chapter

2

Introduction to Database Development

DO NOT COPY, POST, OR DISTRIBUTE

Learning Objectives

This chapter provides an overview of the database development process. After this chapter, the student should have acquired the following knowledge and skills.

- Explain the steps in the information systems life cycle
- Describe the role of databases in an information system
- Explain the goals of database development
- Understand the relationships among phases in the database development process
- Describe features typically provided by CASE tools for database development

OVERVIEW

Chapter 1 provided a broad introduction to database usage in organizations and database technology. You learned about the characteristics of business databases, essential features of database management systems (DBMSs), architectures for deploying databases, and organizational roles interacting with databases. This chapter continues your introduction to database management with a broad focus on database development. You will learn about the context, goals, phases, and tools of database development to facilitate the acquisition of specific knowledge and skills in Parts 3 and 4.

Before you can learn specific skills, you need to understand the broad context for database development. This chapter presents a context for databases as part of an information system. You will learn about components of information systems, the life cycle of information systems, and the role of database development as part of information systems development. This information systems context provides a background for database development. You will learn the phases of database development, the skills used in database development, and software tools that can help you develop databases.

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

2.1 INFORMATION SYSTEMS

Databases exist as part of an information system. Before you can understand database development, you must understand the larger environment that surrounds a database. This section describes the components of an information system and several methodologies to develop information systems.

2.1.1 Components of Information Systems

A system is a set of related components that work together to accomplish defined objectives. A system interacts with its environment and performs functions to accomplish objectives. For example, the human circulatory system, consisting of blood, blood vessels, and the heart, makes blood flow to various parts of the body. The circulatory system interacts with other systems of the body to ensure that the right quantity and composition of blood arrives in a timely manner to various body parts.

An information system is like a physical system (such as the circulatory system) except that an information system manipulates data rather than a physical object like blood. An information system accepts data from its environment, processes data, and produces information for decision making. For example, an information system for processing student loans (Figure 2.1) helps a service provider track loans for lending institutions. This system's environment consists of lenders, students, and government agencies. Lenders send approved loan applications, and students receive cash for school expenses. After graduation, students receive monthly statements and remit payments to retire their loans. If a student defaults, a government agency receives a delinquency notice.

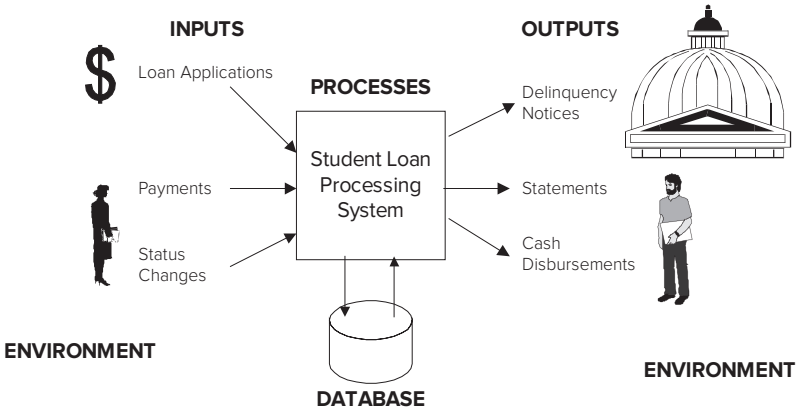
Databases provide long-term memory for information systems, an essential role. The long-term memory contains entities and relationships. The database in Figure 2.1 contains data about students, loans, and payments to generate statements, cash disbursements, and delinquency notices. Information systems without permanent memory or with only a few variables in permanent memory are typically embedded in a device to provide a limited range of functions rather than an open range of functions as business information systems provide.

Databases are not the only components of information systems. Information systems also contain people, procedures, input data, output data, software, and hardware. Thus, developing an information system involves more than developing a database, as discussed in the next subsection.

2.1.2 Information Systems Development Process

Figure 2.2 shows the phases of the traditional systems development life cycle. The phases of the life cycle are not standard. Different authors and organizations have

FIGURE 2.1
Overview of Student Loan Processing System



DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

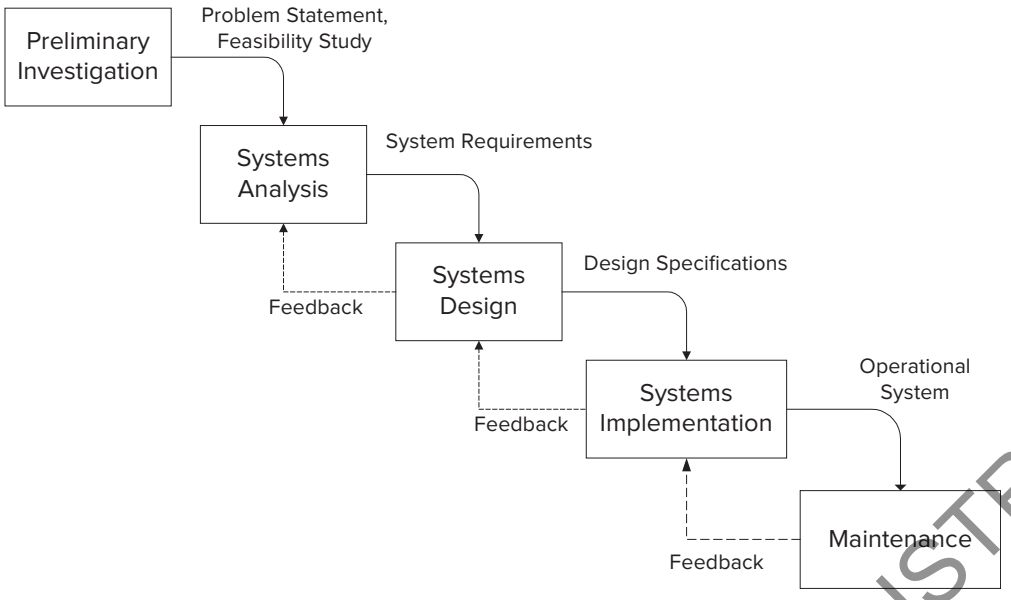


FIGURE 2.2
Traditional Systems Development Life Cycle

proposed from 3 to 20 phases. The traditional life cycle, known as the waterfall model, contains sequential flow in which the result of each phase flows to the next phase. The traditional life cycle is mostly a reference framework. For most systems, the boundary between phases overlaps with considerable backtracking among phases. However, the traditional life cycle is still useful because it describes the activities and shows the addition of detail until an operational system emerges. The following items describe the activities in each phase.

