Designing Databases

Introduction:

Database design has five purposes:

1. Structure the data in stable structures, called **normalized tables**, that are not likely to change over time and that have **minimal redundancy**.
2. Develop a logical database design that reflects the **actual data requirements that exist in the forms** (hard copy and computer displays) **and reports** of an information system. This is why database design is often done in parallel with the design of the human interface of an information system.
3. Develop a logical database design from which we can do physical database design. Because most information systems today use relational database management systems, **logical database design usually uses a relational database model, which represents data in simple tables with common columns to link related tables**.
4. Translate a **relational database model into a technical file and database design** that balances several performance factors.
5. Choose **data storage technologies** (such as Read/ Write DVD or optical disc) that will efficiently, accurately and securely process database activities.

Database Design:

File and database design occurs in **two steps**. You begin by developing a **logical database model**, which describes data using a notation that corresponds to a data organization used by a database management system. This is the system software responsible for storing, retrieving, and protecting data (such as Microsoft Access, Oracle, or SQL Server). The most common style for a logical database model is the relational database model. Once you develop a clear and precise logical database model, you are ready to **prescribe the technical specifications for computer files and databases** in which to store the data. A **physical database design** provides these specifications. You typically do logical and physical database design in parallel with other systems design steps. Thus, you collect the **detailed specifications** of data necessary for logical database design as you design system inputs and outputs. Logical database design is driven not only from the previously developed E-R data model for the application or enterprise but also from **form and report layouts**. You study data elements on these system inputs and outputs andidentify interrelationships among the data.

The Process of Database Design

Figure 9-2 shows that database modeling and design activities occur in **all phases** of the systems development process. In this chapter, we discuss methods that help you finalize logical and physical database designs during the design phase. In logical database design, you use a process called **normalization**, which is a way to build a data model that has the properties of simplicity, non redundancy, and minimal maintenance.

There are four key steps in logical database modeling and design:

1. Develop a logical data model for each known **user interface** (**form and report**) for the application using normalization principles.
2. Combine normalized data requirements from all user interfaces into one consolidated logical database model; this step is called **view integration.**
3. Translate the **conceptual E-R** data model developed without explicit consideration of specific user interfaces, into **normalized data requirements.**
4. Compare the **consolidated logical database design** with **the translated E-R model and produce**, **through view integration**, one final logical database model for the application.

