

CSC3400

DATABASE SYSTEMS

Faculty of Sciences

Study book

Published by

**University of Southern Queensland
Toowoomba Queensland 4350
Australia**

<http://www.usq.edu.au>

© University of Southern Queensland, 2006.1.

Copyrighted materials reproduced herein are used under the provisions of the Copyright Act 1968 as amended, or as a result of application to the copyright owner.

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means electronic, mechanical, photocopying, recording or otherwise without prior permission.

Produced by the Distance and e-Learning Centre using the *GOOD* Publishing System.

Table of contents

Page

Introduction

Notes to the textbook
The purpose of this study material

1 Introduction to databases

Learning objectives	1.1
Introduction	1.1
1.1 Traditional file-based systems	1.1
1.2 Database approach	1.2
1.2.1 What are the DBMS functions, and why are they important?	1.2
1.2.2 What are the components of the DBMS environment?	1.2
1.2.3 Roles in the database environment	1.2
1.2.4 What is a database model?	1.3
1.2.5 Advantages and disadvantages of DBMS	1.3
Review questions 1.1	1.4
Activity 1.1	1.6

2 Database environment

Learning objectives	2.1
Introduction	2.1
2.1 The three level ANSI-SPARC database architecture	2.1
2.2 Database languages	2.2
2.3 Data models and conceptual modelling	2.2
2.4 Functions of a DBMS	2.2
2.5 Components of DBMS	2.3
2.5.1 Components of database manager (DM)	2.3
2.6 Multi-user DBMS architecture	2.4
2.7 System catalog	2.4
Review questions 2.1	2.6

3 The relational model

Learning objectives	3.1
Introduction	3.1
3.1 Relational model terminology	3.1
3.1.1 Properties of relations	3.2
3.1.2 Relational keys	3.2
3.2 Relational integrity	3.3
3.3 Views	3.4
Activity 3.1	3.6

4	Relational algebra and relational calculus	
	Learning objectives	4.1
	Introduction	4.1
	4.1 Relational algebra	4.1
	4.2 The relational calculus	4.3
	4.2.1 Tuple relational calculus	4.4
	4.2.2 Domain relational calculus	4.6
	4.3 Other languages	4.6
	Summary	4.7
	Activity 4.1	4.8
5	Data manipulation	
	Learning objectives	5.1
	Introduction	5.1
	5.1 Objectives of SQL	5.1
	5.1.1 History of SQL	5.2
	5.2 Importance of SQL	5.2
	5.3 Writing SQL Commands	5.2
	5.4 Data Manipulation Language (DML)	5.3
	Activity 5.1	5.6
6	SQL: data definition	
	Learning objectives	6.1
	Introduction	6.1
	6.1 ISO SQL data types	6.1
	6.2 Integrity enhancement feature (IEF)	6.1
	6.2.1 Required data	6.2
	6.2.2 Entity integrity	6.2
	6.2.3 Referential integrity	6.2
	6.2.4 Enterprise constraints	6.3
	6.3 Data definition	6.3
	6.3.1 CREATE A DATABASE	6.4
	6.3.2 ALTER TABLE	6.4
	6.4 Views	6.5
	6.4.1 SQL – DROP VIEW	6.5
	6.4.2 View resolution	6.6
	6.4.3 Restrictions on views	6.6
	6.4.4 View updatability	6.7
	6.4.5 WITH CHECK OPTION	6.7
	6.4.6 View materialization	6.8
	6.5 Transactions	6.8
	6.5.1 Immediate and deferred integrity constraints	6.9
	6.6 Access control – authorization identifiers and ownership	6.9
	6.6.1 Privileges	6.9
	6.6.2 REVOKE	6.10
	Activity 6.1	6.11

7	Query-By-Example & commercial RDBMSs	
	Learning objectives	7.1
	Introduction	7.1
	7.1 Query-by-Example (QBE)	7.1
	7.2 Commercial DBMSs: Access and Oracle	7.3
	Review questions 7.1	7.4
8	Entity-Relationship modelling	
	Learning objectives	8.1
	Introduction	8.1
	8.1 Concepts of the ER model	8.1
	8.1.1 Entity types	8.1
	8.1.2 Relationship types	8.2
	8.1.3 Attribute	8.2
	8.1.4 Strong and weak entity type	8.3
	8.1.5 Attributes on relationship	8.3
	8.1.6 Structural constraints	8.3
	8.1.7 Problems with ER models	8.4
	8.2 Enhanced entity-relationship (EER) model	8.4
	8.2.1 Specialization / generalization	8.4
	8.2.2 Aggregation	8.6
	8.2.3 Composition	8.6
	Activity 8.2	8.7
9	Normalization	
	Learning objectives	9.1
	Introduction	9.1
	9.1 The purpose of normalization	9.1
	9.2 Data redundancy and update anomalies	9.1
	9.2.1 Two important properties of decomposition	9.2
	9.3 Functional dependency	9.2
	9.4 Identifying the primary key for a relation using functional dependencies	9.3
	9.5 Inference rules for functional dependencies	9.3
	9.5.1 Closure of FDs	9.4
	9.5.2 Closure of attribute sets	9.5
	9.5.3 Computing attribute closure of X	9.5
	9.5.4 Minimal set of functional dependencies	9.6
	9.5.5 Example of minimal set of functional dependencies	9.7
	9.6 The process of normalization	9.8
	9.6.1 Decomposition of a relation scheme	9.8
	9.7 Relationship between normal forms	9.8
	9.7.1 Unnormalized form (UNF)	9.9
	9.7.2 First normal form (1NF)	9.9
	9.7.3 Second normal form (2NF)	9.9
	9.7.4 Third normal form (3NF)	9.10
	9.7.5 Boyce-Codd normal form (BCNF)	9.13
	9.8 Correctness criterion of normalization	9.15

9.8.1	Lossless join decompositions	9.15
9.8.2	Dependency preservation	9.16
9.8.3	Qualities of decompositions	9.16
9.9	Comparison of BCNF and 3NF	9.17
9.9.1	Design goals	9.17
9.10	Fourth normal form (4NF)	9.18
9.11	Fifth normal form (5NF)	9.18
	Reference list	9.19
	Activity 9.1	9.20
	Feedback	9.21

10 Database planning, design and administration

	Learning objectives	10.1
	Introduction	10.1
10.1	Database application lifecycle	10.1
10.2	Database planning	10.2
10.3	System definition	10.2
10.3.1	User view	10.2
10.4	Requirements collection and analysis	10.3
10.5	Database design	10.3
10.5.1	Data modelling	10.3
10.5.2	Three phases of database design	10.4
10.6	DBMS selection	10.4
10.7	Application design	10.4
10.7.1	Transaction design	10.5
10.7.2	User interface design	10.5
10.8	Prototyping	10.5
10.9	Implementation	10.5
10.10	Data conversion and loading	10.6
10.11	Testing	10.6
10.12	Operational maintenance	10.6
10.13	CASE tools	10.6
10.14	Data administration (DA) and database administration (DBA)	10.7
10.15	Fact-finding techniques	10.7
10.15.1	When are fact-finding techniques used?	10.8
10.15.2	What types of facts are collected?	10.8
10.16	Using fact-finding techniques – a worked example	10.9
	Activity 10.2	10.11
	Case study 10.1	10.12

11 Methodology – conceptual, logical, and physical database design

	Learning objectives	11.1
	Introduction	11.1
11.1	Conceptual and logical database design	11.2
11.2	Methodology overview – conceptual database design	11.2
11.3	Methodology overview – logical database design for relational model	11.3
11.4	Comparison of logical and physical database design	11.4
11.5	Methodology overview – physical database design for relational databases	11.4
11.6	File organizations	11.5