- *Preliminary Investigation Phase*: Produces a problem statement and feasibility study. The problem statement contains the objectives, constraints, and scope of the system. The feasibility study identifies the costs and benefits of the system. If the system is feasible, systems analysis begins with approval.
- *Systems Analysis Phase*: Produces requirements describing processes, data, and environment interactions. This phase uses diagramming techniques to document processes, data, and environment interactions. To produce requirements, analysts study the current system and interview users of the proposed system.
- *Systems Design Phase*: Produces a plan to implement the requirements efficiently. Analysts produce design specifications for processes, data, and environment interaction. The design specifications focus on choices to optimize resources given constraints.
- *Systems Implementation Phase*: Produces executable code, databases, and user documentation. To implement the system, developers generate code to implement design specifications. Before making the new system operational, project managers devise a transition plan from the old system to the new system. To gain confidence and experience with the new system, an organization may run the old system in parallel to the new system for a period.
- *Maintenance Phase*: Produces corrections, changes, and enhancements to an operating information system. The maintenance phase commences when an information system becomes operational. The maintenance phase is fundamentally different from other phases because it comprises activities from all the other phases. The maintenance phase ends after deploying a replacement system and retiring the current system. Due to the high fixed costs of developing new systems, the maintenance phase can last decades.

DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

The traditional life cycle has been criticized for several reasons. First, an operational system is not produced until late in the process. When a system finally becomes operational, the requirements may have already changed. Second, there is often a rush to begin implementation so that a product is visible. In this rush, appropriate time may not be devoted to analysis and design.

Several alternative methodologies have been proposed to alleviate these difficulties. Spiral development methodologies perform life cycle phases for subsets of a system, progressively producing a larger system until the complete system emerges. Rapid application development methodologies delay producing design documents until requirements are clear. Scaled-down versions of a system, known as prototypes, clarify requirements. Prototypes can be implemented rapidly using graphical development tools for generating menus, forms, reports, and other code. Implementing a prototype allows users to provide meaningful feedback to developers. Often, users may not understand the requirements unless they experience a prototype. Thus, prototyping can reduce the risk of developing an information system because it allows earlier and more direct feedback about the system.

Agile development methodologies are another variation to traditional information systems development. To mitigate rapidly changing software requirements and risks caused by long development cycles, agile development methodologies promote active user involvement and team empowerment, viewing software development as an empirical process. Requirements evolve in agile development, but the timescale of development is fixed. Agile development involves iteration through small incremental releases with testing integrated throughout the project lifecycle. Extreme programming, a prominent agile development approach, features a set of primary technical practices and a set of corollary technical practices. Scrum, a subset of agile, provides a set of concepts and practices for reducing software development overhead and maximizing productive work.

All development methodologies produce graphical models of the data, processes, and environment interactions. The data model describes the entity types and relationships. The process model describes relationships among processes. A process can provide input data used by other processes and use the output data of other processes. The environment interaction model describes relationships between events and processes. An event such as the passage of time or an action from the environment can trigger a process to start or stop. The systems analysis phase produces an initial version of these models. The systems design phase adds more details for the efficient implementation of the models.

Even though models of data, processes, and environment interactions are necessary to develop an information system, this book emphasizes data models only. In many information systems development efforts, the data model is the most important. For business information systems, development processes usually produce the process and environment interaction models after the data model. Rather than present notation for the process and environment interaction models, this book emphasizes form and report development to depict connections among data, processes, and the environment.

2.2 GOALS OF DATABASE DEVELOPMENT

Broadly, the goal of database development involves the creation of a database that provides an important resource for an organization. To fulfill this broad goal, the database should serve a large community of users, support organizational policies, contain high-quality data, and provide efficient access. The remainder of this section describes the goals of database development in more detail.

2.2.1 Develop a Common Vocabulary

A database provides a common vocabulary for an organization. Before implementing a common database, different parts of an organization may have different terminology.

DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

For example, there may be multiple formats for addresses, multiple ways to identify customers, and different ways to calculate interest rates. After implementing a database, communication can improve among different parts of an organization. Thus, a database can unify an organization by establishing a common vocabulary.

Achieving a common vocabulary is not easy. Developing a database requires compromise to satisfy a large community of users. In some sense, a good database designer shares some characteristics with a good politician. A good politician often finds compromise solutions with a level of approval and disapproval. In establishing a common vocabulary, a good database designer also finds similar imperfect solutions. Forging compromises can be difficult, but the results can improve productivity, customer satisfaction, and other organizational performance measures.

2.2.2 Define Business Rules

A database contains business rules to support organizational policies. Defining business rules is the essence of defining the semantics or meaning of a database. For example, in an order entry system, an order must precede a shipment, a fundamental rule of order processing. A database can contain integrity constraints to support this rule. Defining business rules enables a database to support organizational policies actively. This active role contrasts with the more passive role that databases have in establishing a common vocabulary.

In defining business rules, a database designer must choose constraint levels to balance the competing needs of different groups. Overly strict constraints may force workaround solutions to handle exceptions. In contrast, loose constraints may allow incorrect data in a database. For example, in a university database, a designer must decide if a course offering can be stored without knowing the instructor. Some user groups may want the initial entry of the instructor to ensure that course commitments can be met. Other user groups may want more flexibility to be able to release course schedules early. Forcing an entry of the instructor name at the time a course offering is stored may be too strict. If a database contains this constraint, users may use workarounds by using a default value such as TBA (to be announced). The appropriate constraint (forcing an entry of the instructor name or allowing later entry) depends on the importance of the needs of the user groups compared to the goals of the organization.

2.2.3 Ensure Data Quality

The importance of data quality is analogous to the importance of product quality in manufacturing. Poor product quality can lead to loss of sales, litigation, and customer dissatisfaction. Because data are the product of an information system, data quality is equally important. Poor data quality can lead to poor decision-making about communicating with customers, identifying repeat customers, tracking sales, and resolving customer problems. For example, communicating with customers can be difficult if addresses are outdated or customer names are inconsistently spelled on different orders.

Data quality has many dimensions or characteristics, as depicted in Table 2-1. The importance of data quality characteristics can depend on the part of the database in which they are applied. For example, in the product part of a retail grocery database, important characteristics of data quality may be the timeliness and consistency of prices. For other parts of the database, other characteristics may be more important.