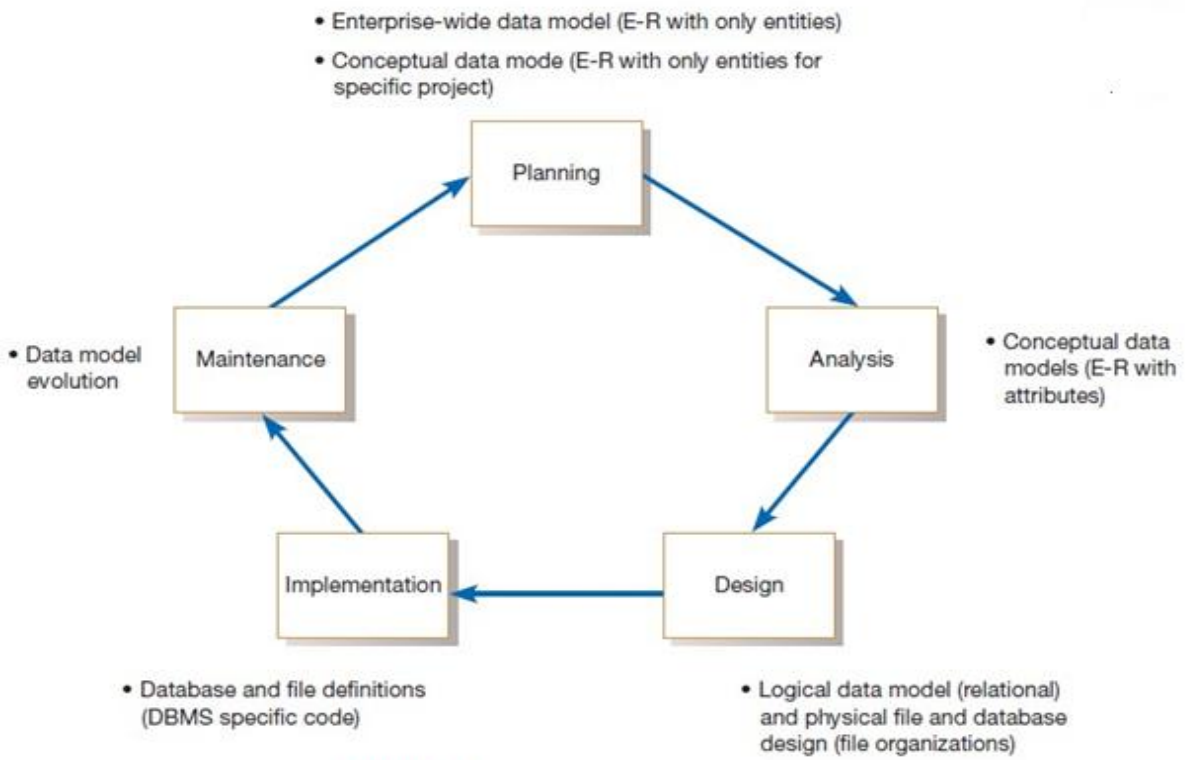

FIGURE 9-2
Relationship between data modeling and the SDLC

Physical data base design include following steps:

1. Choosing the storage format (called data type) for each attribute from the logical database model.
2. Grouping attributes from the **logical database model into physical records.**
3. Arranging related records in secondary memory (hard disks and magnetic tapes) so that individual records and groups of records can be stored, retrieved, and updated rapidly (**called file organization**).
4. Selecting media and structures for storing data to make access more efficient. The choice of media affects the utility of different file organizations. The primary structure used today to make access to data more rapid is **key indexes** on **unique and non unique keys**.

The Relational Database Model

EMPLOYEE1

Emp_ID	Name	Dept	Salary
100	Margaret Simpson	Marketing	75,000
140	Allen Beeton	Accounting	95,000
110	Chris Lucero	Info Systems	90,000
190	Lorenzo Davis	Finance	90,000
150	Susan Martin	Marketing	62,000

The **relational database model** represents data in the form of **related tables**, or relations. A relation is a named, two dimensional table of data. Each relation (or table) consists of a set of named **columns** and an arbitrary number of unnamed **rows**. Each column in a relation corresponds to an attribute of that relation. Each row of a relation corresponds to a record that contains data values for an entity. Above figure (9-5) shows an example of a relation named EMPLOYEE1. This relation contains the following attributes describing employees: Emp_ID, Name, Dept, and Salary. This table has five sample rows, corresponding to five employees.

The identifier attribute (called the primary key of the relation)isunderlined. For example, you would express EMPLOYEE1 as follows:
EMPLOYEE1(**Emp_ID**,Name,Dept,Salary)

Not all tables are relations. Relations have several properties thatdistinguish them from non relational tables:

1. Entries in cells are simple.
2. Entries in a given column are from the same set of values.
3. Each row is unique. Uniqueness is guaranteed because the relation has a nonempty **primary key value**.
4. The sequence of columns can be interchanged without changing themeaning or use of the relation.
5. The rows may be interchanged or stored in any sequences.

Well-structured Relations

What constitutes a **well-structured relation** (also known as a table)? Intuitively, a well structured relation contains a **minimum amount of redundancy** and allows users to insert, modify, and delete the rows in a table without errors or inconsistencies.

EMPLOYEE2

Emp_ID	Name	Dept	Salary	Course	Date_Completed
100	Margaret Simpson	Marketing	42,000	SPSS	6/19/2017
100	Margaret Simpson	Marketing	42,000	Surveys	10/7/2017
140	Alan Beeton	Accounting	39,000	Tax Acc	12/8/2017
110	Chris Lucero	Info Systems	41,500	SPSS	1/22/2017
110	Chris Lucero	Info Systems	41,500	C++	4/22/2017
190	Lorenzo Davis	Finance	38,000	Investments	5/7/2017
150	Susan Martin	Marketing	38,500	SPSS	6/19/2017
150	Susan Martin	Marketing	38,500	TQM	8/12/2017

FIGURE 9-6
Relation with redundancy

EMPLOYEE2 (above fig 9.6) contains data about employees and the courses they have completed. **This is not a well-structured relation**. If you examine the sample data in the table, you notice a considerable amount of redundancy. For example, the Emp_ID, Name, Dept, and Salary values appear in two separate rows for employees **100, 110**, and **150**. Consequently, if the salary for employee 100 changes, we must record this fact in two rows (or more, for some employees).

The problem with this relation is that it contains data about two entities: EMPLOYEE and COURSE. You will learn to use principles of normalization to divide EMPLOYEE2 into two relations. One of the resulting relations is EMPLOYEE1 (Figure 9-5). The other we will call EMP COURSE, which appears with sample data in Figure 9-7. The primary key of this relation is the combination of Emp_ID and Course (we emphasize this by underlining the column names for these attributes).

