Design and implementation of bibliographic database

Kiranmayi Karamcheti
University of Nevada, Las Vegas

Follow this and additional works at: https://digitalscholarship.unlv.edu/rtds

Repository Citation
Karamcheti, Kiranmayi, "Design and implementation of bibliographic database" (2007). UNLV Retrospective Theses & Dissertations. 2241.
https://digitalscholarship.unlv.edu/rtds/2241

This Thesis is brought to you for free and open access by Digital Scholarship@UNLV. It has been accepted for inclusion in UNLV Retrospective Theses & Dissertations by an authorized administrator of Digital Scholarship@UNLV. For more information, please contact digitalscholarship@unlv.edu.
DESIGN AND IMPLEMENTATION OF BIBLIOGRAPHIC DATABASE

By

Kiranmayi Karamcheti

Bachelor of Technology in Computer Science and Information Technology
Jawaharlal Nehru Technological University, India
June 2005

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science Degree in Computer Science
Department of Computer Science
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
December 2007
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.
Thesis Approval
The Graduate College
University of Nevada, Las Vegas

NOVEMBER 19TH, 2007

The Thesis prepared by

KIRANMAYI KARAMCHETI

Entitled

DESIGN AND IMPLEMENTATION OF A BIBLIOGRAPHIC DATABASE

is approved in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
ABSTRACT

Design and Implementation of Bibliographic Database

By

Kiranmayi Karamcheti

Dr. Kazem Taghva, Examination Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

In this thesis, we investigate the issues associated with the design and implementation of a comprehensive bibliographic database. Of particular interest is identification and implementation of recursive queries.

The design process will be dictated by dependencies and constraints as set by the scientific communities. The final design will be represented in the Unified Modeling Language (UML). The physical database and associated recursive queries are implemented in the Oracle database management system.
TABLE OF CONTENTS

ABSTRACT .................................................................................................................................iii

LIST OF TABLES .......................................................................................................................vi

LIST OF FIGURES ....................................................................................................................vii

ACKNOWLEDGEMENTS ......................................................................................................viii

CHAPTER 1 INTRODUCTION .............................................................................................1
1.1 Building a Database .............................................................................................................1
1.2 Database Management Systems ..........................................................................................1
1.3 Relations (or) Tables ............................................................................................................2
1.4 Structured Query Language ...............................................................................................4
1.5 Database Design ..................................................................................................................5
  1.5.1 Phases of Database Design .......................................................................................5
1.6 Models and Languages ......................................................................................................5
1.7 Database Design using UML .............................................................................................6
  1.7.1 The UML Class ..........................................................................................................6
  1.7.2 Relation Scheme .........................................................................................................8
  1.7.3 Table Structure ...........................................................................................................9
    1.7.3.1 Basic Structure: Rows and Columns ....................................................................10
  1.7.4 The UML Association ...............................................................................................11
1.8 Relation Scheme Diagram ..............................................................................................12
1.9 Normalization ....................................................................................................................14
1.10 Basic Queries: SQL .........................................................................................................15
  1.10.1 Retrieving data with SQL SELECT ....................................................................15
  1.10.2 Basic SQL statements: DDL and DML ...............................................................15

CHAPTER 2 INTRODUCTION TO RELATIONAL DATABASE ......................................18
2.1 Relational Database .........................................................................................................18
2.2 Relational Database Design ............................................................................................19
  2.2.1 The Entities ...............................................................................................................19
  2.2.2 The Relationship ......................................................................................................20
      2.2.2.1 Degree of Relationship ..................................................................................21
  2.2.3 The Attributes .........................................................................................................23
2.3 Unified Modeling Language ...........................................................................................24

CHAPTER 3 BIBLIOGRAPHIC DATABASE- ACTUAL DESIGN .........................26

iv

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
LIST OF TABLES

Table 1.1: Normalization ................................................................. 15
Table 3.1: Master_Entries ................................................................. 27
Table 3.2: Article ........................................................................... 28
Table 3.3: Book ............................................................................. 29
Table 3.4: Proceedings ................................................................. 30
Table 3.5: Inbook .......................................................................... 31
Table 3.6: Inproceedings ................................................................. 33
Table 3.7: Journals ....................................................................... 34
Table 3.8: Magazines ................................................................. 35
Table 3.9: Techreports ................................................................. 36
Table 3.10: Manuals ..................................................................... 37
Table 3.11: Conference ................................................................. 38
Table 3.12: Annual Reports ........................................................... 39
Table 3.13: Periodicals ................................................................. 41
Table 3.14: BookonCD ................................................................. 42
Table 3.15: BookonTape ............................................................... 43
Table 3.16: BookonVCD ............................................................... 44
Table 3.17: OnlineSource .............................................................. 45
Table 3.18: Relationship ............................................................... 46
Table 3.19: Author ...................................................................... 46
Table 3.20: Publishers ................................................................. 47
Table 3.21: Organizations ............................................................. 47
Table 3.22: Month Year ................................................................. 48
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Class Diagram – Article Table</td>
<td>8</td>
</tr>
<tr>
<td>1.2</td>
<td>Attributes in the Article Table</td>
<td>8</td>
</tr>
<tr>
<td>1.3</td>
<td>Example of Schema, Attributes and Tuples</td>
<td>10</td>
</tr>
<tr>
<td>1.4</td>
<td>Class Diagram – Relationship between Article and Relationship table</td>
<td>12</td>
</tr>
<tr>
<td>1.5</td>
<td>Showing the relationship between tables</td>
<td>13</td>
</tr>
<tr>
<td>1.6</td>
<td>Class Diagram – Relationship among Article and Master Entries table and Relationship table</td>
<td>14</td>
</tr>
<tr>
<td>2.1</td>
<td>Representation of an Entity</td>
<td>20</td>
</tr>
<tr>
<td>2.2</td>
<td>Representation of a Relationship</td>
<td>20</td>
</tr>
<tr>
<td>2.3</td>
<td>One-to-many Relationship</td>
<td>21</td>
</tr>
<tr>
<td>2.4</td>
<td>Many-Many Relationship</td>
<td>22</td>
</tr>
<tr>
<td>2.5</td>
<td>Many-to-many Relationships – a solution</td>
<td>23</td>
</tr>
<tr>
<td>2.6</td>
<td>Representation of Attributes</td>
<td>24</td>
</tr>
<tr>
<td>4.1</td>
<td>Example for Circular Reference</td>
<td>51s</td>
</tr>
</tbody>
</table>
ACKNOWLEDGEMENTS

Words would not suffice, but I would like to take this opportunity to express my gratitude to those who have helped me bring this thesis to a successful end with their knowledge, help and co-operation.

I wish to express my sincerest gratitude to my mentor Dr. Kazem Taghva for giving me the opportunity to carry out this thesis. I am especially indebted to him for all his valuable suggestions and help. I express the heartfelt debt that I owe to Dr. Ajoy K Datta for corroborating me in various ways both as a Graduate coordinator and my thesis committee member. I am also grateful to Dr. Wolfgang Bein for being in my committee. I am glad that I have taken one of his courses which have helped me in improving my problem solving skills. I would also like to thank Dr. Venkatesan Muthukumar who has accepted to be my graduate college representative.