Activity 11.2 11.7

12 Web Technology and DBMSs

Learning objectives 12.1
Introduction 12.1
12.1 Introduction to the Internet and the Web 12.1
12.2 Dynamic Websites and Databases 12.1

13 Solutions

Introduction 13.1
13.1 Chapter 4 Relational Algebra and Relational Calculus 13.1
13.2 Chapter 5 SQL: Data Manipulation 13.4
13.3 Chapter 6 SQL: Data Definition 13.7
13.4 Chapter 9 Database Planning, Design, and Administration 13.10
13.5 Chapter 10 Fact-Finding Techniques 13.11
13.6 Chapter 11 Entity-Relationship Modeling 13.11
13.7 Chapter 12 Enhanced Entity-Relationship Modeling 13.13
13.8 Chapter 13 Normalization 13.13

Introduction

Notes to the textbook

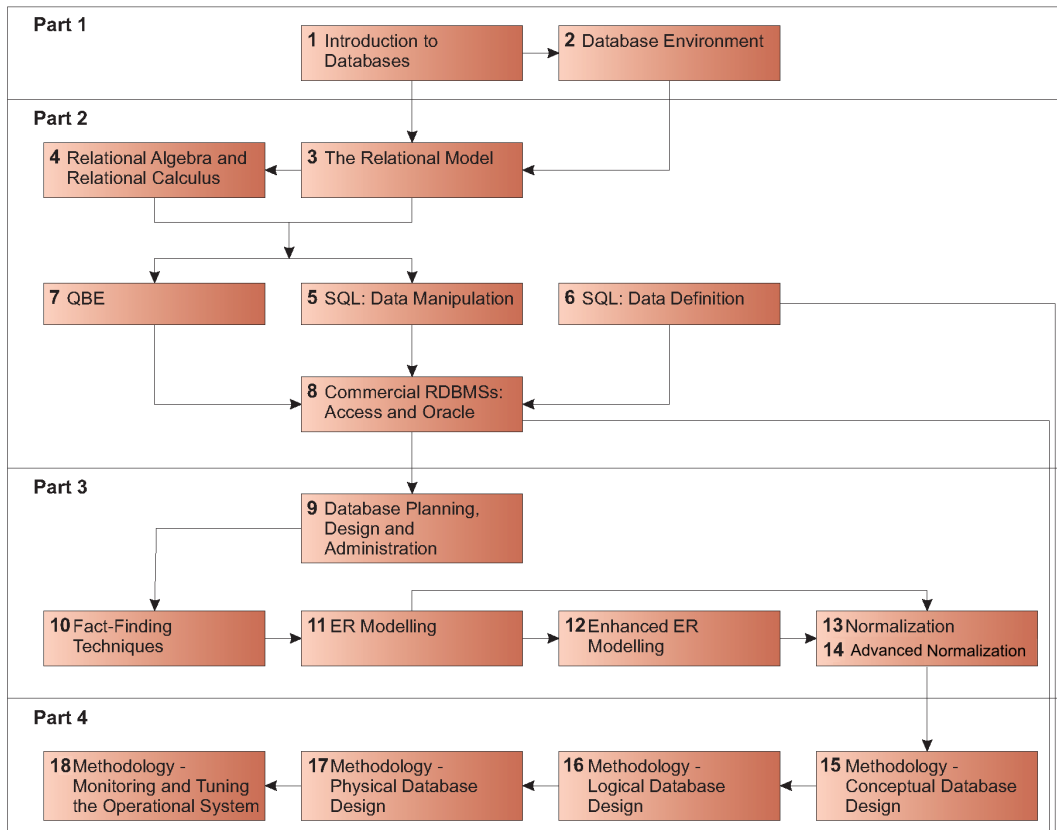
The textbook used for this course is: *Database Systems – A Practical Approach to Design, Implementation, and Management* by Thomas M. Connolly & Carolyn E. Begg, published by Addison-Wesley.

It is a good book for database design and management, providing an easy-to-use, step-by-step methodology for the three main phases of relational database design (conceptual, logical and physical). You should be able to master this methodology after studying the course.

The textbook provides ‘*Chapter Objectives*’ and a ‘*Chapter Summary*’ for each chapter. You should constantly refer to the objectives whilst reading each chapter, and check your understanding by attempting the ‘*Review Questions*’. The ‘*Chapter Summary*’ refreshes you with some important concepts covered in each chapter.

The purpose of this study material

This material will provide learning objectives for each chapter in the textbook. You must read the textbook while using the notes in the study material as a guide. The following figure shows the logical organization of the textbook and the suggested paths through it.



This course will cover the first four parts of the textbook and a brief introduction of Part 8:

1. Part 1 Background

- Chapter 1 Introduction to Databases
- Chapter 2 Database Environment

2. Part 2 The Relational Model and Languages

- Chapter 3 The Relational Model
- Chapter 4 The Relational Algebra and Relational Calculus
- Chapter 5 SQL: Data Manipulation
- Chapter 6 SQL: Data Definition
- Chapter 7 Query-by-Example
- Chapter 8 Commercial RDBMSs: Access and Oracle

3. Part 3 Database Analysis and Design Techniques

- Chapter 9 Database Planning, Design, and Administration
- Chapter 10 Fact-Finding Techniques
- Chapter 11 Entity-Relationship Modeling
- Chapter 12 Enhanced Entity-Relationship Modeling
- Chapter 13, 14 Normalization

4. Part 4 Methodology

- Chapter 15 Methodology – Conceptual Database Design
- Chapter 16 Methodology – Logical Database Design
- Chapter 17 Methodology – Physical Database Design for Relational DBMSs
- Chapter 29 Web Technology and DBMSs

As this guide is written principally to accompany the prescribed text, it follows exactly in the sequence and numbering of items. Many of the major headings, highlighted points and definitions have been replicated, with permission from Pearson Education.

Module **1**

Introduction to databases

Learning objectives

On successful completion of this module, you should be able to:

- define the terms: Database and Database Management System (DBMS)
- list common uses of database systems; characteristics of file-based systems; problems with the file-based approach; major components and personnel of the DBMS environment; and the advantages and disadvantages of DBMS.

Introduction

The material in Chapter 1 of the textbook covers a wide area, and introduces many basic concepts.

1.1 Traditional file-based systems

From *section 1.2 File-based Systems*, you should understand the limitations of the file-based approach, what constitutes a computer file system, and what some of its problems are.

Note that a computer file system stores data in independent, unrelated files on disk. The sharing, security and integrity of the data can't be enforced efficiently because of the following problems:

- Separation and isolation of data:
Each program maintains its own set of data. Users of one program may be unaware of potentially useful data held by other programs.
- Data dependence:
File structure is defined in the program code.
- Duplication of data:
The same data is held by different programs. There is wasted space and potentially different values and/or different formats for the same item.
- Incompatible file formats:
Programs are written in different languages, and so cannot easily access each others files.
- Fixed Queries/Proliferation of application programs:
Programs are written to satisfy particular functions. Any new requirement needs a new program.

1.2 Database approach

The textbook gives definitions for the database, the system catalog (data dictionary or metadata), and the Database Management System (DBMS).

The DBMS is a software system that:

- enables users to define, create, and maintain the database.
- provides controlled access to this database.

1.2.1 What are the DBMS functions, and why are they important?

Read *section 1.3.2* of the text, on how the DBMS provides the facilities for database management.

The two languages of databases are:

- Data definition language (DDL).
Permits specification of data types, structures and any data constraints. All specifications are stored in the database.
- Data manipulation language (DML).
General enquiry facility (query language) of the data.