Normalization: Normalization is a process for converting complex datastructures into simple, stable data structures.

Physical File and Database Design

- **Designing physical files and databases requires certain information that should have been collected and produced during prior** SDLC phases. This information includes the following:
- Normalized relations, including volume estimates
- Definitions of each attribute
- Descriptions of where and when data are used: entered, retrieved, deleted, and updated (including frequencies)
- Expectations or requirements for response time and data integrity
- Descriptions of the technologies used for implementing the files and database so that the range of required decisions and choices for each is known.

Normalized relations are, of course, the result of logical database design. Statistics on the number of rows in each table as well as the other information listed above may have been collected **during requirements determination in systems analysis**. If not, these items need to be discovered to proceed with database design.

Designing Fields

A field is the smallest unit of application data recognized by system software, such as a programming language or database management system. An attribute from a logical database model may be represented by several fields. For example, a student name attribute in a normalized student relation might be represented as three fields: last name, first name, and middle initial. In general, you will represent each attribute from each normalized relation as one or more fields. The basic decisions you must make in specifying each field concern the **type of data** (or storage type) used to represent the field and data integrity controls for the field.

Choosing Data types: You want to choose a data type for a field that minimizes space, represents every possible valid value for the associated attribute, and allows the data to be manipulated as needed. You would select a length for this field that would handle the maximum value.

TABLE 9-2 Commonly Used Data Types in Oracle 10g

Data Type	Description
VARCHAR2	Variable-length character data with a maximum length of 4000 characters; you must enter a maximum field length (e.g., VARCHAR2(30) for a field with a maximum length of 30 characters). A value less than 30 characters will consume only the required space.
CHAR	Fixed-length character data with a maximum length of 255 characters; default length is 1 character (e.g., CHAR(5) for a field with a fixed length of five characters, capable of holding a value from 0 to 5 characters long).
LONG	Capable of storing up to two gigabytes of one variable-length character data field (e.g., to hold a medical instruction or a customer comment).
NUMBER	Positive and negative numbers in the range 10^{-130} to 10^{126}; can specify the precision (total number of digits to the left and right of the decimal point) and the scale (the number of digits to the right of the decimal point) (e.g., NUMBER(5) specifies an integer field with a maximum of 5 digits and NUMBER(5, 2) specifies a field with no more than five digits and exactly two digits to the right of the decimal point).
DATE	Any date from January 1, 4712 BC to December 31, 4712 AD; date stores the century, year, month, day, hour, minute, and second.
BLOB	Binary large object, capable of storing up to four gigabytes of binary data (e.g., a photograph or sound clip).

Calculated Fields: It is common for an attribute to be mathematically related to other data. A field that can be derived from other database fields is called a **calculated field (or a computed field** or a derived field). If you specify a field as calculated, you would then usually be prompted to enter the **formula for the calculation**; the formula can involve other fields from the same record and possibly fields from records in related files. The database technology will either store the calculated value or compute it when requested.

Coding and Compression Techniques: Some attributes have very few values from a large range of possible values. For example, suppose that each product from PVF has a **finish attribute**, with possible values of **Birch, Walnut, Oak, and so forth**. To store this attribute as text might require **12, 15, or even 20 bytes to represent the longest finish value**. Suppose that even a liberal estimate is that PVF will never have more than 25 finishes. Thus, a single alphabetic or alphanumeric character would be more than sufficient. We not only reduce storage space but also increase integrity (by restricting input to only a few values. **Codes also have disadvantages**. If used in system inputs and outputs, they can be more difficult for users to remember, and programs must be written to decode fields if codes will not be displayed.

Designing Physical tables: A relational database is a set of related tables (tables are related by foreign keys referencing primary keys). **In logical database design**, you grouped into a relation those attributes that concern some unifying, normalized business concept, such as a customer, product, or employee. **Physical table is a named set of rows and** columns that specifies the fields in each row of the table. **The design of a physical table**

has two goals : **efficient use of secondary storage** and **data processing speed.**

The efficient use of secondary storage (disk space) relates to **how data are loaded on disks**. Disks are physically divided into **units (called pages) that can be read or written in one machine operation**. Space is used efficiently when the physical length of a table row divides close to evenly into the length of the storage unit. For many information systems, this even division is very difficult to achieve because it depends on factors, such as operating system parameters, outside the control of each database.

A second and often more important consideration when selecting a physical table design is efficient data processing. Data are most efficiently processed when they are stored close to one another in secondary memory, thus minimizing the number of input/output (I/O) operations that must be performed. Typically, the data in one physical table (all the rows and fields in those rows) are stored close together on disk.