I shall remain ever obliged to my parents and my brother for their continuous support in all my endeavors.
CHAPTER 1

INTRODUCTION

1.1 Building a Database

A database is a collection of related data. It is also defined as a self describing collection of integrated records. Data acts as a bridge between the machine components and the human components. The database contains both the operational data and the meta-data (the data about the data). The structure of the database is called the schema. A schema is a named collection of schema objects, such as tables, views, clusters, and procedures, associated with a particular user.

1.2 Database Management Systems

DBMS is a software that enables users to define, create, maintain and control access to the database. It interacts with the users' application program and the database. It provides the following facilities.

- It allows users to define the database, usually through a Data Definition Language (DDL). It allows users to specify the data types and structures and the constraints on the data to be stored in the database.

- It allows users to insert, delete, and retrieve data from the database, usually through a Data Manipulation Language (DML). DML provides a general inquiry facility
to the data in the repository using a query language. The most common query language is the Structured Query Language (SQL). A database system is a collection of application programs that interact with the database.

1.3 Relations or Tables

Consider a database which handles bibliography information. The design should keep the collection of the bibliographic entries in an Oracle database and allow manipulation of these entities via an application program. A relation is defined as a set of tuples that all have the same attributes. This is usually represented by a table, which is data organized in rows and columns. An entity is a distinct object (a person, place, thing, concept, or event) in the organization that is to be represented in the database.

In our database, the entities are classified into various categories such as: Article, Book, Inbook, Proceedings, Inproceedings, Techreport, Manual, Conference and so on. An entry in each of these categories has some required fields and some optional fields. Each entity consists of several attributes. An attribute is a property that describes some aspect of the object that we wish to record. The following are some of the entities with attributes defined for each entity.

```
MASTER_ENTRIES(CITE_KEY, entry_type)

ARTICLE(CITE_KEY, author, title, journal, volume, number, pages, month, year, note)

BOOK(CITE_KEY, author, editor, title, publisher, address, volume, edition, series, month, year, note)

PROCEEDINGS(CITE_KEY, editor, title, publisher, organization, address, month, year, note)
```
Each entry is identified by a unique key. A relation is defined as the set of tuples that all have the same attributes. A tuple usually represents some object and its associated data, whether that object is a physical object or a concept. A key is a kind of constraint which requires that the object, or critical information about the object, isn't duplicated. A Candidate key is the minimal set of attributes that uniquely identifies each occurrence of an entity type.

For example, a family might like to have a constraint such that no two people in the immediate family have the same name. If information about family members were stored in a database, a key could be placed over the family member's name. In a University, they have no such luxury. Each student is typically assigned a Student ID, which are used as keys for individual students stored in the school database. StudentID is the primary key.

Keys can have more than one column, for example, a nation may impose a restriction that a province can't have two cities by the same name. So, when cities are stored in a relation, there would be a key defined over province and city name. This would allow for two different provinces to have a town called Springfield (because their province would be different), but not two cities with the same name in the same province.

A key, in this context, refers to any set of attributes which uniquely span the relation. In particular, this is called a superkey. A candidate key is a minimal superkey, meaning that, none of the attributes in the key could be removed from the key, and still has that attribute set been a key. Many DBMSs have a concept of a primary key. Primary key is
the candidate key that is selected to uniquely identify each occurrence of an entity type. A key over more than one attribute is called a compound key. Theoretically, a key can even be over zero attributes. This would enforce that there cannot be more than one tuple in the relation. In our database CITE_KEY is the primary key for MASTER_ENTRIES table and this acts as a foreign key for all the other tables. A relationship is an association between two entities.

1.4 Structured Query Language

Objectives

- To create the database and relation structures
- To perform basic data management tasks, such as insertion, modification, and deletion of data from the relations.
- To perform both simple and complex queries.

The command structure consists of standard English words such as CREATE TABLE, INSERT, SELECT. For Example:

1. CREATE TABLE MASTER_ENTRIES(CITE_KEY Varchar(25), Entry_type Varchar(25));

2. INSERT INTO MASTER_ENTRIES VALUES("AOI", "ARTICLE");

3. SELECT Entry_type
   FROM MASTER_ENTRIES
   WHERE CITE_KEY = "AOI";

1.5 Database Design

Database design is the process of creating a design for a database that will
support the enterprise's operations and objectives. The two main approaches to the design of a database are referred to as 'bottom-up' and 'top-down'. The bottom-up approach begins at the fundamental level of attributes (i.e. properties of entities and relationships), which through analysis of the associations between attributes, are grouped into relationships between the attributes. A more appropriate strategy for the design of complex databases is to use the top-down approach.

1.5.1 Phases of Database Design

Database design is made up of three main phases, namely conceptual, logical, and physical design.

Conceptual database design: The process of constructing a model of the information used in an enterprise, independent of all physical considerations.

Logical database design: The process of constructing a model of the information used in an enterprise based on a specific data model, but independent of a particular DBMS and other physical considerations.

Physical database design: The process of producing a description of the implementation of the database on secondary storage; it describes the base relations, file organizations, and indexes used to achieve efficient access to the data, and any indexes used to achieve efficient access to the data, and any associated integrity constraints and security measures.

1.6 Models and Languages

Database design is a process of modeling an enterprise in the real world. Many models and languages—some formally and mathematically defined, some informal and intuitive—are used by designers.
• The Unified Modeling Language (UML) was designed for software engineering of large systems using object-oriented (OO) programming languages. UML is used to model an enterprise that will be represented in the database.

• The Entity-Relationship (ER) model is used in many database development systems. There are many different graphic standards that can represent the ER model. It is very similar to the UML class diagram, but may also include elements of the relational model.

• The Relational Model (RM) is the formal model of a database that was developed for IBM in the early 1970s by Dr. E.F. Codd. It is largely based on set theory, which makes it both powerful and easy to implement in computers. All modern relational databases are based on this model. It is used to represent information that does not appear in the UML model but is needed to build functioning databases.

• Relational Algebra (RA) is a formal language used to symbolically manipulate objects of the relational model.

• The table model is an informal set of terms for relational model objects.

• The Structured Query Language SQL is used to build and manipulate relational databases. It is based on relational algebra. It is a declarative, rather than a procedural programming language.

1.7 Database Design using UML

1.7.1 The UML class

A UML class (ER term: entity) is any “thing” in the enterprise that is to be represented in our database. It could be a physical “thing” or simply a fact about the enterprise or an event that happens in the real world.
Example: We’ll build a bibliographic database—it could be for any kind of entry type. To get the hierarchy of references, we need any entry type, so an Article will be our first class (entity) type.

• The first step in modeling a class is to describe it in natural language. This helps us to know exactly what this class ("thing") means in the enterprise. We can describe an Article like this:

  "An Article is an entry type which has set of references that refers to any entry type. We need to know the article's id, cite key, title, volume, publisher, and so on to identify what the article describes."

• Each class is uniquely defined by its set of attributes (UML and ER), also called properties in some OO languages. Each attribute is one piece of information that characterizes each member of this class in the database. Together, they provide the structure for database tables or code objects.

• In UML, only descriptive attributes are identified—those which actually provide real-world information (relevant to the enterprise) about the class that are used for modeling. (These are sometimes called natural attributes.)

Class Diagram

The class diagram shows the class name (always a singular noun) and its list of attributes.