A database design should help achieve adequate data quality. When evaluating alternatives, a database designer should consider data quality characteristics. For example, in a customer database, a database designer should consider the possibility that some customers may not have U.S. addresses. Therefore, the database design may be incomplete if it fails to support non-U.S. addresses.

Achieving adequate data quality may require a cost-benefit trade-off. For example, in a grocery store database, the benefits of timely price updates are reduced consumer complaints and less loss in fines from government agencies. Achieving data quality

DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

TABLE 2-1
Common Characteristics of Data Quality

Characteristic	Meaning
Completeness	Database represents all important parts of the information system.
Lack of ambiguity	Each part of the database has only one meaning.
Correctness	Database contains values perceived by the user.
Timeliness	Business changes are posted to the database without excessive delays.
Reliability	Failures or interference do not corrupt database.
Consistency	Different parts of the database do not conflict.

can be costly both in preventative and monitoring activities. For example, to improve the timeliness and accuracy of price updates, automated data entry may be used (preventative activity) as well as sampling the accuracy of the prices charged to consumers (monitoring activity).

The cost-benefit trade-off for data quality should consider long-term and short-term costs and benefits. Often the benefits of data quality are long-term, especially data quality issues that cross individual databases. For example, consistency of customer identification across databases can be a crucial issue for strategic decision-making. The issue may not be important for individual databases. Chapter 14 on data integration addresses issues of data quality related to strategic decision-making.

Organizations increasingly recognize that poor data quality can bring extra risks to an organization especially related to litigation and government regulations. Many businesses and government agencies have data governance organizations that deal with data quality, privacy, and security issues in a broad context. For data quality improvements, data governance initiatives typically focus on the development of data quality measures, reporting the status of data quality, and establishing decision rights and accountabilities. Chapter 16 provides details about data governance processes and tools covering data quality issues.

2.2.4 Find an Efficient Implementation

Even if the other design goals are met, a slow-performing database will not be used. Thus, finding an efficient implementation is paramount. However, an efficient implementation should respect the other goals as much as possible. An efficient implementation that compromises the meaning of the database or database quality may be rejected by database users.

Finding an efficient implementation is an optimization problem with an objective and constraints. Informally, the objective is to maximize performance subject to constraints about resource usage, data quality, and data meaning. Finding an efficient implementation can be difficult because of the number of choices available, the interaction among choices, and the difficulty of describing inputs. In addition, finding an efficient implementation is a continuing effort. Performance should be monitored and design changes should be made if warranted.

2.3 DATABASE DEVELOPMENT PROCESS

This section describes the phases of the database development process and discusses relationships to the information systems development process. The chapters in Parts 3 and 4 elaborate on the framework provided here.

2.3.1 Phases of Database Development

The goal of the database development process is to produce an operational database for an information system. To produce an operational database, you need to define the

DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

three schemas (external, conceptual, and internal) and populate (supply with data) the database. To create these schemas, you can follow the process depicted in Figure 2.3. The first two phases are concerned with the information content of the database while the last two phases are concerned with efficient implementation. These phases are described in more detail in the remainder of this section.

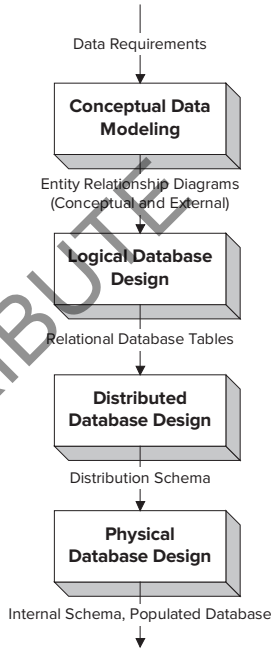


FIGURE 2.3
Phases of Database Development

Conceptual Data Modeling The conceptual data modeling phase uses data requirements and produces entity relationship diagrams (ERDs) for the conceptual schema and each external schema. Data requirements can have many formats such as interviews with users, documentation of existing systems, and proposed forms and reports. The conceptual schema should represent all the requirements and formats. In contrast, the external schemas (or views) represent the requirements of a particular usage of the database such as a form or report, rather than all requirements. Thus, external schemas are generally much smaller than the conceptual schema.

The conceptual and external schemas follow the rules of the Entity Relationship Model, a graphical representation that depicts things of interest (entities) and relationships among entities. Figure 2.4 depicts an entity relationship diagram (ERD) for part of a student loan system. The rectangles (*Student* and *Loan*) represent entity types, and labeled lines (*Receives*) represent relationships. Attributes or properties of entities are listed inside the rectangle. The underlined attribute, known as the primary key, provides a unique identification for the entity type. Chapter 3 provides a precise definition of primary keys. Chapters 5 and 6 present more details about the Entity Relationship Model. Because the Entity Relationship Model is not fully supported by any DBMS, the conceptual schema is not biased toward any specific DBMS.

Logical Database Design The logical database design phase transforms the conceptual data model into a format understandable by a commercial DBMS. The logical design phase is not concerned with efficient implementation. Rather, the logical design phase is concerned with refining the conceptual data model. The refinements preserve the information content of the conceptual data model while enabling implementation on a commercial DBMS. Because most business databases are implemented on relational DBMSs, the logical design phase usually produces a table design compliant with the SQL standard.

The logical database design phase consists of two refinement activities: conversion and normalization. The conversion activity transforms ERDs into table designs using conversion rules. As you will learn in Chapter 3, a table design includes tables, columns, primary keys, foreign keys (links to other related tables), and other constraints. For example, the ERD in Figure 2.4 is converted into two tables, as depicted in Figure 2.5. The normalization activity removes redundancies in a table design using constraints or dependencies among columns. Chapter 6 presents conversion rules, while Chapter 7 presents normalization techniques.

Distributed Database Design The distributed database design phase marks a departure from the first two phases. The distributed database design and physical database design phases are both concerned with an efficient implementation. In contrast, the first two phases (conceptual data modeling and logical database design) are concerned with the information content of the database.