1.2.2 What are the components of the DBMS environment?

1. Hardware
 - can range from a PC to a network of computers.
2. Software
 - DBMS, operating system, network software (if necessary) and also the application programs.
3. Data
 - data constitutes the database's central component through which information is generated
4. Procedures
 - instructions and rules that should be applied to the design and use of the database and DBMS.
 - designed to accomplish desired activities
5. People
 - perform different functions within the environment

1.2.3 Roles in the database environment

There are four distinct types of people that participate in the DBMS environment.

1. Data administrator (DA) & database administrator (DBA)
2. Database designers (Logical and Physical)
3. Application programmers
4. End users (naive and sophisticated)

1.2.4 What is a database model?

A database model is a collection of logical constructs used to represent the database's data structure as well as the data relationship(s) found within that structure.

1.2.5 Advantages and disadvantages of DBMS

Finally, from *section 1.6 Advantages and Disadvantages of DBMS*, a proper database system eliminates data dependence and provides:

- Control of data redundancy.
- Data consistency.
- More information from the same amount of data.
- Sharing of data.
- Improved data integrity.
- Improved security.
- Enforcement of standards.
- Economy of scale.
- Balanced conflicting requirements.
- Improved data accessibility and responsiveness.
- Increased productivity.
- Improved maintenance through data independence.
- Increased concurrency.
- Improved backup and recovery services.

Disadvantages of DBMS are the complexity, size and cost of DBMS etc.

Read the *Chapter Summary* at the end of the textbook to review the topics covered in the chapter.

Review questions 1.1

Attempt the *Review Questions* on page 32 of the textbook. Below you will find listed some helpful notes for each:

1.1 List four examples of database systems other than those listed in Section 1.1.

Some examples could be:

- A system that maintains component part details for a car manufacturer;
- An advertising company keeping details of all clients and adverts placed with them;
- A training company keeping course information and participants' details;
- An organization maintaining all sales order information.

1.2 Discuss each of the following terms:

- data (See Section 1.3.3)
- database (See Section 1.3.1)
- database management system (See Section 1.3.2)
- Data Independence (See Sections 1.2.2 and 1.3.1)
- Security (See Section 1.6)
- Integrity (See Section 1.6)
- Views (See Section 1.3.2)

1.3 Describe the approach taken to the handling of data in the early file-based systems. Discuss the disadvantages of this approach.

(See Section 1.2). (See Section 1.2.2).

1.4 Describe the main characteristics of the database approach and contrast it with the file-based approach.

(See Section 1.3)

1.5 Describe the five components of the DBMS environment and discuss how they relate to each other.

See Section 1.3.3.

1.6 Discuss the roles of the following personnel in the database environment:

- Data Administrator See Section 1.4.1
- Database Administrator See Section 1.4.1
- Logical Database Designer See Section 1.4.2
- Physical Database Designer See Section 1.4.2
- Application Programmer See Section 1.4.3

- End Users See Section 1.4.4

1.7 Discuss the advantages and disadvantages of database processing.

See Section 1.6

Activity 1.1

Attempt the following exercises from page 32 of the textbook:

Exercise 1.8

Interview some users of database systems. Which DBMS facilities do they find most useful and why? Which DBMS facilities do they find least useful and why? What do these users perceive to be the advantages and disadvantages of the DBMS?

Note

Select a variety of users for a particular DBMS. If the users are using different DBMSs, group the answers for the different systems, which will give an overall picture of specific systems.

Additional Activities

Exercise 1.9

Write a small program that allows entry and display of renter details including a renter number, name, address, telephone number, preferred number of rooms and maximum rent. The details should be stored in a file. Enter a few records and display the details. Now repeat this process but rather than writing a special program, use any DBMS that you have access to. What can you conclude from these two approaches?

Note

The program can be written in any appropriate programming language, such as Pascal, FORTRAN or C. It should adhere to basic software engineering principles including being well structured, modular, and suitably commented. It is important to appreciate the process involved even in developing a small program such as this. The DBMS facilities to structure, store, and retrieve data are used to the same effect. The differences in the approaches, such as the effort involved, potential for extension, and ability to share the data should be noted.

Exercise 1.10

Study the *DreamHome* case study presented in Section 10.4. In what ways would a DBMS help this organization? What data can you identify that needs to be represented in the database? What relationships exist between the data? What queries do you think are required?

Note

It may be useful to review the file-based approach and the database approach here before tackling the first part of the exercise. Careful reading and thinking about how people might use the applications should help in carrying out the rest of the exercise.

Module 2

Database environment

Learning objectives

On successful completion of this module, you should be able to:

- describe the purpose and origin of the three-level database architecture; the content of the external, conceptual and internal levels; the mappings of the external/conceptual and the conceptual/internal; the concept of logical and physical data independence
- distinguish between a Data Definition Language and a Data Manipulation Language
- classify the types of data models
- express the purpose and importance of conceptual modelling
- explain the typical functions and services a DBMS should provide; the components of a DBMS; the meaning of the client-server architecture and the advantages of this type of architecture for a DBMS
- discuss the function and importance of the system catalog.

Introduction

This module covers chapter 2 of the textbook. Chapter 2 presents the 3-level architecture for a database system and examines each level together with the different database languages that can be associated with particular levels. It introduces different data models and the importance of conceptual modeling. The chapter also describes the functions of a DBMS and examines the software components. It presents common multi-user DBMS architectures and structures.

The following sections point out important concepts from each section in Chapter 2.

2.1 The three level ANSI-SPARC database architecture

The ANSI-SPARC database architecture uses three levels of abstraction: external, conceptual, and internal.

1. The external level consists of the users' view of the database.
2. The conceptual level is the community view of the database. It specifies the information content of the entire database, independent of storage considerations.
3. The internal level is the computer's view of the database. It specifies how data is represented, how records are sequenced, what indexes and pointers exist, and what hashing scheme, if any, is used.

The **external/conceptual mapping** transforms requests and results between the external and conceptual levels. The **conceptual/internal mapping** transforms requests and results between the conceptual and internal levels.

A **database schema** is a description of the database structure. Data independence makes each level immune to changes to lower levels. **Logical data independence** refers to the immunity of external schemas to changes in the conceptual schema. **Physical data independence** refers to the immunity of the conceptual schema to changes in the internal schema.

2.2 Database languages

There are two types of database languages:

1. **Data Definition Language (DDL):**

The DDL is used to specify the database schema.

2. **Data Manipulation Language (DML):**

The DML is used to both read and update the database. The part of DML that involves data retrieval is called a **query language**.

2.3 Data models and conceptual modelling

A **data model** is a collection of concepts that can be used to describe a set of data, the operations to manipulate the data, and a set of integrity rules for the data. They fall into three broad categories:

1. **object-based** data models:

include the Entity-Relationship, semantic, functional, and object-oriented models

2. **record-based** data models:

include the relational, network, and hierarchical models

3. **physical** data models:

The first two are used to describe data at the conceptual and external levels; the latter is used to describe data at the internal level.

Conceptual modelling is the process of constructing a detailed architecture for a database that is independent of implementation details, such as the target DBMS, application programs, programming languages, or any other physical considerations. The design of the conceptual schema is critical to the overall success of the system. It is worth spending the time and energy necessary to produce the best possible conceptual design.