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
1.7.2 Relation Scheme

In an OO programming language, each class is instantiated with objects of that class. In building a relational database, each class is first translated into a relational model scheme. The scheme is identified by the plural form of the class name, and starts with all of the attributes from the class diagram.

Schema name \( \rightarrow \) Articles

\textit{Attribute names}

<table>
<thead>
<tr>
<th>Article_Id</th>
<th>CITE_KEY</th>
<th>Title</th>
<th>Journal</th>
<th>Volume</th>
<th>Number</th>
<th>Pages</th>
</tr>
</thead>
</table>

Figure 1.2: Attributes in the Article table

- In the relational model, a scheme is defined as a set of attributes, together with
an assignment rule that associates each attribute with a set of legal values that may be
assigned to it. These values are called the domain of the attribute. In set notation:

Articles Scheme = \{Article\_Id, CITE\_KEY, Title, Journal, Volume, Number, Pages,
MonthYear\_Id, Note, Publisher\_Id, Edition\}

1.7.3 Table Structure

When we actually build the database, each relation scheme becomes the structure
for one table. The SQL syntax for creating the table includes a data type for each
attribute, which is needed for the database but is not the same as the domain of the
attribute.

CREATE TABLE Article (  
    CITE\_KEY Varchar (10) NOT NULL,
    ARTICLE\_ID Varchar(10) NOT NULL,
    Title Varchar(50) NOT NULL,
    Journal Varchar (50),
    Number Integer,
    Pages Varchar (10),
    MonthYear\_Id Integer,
    Note Varchar (10),
    Publisher\_Id Integer,
    Edition Varchar (50))

In this example, VARCHAR is simply a variable-length character string of no more than
the number of characters in parentheses.
1.7.3.1 Basic Structures: Rows and Tables

Representing Data in Rows

Each real-world individual of a class (for example, each Article) is represented by a row of information in a database table. The row is defined in the relational model as a tuple that is constructed over a given scheme. Mathematically, the tuple is a function that assigns a constant value from the attribute domain to each attribute of the scheme. Notice that because the scheme is a set of attributes, we could show them in any order without changing the meaning of the data in the row (tuple).

![Figure 1.3: Example of Schema, Attributes and Tuples](image)

In formal notation, we could show the assignments explicitly, where \( t \) represents a tuple:

\[
t_{IJ} = \{\text{Article\_Id := 'A0001'}, \text{CITE\_KEY := 'A01'}, \text{Title := 'A Comparison of Automatic and Manual Zoning'}, \text{Journal := 'Document Analysis and Recognition'}, .............\}
\]

In practice, when we create a table row in SQL, we are actually making the assignment of domain values to attributes, just as in the tuple definition.
1.7.4 The UML Association

The UML association (ER term: relationship) is the way that two classes are functionally connected to each other.

Example: We want our Article to be able to have set of references, so we need to model the Relationship class and its association with the Article. Notice that while the Article class represents a physical “thing,” the Relationship class represents an event that happens in the enterprise. Both are equally valid class types. We will first describe the Relationship:

“A Relation is created when an Article refers to another article or any other entry type. We need to know to what entry type the article is referring to, and the Parent_Id and Child_Id of the entry_type in the Relationship class.”

The association between the article and the relationship will tell us to what entry type the article is referring to. We will describe the association in natural language just as we described the classes, but we will also include information about how few (at minimum) and how many (at maximum) individuals of one class may be connected to a single individual of the other class. This is called the multiplicity of the association (ER term: cardinality), and we describe it in both directions.

“Each article refers to one or more entry types.”
Each article is referred by one or more entry types.

Class diagram

![Class Diagram](image)

Figure 1.4: Class Diagram showing the relationship between Article and Relationship table

1.8 Relation Scheme Diagram

The relation scheme for the Relationship table contains all of the attributes from the class diagram, as before. But we also need to represent the association in the database. We do this by copying the PK attributes of the MASTER_ENTRIES into the Article scheme. The copied attributes are called a foreign (FK), which is simply an image of the linked relation’s primary key.
Design Pattern – Many to Many

The UML association class represents the attributes of a many-to-many association, but can only be used if there is at most one pairing of any two individuals in the relationship. This means, for example in Article entry, that there can be only one title for each Article. This constraint is consistent with the enterprise being modeled. At times the same two individuals in a many-to-many association will be paired more than once. This frequently happens when we need to keep a history of events over time.

Example: In a library, customers can borrow many books and each book can be borrowed by many customers, so this seems to be a simple many-to-many association between customers and books. But any one customer may borrow a book, return it, and then borrow the same book again at a later time. The library records each book loan separately. There is no invoice for each set of borrowed books and therefore no equivalent here of the Entry Types in our bibliographic database.
1.9 Normalization

Normalization is usually thought of as a process of applying a set of rules to the database design, mostly to achieve minimum redundancy in the data.

- In theory, single relation schemes (sometimes called the universal scheme or U) contains all of the attributes in the database—then apply these rules recursively to develop a set of increasingly-normalized sub-relation schemes. When all of the schemes are in third normal form, then the whole database is properly normalized.
<table>
<thead>
<tr>
<th>Normal form</th>
<th>Traditional definition</th>
<th>As presented here</th>
</tr>
</thead>
<tbody>
<tr>
<td>First normal form (1NF)</td>
<td>• All attributes must be atomic, and</td>
<td>• Eliminate multi-valued attributes, and</td>
</tr>
<tr>
<td></td>
<td>• No repeating groups</td>
<td>• Eliminate repeated attributes</td>
</tr>
<tr>
<td>Second normal form (2NF)</td>
<td>• First normal form, and</td>
<td>• Eliminate subkeys (where the subkey is part of a composite primary key)</td>
</tr>
<tr>
<td></td>
<td>• No partial functional dependencies</td>
<td></td>
</tr>
<tr>
<td>Third normal form (3NF)</td>
<td>• Second normal form, and</td>
<td>• Eliminate subkeys (where the subkey is not part of the primary key)</td>
</tr>
<tr>
<td></td>
<td>• No transitive functional dependencies</td>
<td></td>
</tr>
</tbody>
</table>

1.10 Basic queries: SQL

1.10.1 Retrieving data with SQL SELECT

To look at the data in our tables, we use the select (SQL) statement. Although the result table is not stored in the database like the named tables are, we can also use it as part of other select statements. The basic syntax consists of four clauses:

SELECT <attribute names>
FROM <table names>
WHERE <condition to pick rows>
ORDER BY <attribute names>;

1.10.2 Basic SQL Statements: DDL and DML

• As discussed earlier, SQL statements are divided into two major categories: data definition language (DDL) and data manipulation language (DML).
Data Definition Language

DDL statements are used to build and modify the structure of tables and other objects in the database. When we execute a DDL statement, it takes effect immediately.