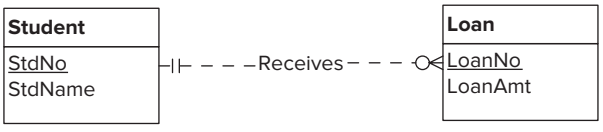


FIGURE 2.4
Partial ERD for the Student Loan System

DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

FIGURE 2.5
Conversion of Figure 2.4

```
CREATE TABLE Student
(  StdNo    INTEGER     NOT NULL,
   StdName  CHAR(50),
   …
PRIMARY KEY (StdNo)     );
CREATE TABLE Loan
( LoanNo    INTEGER     NOT NULL,
  LoanAmt   DECIMAL(10,2),
  StdNo     INTEGER     NOT NULL,
  …
PRIMARY KEY (LoanNo),
FOREIGN KEY (StdNo) REFERENCES Student  );
```

Distributed database design involves choices about the location of data and processes to improve performance and provide local control of data. Performance can be measured in many ways, such as reduced response time, improved data availability, and improved control. For data location decisions, the database can be split in many ways to distribute it among computer sites. For example, a loan table can be distributed according to the location of the bank granting the loan. Another technique to improve performance is to replicate or make copies of parts of the database. Replication improves the availability of the database but makes updating more difficult because multiple copies must be kept consistent.

Data location decisions should respect data ownership. An organization that controls some part of a database should control access to its data. For example, a franchise store should have control over access to its locally generated data. Distributed database technology presented in Chapter 18 enables an organization to align data location with data control.

For process location decisions, some of the work is typically performed on a server and some of the work is performed by a client. For example, the server often retrieves data and sends them to the client. The client displays the results in an appealing manner. There are many other options about the location of data and processing that are explored in Chapter 18.

Physical Database Design The physical database design phase, like the distributed database design phase, is concerned with an efficient implementation. Unlike distributed database design, physical database design involves performance at one computer location only. If a database is distributed, physical design decisions must be made for each location. An efficient implementation minimizes response time without using excessive resources such as disk space and main memory. Because response time is difficult to directly measure, other measures such as the amount of disk input-output activity are often used as a substitute.

In the physical database design phase, two important choices involve indexes and data placement. An index is an auxiliary file that can improve performance. For each table column, the designer decides whether an index can improve performance. An index can improve performance on retrievals but reduce performance on updates. For example, indexes on the primary keys (*StdNo* and *LoanNo* in Figure 2.5) can usually improve performance. For data placement, a designer makes decisions about clustering to locate data close together on a disk. For example, performance might improve by placing student rows near the rows of associated loans. Chapter 8 describes details of physical database design, including index selection and data placement.

Splitting Conceptual Design for Large Projects The database development process shown in Figure 2.3 works well for moderate-size databases. For large databases, the conceptual modeling phase is usually modified. Designing large databases is a time-consuming and labor-intensive process often involving a team of designers. The develop-

DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

ment effort can involve requirements from many different groups of users. To manage complexity, a divide and conquer strategy is used in many areas of computing. Dividing a large problem into smaller problems allows the smaller problems to be solved independently. The solutions to the smaller problems are then combined into a solution for the entire problem.

View design and integration (Figure 2.6) is an approach to managing the complexity of large database development efforts. In view design, an ERD is constructed for each group of users. A view is typically small enough for a single person to design. Multiple designers can work on views covering different parts of the database. The view integration process merges the views into a complete and consistent conceptual schema. Integration involves recognizing and resolving conflicts. To resolve conflicts, it is sometimes necessary to revise the conflicting views. Compromise is an important part of conflict resolution in the view integration process.

Cross-Checking with Application Development The database development process does not exist in isolation. Database development sometimes occurs concurrently with activities in the systems analysis, systems design, and systems implementation phases. The conceptual data modeling phase is part of the systems analysis phase. The logical database design phase is performed during systems design. The distributed database design and physical database design phases are usually divided between systems design and systems implementation. Most of the preliminary decisions for the last two phases can be made in systems design. However, many physical design and distributed design decisions must be tested on a populated database. Thus, some activities in the last two phases occur in systems implementation.

To fulfill the goals of database development, the database development process must be tightly integrated with other parts of information systems development. To produce data, process, and interaction models that are consistent and complete, cross-checking can be performed, as depicted in Figure 2.7. The information systems development process can be split between database development and applications development. The database development process produces ERDs, table designs, and so on as described in this section. The applications development process produces process models, interaction models, and prototypes. Prototypes are especially important for cross-checking. A database has no value unless it supports intended applications such as forms and reports. Prototypes can help reveal mismatches between the database and applications using the database.

DO NOT COPY, POST, OR DISTRIBUTE

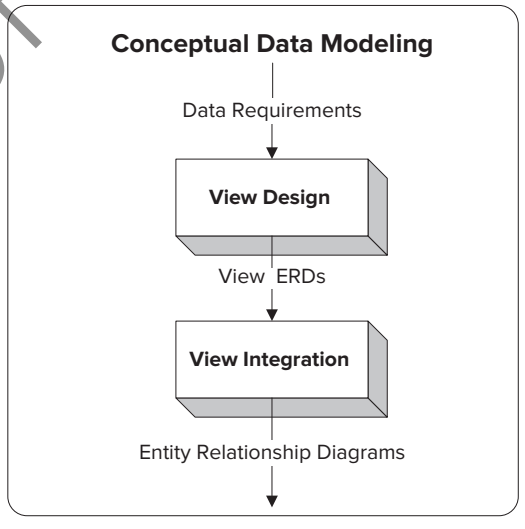
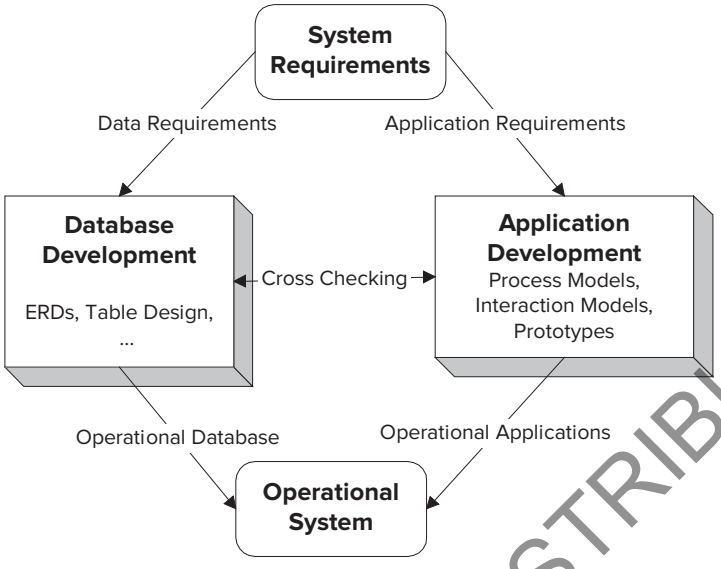


FIGURE 2.6
Splitting of Conceptual Data Modeling into View Design and View Integration

