS2KLAST: CHAR(25)
CS2KLEVL: SMALLINT
CS2KNAME: CHAR(25)
CS2KFRST: CHAR(25)
CS2KMIDD: CHAR(10)
CS2KCREG: CHAR(1)
CS2KCYR3: CHAR(2)
CS2KCMO3: CHAR(1)
CS2KCSTA: CHAR(1)
CS2KCCLA: CHAR(1)
CS2KCCOL: CHAR(2)
CS2KCLNA: CHAR(35)
CS2KCMAJ: CHAR(3)
CS2KCDES: CHAR(40)
CS2KCLOA: DECIMAL(3,1)
CS2KCNEE: CHAR(1)
CS2KEMPL: CHAR(1)
CS2KCDUA: CHAR(1)
CS2KCHDE: CHAR(1)
CS2KCNDE: CHAR(1)
CS2KCAPP: CHAR(2)
CS2KCTOD: CHAR(2)
CS2KCPRB: CHAR(1)
CS2KCPRO: CHAR(1)
CS2KMAPP: CHAR(1)
CS2KCDTA: CHAR(6)
CS2KUGPA: DECIMAL(3,2)
CS2KUTRN: DECIMAL(4,1)
CS2KUTOT: DECIMAL(4,1)
CS2KUGPT: DECIMAL(4,1)

TDWMAJS0
MAJSCOLL: CHAR(2)
MAJSMAJR: CHAR(3)
MAJSLEVL: SMALLINT
MAJSDESC: CHAR(40)
MAJSAPGM: CHAR(7)
MAJSLEVB: CHAR(1)
MAJSLEVM: CHAR(1)
MAJSLEVG: CHAR(1)
MAJSLEVH: CHAR(1)
MAJSLEVP: CHAR(1)
MAJSBDTE: CHAR(6)
MAJSEDTE: CHAR(6)
MAJSNAME: CHAR(70)
MAJSOCIP: CHAR(6)
MAJSDB2K: CHAR(8)
MAJSED2K: CHAR(8)

TDWCOLL0
COLLCOLL: CHAR(2)
COLLLEVL: SMALLINT
COLLNUMB: CHAR(2)
COLLDESC: CHAR(20)
COLLNAME: CHAR(35)
COLLTYPE: CHAR(1)
COLLAPGM: CHAR(7)
COLLBUDG: CHAR(3)