The create table statement does exactly that:

```
CREATE TABLE <table name> (  
  <attribute name 1> <data type 1>,  
  ...  
  <attribute name n> <data type n>);  
```

The data types that we use most frequently are character strings, which might be called VARCHAR or CHAR for variable or fixed length strings; numeric types such as NUMBER or INTEGER, which will usually specify a precision; and DATE or related types. Data type syntax is variable from system to system;

- The alter table statement may be used to specify primary and foreign key constraints, as well as to make other modifications to the table structure. Key constraints may also be specified in the CREATE TABLE statement.

```
ALTER TABLE <table name>  
ADD CONSTRAINT <constraint name> PRIMARY KEY (<attribute list>);  
```

- In the foreign key constraint, we have to specify both the FK attributes in this (child) table, and the PK attributes that they link to in the parent table.

```
ALTER TABLE <table name>  
ADD CONSTRAINT <constraint name> FOREIGN KEY (<attribute list>)  
REFERENCES <parent table name> (<attribute list>);  
```
• Any object created can be deleted with a drop statement. The syntax is different for tables and constraints.

    DROP TABLE <table name>;

    ALTER TABLE <table name>
    DROP CONSTRAINT <constraint name>;

The DROP TABLE statement gets rid of its own PK constraint, but won’t work until you separately drop any FK constraints (or child tables) that refer to this one. It also gets rid of all data that was contained in the table.

*Data Manipulation Language*

DML statements are used to work with the data in tables.

• The insert statement is used, obviously, to add new rows to a table.

    INSERT INTO <table name>
    VALUES (<value 1>... <value n>);

• The update statement is used to change values that are already in a table.

    UPDATE <table name>
    SET <attribute> = <expression>
    WHERE <condition>;

• The delete statement does just that, for rows in a table.

    DELETE FROM <table name>
    WHERE <condition>;
CHAPTER 2

INTRODUCTION TO RELATION DATABASE

2.1 Relational Model

A relational database is a database that conforms to the relational model, and could also be defined as a set of relations or a database built in an RDBMS.

A relational database management system (RDBMS) is a system that manages data using the relational model. Most current RDBMS (for example: MySQL, PostgreSQL, Oracle, Microsoft SQL Server) are more accurately called SQL database management products. (RDBMS - relational database management system) A database based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organized in tables. A table is a collection of records and each record in a table contains the same fields.

Properties of Relational Tables:

• Values Are Atomic
• Each Row is Unique
• Column Values Are of the Same Kind
• The Sequence of Columns is Insignificant
• The Sequence of Rows is Insignificant
Each Column Has a Unique Name

Certain fields may be designated as a key, which means that searches for specific values of that field will use indexing to speed them up. Where fields in two different tables take values from the same set, a join operation can be performed to select related records in the two tables by matching values in those fields. Often, but not always, the fields will have the same name in both tables. For example, an "orders" table might contain (customer-ID, product-code) pairs and a "products" table might contain (product-code, price) pairs so to calculate a given customer's bill you would sum the prices of all products ordered by that customer by joining on the product-code fields of the two tables. This can be extended to joining multiple tables on multiple fields. Because these relationships are only specified at retrieval time, relational databases are classed as dynamic database management system. The RELATIONAL database model is based on the Relational Algebra.

2.2 Relational Database - Design

2.2.1 The Entities

An entity is represented diagrammatically by a box with rounded corners and a name written in the singular. The name for an entity is one that represents a class of things in general - not specific instances.

The technique is called an entity-attribute-relationship model.
2.2.2 The Relationship

A relationship is a significant association between two entities. It is represented by a line that joins two entity boxes. Each relationship has two ends, for each of which there is a name.

There may well be more than one solution when the problem is a complex one.
2.2.2.1 Degree of Relationship

One-to-many relationships

In this kind of relationship one of the entities can be related many times to another entity. For example, one excavation can contain many finds; one person can commit many crimes. One to many relationships are common and are well handed by relational databases.

Many-to-many relationships

Here there can be many occurrences of one entity related to many occurrences of another entity. For example there can be many witnesses to each charter and many charters can be witnessed by any witness. These relationships can be difficult to identify and to resolve. As a rule of thumb, if some relationship puzzles you, there is a good chance that there is a many to many relationship.
Many-to-many relationships - a solution

Many-to-many relationships are dealt with by the many-to-many relationship being decomposed into two relationships, with a newly created entity intervening between the original entries. Thus a many-to-many relationship can be replaced by an entity which has a many-to-one relationship with each of the original entities, as illustrated by the example databases.

Here the idea of the 'chart sign' is that at the signing of any one charter there is one charter but many witnesses. Any witness can sign a number of charters.
The degree of the relationship is often difficult to identify in the early stages but if they are missed, it will show up when you come to create a real database and enter data. Decide on the characteristics (attributes) of the entities.

Attributes

The next step is to add details about each entity. It is often useful to write them on the diagram so that they are clearly associated with the entity.

2.2.3 The Attributes

Attributes are the details about the state of an entity; they are any description of an entity; they are properties of things we want to know about entities. For example, we might want some information about a person, such as their name or date of birth.

Attributes represent the data that is to be kept about the entities - and thus become the rows in the tables.

At this stage it is also useful to identify how the attributes to be related to one
another can be linked by a unique 'key'.

![Diagram of person and subject attributes]

Figure 2.6: Representation of Attributes

2.3 Unified Modeling Language

The Unified Modeling Language (UML) is a standardized specification language for object modeling. UML is a general-purpose modeling language that includes a graphical notation used to create an abstract model of a system, referred to as a UML model. UML is officially defined at the Object Management Group (OMG) by the UML metamodel, a Meta-Object Facility metamodel (MOF). UML is also used for business process modeling, systems engineering modeling and representing organizational structures. In UML there are 13 types of diagrams

Structure diagrams emphasize what things much be in the system being modeled:

- Class Diagram
- Component Diagram
- Composite Structure Diagram
- Deployment Diagram
- Object Diagram
- Package Diagram

*Behavior Diagrams* emphasize what must happen in the system being modeled.

- Activity Diagram
- State Machine Diagram
- Use Case Diagram

*Interaction Diagrams*, a subset of behavior diagrams, emphasize the flow of control and data among the things in the system being modeled:

- Communication diagram
- Interaction overview diagram (UML 2.0)
- Sequence diagram
- UML Timing Diagram(UML 2.0)
BIBLIOGRAPHIC DATABASE – ACTUAL DESIGN

Biographic databases are used to store collections of bibliographic records. Many traditional bibliographic databases contained fields to store information about printed works, books, articles, proceedings etc. An example of this type database is BibTex which is used with the Latex word-processing application.

The bibliographic entries are classified into various categories: Article, Book, Inbook, Proceedings, Inproceedings, Techreport, Manual, Conference, and so on. An entry in each of these categories has some required fields and some optional fields. The MASTER ENTRIES table contains the CITE_KEY and the type of entry (article, book, etc.) for all the bibliographic entries. There is one table for each entry type that maintains all the entries under that particular category. The REQUIRED_FIELDS table records information about the required fields for each type of entry.

The design of the relational database results in the following tables:
The MASTER_ENTRIES table consists of the fields CITE_KEY and entry_type. Each entry is identified by a unique CITE_KEY.

Table Name: MASTER_ENTRIES
<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Primary Key (PK)</td>
</tr>
<tr>
<td>Entry_type</td>
<td>Varchar2(50)</td>
<td>Holds values like Article, Book and so on</td>
</tr>
</tbody>
</table>

The table ARTICLE contains all the bibliographic entries that are articles. This also contains a CITE_KEY which is used to identify the type of entry. Each Article is uniquely identified by an ARTICLE_ID. This table contains Publisher_Id which refers to the Publisher table.