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

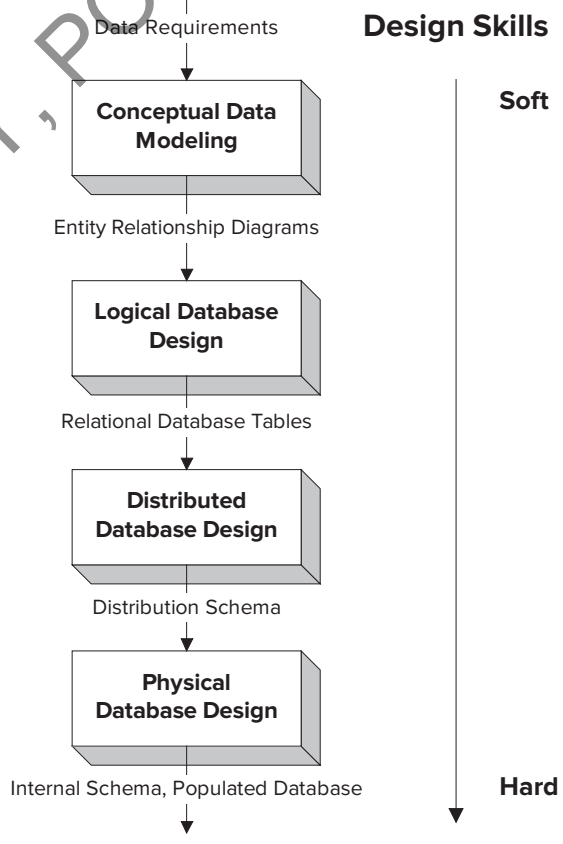
FIGURE 2.7
Interaction between Database and Application Development



2.3.2 Skills in Database Development

As a database designer, you need two different kinds of skills, as depicted in Figure 2.8. The conceptual data modeling and logical database design phases involve mostly soft skills. Soft skills are qualitative, subjective, and people-oriented. Qualitative skills emphasize the generation of feasible alternatives rather than the best alternatives. As a database designer, you want to generate a range of feasible alternatives. The choice among feasible alternatives can be subjective. You should note the assumptions in

FIGURE 2.8
Design Skills Used in Database Development



DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

which each feasible alternative is preferred. The alternative chosen is often subjective based on the designer's assessment of the most reasonable assumptions. Conceptual data modeling is especially people-oriented. In performing data modeling, you need to obtain requirements from diverse groups of users. Compromise and effective listening are essential skills in data modeling.

Distributed database design and physical database design involve mostly hard skills. Hard skills are quantitative, objective, and data-intensive. A background in quantitative disciplines such as statistics and operations management can be useful to understand mathematical models used in these phases. Many of the decisions in these phases can be modeled mathematically using an objective function and constraints. For example, the objective function for index selection is to minimize disk reads and writes with constraints about the amount of disk space and response time limitations. Many decisions cannot be based on objective criteria alone because of uncertainty about database usage. To resolve uncertainty, intensive data analysis can be useful. The database designer should collect and analyze data to understand patterns of database usage and database performance.

Because of the diverse skills and background knowledge required in different phases of database development, role specialization can occur. Large organizations typically provide specialization in database design roles between data modelers and database performance experts. Data modelers perform conceptual data modeling and logical database design phases. Database performance experts mostly perform tasks in the distributed and physical database design phases. Because the skills are different in these roles, the same person will not perform both roles in large organizations. Small organizations typically lack role diversification with the same person fulfilling multiple roles.

2.4 TOOLS FOR DATABASE DEVELOPMENT

To improve productivity in developing information systems, computer-aided software engineering (CASE) tools have been created. CASE tools can help improve the productivity of information systems professionals working on large projects as well as end users working on small projects. Several studies have provided evidence that CASE tools facilitate improvements in the early phases of systems development, leading to lower cost, higher quality, and faster implementations.

Most CASE tools support the database development process. Some CASE tools support database development as a part of information systems development. Other CASE tools target various phases of database development without supporting other aspects of information systems development.

CASE tools can be classified as front-end or back-end tools. Front-end CASE tools help designers diagram, analyze, and document models used in the database development process. Back-end CASE tools create prototypes and generate code that can be used to cross-check a database with other components of an information system. This section presents features of CASE tools for database development and demonstrates a commercial CASE tool, Aqua Data Studio, with a focus on database development.

2.4.1 Diagramming

Diagramming is the most important and widely used function in CASE tools. Most CASE tools provide predefined shapes and connections among the shapes. The connection tools typically allow shapes to be moved while remaining connected as though glued. This glue feature provides important flexibility because symbols on a diagram typically are rearranged many times.

For large drawings, CASE tools provide several features. Most CASE tools allow diagrams to span multiple pages. Multiple-page drawings can be printed so that the pages can be pasted together to make a wall display. Layout can be difficult for large

DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

drawings. Some CASE tools try to improve the visual appeal of a diagram by performing an automatic layout. The automatic layout feature may minimize the number of crossing connections in a diagram. Although the automated layout is not typically sufficient by itself, a designer can use it as a first step to improve the visual appearance of a large diagram.

2.4.2 Documentation

Documentation is one of the oldest and most valuable functions of CASE tools. CASE tools store various properties of a data model and link the properties to symbols on the diagram. Example properties stored in a CASE tool include alias names, integrity rules, data types, and owners. In addition to properties, CASE tools store text describing assumptions, alternatives, and notes. Both the properties and text are stored in the data dictionary, the database of the CASE tool. The data dictionary is also known as the repository or encyclopedia.

To support system evolution, many CASE tools can document versions. A version is a group of changes and enhancements to a system that is released together. Because of the volume of changes, groups of changes rather than individual changes are typically released together. In the life of an information system, many versions can be made. To aid in understanding relationships among versions, many CASE tools support documentation for individual changes and entire versions.

2.4.3 Analysis

CASE tools can provide active assistance to database designers through analysis functions. In documentation and diagramming, CASE tools help designers become more proficient. In analysis functions, CASE tools can perform the work of a database designer. An analysis function is any form of reasoning applied to specifications produced in the database development process. For example, an important analysis function is to convert between an ERD and a table design. Converting from an ERD to a table design is known as forward engineering and converting in the reverse direction is known as reverse engineering.