2.4 Functions of a DBMS

You will find 10 functions described on pages 48–53 of the textbook. You should be able to explain the purpose of the following 10 functions:

1. Data storage, retrieval and update
2. A user-accessible catalog
3. Transaction support
4. Concurrency control services
5. Recovery services
6. Authorization services
7. Support for data communication
8. Integrity services
9. Services to promote data independence
10. Utility services

2.5 Components of DBMS

- Query processor
- Database manager (DM)
- File manager
- DML preprocessor
- DDL compiler
- Catalog manager

2.5.1 Components of database manager (DM)

- Authorization control
- Command processor
- Integrity checker
- Query optimizer
- Transaction manager
- Scheduler
- Recovery manager
- Buffer manager

2.6 Multi-user DBMS architecture

The common architectures:

1. Teleprocessing
 - Traditional architecture
 - Single mainframe with a number of terminals attached.
2. File-server
 - File-server is connected to several workstations across a network.
 - Database resides on file-server.
 - DBMS and applications run on each workstation.
3. Client-server

Client-server architecture refers to the way in which software components interact. There is a client process that requires some resource, and a server that provides the resource. Typically, the client handles the user interface and the server handles the database functionality.

Advantages include:

- wider access to existing databases
- increased performance
- possible reduction in hardware costs
- reduction in communication costs
- increased consistency.

2.7 System catalog

The system catalog is one of the fundamental components of a DBMS. It is a repository of information (metadata) describing the data in the database. It contains ‘data about the data’ or meta-data. The catalog should be accessible to users.

It typically stores:

- names of authorized users
- names of data items in the database
- constraints on each data item
- data items accessible by a user and the type of access.

The system catalog is used by modules such as:

- authorization control
- integrity checker.

The **Information Resource Dictionary System** is a recent ISO standard that defines a set of access methods for a data dictionary. This allows dictionaries to be shared and transferred from one system to another.

Review questions 2.1

Attempt the *Review Questions* on page 65 of the textbook.

1. For Question 1 – see Section 2.1.5 for answer.
 2. For Question 2 – see Section 2.1 for answer.
 3. For Question 3 – see also Section 2.3 for answer.
 4. For Question 4 – see Section 2.3.4 for answer.
-

Module 3

The relational model

Learning objectives

On successful completion of this module, you should be able to:

- describe the origins of the relational model
- explain the terminology of the relational model
- represent data using tables
- interpret the connection between mathematical relations and relations in the relational model
- list the properties of database relations
- identify candidate, primary and foreign keys
- define the terms: entity integrity and referential integrity
- use views in relational systems.

Introduction

Chapter 3 of the textbook discusses the terminology and basic structural concepts of the relational data model. The terminology and the concepts are fundamental for relational database design, you must familiarize yourself with them.

3.1 Relational model terminology

Confusingly, many fundamental concepts in database theory and practice have three alternative terms associated with them. This stems from the fact that one can see a relational database from a theoretical, a practical, and an implementation point of view. Relational databases are soundly built on mathematical theory, so researchers usually adopt mathematical terms. Users of databases relate concepts to every-day terminology, while programmers use low-level programming terminology. The following table gives an overview of synonyms that are often used. As computer scientists, you should preferably use the mathematical terminology.

Mathematical terms	Practical terms	Low-level terms
relation	table	file
tuple	row	record
attribute	column	field

A mathematical **relation** is a subset of the Cartesian product of two or more sets. In database terms, a relation is any subset of the Cartesian product of the domains of the attributes. A relation is normally written as a set of n-tuples, in which each element is chosen from the appropriate domain.

Relations are physically represented as **tables**, with the rows corresponding to individual tuples and the columns to attributes. A relation is a table with columns and rows. It only applies to the logical structure of the database, not the physical structure.

- **Attribute** is a named column of a relation.
- **Domain** is a set of allowable values for one or more attributes.
- **Tuple** is a row of a relation.
- **Degree** is a number of attributes in a relation.
- **Cardinality** is a number of tuples in a relation.
- **Relational Database** is a collection of normalized relations.

The structure of the relation, with domain specifications and other constraints, is part of the **intension** of the database, while the relation with all its tuples written out represents an instance or **extension** of the database.

3.1.1 Properties of relations

Properties of database relations are:

- relation name is distinct from all other relations
- each cell of relation contains exactly one atomic (single) value
- each attribute has a distinct name
- values of an attribute are all from the same domain
- order of attributes has no significance
- each tuple is distinct; there are no duplicate tuples
- order of tuples has no significance, theoretically.

3.1.2 Relational keys

There are two kinds of keys in relations. The first are identifying keys: the primary key is the main concept, while two other keys – super key and candidate key – are related concepts. The second kind is the foreign key.

Identity Keys

Super Keys

A super key is a set of attributes whose values can be used to uniquely identify a tuple within a relation. A relation may have more than one super key, but it always has at least one: the set of all attributes that make up the relation.

Candidate Keys

A candidate key is a super key that is minimal; that is, there is no proper subset that is itself a superkey. A relation may have more than one candidate key, and the different candidate keys may have a different number of attributes. In other words, you should not interpret 'minimal' to mean the super key with the fewest attributes.

A candidate key has two properties:

- (i) in each tuple of R, the values of K uniquely identify that tuple (uniqueness)
- (ii) no proper subset of K has the uniqueness property (irreducibility).

Primary Key

The primary key of a relation is a candidate key especially selected to be the key for the relation. In other words, it is a choice, and there can be only one candidate key designated to be the primary key.

Relationship between identity keys

The relationship between keys:

Superkey \supseteq Candidate Key \supseteq Primary Key

Foreign keys

The attribute(s) within one relation that matches a candidate key of another relation. A relation may have several foreign keys, associated with different target relations.

Foreign keys allow users to link information in one relation to information in another relation. Without FKs, a database would be a collection of unrelated tables.

3.2 Relational integrity

Before discussing the various kinds of relational integrity, we briefly explain the concept of 'Null' in relational databases.

Null

- Used in the place of a value of an attribute that is currently unknown or is not applicable for this tuple.
- Deals with incomplete or exceptional data.
- Represents the absence of a value and is not the same as zero or spaces, which are values.
- Thus, Null is not a value!

Entity integrity

It is the constraint for a base relation that no attribute of a primary key can be null.

A relational database will automatically enforce entity integrity on all attributes that were designated as making up the primary key.

Referential integrity

It is the constraint between two relations.

A value of a foreign key in a relation must match a candidate key value of the base relation, or the foreign key attribute should be null.

A relational database will usually automatically enforce referential integrity on all attributes designated as being part of some foreign key in the relation.

Enterprise constraints

Some constraints for additional rules specified by users or database administrators.

A relational database will not automatically enforce such constraints. Instead, they must be specified either in a relational query language, or through special purpose application code written by programmers within the enterprise.

3.3 Views

A view in the relational model is a virtual relation. The view provides security and allows the designer to customize a user's model. Views are created dynamically for users. Not all views are updateable.

Base relation

A named relation, corresponding to an entity in a conceptual schema, whose tuples are physically stored in database.

View

The dynamic result of one or more relational operations operating on the base relations to produce another relation.

- A view is a virtual relation that does not actually exist in the database but is produced upon request, at time of request.
- Contents of a view are defined as a query on one or more base relations.
- Views are dynamic, meaning that changes made to base relations that affect view attributes are immediately reflected in the view.

Purpose of views

Provides a powerful and flexible security mechanism by hiding parts of the database from certain users.

Permits users to access data in a customized way, so that same data can be seen by different users in different ways, at same time. It can simplify complex operations on base relations.

Updating views

All updates to a base relation should be immediately reflected in all views that reference that base relation.

If a view is updated, the underlying base relation should reflect the change. However, there are restrictions on types of modifications that can be made through views:

- updates are allowed if the query involves a single base relation and contains a candidate key of base relation
- updates are not allowed involving multiple base relations
- updates are not allowed involving aggregation or grouping operations
- classes of views are defined as theoretically not updateable, theoretically updateable and partially updateable.