Table Name: ARTICLE

The table BOOK contains all the bibliographic entries that are books. This also contains a CITE_KEY which is used to identify the type of entry. Each Book is uniquely identified by a BOOK_ID. This table contains Publisher_Id which refers to the Publisher table and MonthYear_Id which refers to the Month_Year table.

Table Name: BOOK

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
Table 3.2: ARTICLE

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>ARTICLE_ID</td>
<td>Integer</td>
<td>Primary Key (PK)</td>
</tr>
<tr>
<td>Title</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Journal</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Volume</td>
<td>Varchar(50)</td>
<td></td>
</tr>
<tr>
<td>Pages</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td>Foreign key (Referencing Month_Year table)</td>
</tr>
<tr>
<td>Note</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Publisher_Id</td>
<td>Integer</td>
<td>Foreign key (Referencing PUBLISHERS table)</td>
</tr>
<tr>
<td>Edition</td>
<td>Varchar(50)</td>
<td></td>
</tr>
</tbody>
</table>
Table 3.3: BOOK

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>BOOK_ID</td>
<td>Integer</td>
<td>Primary Key(PK)</td>
</tr>
<tr>
<td>Editor</td>
<td>Varchar(50)</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Publisher_Id</td>
<td>Integer</td>
<td>Foreign key (Referencing PUBLISHERS table)</td>
</tr>
<tr>
<td>Pages</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Volume</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Edition</td>
<td>Varchar(50)</td>
<td></td>
</tr>
<tr>
<td>Series</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Note</td>
<td>Varchar(50)</td>
<td></td>
</tr>
</tbody>
</table>

The table PROCEEDINGS contains all the bibliographic entries that are proceedings.

This also contains a CITE_KEY which is used to identify the type of entry.

Each Proceeding is uniquely identified by a PROCEEDINGS_ID

This table contains Publisher_Id which refers to the Publisher table, Organization_Id which refers to the Organizations table.

MonthYear_Id which refers to Month_Year table.
Table Name: PROCEEDINGS

Table 3.4: PROCEEDINGS

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>PROCEEDINGS_ID</td>
<td>Integer</td>
<td>Primary Key (PK)</td>
</tr>
<tr>
<td>Editor</td>
<td>Varchar(50)</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Publisher_Id</td>
<td>Integer</td>
<td>Foreign key (Referencing PUBLISHERS table)</td>
</tr>
<tr>
<td>Organization_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing ORGANIZATIONS table)</td>
</tr>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing MONTH_YEAR table)</td>
</tr>
<tr>
<td>Note</td>
<td>Varchar(50)</td>
<td></td>
</tr>
</tbody>
</table>

The table INBOOK contains all the bibliographic entries that are inbook.
This also contains a CITE_KEY which is used to identify the type of entry.
Each Inbook is uniquely identified by an INBOOK_ID
This table contains Publisher_Id which refers to the Publisher table,
MonthYear_Id which refers to Month_Year table.

Table Name: INBOOK

Table 3.5: INBOOK

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>INBOOK_ID</td>
<td>Integer</td>
<td>Primary Key (PK)</td>
</tr>
<tr>
<td>Editor</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Publisher_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing PUBLISHERS table)</td>
</tr>
<tr>
<td>Volume</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Edition</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Series</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Chapter</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Pages</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing MONTH_YEAR table)</td>
</tr>
<tr>
<td>Note</td>
<td>Varchar(10)</td>
<td></td>
</tr>
</tbody>
</table>
The table INPROCEEDINGS contains all the bibliographic entries that are inproceedings.

This also contains a CITE_KEY which is used to identify the type of entry.

Each Inproceeding is uniquely identified by an INPROCEEDING_ID.

This table contains Publisher_Id which refers to the Publisher table,

Organization_Id which refers to the Organizations table,

MonthYear_Id which refers to Month_Year table.

The table JOURNALS contains all the bibliographic entries that are journals.

This also contains a CITE_KEY which is used to identify the type of entry.

Each Journal is uniquely identified by a JOURNAL_ID.

This table contains Publisher_Id which refers to the Publisher table,

MonthYear_Id which refers to Month_Year table.
Table Name: INPROCEEDINGS

Table 3.6: INPROCEEDINGS

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER ENTRIES table)</td>
</tr>
<tr>
<td>INPROCEEDINGS_ID</td>
<td>Integer</td>
<td>Primary Key(PK)</td>
</tr>
<tr>
<td>Editor</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Book_title</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Publisher_Id</td>
<td>Integer</td>
<td>Foreign key (Referencing PUBLISHERS table)</td>
</tr>
<tr>
<td>Organization_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing ORGANIZATIONS table)</td>
</tr>
<tr>
<td>Pages</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing MONTH_YEAR table)</td>
</tr>
<tr>
<td>Note</td>
<td>Varchar(50)</td>
<td></td>
</tr>
</tbody>
</table>
Table Name: JOURNALS

Table 3.7: JOURNALS

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>JOURNAL_ID</td>
<td>Integer</td>
<td>Primary Key (PK)</td>
</tr>
<tr>
<td>Title</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Volume</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>Pages</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing MONTH_YEAR table)</td>
</tr>
<tr>
<td>Note</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Publisher_Id</td>
<td>Integer</td>
<td>Foreign key (Referencing PUBLISHERS table)</td>
</tr>
<tr>
<td>Edition</td>
<td>Varchar(10)</td>
<td></td>
</tr>
</tbody>
</table>

The table MAGAZINES contains all the bibliographic entries that are magazines.

This also contains a CITE_KEY which is used to identify the type of entry.

Each Magazine is uniquely identified by a MAGAZINE_ID.

This table contains Publisher_Id which refers to the Publisher table,
MonthYear_Id which refers to Month_Year table.

Table Name: MAGAZINES

Table 3.8: MAGAZINES

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>MAGAZINE_ID</td>
<td>Integer</td>
<td>Primary Key (PK)</td>
</tr>
<tr>
<td>Name</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Publisher_Id</td>
<td>Integer</td>
<td>Foreign key (Referencing PUBLISHERS table)</td>
</tr>
<tr>
<td>Pages</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing MONTH_YEAR table)</td>
</tr>
<tr>
<td>Note</td>
<td>Varchar(10)</td>
<td></td>
</tr>
</tbody>
</table>

The table TECHREPORTS contains all the bibliographic entries that are techreports. This also contains a CITE_KEY which is used to identify the type of entry. Each Techreport is uniquely identified by TECHREPORT_ID. This table contains MonthYear_Id which refers to Month_Year table.

Table Name: TECHREPORTS
Table 3.9: TECHREPORTS

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>TECHREPORT_ID</td>
<td>Integer</td>
<td>Primary Key(PK)</td>
</tr>
<tr>
<td>Title</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing MONTH_YEAR table)</td>
</tr>
<tr>
<td>Subtitles</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Pages</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Note</td>
<td>Varchar(10)</td>
<td></td>
</tr>
</tbody>
</table>

The table MANUALS contains all the bibliographic entries that are manuals.

This also contains a CITE_KEY which is used to identify the type of entry.

Each Manual is uniquely identified by MANUAL_ID.