Analysis functions can be provided in each phase of database development. In the conceptual data modeling phase, analysis functions can reveal conflicts in an ERD. In the logical database design phase, conversion and normalization are common analysis functions. Conversion produces a table design from an ERD. Normalization removes redundancy in a table design. In the distributed database design and physical database design phases, analysis functions can suggest decisions about data location and index selection. In addition, analysis functions for version control can cross database development phases. Analysis functions can convert between versions and show a list of differences between versions.

Because analysis functions are advanced features in CASE tools, the availability of analysis functions varies widely. Some CASE tools support little or no analysis functions, while others support extensive analysis functions. Because analysis functions can be useful in each phase of database development, no single CASE tool provides a complete range of analysis functions. CASE tools tend to specialize by the phases supported. CASE tools independent of a DBMS typically specialize in analysis functions in the conceptual data modeling phase. In contrast, CASE tools offered by a DBMS vendor often specialize in physical database design phases.

2.4.4 Prototyping Tools

Prototyping tools provide a link between database development and application development. Prototyping tools can be used to create forms and reports that use a database. Because prototyping tools may generate code (SQL statements and programming language code), they are sometimes known as code generation tools. Prototyping tools are often provided as part of a DBMS. The prototyping tools may provide wizards to

DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

aid a developer in quickly creating applications that can be tested by users. Prototyping tools can also create an initial database design by retrieving existing designs from a library of designs. This kind of prototyping tool can be very useful to end users and novice database designers.

2.4.5 Commercial CASE Tools

Table 2-2 summarizes major CASE tools that provide extensive features for database development. Each product in Table 2-2 supports multiple steps in database development, although the quality, depth, and breadth of features vary across products. In addition, most of the products in Table 2-2 provide several versions that vary in price and features. All of the products are relatively neutral to a particular DBMS even though two products are offered by organizations with major DBMS products. Besides the full-featured products listed in Table 2-2, other companies offer drawing tools for database diagrams.

ER Modeler in Aqua Data Studio To depict features of commercial CASE tools, this section concludes with an overview of the ER Modeler component of Aqua Data Studio. The ER Modeler provides excellent drawing capabilities, forward and reverse engineering

TABLE 2-2
Prominent CASE Tools for Database Development

Tool	Vendor	Innovative Features
SAP PowerDesigner	SAP	Forward and reverse engineering for relational databases and many programming languages; model management support for comparing and merging models; application code generation; UML support; business process modeling; XML code generation; version control; data integration support; physical design support; support for industry-standard enterprise architecture frameworks
Oracle SQL Developer Data Modeler	Oracle	Forward and reverse engineering for relational databases; data warehouse modeling; code generation for other DBMSs; compare and merge models; version control; name standardization; design rules; impact analysis; wizards for view creation, view discovery, and foreign key discovery
ERWin Data Modeler	ERWin	Forward and reverse engineering for relational databases; model reuse tools; bi-directional compare; model change impact analysis; schema and design analysis; version control; sub modeling support; workgroup support
ER/Studio Data Architect	IDERA	Forward and reverse engineering for relational databases; automated diagram layout; visual data lineage; model management support for comparing and merging models; UML support; version control; schema patterns for model reuse; workgroup support; data integration support
Visible Analyst	Visible Systems	Forward and reverse engineering for relational databases and XML; model management support for comparing and merging models; version control; database view design; data warehouse design diagrams; business requirements traceability; process integration with data; Enterprise Edition supports Zachman Framework for enterprise architecture design
Aqua Data Studio	AquaFold	Forward and reverse engineering, schema comparison, version control, DBA tools, query builder, schema object management
Visual Paradigm	Visual Paradigm	Forward and reverse engineering, editors for tables and views, generation of database patch scripts, trigger and stored procedure support, support for project management, enterprise architecture, system modeling, business modeling, user interface requirements, and software requirements in Visual Paradigm tool

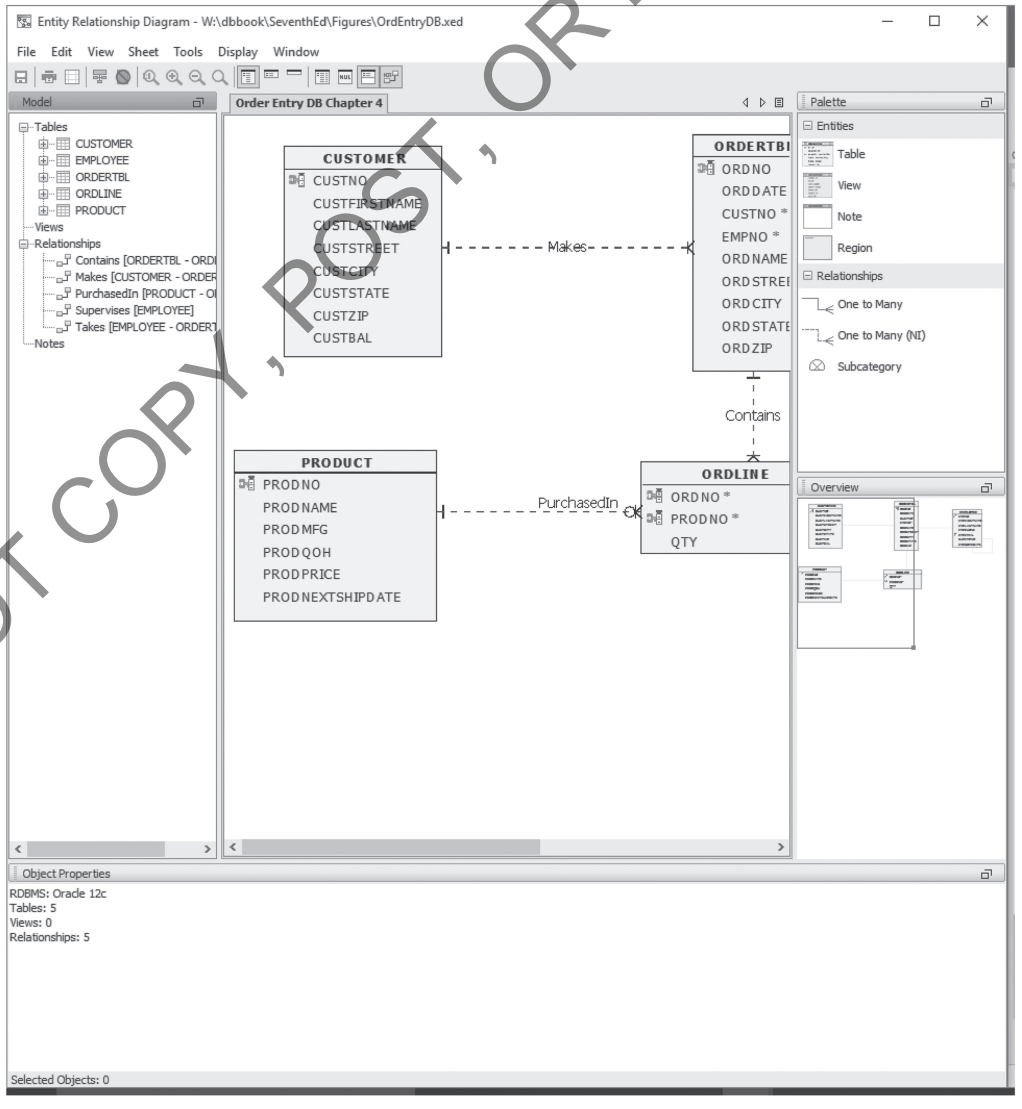
DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