Activity 3.1

- Attempt the *Review Questions* on page 86 of the textbook.
 - Attempt *exercises* 3.8 and 3.9 on page 87 of the textbook.
-

Module 4

Relational algebra and relational calculus

Learning objectives

On successful completion of this module, you should be able to:

- define the term ‘relational completeness’
- write queries in relational algebra using five basic operations: Selection; Projection; Cartesian Product; Union; Set Difference
- create various queries in the tuple relational calculus
- formulate queries in domain relational calculus
- describe the categories of relational Data Manipulation Languages (DMLs).

Introduction

This module covers chapter 4 of the textbook.

4.1 Relational algebra

Relational algebra operations work on one or more relations to define another relation without changing the original relations.

Thus, both operands and results are relations, so output from one operation can become input to another operation.

This allows expressions to be nested, just as in arithmetic. This property is called closure.

There are five basic operations in relational algebra:

1. Selection
2. Projection
3. Cartesian Product
4. Union
5. Set Difference

These perform most of the data retrieval operations needed.

There are also Join, Intersection, and Division operations that can be expressed in terms of five basic operations.

1. Selection operation: $\sigma_{\text{predicate}}(R)$

Selection operation works on a single relation R and defines a relation that contains only those tuples (rows) of R that satisfy the specified condition (predicate).

Example – Selection

Select all staff whose salary is greater than 10000.

$\sigma_{\text{salary} > 10000}(\text{Staff})$

2. Projection: $\Pi_{\text{col1}, \dots, \text{coln}}(R)$

Projection operation works on a single relation R and defines a relation that contains a vertical subset of R , extracting the values of specified attributes and eliminating duplicates.

Notice that the result of the projection may have less tuples of the base relation R if the specified attributes are not containing the key of R .

Example – Projection

Produce a list of names for all staff, showing only the FName, LName, details.

$\Pi_{\text{fname}, \text{lname}}(\text{Staff})$

If more than one staff have the same first name and last name then the duplicated tuples will be deleted.

3. Cartesian product: $R \times S$

The Cartesian product operation defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S .

The Cartesian product and Selection can be reduced to a single operation called a **join**.

4. Union $R \cup S$

Union of two relations R and S with I and J tuples, respectively, is obtained by concatenating them into one relation with a maximum of $(I + J)$ tuples, duplicate tuples being eliminated. R and S must be union-compatible.

5. Set difference $R - S$

The set difference operation defines a relation consisting of the tuples that are in relation R , but not in S . R and S must be union-compatible.

6. Join operations

Join is a special case of Cartesian product. It is a derivative of the Cartesian product. It is equivalent to performing a selection, using the join predicate as the selection formula, over the Cartesian product of the two operand relations.

Join is one of the most difficult operations to implement efficiently in a relational DBMS and one of the reasons why relational systems have intrinsic performance problems.

There are various forms of join operation

- theta-join
- equi-join (a particular type of theta-join)
- natural join
- outer join
- semi-join

Table 4.1 on page 100 of the textbook is a summary of all relational algebra operations. You should become familiar with all notations and semantics.

7. Grouping and Counting

Section 4.1.5 in the text book discusses aggregation and grouping operations in the Relational Algebra. As these operations were added to the language much later than the others, and since the operations are non-standard, you should not study this section.

Hence, we will assume that grouping is not possible in the Algebra, and that almost no aggregation is possible either. A noteworthy exception to this is Counting. A limited degree of counting is possible by using a so-called self-join. This is a join of a table with itself. Performing the correct selection condition of a self-join allows one to find tuples of the same relation that agree on some attributes but disagree on others. Hence, given a relation Sees(person, movie), it is possible to express the query “give all persons who have seen at least two movies” in the Relational Algebra without using an aggregation function. The solution goes like this: $\Pi \text{ person } (\sigma \text{ person1}=\text{person2} \text{ and movie1} \neq \text{movie2} (\text{Sees} \times \text{Sees}))$.

4.2 The relational calculus

Relational calculus query specifies **what** is to be retrieved rather than **how** to retrieve it. There is no description of how to evaluate a query. It is based on a branch of symbolic logic called **predicate calculus**.

When applied to databases, relational calculus is in two forms:

1. tuple-oriented and
2. domain-oriented.

Relational calculus is in first-order logic or predicate calculus, a **predicate** is a truth-valued function with arguments.

When we substitute values for the arguments, the function yields an expression, called a **proposition**, which can be either true or false.

If a predicate contains a variable, as in ‘x is a member of staff’, there must be a range for x. When we substitute some values of this range for x, the proposition may be true; for other values, it may be false.

If P is a predicate, then we write the set of all x such that P is true for x, as

$$\{x \mid P(x)\}$$

Predicates can be connected using \wedge (AND), \vee (OR) and \sim (NOT)

4.2.1 Tuple relational calculus

We are interested in finding tuples for which a predicate is true. The calculus is based on use of tuple variables.

A tuple variable is a variable that ‘ranges over’ a named relation: that is, a variable whose only permitted values are tuples of the relation.

To specify the range of a tuple variable S as the Staff relation.

$$\text{Staff}(S)$$

To find the set of all tuples S such that P(S) is true.

$$\{S \mid P(S)\}$$

Example of tuple relational calculus:

To find **all** information (i.e., Sno, FName, LName, Address, Tel_No, Position, Sex, DOB, Salary, NIN, and Bno) of all staff earning more than £10,000, we write

$$\{S \mid \text{Staff}(S) \wedge S.\text{salary} > 10000\}$$

S.salary means the value of the Salary attribute for the tuple S.

To find a **particular attribute**, such as Salary, we write.

$$\{S.\text{salary} \mid \text{Staff}(S) \wedge S.\text{salary} > 10000\}$$

We can use two **quantifiers** to tell how many instances the predicate applies to.

- Existential quantifier \exists (‘there exists’)
- Universal quantifier \forall (‘for all’)

Existential quantifier used in formulae that must be true for at least one instance, such as:

$$\exists B (B.\text{Bno} = S.\text{Bno} \wedge B.\text{City} = \text{‘London’})$$

Means ‘There exists a Branch tuple that has the same Bno as the Bno of the current Staff tuple, S, and is located in London’.

Universal quantifier is used in statements about every instance, such as:

$\forall B (B.City \neq \text{‘Paris’})$

Means ‘For all Branch tuples, the address is not in Paris’.

Can also use $\sim \exists B (B.City = \text{‘Paris’})$ which means ‘There are no branches with an address in Paris’.

De Morgan’s laws are very useful for the existential and universal quantifiers.

$(\exists \mathbf{x})(\mathbf{F}(\mathbf{x})) \equiv (\sim \forall \mathbf{x})(\sim \mathbf{F}(\mathbf{x}))$

$(\forall \mathbf{x})(\mathbf{F}(\mathbf{x})) \equiv (\sim \exists \mathbf{x})(\sim \mathbf{F}(\mathbf{x}))$

$(\exists \mathbf{x})(\mathbf{F}_1(\mathbf{x}) \wedge \mathbf{F}_2(\mathbf{x})) \equiv (\sim \forall \mathbf{x})(\sim \mathbf{F}_1(\mathbf{x}) / \sim \mathbf{F}_2(\mathbf{x}))$

$(\forall \mathbf{x})(\mathbf{F}_1(\mathbf{x}) \wedge \mathbf{F}_2(\mathbf{x})) \equiv \sim (\exists \mathbf{x})(\sim \mathbf{F}_1(\mathbf{x}) / \sim \mathbf{F}_2(\mathbf{x}))$

Tuple variables are called **free** variables. If qualified by \forall or \exists they are called **bound** variables.

Formulae should be unambiguous and make sense.

You can find the definition for a (well-formed) formula in predicate calculus on page 103 of the textbook.