This table contains Publisher_Id which refers to Publisher table,

MonthYear_Id which refers to Month_Year table.
Table Name: MANUALS

Table 3.10: MANUALS

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>MANUAL_ID</td>
<td>Integer</td>
<td>Primary Key(PK)</td>
</tr>
<tr>
<td>Title</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Publisher_Id</td>
<td>Integer</td>
<td>Foreign key (Referencing PUBLISHERS table)</td>
</tr>
<tr>
<td>Pages</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Volume</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Edition</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Series</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing MONTH_YEAR table)</td>
</tr>
<tr>
<td>Note</td>
<td>Varchar(10)</td>
<td></td>
</tr>
</tbody>
</table>

The table CONFERENCE contains all the bibliographic entries that are manuals.

This also contains a CITE_KEY which is used to identify the type of entry.

Each Conference is uniquely identified by CONFERENCE_ID.
This table contains Organization_Id which refers to Organization table.

MonthYear_Id which refers to Month_Year table.
Table Name: CONFERENCE

Table 3.11: CONFERENCE

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>CONFERENCE_ID</td>
<td>Integer</td>
<td>Primary Key (PK)</td>
</tr>
<tr>
<td>Name</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Organization_Id</td>
<td>Integer</td>
<td>Foreign key (Referencing ORGANIZATIONS table)</td>
</tr>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing MONTH_YEAR table)</td>
</tr>
</tbody>
</table>

The table ANNUALREPORTS contains all the bibliographic entries that are annual reports.

This also contains a CITE_KEY which is used to identify the type of entry.

Each Annual Report is uniquely identified by ANNUALREPORT_ID.

This table contains Publisher_Id which refers to Publisher table.

Organization_Id which refers to Organization table.

MonthYear_Id which refers to Month_Year table.
The table PERIODICALS contains all the bibliographic entries that are periodicals.
This also contains a CITE_KEY which is used to identify the type of entry. Each Periodical is uniquely identified by PERIODICAL_ID.

This table contains Publisher_Id which refers to Publisher table.

Organization_Id which refers to Organization table.

MonthYear_Id which refers to Month_Year table.

The table BOOKONCD contains all the bibliographic entries that are CD’s. This also contains a CITE_KEY which is used to identify the type of entry. Each CD is uniquely identified by BOOKONCD_ID.

This table contains Publisher_Id which refers to Publisher table.

Organization_Id which refers to Organization table.

MonthYear_Id which refers to Month_Year table.
Table Name: PERIODICALS

Table 3.13: PERIODICALS

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>PERIODICAL_ID</td>
<td>Integer</td>
<td>Primary Key (PK)</td>
</tr>
<tr>
<td>Editor</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Title</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Publisher_Id</td>
<td>Integer</td>
<td>Foreign key (Referencing PUBLISHERS table)</td>
</tr>
<tr>
<td>Organization_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing ORGANIZATIONS table)</td>
</tr>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing MONTH_YEAR table)</td>
</tr>
<tr>
<td>Note</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Pages</td>
<td>Varchar(10)</td>
<td></td>
</tr>
</tbody>
</table>
Table Name: BOOKONCD

Table 3.14: BOOKONCD

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>BOOKONCD_ID</td>
<td>Integer</td>
<td>Primary Key(PK)</td>
</tr>
<tr>
<td>CD_Name</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Organization_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing ORGANIZATIONS table)</td>
</tr>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing MONTH_YEAR table)</td>
</tr>
<tr>
<td>Volume</td>
<td>Varchar(10)</td>
<td></td>
</tr>
</tbody>
</table>

The table BOOKONTAPE contains all the bibliographic entries that are tapes.

This also contains a CITE_KEY which is used to identify the type of entry.

Each TAPE is uniquely identified by BOOKONTAPE_ID.

This table contains Organization_Id which refers to Organization table,

    MonthYear_Id which refers to Month_Year table.
Table Name: BOOKONTAPE

Table 3.15: BOOKONTAPE

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>BOOKONTAPE_ID</td>
<td>Integer</td>
<td>Primary Key(PK)</td>
</tr>
<tr>
<td>Tape_Name</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Organization_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing ORGANIZATIONS table)</td>
</tr>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing MONTH_YEAR table)</td>
</tr>
<tr>
<td>Volume</td>
<td>Varchar(10)</td>
<td></td>
</tr>
</tbody>
</table>

The table BOOKONVCD contains all the bibliographic entries that are VCD's.

This also contains a CITE_KEY which is used to identify the type of entry.

Each VCD is uniquely identified by BOOKONVCD_ID.

This table contains Organization_Id which refers to Organization table.

MonthYear_Id which refers to Month_Year table
Table Name: BOOKONVCD

The table ONLINE SOURCE contains all the bibliographic entries that are online sources. This also contains a CITE KEY which is used to identify the type of entry.

Each ONLINE SOURCE is uniquely identified by ONLINE SOURCE ID.

This table contains Organization_Id which refers to Organization table.

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>BOOKONVCD_ID</td>
<td>Integer</td>
<td>Primary Key(PK)</td>
</tr>
<tr>
<td>CD_Name</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Organization_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing ORGANIZATIONS table)</td>
</tr>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing MONTH_YEAR table)</td>
</tr>
<tr>
<td>Volume</td>
<td>Varchar(10)</td>
<td></td>
</tr>
</tbody>
</table>
Table Name: ONLINESOURCE

Table 3.17: ONLINESOURCE

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>ONLINESOURCE_ID</td>
<td>Integer</td>
<td>Primary Key (PK)</td>
</tr>
<tr>
<td>Topic_Name</td>
<td>Varchar(150)</td>
<td></td>
</tr>
<tr>
<td>Posted_Date</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>Retrieved_Date</td>
<td>Date</td>
<td></td>
</tr>
<tr>
<td>URL</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Organization_Id</td>
<td>Integer</td>
<td>Foreign Key (Referencing ORGANIZATIONS table)</td>
</tr>
<tr>
<td>Entry_type</td>
<td>Varchar2(10)</td>
<td>Referencing MASTER_ENTRIES table</td>
</tr>
</tbody>
</table>

The table RELATIONSHIP contains Parent_Id and Child_Id which are used to keep track of all the references for all the entry_types.
Table Name: RELATIONSHIP

Table 3.18: RELATIONSHIP

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent_Id</td>
<td>Integer</td>
<td>Entry_type Id</td>
</tr>
<tr>
<td>Child_Id</td>
<td>Integer</td>
<td>Entry_type Id</td>
</tr>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
</tbody>
</table>

The table AUTHOR is used to hold the list of all the authors. An entry type can have any number of Authors.

Table Name: AUTHOR

Table 3.19: AUTHOR

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITE_KEY</td>
<td>Varchar2(10)</td>
<td>Foreign Key (Referencing MASTER_ENTRIES table)</td>
</tr>
<tr>
<td>Id</td>
<td>Integer</td>
<td>Id's of all the Entry_type's</td>
</tr>
<tr>
<td>Authors</td>
<td>Varchar(50)</td>
<td>Set of values</td>
</tr>
</tbody>
</table>
The PUBLISHERS table records the information about the Publisher and its address.

Table Name: PUBLISHERS

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Publisher</td>
<td>Varchar(50)</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Varchar(50)</td>
<td>Default value &quot;N.A.&quot;</td>
</tr>
</tbody>
</table>

The ORGANIZATIONS table records the information about the organization and its address.