tools, and schema comparison tools. In addition to the ER Modeler component, Aqua Data Studio provides DBA tools for managing databases in a variety of DBMSs, a query builder, and code generation. Thus, Aqua Data Studio supports traditional CASE tool features as well as features to manage operations of databases.

The ER Modeler window contains panes for a drawing area, model objects, a palette of diagram shapes, an overview pane for managing large drawings, and an object summary as shown in Figure 2.9. The drawing pane contains a number of drawing sheets, each containing a database diagram. In Figure 2.9, the drawing pane contains one sheet showing a database diagram for an order entry database. The Palette pane shows entities (table, view, note, and region) and relationships (One to Many, One to Many (NI), and Subcategory) that can be placed in a drawing sheet. The Overview pane compresses the entire diagram with a red rectangle surrounding the visible part of the diagram. The Model pane displays the objects in a diagram (tables and relationships) with expansion to display details. In Figure 2.9, the Model pane expands tables and relationships to show the objects in the diagram. The Object Properties pane lists properties of the object selected in the drawing sheet. In Figure 2.9, the Object Properties pane lists properties of the entire diagram because no object in the diagram is selected.

FIGURE 2.9
ER Modeler Window



DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

The ER Modeler provides multiple levels of detail in the drawing pane. Figure 2.9 shows the attribute level with table and column names. Relationship names can be added to the attribute level display, as shown in Figure 2.9. The ER Modeler supports less detail with the primary key level (table and primary key names) and the entity level (just table names) and more detail with the physical schema level (data types added to the attribute level), nullable columns (attribute level and null constraints), and the comment level (attribute level and comments).

The ER Modeler provides a data dictionary with details of each object in a diagram. To edit properties in the data dictionary, you use the properties window for a specified object. Figure 2.10 displays the properties window for the *Product* table with tabs separating different collections of properties. The General tab shows the column names, data types, lengths and nulls allowed values for each column. Figure 2.11 displays properties for the *PurchasedIn* relationship with tabs for several collections of properties. The General tab contains the most prominent properties, including cardinality, type, and nulls.