Look at Example 4.14 for tuple relational calculus from page 105 to 106 to understand how to make queries using tuple relational calculus expressions.

Especially, look at examples (c) & (d) on page 106 to understand the applications of De Morgan’s laws.

The following two expressions are equivalent for the query (c):

$\{S.fname, S.lname \mid Staff(S) \wedge (\sim (\exists P) (PropertyForRent(p) \wedge (S.sno = P.sno)))\}$

or

$\{S.fname, S.lname \mid Staff(S) \wedge \forall P (\sim (PropertyForRent(p) \vee (S.sno = P.sno)))\}$

to list the names of staff who currently do not manage any properties.

This example uses De Morgan’s laws $(\sim (\exists \mathbf{x})(\mathbf{F}_1(\mathbf{x}) \wedge \mathbf{F}_2(\mathbf{x})) \equiv (\forall \mathbf{x})(\sim \mathbf{F}_1(\mathbf{x}) \vee \sim \mathbf{F}_2(\mathbf{x}))$

Expressions can generate an infinite set. This is avoided by using range variables defined by a RANGE statement.

However, you can also define a range explicitly within a formula. For example, $\{S \mid \sim (S \in \text{Staff})\}$ means the set of tuples that are not in the Staff relation (Expression is unsafe).

To avoid this, add a restriction that all values in the result must be values in the domain of the formula.

4.2.2 Domain relational calculus

Uses variables that take values from domains instead of tuples of relations.

If $P(d_1, d_2, \dots, d_n)$ stands for a predicate with variables d_1, d_2, \dots, d_n , then $\{d_1, d_2, \dots, d_n \mid P(d_1, d_2, \dots, d_n)\}$ means the set of all domain variables d_1, d_2, \dots, d_n for which the predicate, or formula, $P(d_1, d_2, \dots, d_n)$ is true.

We often test for a membership condition to determine whether values belong to a relation.

The expression $R(x, y)$ evaluates to true **if and only if** there is a tuple in relation R with values x, y for its two attributes.

Example – domain relational calculus

(c) List the names of staff who do not manage any properties for rent.

$$\{fname, lname \mid \exists sno \text{ Staff}(sno, fname, lname, position, sex, DOB, sa, bN) \wedge (\sim (\exists sNo1)(\text{Property_for_Rent}(pno, st, city, pc, tup.rms, rmi.oN, sNo1, bN1) \wedge P.city =))\}$$

Each attribute has a (variable) name. Condition Staff (lname, position, salary) ensures domain variables are restricted to attributes of the same tuple.

$$fname, lname \mid \exists position, \exists salary$$

$$(\text{Staff}(lname, position, salary) \wedge position = \text{'Manager'} \wedge salary > 25000)\}$$

When domain relational calculus is restricted to safe expressions, it is equivalent to tuple relational calculus restricted to safe expressions, which is equivalent to relational algebra.

This means that every relational algebra expression has an equivalent relational calculus expression, and vice versa.

4.3 Other languages

Transform-oriented languages are non-procedural languages that use relations to transform input data into required outputs (e.g. SQL).

Graphical languages provide the user with a picture or illustration of the structure of the relation. The user fills in an example of what is wanted and the system returns the required data in that format (e.g. QBE).

Fourth-generation languages (4GLs) can create a complete customized application using a limited set of commands in a user-friendly, often menu-driven environment.

Some systems accept a form of **natural language**, sometimes called a **fifth-generation language** (5GL), although this development is still in its infancy.

Summary

Relational data manipulation languages are sometimes classified as **procedural or non-procedural, transform-oriented, graphical, fourth-generation, or fifth-generation**.

You should understand the different query languages and the classification of the language, such as:

- **Relational algebra** is a formal procedural language. Its operations include selection, projection, Cartesian product, union, intersection, set difference, division, and several types of joins.
- **Relational calculus** is a formal non-procedural language that uses predicates. It includes:
 - **Tuple relational calculus**
 - **Domain relational calculus**.
- Relational algebra is logically equivalent to a safe subset of relational calculus (and vice versa).

Activity 4.1

- Read the *Chapter Summary* and answer the *Review Questions: 4.1– 4.7* on page 110 of the textbook.
 - Attempt *exercises 4.8, 4.9, 4.12* from page 111 of the textbook.
-

Module 5

Data manipulation

Learning objectives

On successful completion of this module, you should be able to:

- explain the purpose and importance of Structured Query Language (SQL)
- describe the history and development of SQL
- use SQL commands
- retrieve data from a database using the SELECT statement
- build SQL statements that: use compound WHERE conditions; sort query results using ORDER BY; use the aggregate functions of SQL; group data using GROUP BY and HAVING; use subqueries; join tables together; perform set operations (UNION, INTERSECT, EXCEPT)
- perform database updates using INSERT, UPDATE, and DELETE.

Introduction

This module covers chapter 5 of the textbook – the Data Manipulation function of SQL. From section 5.1 of the textbook, read about the objectives, history, and the importance of SQL.

5.1 Objectives of SQL

SQL is a transform-oriented language with two major components:

1. A DDL for defining the database structure.
2. A DML for retrieving and updating data.

SQL does not contain flow control commands. These must be implemented using a programming or job-control language, or interactively by the decisions of the user.

SQL is relatively easy to learn:

1. It is a non-procedural language – you specify **what** information you require, rather than **how** to get it.
2. It is essentially free-format.

SQL using standard English words:

```
CREATE TABLE staff(sno VARCHAR(5),
                    lname VARCHAR(15),
                    salary DECIMAL(7,2));
INSERT INTO staff
VALUES ('SG16', 'Brown', 8300);
SELECT sno, lname, salary
FROM staff
WHERE salary > 10000;
```

SQL can be used by a range of users including DBAs, management, application programmers, and other types of end users.

An ISO standard now exists for SQL, making it both the formal and *de facto* standard language for relational databases.

5.1.1 History of SQL

Through reading of Section 5.1.2 in the textbook you should become familiar with the development of SQL and recent versions of SQL.

5.2 Importance of SQL

- SQL is the strategic choice of many large and influential organizations (e.g. X/OPEN).
- SQL has become a Federal Information Processing Standard (FIPS) to which conformance is required for all sales of databases to the American government.
- The SQL Access Group defined enhancements that would support interoperability across disparate systems.
- SQL is used in other standards and even influences the development of other standards as a definitional tool. Examples include:
 - ISO's Information Resource Directory System (IRDS) Standard.
 - Remote Data Access (RDA) Standard.

5.3 Writing SQL Commands

An SQL statement consists of **reserved words** and **user-defined words**.

- Reserved words are a fixed part of SQL and must be spelled exactly as required and cannot be split across lines.

- User-defined words are made up by the user and represent names of various database objects such as relations, columns, and views.

Most components of an SQL statement are **case insensitive**, except for literal character data.

Please read the textbook Section 5.2 for conventions of SQL statements.

5.4 Data Manipulation Language (DML)

SQL DML statements include:

- SELECT
- INSERT
- UPDATE
- DELETE

The **SELECT** statement is used to express a query, and is the most important statement in the language. It combines the three fundamental relational operations of selection, projection, and join.

The output of a SELECT statement is a table consisting of one or more columns and zero or more rows.

SELECT Statement

```
SELECT    [DISTINCT | ALL]
          { * | [column_expression [AS new_name]] [, ...] }
FROM      table_name [alias] [, ...]
[WHERE    condition]
[GROUP BY column_list]          [HAVING condition]
[ORDER BY column_list]
```

SELECT Specifies which columns are to appear in the output.

FROM Specifies table(s) to be used.

WHERE Filters rows.