Table Name: ORGANIZATIONS

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization</td>
<td>Varchar(50)</td>
<td></td>
</tr>
<tr>
<td>Address</td>
<td>Varchar(100)</td>
<td>Default value &quot;N.A.&quot;</td>
</tr>
</tbody>
</table>
The MONTH_YEAR table records the information about the month and year in which
the entry type is published.

Table Name: MONTH_YEAR

Table 3.22: MONTH_YEAR

<table>
<thead>
<tr>
<th>Field</th>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MonthYear_Id</td>
<td>Integer</td>
<td>Primary Key (PK)</td>
</tr>
<tr>
<td>Month</td>
<td>Varchar(10)</td>
<td></td>
</tr>
<tr>
<td>Year</td>
<td>Integer</td>
<td></td>
</tr>
</tbody>
</table>
CHAPTER 4

IMPLEMENTATION - TABLE CREATION, QUERIES

4.1 Table Creation

In this design, we have a master table named 'MASTER_ENTRIES', which has the fields CITE_KEY and ENTRY_TYPE. We have various other tables for each entry types like BOOK, ARTICLE, PROCEEDINGS, MANUALS, etc. All the entry type tables are linked to master table through CITE_KEY. When a new data gets inserted into the entry table say book, it will have unique cite key which is a foreign key and unique entry type id which acts as primary key. The cite key should be updated in both the master table entry as well as the corresponding entry type table, here it is book.

All the entries in the entry type tables can be traced from master table. Each entry in the master table contains the data, cite key and the entry type. Let us say cite key value is 1 and entry type is 'book'. It means that, in the book table, there is an entry which has the cite key as 1. In this manner, all the entries in all other tables like articles, proceedings etc., can be traced.

We have a table called 'RELATIONSHIP'. This table contains the relationship among various entry types. Let us say a book 'A' refers article 'B' and the article 'B' refers a manual 'C', then this table contains parent child relationships to indicate: A→B→C. Note that the relationships can be circular too. That is, C→A. The current queries design
handles this circular references scenario too.

There is a table called 'Author'. It contains various authors and their relationship to the entry types which they have written. There can be many authors for a single entry. The current design handles that too.

Apart from entry types which are in paper printed format, there are types which are magnetic materials too. The tables 'BOOKONCD', 'BOOKONVCD', etc., denotes this representation. Each entry type contains the publisher details, month and year it was published etc., The current design has a separate table to have the information for various publishers and their details. The entry type tables are related with the publisher table through publisher ID field. We have various organizations, who publish manuals, annual reports, conferences etc., these tables are linked with organization table through organization ID.

The current design handles both publishers-entry type relationship and organization-entry type relationship.

4.2 Algorithms

ALGORITHM1: All papers referred by X implicitly and explicitly

INPUT: PID - parent entry type ID

OUTPUT: All the relations in the relationship table where PID is the root of the all derived relations.

PRE-CONDITION: All the entries in the relationship table should have both non-empty parent and child fields. The entries which are at the bottom in the tree structure will have its child field as -1 to denote the end of the relationship tree structure.
Consider the following tree structure.

![Diagram of a tree structure](image)

Figure 4.1: Example for Circular Reference

In the relationship tree structure above, 16 is the top root node. It is given as the input. The algorithm should output all its child nodes in the relationship tree. It should consider the cyclic relationships mentioned in the diagram above. For example, take 63.
It has three children, 43, 70 and 21.

Consider the node 10. It has children 11 and 20. The line relationship between (10, 20), (22, 11) indicates that they have cyclic relationship. (-1) indicates that it does not have any child.

OUTPUT:

In the above tree structure, the following output should be computed by this algorithm.

20 → 43 → 10 → 11 → 12 → 38 → 22 → 13 → 24 → 70 → 63 → 21 → 35 → 17

Note that, even though cyclic relationships were present, all the child nodes of 16 were printed exactly once. Also note that the child nodes can be printed in any order.

DESIGN:

The current algorithm prints the relationship in pre-order traversal. It maintains two package level tables. One is to store the processed child IDs which need to be printed. The other one is to implement the recursive calls if the current node has one or more children.

PROCEDURE FIND_RELATIONS (PID: INPUT)

P_TABLE1 = contains the child node entries to which relationships to be processed recursively. Initially it will contain the input PID value.

P_TABLE2 = contains the final list of child nodes which need to be printed. It will only have unique entries. Initially it will be empty.

Step 1: Create a CURSOR variable for the RELATIONSHIP table where it points to the entries which contain PID as the parent.

Step 2: FOR each CHILD_ID entry the CURSOR variable points to, Do Step 3 to Step 6:
Step 3: Check in the P_TABLE1 whether the entry is already present or not.
Step 4: If it is present, then omit this child entry. Go to the next one the cursor points to.
Step 5: If it is a new entry and it is not equal to -1, then push this entry into both P_TABLE1 and P_TABLE2. -1 means it is the end of the tree hierarchy.
Step 6: CALL FIND_RELATIONS (CHILD_ID) recursively with CHILD_ID as the PID.
Step 7: Remove the PID from PTABLE_1 since it is processed.
Step 8: PRINT the records to which PTABLE_2 entries point to, since they contain the entire child Ids for the input PID.
Step 9: END

ALGORITHM 2: This algorithm prints the number of entry types a publisher ‘A’ published in a particular month and year.

This algorithm takes author ID as the input and finds out all the publications done by the author like books, articles, etc., The AUTHOR table contains all the author and his publications details. The AUTHOR table contains only Ids for both author and his publications. So the algorithm traces back to the MASTER_TABLE and finds out the exact type of the entry and prints it out.
Step 1: OPEN the cursor to the AUTHOR table where it points to all entries to which, AUTHOR_ID is the author.
Step 2: For each AUTHOR_ID entry in the table the CURSOR points to, do the following steps, from Step 3 to Step 6.
Step 3: Get the corresponding master table cite key from the AUTHOR table to which
CURSOR variable points to.

Step 4: Get the corresponding record from the master table for the current master cite key.

Step 5: If the entry type pointed by the master cite key is BOOK, then go to the other corresponding BOOK table using master cite key as foreign key and print out its title and necessary details.

Step 6: Go to the next entry the CURSOR points to.

Step 7: END

UTILITIES_PKG:

ALGORITHM 3: PROCEDURE PRINT_ALL_PUBLISHERS_COUNT

This algorithm prints how many publications each publisher has done. For example, Lets say Publisher A has done 3 books, 3 articles, 3 periodicals till now. So it prints like this: The total publications from publisher A is: 9. It prints for all the publishers in the PUBLISHER table.

PROCEDURE PRINT_ALL_PUBLISHERS_COUNT

Step 1: Open a CURSOR to the PUBLISHER table.

Step 2: For each entry PUBLISHER ID in the PUBLISHER table, to which the CURSOR points to, do the following steps from Step 3 to Step 4:

Step 3: Get the COUNT for books, articles, proceedings, inbooks, inproceedings, journals, magazines, manuals, annual reports and periodicals WHERE the publisher id = PUBLISHER ID.