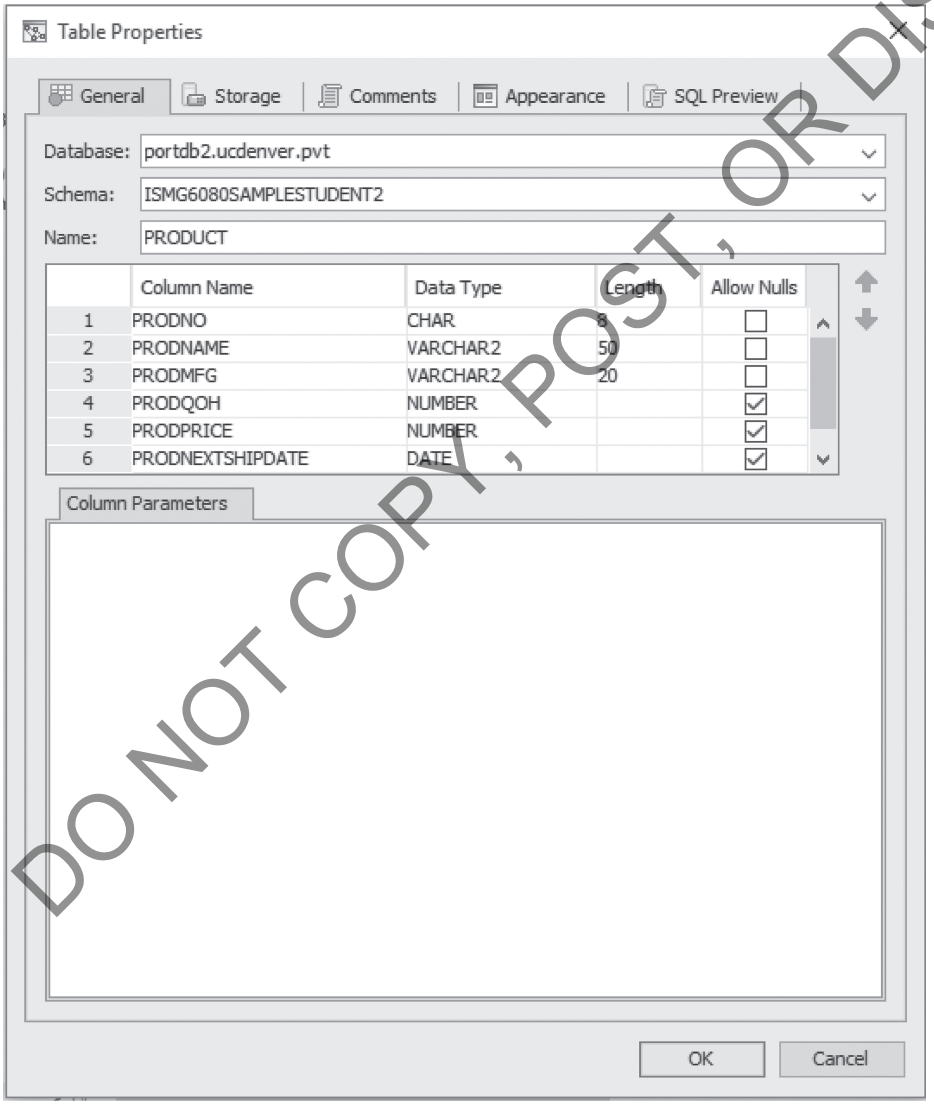


FIGURE 2.10
Table Properties Window for the *Product* Table

DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

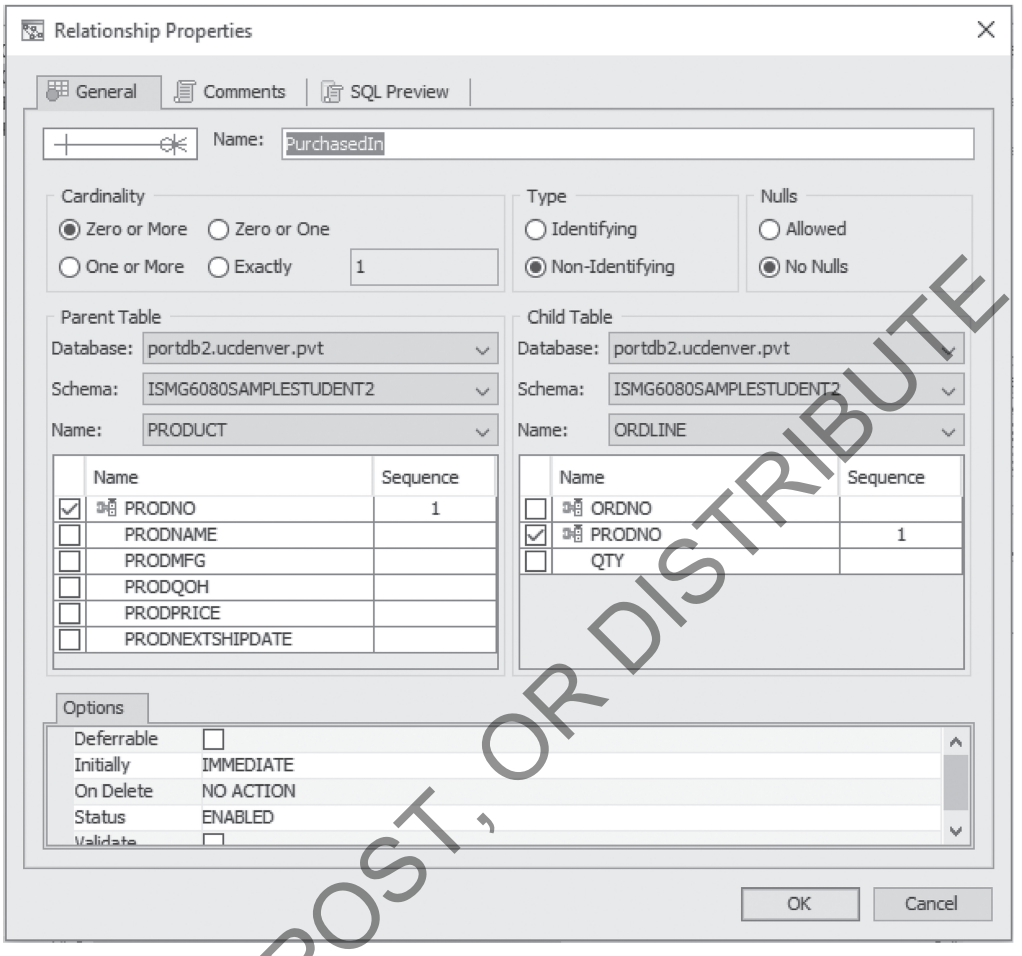


FIGURE 2.11
Relationships Properties Window for the *PurchasedIn* Relationship

CLOSING THOUGHTS

This chapter initially described the role of databases in information systems and the nature of the database development process. Information systems are collections of related components that produce data for decision-making. A database provides the permanent memory for an information system. Development of an information system involves a repetitive process of analysis, design, and implementation. Database development occurs in all phases of systems development. Because a database is often a crucial part of an information system, database development can be the dominant part of information systems development. Development of the processing and environment interaction components are often performed after the database development. Cross-checking between a database and applications connects the database development process to the information systems development process.

After presenting the role of databases and the nature of database development, this chapter described the goals, phases, and tools of database development. The goals emphasize both the information content of the database and efficient implementation. The phases of database development first establish the information content of the database and then find an efficient implementation. The conceptual data modeling and logical database design phases involve the information content of the database. The distributed database design and physical database design phases involve

DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

efficient implementation. Because developing databases can be a challenging process, computer-aided software engineering (CASE) tools have been created to improve productivity. CASE tools can be essential in helping the database designer to draw, document, and prototype the database. In addition, some CASE tools provide active assistance with analyzing a database design.

This chapter provides a context for the chapters in Parts 3 and 4 and you might want to reread this chapter after completing those parts of the book. The chapters in Parts 3 and 4 provide details about the phases of database development. Chapters 5 and 6 present details of the Entity Relationship Model, data modeling practice using the Entity Relationship Model, and conversion from the Entity Relationship Model to the Relational Model. Chapter 7 presents normalization techniques for relational tables. Chapter 8 presents physical database design techniques.

REVIEW CONCEPTS

- System: related components that work together to accomplish objectives
- Information system: a system that accepts, processes, and produces data
- Waterfall model of information systems development: reference framework for activities in the information systems development process
- Spiral development methodologies, rapid application development methodologies, and Agile development methodologies to alleviate the problems in the traditional waterfall development approach
- Role of databases in information systems: provide permanent memory
- Define a common vocabulary to unify an organization
- Define business rules to support organizational processes
- Ensure data quality to improve the quality of decision making
- Evaluate investment in data quality using a cost-benefit approach
- Find an efficient implementation to ensure adequate performance while not compromising other design goals
- Conceptual data modeling to represent the information content independent of a target DBMS
- View design and view integration to manage the complexity of large data modeling efforts
- Logical database design to refine a conceptual data model to a target DBMS
- Distributed database design to determine locations of data and processing to achieve an efficient and reliable implementation
- Physical database design to achieve efficient implementations on each computer site
- Develop prototype forms and reports to cross-check among the database and applications using the database
- Soft skills for conceptual data modeling: qualitative, subjective, and people-oriented
- Hard skills for finding an efficient implementation: quantitative, objective, and data-intensive
- Computer-aided software engineering (CASE) tools to improve productivity in the database development process
- Fundamental assistance of CASE tools: drawing and documenting
- Active assistance of CASE tools: analysis and prototyping

DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.

PROBLEMS

Because of the introductory nature of this chapter, there are no problems in this chapter. Problems appear at the end of chapters in Parts 3 and 4.

DO NOT COPY, POST, OR DISTRIBUTE

Copyright (c)2024 by Sage Publications, Inc.
This work may not be reproduced or distributed in any form or by any means without express written permission of the publisher.