TDWSECT0
SECTSECT: CHAR(4)
SECTYEAR: CHAR(4)
SECTTERM: CHAR(1)
SECTWEEK: CHAR(1)
SECTTYPE: CHAR(1)
SECTINSR: CHAR(1)
SECTDCOD: CHAR(9)
SECTPSTU: CHAR(1)
SECTPREF: CHAR(3)
SECTCSEN: CHAR(4)
SECTSUFX: CHAR(1)
SECTCDLR: CHAR(3)
SECTCRED: CHAR(4)
SECTADDS: CHAR(1)
SECTOTHR: CHAR(1)
SECTREIN: CHAR(1)
SECTCRCH: CHAR(1)
SECTPLCH: CHAR(1)
SECTTICH: CHAR(1)
SECTROTA: CHAR(1)
SECTPROF: CHAR(1)
SECTCANC: CHAR(1)
SECTMULT: CHAR(1)
SECTCONF: CHAR(1)
SECTCLOS: CHAR(1)
SECTSUOP: CHAR(1)
SECTEXMT: CHAR(1)
SECTHOLD: CHAR(1)
SECTEXAM: CHAR(1)
SECTDUMM: CHAR(1)
SECTENRL: DECIMAL(3,0)
```

Figure 6. Physical Design of Data Models

The physical design includes the following activities:

- Define the naming standards – Set up a primary word to describe the data element's subject. For example: database is UDW (University Data Warehouse), table is TDW, table space is SDW, and index is IDW.

- Determine the data types, primary key, foreign keys, and how data will be passed between tables.
- Define and determine the parameters of the table storage.
- Estimate the size of the table storage and the entire data warehouse – The lengths of each attribute, the number of rows for the initial prototype, the full historical load, and incremental rows per load when in production.
- Develop the initial indexing plan and define the indexes – Overview the indexes and query strategies to optimize the database in the process. We build many sets of indexes for queries to increase the efficiency of data retrieval. We use query analyzers to view the results of queries, optimize the queries, and improve the indexes on the query.

When the physical design of the data models was completed, all tables, table space, and indexes were created. The indexes that include primary index, alternate index, and cluster index, are to maintain the process efficiently to allow the end user to share disk architectures. Finally, we uploaded the data models with File Transfer Protocol (FTP) to build tables in DB2 OS/390.

IV. Application Server

Once the data is loaded into the data warehouse, the next step is to deliver these data to an appropriate web page for end user viewing by communicating with an application server. The application server, the mainframe system, is the platform where the data is stored for end users' direct query, reporting systems, and other applications. Figure 7 illustrates the communications between the data warehouse and an end user's browser.

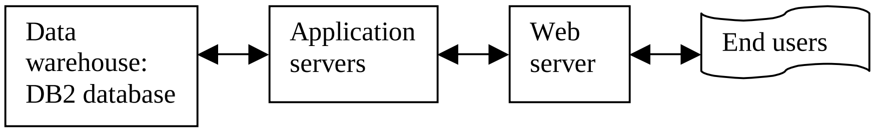


Figure 7. Communication between the data warehouse and the end user's browser

The University of Florida has developed EAGLE, **E**nhanced **A**pplication **G**eneration **L**anguage for the **E**nterprise, a CICS-based application server that enables direct Internet access to mainframe databases and the university's CICS resources. EAGLE provides both dynamic and static access to DB2 data, including singleton and cursor select, insert, update, and delete. The EAGLE server is a communication bridge that migrates administrative and/or student-based data between computer platforms to a Web-based environment. It sends or receives data via TCP/IP to any device with a socket listener, and enables linking of different sites within one logical site. Currently, EAGLE uses proprietary Electronic Data Interchange (EDI) and XML data formats to pass data to the other servers.

EAGLE provides page management, which enables the programmer to generate Web pages that invoke existing business logic to display information contained in mainframe database. These pages allow the Internet user to manipulate the data. We used a feature of EAGLE called Eagle Server Page (ESP) language and HTML to code SQL queries to extract data from the data warehouse and to display these data on Web pages.

The ESP is a tag-based mark-up language for creating dynamic Web pages that access DB2 data. It uses a simple set of XML-like tags to integrate HTML or XML with data from dynamic queries. It does not require any user-written CICS programs for data access. We designed the Web page format using ESP and HTML. We also used a DB2 program generator as an Eagle application to analyze SQL select statements and to build a DB2 I/O subroutine to access the table or view used in the select. We manage queries between our end users and the data by using an EAGLE application to maintain DB2 data.

V. Implementation

Since the data warehouse will grow over time, implementation issues are very important. The data warehouse must provide data services to numerous administrative units on campus. It is both a parallel and serial processing environment, which executes different tasks and requests concurrently. Implementation goals include:

- Dramatic performance gains for as many categories of user queries as possible,
- A reasonable amount of extra data storage to the warehouse,
- Complete transparency to end users and to application designers,
- Direct benefit to all users, regardless of which query tool they use,
- Impact the cost of the data system as little as possible, and
- Impact the DBA’s administrative responsibilities as little as possible.

VI. Conclusion

This paper summarizes the UF data warehouse design and modeling experience. This data warehouse is a user-centered design that has met the needs of the whole university community, including administrators, faculty, and students. We have analyzed the UF data warehouse requirements and designed its architecture to fit the needs of our customers.

The logical design focuses on the correctness and completeness of a desired data warehouse so that the user's business requirements and environment can be represented accurately. The physical design must consider cost, data security, data relationships, naming standards for data types, tables, indexes, etc..

We chose to use DB2 OS/390 because this mainframe system is already in use, and we can have the benefit of minimizing the initial implementation cost. We have successfully developed and tested a prototype data warehouse to deliver data from source systems to the end user's desktop. We currently are looking into implementation issues such as result interpretation, data selection and representation. We hope to attain a fully functional, responsive, and high quality data warehouse in the near future.

VII. Bibliography

1. M. Carla DeAngelis – "Data Modeling with Erwin," 2000, Sams, U.S.A.

2. Ralph Kimball, Laura Reeves, Margy Ross, and Warren Thornthwaite – "The Data Warehouse Lifecycle Toolkit", 1998, John Wiley and Sons, New York.

3. David Marco – "Building and Managing the Meta Data Repository," 2000, John Wiley and Sons, New York.

4. Paul I-Hai Lin – "Building Web Application with HTML", 2000, Indiana-Purdue University at Fort Wayne.

5. The Office of the CIO – "Eagle Developer's Manual", "Eagle User's Guide", and "Eagle Sever Page Menu", 2000, University of Florida

VIII. Biographical

Melissa Lin is a database administrator at the University of Florida's Office of the CIO. She currently is in charge of designing and implementing the university's first data warehouse. Previously, she was a Visiting Professor in the Computer Science Department of Indiana University - Purdue University, Fort Wayne, during 1999. Before that, she worked as systems analyst at GTE Data Service, Fort Wayne, Indiana, for 11 years, responsible for E911 and CBSS billing supports. Lin's current interests include data warehouse modeling and Web-based user interface applications.