Step 4: PRINT the COUNT values for each entry type.
ALGORITHM 4: PROCEDURE PRINT_HIGHEST_PUBLISHER_COUNT

This algorithm prints the highest number of publications for all the entry types.

Step 1: Open a CURSOR to the PUBLISHER table.

Step 2: Initialize HIGHEST_COUNT = 0

Step 3: For each entry PUBLISHER_ID in the PUBLISHER table, to which the CURSOR points to, do the following steps from Step 3 to Step 4.

Step 4: Get the COUNT for books, articles, proceedings, inbooks, inproceedings, journals, magazines, manuals, annual reports and periodicals WHERE the publisher id = PUBLISHER_ID.

Step 5: If the COUNT for the current PUBLISHER_ID is greater than HIGHEST_COUNT values, then assign COUNT as the HIGHEST_COUNT.

Step 6: PRINT the HIGHEST_COUNT and its corresponding publisher.

Step 7: END

ALGORITHM 5: PROCEDURE PRINT_CO_AUTHOR

This algorithm prints the various entry types like books, articles etc., written by two authors. It takes two author Ids as inputs.

Step 1: OPEN CURSOR 1 from AUTHOR table WHERE the author ID is AUTHOR ID1.

Step 2: OPEN CURSOR 2 from AUTHOR table WHERE the author ID is AUTHOR ID2.
Step 3: Repeat the steps 4 to 5, for each author entry in the table pointed by the CURSOR1

Step 4: Repeat the step 5, for each author entry in the table pointed by the CURSOR 2

Step 5: If the CITE KEY of the entry pointed by CURSOR 1 = CITE KEY of entry pointed by CURSOR 2 THEN PRINT the entry type by referencing the corresponding entry type table.

Step 6: END

ALGORITHM 6: This algorithm prints the pairs of organization id’s which have the same organization name.

Step 1: OPEN a CURSOR for Organizations table where v_O1Id and v_O2Id are the Organization id

Step 2: For each entry ORGANIZATION_ID in the ORGANIZATION table, to which the CURSOR points to, do the following steps from Step 3 to Step 4

Step 3: Get the pairs of organization id’s which have the same organization name.

Step 4: PRINT the PAIRS of organization ids for each entry type.

Step 5: END

ALGORITHM 7: This algorithm stores the Magazine Name and its Publisher in a temporary table and populates them

Step 1: OPEN a CURSOR for Magazine and Publisher tables where Publisher_Id is the Id of the publisher used to join the two tables.

Step 2: For each entry Magazine Name in the Magazine table and the Publisher_Id, to
which the CURSOR points to, do the following steps from Step 3 to Step 4

Step 3: Get the Magazine Name and its Publisher and store it in a temporary table.

Step 4: PRINT the Magazine Name and Publisher from the temporary table.

Step 5: END

ALGORITHM 8: This algorithm prints the list of first ten entries from the Master entry table.

Step 1: OPEN a CURSOR for Master_Entries table.

Step 2: For each Entry_type in the Master_Entries table to which the CURSOR points to, do the following steps from Step 3 to Step 4 10 times.

Step 3: Get the list of first ten Entry type Id’s from the Master_Entries table.

Step 4: PRINT the Entry_TypeId’s.

Step 5: END
CHAPTER 5

CONCLUSION AND FUTURE WORK

The thesis, “Design and Implementation of Bibliographic Database” studies the various DBMS concepts and their interaction mechanisms. It builds a database by creating various tables defining their primary key, foreign key and other related fields. The database contains a master table which contains the cite key entries for all the entries in any of the entry type table along with entry type. The database entity relationship is depicted in UML notations which is universal standard.

The bibliographic database is designed in relational database, using Oracle as the database engine. So the SQL and PL/SQL commands used in the bibliographic database system is specific to Oracle. The bibliographic database design flows from the master table to other entry type tables. Here the entry type can be book or article or manuals etc., the master table is designed such that the cite key field is used as foreign key field in all the entry types and hence building the relationship among them. Whenever any creation/ updation/ deletion takes place in the entry type tables, the corresponding master table entry should be kept intact.

The relational database is mainly consisting of four levels.

1) Relationships captured for various entry types and their corresponding fields
2) Various publishers who publish these entry types

3) The relationship between various entry types and their authors

4) The relationship among the entry types itself since it can refer one or multiple entry types

5) Various organizations who publish some of the entry types

The current bibliographic database system builds the system based on various queries constructed for the above mentioned relationships. In this exercise, various SQL, PL/SQL constructs like DDL/DML statements, collection data types, cursors, procedures, functions, packages, exceptions are understood and utilized for the system implementation.

Table structures for entry types like book, article, manuals etc., are defined and the data is populated in the table. The tables contain various fields to have relationship with other tables like publisher or organization who publish them. There is a separate table structure for capturing the relationship among the authors and the entry types they have authored. Also there is a table structure for entry type references relationships where it gives the information of which entry type refers to which one.

One of the key features of the current system is: it handles the circular references between the various entity types. An entry type can refer any entry type. So when the user wants to resolve for a relationship A -> B -> C -> A, the circular reference is handled here. Any level of direct or indirect circular reference relationship is automatically handled. In this scenario, the usage of recursive functions, nested data types, manipulating with package level data and cursors are studied.

The current system handles the following query: given a publisher, it prints all
the publications done by the publisher. If the user wants the publications only on a particular year/month, it handles that too. In this scenario, the usage of various PL/SQL structures like procedures, cursors are studied.

The system also studied various authors and their relationship with various entry types. An author might have written various entry types and might have referred various entries for any of his work. This relationship is captured in 'author' table and was studied with various queries implementation.

The relationship between an author and a publisher is also studied. The author might have written many entry types for one or more publications. The user can fetch the information from the current system such as: how many books, the author A has published through publisher B, etc.

An entry type can be written by more than one author. The current system can output all the entries written by two authors. However it can be extended such that the system can find all the entries written by a common set of authors. This case uses looping mechanisms and cursor variable interaction.

Overall, this system studies various SQL features, database management system constructs, statements and their linkage with PL/SQL programming structure. This exercise helps in understanding the DBMS concepts more clearly, in the practical way. In future set of all the rules involved in implementing the logical queries will be defined. XML representation of the DB will be created and executes logic queries on the same using Xquery.


3. T. Connolly, C. Begg. Database Systems - practical approach for design implementation and management


8. Dr. Lorna Scammell, Design your own database http://www.campus.ncl.ac.uk/databases/design/design.html


10. Oracle Database 10g PL/SQL Programming, Chapter 3: PL/SQL Basics


Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
14. An Introduction to Database Systems [Date: Date 1995]; Carter: 1992

VITA

Graduate College
University of Nevada, Las Vegas

Kiranmayi Karamcheti

Address:
4850 156th AVE NE, Apt#188
Redmond, WA 98052

Degree:
• Bachelor of Engineering, Computer Science and Information Technology, 2005
  Jawaharlal Nehru Technological University, India

Thesis Title: Design and Implementation of a Bibliographic Database

Thesis Examination Committee:
• Chairperson, Dr. Kazem Taghva, Ph.D.
• Committee Member, Dr. Ajoy K. Datta, Ph.D.
• Committee Member, Dr. Wolfgang Bein, Ph.D.
• Graduate College Representative, Dr. Venkatesan Muthukumar, Ph.D.