The **WHERE** clause selects rows to be included in the result table by applying a search condition to the rows of the named table(s).

GROUP BY Forms groups of rows with the same column value.

HAVING Filters groups subject to some condition.

The **HAVING** clause acts as a **WHERE** clause for groups, restricting the groups that appear in the final result table. However, unlike the **WHERE** clause, the **HAVING** clause can include aggregate functions.

ORDER BY Specifies the order of the output.

The **ORDER BY** clause allows the result table to be sorted on the values in one or more columns. Each column can be sorted in ascending or descending order. If specified, the **ORDER BY** clause must be the last clause in the **SELECT** statement.

The **GROUP BY** clause allows summary information to be included in the result table. Rows that have the same value for one or more columns can be grouped together and treated as a unit for using the aggregate functions. In this case, the aggregate functions take each group as an argument and compute a single value for each group as the result.

Note:

- Order of the clauses cannot be changed.
- **SELECT** and **FROM** are mandatory in a query.

Read the examples in the textbook of various queries:

- **SELECT . . . FROM**
 - all columns, all rows
 - using * as an abbreviation for ‘all columns’
 - specific Columns, all Rows
 - use of **DISTINCT**
 - calculated fields
- **SELECT . . . FROM . . . WHERE**
 - comparison search condition
 - compound comparison search condition
 - range search condition
 - set membership
 - pattern matching
 - **NULL** search condition
- **SELECT . . . FROM . . . ORDER BY**
 - single column ordering
 - multiple column ordering
- **SELECT Statement – Aggregates**

SQL supports the following five aggregate functions that take an entire column as an argument and compute a single value as the result. It is illegal to mix aggregate functions with column names in a **SELECT** clause, unless the **GROUP BY** clause is used.

- **COUNT** – returns number of values in a specified column.
- **SUM** – returns sum of values in a specified column.

- AVG – returns average of values in a specified column.
- MIN – returns smallest value in a specified column.
- MAX – returns largest value in a specified column.

Complex queries:

- Subquery with equality
- Subquery with aggregate
- Nested subquery: use of IN, ANY and ALL

A subquery is a complete SELECT statement embedded in another query. Subqueries may appear within the WHERE or HAVING clauses. Conceptually, a subquery produces a temporary table whose contents can be accessed by the outer query. A subquery can be embedded in another subquery.

If the columns of the result table come from more than one table, a join must be used, by specifying more than one table in the FROM clause and typically including a WHERE clause to specify the join column(s). The ISO standard allows outer joins to be defined. It also allows the set operations of union, intersection, and difference to be used with the UNION, INTERSECT, and EXCEPT set operations.

Multi-Table Queries

- Simple join
- Alternative JOIN constructs
- Sorting a join
- Multiple grouping columns
- Computing a join
- Outer joins
- Left outer join
- Right outer join
- Full outer join
- EXISTS and NOT EXISTS

As well as SELECT, the SQL DML includes the INSERT statement to insert a single row of data into a named table or to insert an arbitrary number of rows from another table using a subselect; the UPDATE statement to update one or more values in a specified column of a named table; the DELETE statement to delete one or more rows from a named table.

The ISO standard provides six base data types: character, bit, exact numeric, approximate numeric, datetime, and interval.

Activity 5.1

Attempt questions 5.7–5.29 on page 155 of the textbook.

Module 6

SQL: data definition

Learning objectives

On successful completion of this module, you should be able to:

- apply the data types supported by SQL standard
- describe the purpose of the integrity enhancement feature of SQL
- define integrity constraints using SQL
- use the integrity enhancement feature in CREATE and ALTER TABLE statements
- explain the purpose of views
- create and delete views using SQL
- explain: how the DBMS performs operations on views; under what conditions views are updateable; advantages and disadvantages of views; how the ISO transaction model works
- apply the GRANT and REVOKE statements as a level of security.

Introduction

This module covers chapter 6 of the textbook.

6.1 ISO SQL data types

Section 6.1 described SQL identifiers, SQL scalar data types and operators defined in the ISO standard. You will use them to define database schemas and domains of the attributes in SQL.

6.2 Integrity enhancement feature (IEF)

The ISO SQL standard provides clauses in the CREATE and ALTER TABLE statements to define integrity constraints that handle five types of integrity constraints:

1. Required data.
2. Domain constraints.
3. Entity integrity.
4. Referential integrity.
5. Enterprise constraints.

6.2.1 Required data

Required data can be specified using NOT NULL.

```
position VARCHAR(10) NOT NULL
```

Domain constraints

Domain constraints can be specified using the CHECK clause or by defining domains using the CREATE DOMAIN statement.

(a) CHECK

```
sex CHAR NOT NULL  
CHECK (sex IN ('M', 'F'))
```

(b) CREATE DOMAIN

```
CREATE DOMAIN DomainName [AS] dataType  
[DEFAULT defaultOption]  
[CHECK (searchCondition)]
```

See example from the textbook for the CHECK clause.

6.2.2 Entity integrity

Entity integrity is a constraint for the relationship of columns in one table. It includes primary and candidate keys.

Primary keys should be defined using the PRIMARY KEY clause, and alternate keys using the combination of NOT NULL and UNIQUE.

The Primary key of a table must contain a unique, non-null value for each row.

There can only be one PRIMARY KEY (PK) clause per table. It is still possible to ensure uniqueness for alternate keys using UNIQUE:

```
UNIQUE(telNo)
```

6.2.3 Referential integrity

Referential integrity is a constraint between two tables.

A foreign key (FK) is a column or a set of columns that links each row in the child table containing FK to rows of the parent table containing matching PK.

Referential integrity means that, if FK contains a value, that value must refer to an existing row in the parent table.

Foreign keys should be defined using the FOREIGN KEY clause, and update and delete rules using the subclauses ON UPDATE and ON DELETE.

ISO standards supports the definition of FKs with the FOREIGN KEY clause in CREATE and ALTER TABLE:

```
FOREIGN KEY(branchNo) REFERENCES Branch
```

Any INSERT/UPDATE that attempts to create FK value in a child table without matching a candidate key value in the parent is rejected.

Action taken that attempts to update/delete a candidate key value in the parent table with matching rows in a child is dependent on referential action specified using ON UPDATE and ON DELETE subclauses:

- CASCADE
- SET NULL
- SET DEFAULT
- NO ACTION

You should become familiar with these four options for the update/delete action, and the reasons why they are necessary.

6.2.4 Enterprise constraints

Enterprise constraints are constraints involving more than two tables.

They can be defined:

- using CHECK/UNIQUE in CREATE and ALTER TABLE
- or by using:

```
CREATE ASSERTION AssertionName CHECK (searchCondition)
```

(which is very similar to the CHECK clause.)

6.3 Data definition

SQL DDL allows database objects such as schemas, domains, tables, views, and indexes to be created and destroyed.

The main SQL DDL statements are:

CREATE SCHEMA DROP SCHEMA

CREATE/ALTER DROP DOMAIN
DOMAIN

CREATE/ALTER TABLE DROP TABLE

CREATE VIEW DROP VIEW

Many DBMSs also provide:

CREATE INDEX DROP INDEX

6.3.1 CREATE A DATABASE

You should be aware of the relationship of a database within the DBMS environment. The following are basic definitions related to a database:

- Relations and other database objects exist in an **environment**.
- Each environment contains one or more **catalogs**, and each catalog consists of a set of schemas.
- A Schema is a named collection of related database objects.
- Objects in a schema can be tables, views, domains, assertions, collations, translations, and character sets. All have the same owner.

Read the textbook to see how to create a schema/table/index/view in SQL.

With the CREATE TABLE clause, you should be able to correctly define the data types of attributes, the primary key, foreign keys and the constraints for the following requirements.

- Creates a table with one or more columns of the specified **dataType**.
- With NOT NULL, the system rejects any attempt to insert a null in the column.
- Can specify a DEFAULT value for the column.
- Primary keys should always be specified as NOT NULL.
- FOREIGN KEY clause specifies FK along with the referential action

6.3.2 ALTER TABLE

With the ALTER TABLE clause you can change or delete the tables.

- Add a new column to a table.
- Drop a column from a table.
- Add a new table constraint.

- Drop a table constraint.
- Set a default for a column.
- Drop a default for a column.

Refer to the textbook for examples.

6.4 Views

A view is a virtual relation that does not necessarily actually exist in the database but is produced upon request, at the time of request.

Contents of a view are defined as a query on one or more base relations. A view is created using the `CREATE VIEW` statement by specifying a defining query. It is not a physically stored table, but is recreated each time it is referenced.

Views can be used to simplify the structure of the database and make queries easier to write. They can also be used to protect certain columns and/or rows from unauthorized access. Not all views are updateable.

Using the SQL `CREATE VIEW` clause you can create various views, the textbook has given a number of examples:

- Example 6.3 – Create Horizontal View
- Example 6.4 – Create Vertical View
- Example 6.5 – Grouped and Joined Views
- Example 6.3 – Grouped and Joined Views

6.4.1 SQL – DROP VIEW

```
DROP VIEW ViewName [RESTRICT | CASCADE]
```

Causes a definition of the view to be deleted from the database.

For example:

```
DROP VIEW Manager3Staff;
```

With the `CASCADE` option, all related dependent objects are deleted; i.e. any views defined on a view being dropped.

With `RESTRICT` (default), if any other objects depend for their existence on the continued existence of a view being dropped, the command is rejected.

6.4.2 View resolution

Views can be used as normal tables for queries. The system process the queries on the base tables, this is doing the view resolution.

With view resolution, any operations on a view are automatically translated into operations on the relations from which it is derived.

For example:

The following is the definition of view StaffPropCnt:

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
AS SELECT s.branchNo, s.staffNo, COUNT(*)
   FROM Staff s, PropertyForRent p
   WHERE s.staffNo = p.staffNo
   GROUP BY s.branchNo, s.staffNo;
```

If we want to count the number of properties managed by each member at branch B003, we can make a query on view StaffPropCnt:

```
SELECT staffNo, cnt
FROM StaffPropCnt
WHERE branchNo = 'B003'
ORDER BY staffNo;
```

The final merged query is the following query for execution on the database to produce the result:

```
SELECT s.staffNo, COUNT(*)
FROM staff s, PropertyForRent p
WHERE s.staffNo = p.staffNo AND
      branchNo = 'B003'
GROUP BY s.branchNo, s.staffNo
ORDER BY s.staffNo;
```

6.4.3 Restrictions on views

Since a view is not a physically existing table, SQL imposes several restrictions on creation and use of views.

- (a) If column in view is based on an aggregate function:
 - Column may appear only in SELECT and ORDER BY clauses of queries that access view.
 - Column may not be used in WHERE nor be an argument to an aggregate function in any query based on view.
- (b) Grouped view may never be joined with a base table or a view.

For example, StaffPropCnt view is a grouped view, so any attempt to join this view with another table or view fails.

6.4.4 View updatability

All updates to the base table are reflected in all views that encompass the base table. However, the converse may be not true.

Read the textbook to see the problems caused by an insertion operation to a view.

ISO specifies the views that must be updateable in a system that conforms to the standard.

A view is updateable if:

- DISTINCT is not specified.
- Every element in SELECT list of the defining query is a column name and no column appears more than once.
- FROM clause specifies only one table, excluding any views based on a join, union, intersection or difference.
- No nested SELECT referencing an outer table.
- No GROUP BY or HAVING clause.

Also, every row added through view must not violate integrity constraints of the base table.

For view to be updateable, DBMS must be able to trace any row or column back to its row or column in the source table.

6.4.5 WITH CHECK OPTION

- Rows exist in a view because they satisfy the WHERE condition of a defining query.
- If a row changes and no longer satisfies the condition, it disappears from the view.
- New rows appear within view when insert/update on view cause them to satisfy the WHERE condition.
- Rows that enter or leave a view are called **migrating rows**.
- WITH CHECK OPTION prohibits a row migrating out of the view.
- LOCAL/CASCADED apply to view hierarchies.
- With LOCAL, any row insert/update on view and any view directly or indirectly defined on this view must not cause the row to disappear from view unless it also disappears from the derived view/table.
- With CASCADED (default), any row insert/update on this view and on any view directly or indirectly defined on this view must not cause the row to disappear from the view.

Read Example 6.6 WITH CHECK OPTION from the textbook for the update on views and how to prevent anomalies.

6.4.6 View materialization

The view materialization is used to solve the problem of the slow proves of view resolution mechanism. It stores the view as a temporary table when the view is first queried. Thereafter, queries based on the materialized view can be faster than re-computing the view each time. It is particularly useful if the view is accessed frequently.

The difficulty is to maintain the currency of a view while base tables(s) are being updated.

The view maintenance aims to apply only those changes necessary to keep a view current.

6.5 Transactions

A **transaction** is logical unit of work with one or more SQL statements guaranteed to be atomic with respect to recovery.

SQL defines a transaction model based on COMMIT and ROLLBACK.

The COMMIT statement signals successful completion of a transaction and all changes to the database are made permanent. The ROLLBACK statement signals that the transaction should be aborted and all changes to the database are undone.

An SQL transaction automatically begins with a **transaction-initiating** SQL statement (e.g., SELECT, INSERT).

Changes made by a transaction are not visible to other concurrently executing transactions until a transaction completes.

Transactions can complete in one of four ways:

1. COMMIT ends transaction successfully, making changes permanent.
2. ROLLBACK aborts transaction, backing out any changes made by transaction.
3. For programmatic SQL, successful program termination ends final transaction successfully, even if COMMIT has not been executed.
4. For programmatic SQL, abnormal program end aborts transaction.

A new transaction starts with a next transaction-initiating statement.

SQL transactions cannot be nested.

6.5.1 Immediate and deferred integrity constraints

Do not always want constraints to be checked immediately, but instead at transaction commit.

Constraints may be defined as `INITIALLY IMMEDIATE` or `INITIALLY DEFERRED`, indicating the mode a constraint assumes at the start of each transaction.

In the former case, it is also possible to specify whether the mode can be changed subsequently using the qualifier `[NOT] DEFERRABLE`.

The default mode is `INITIALLY IMMEDIATE`.

6.6 Access control – authorization identifiers and ownership

SQL access control is built around the concepts of authorization identifiers, ownership, and privileges.

- Authorization identifiers are assigned to database users by the DBA and identify a user. The authorization identifier is a normal SQL identifier used to establish the identity of a user. Usually it has an associated password. It is used to determine which objects the user may reference and what operations may be performed on those objects.
- Each object that is created in SQL has an owner. The owner can pass privileges on to other users using the `GRANT` statement and can revoke the privileges passed on using the `REVOKE` statement.
- The privileges that can be passed on are `USAGE`, `SELECT`, `DELETE`, `INSERT`, `UPDATE`, and `REFERENCES`; the latter three can be restricted to specific columns. A user can allow a receiving user to pass privileges on using the `WITH GRANT OPTION` clause and can revoke it using the `GRANT OPTION FOR` clause.

6.6.1 Privileges

Actions the user is permitted to carry out on a given base table or view:

<code>SELECT</code>	Retrieve data from a table.
<code>INSERT</code>	Insert new rows into a table.
<code>UPDATE</code>	Modify rows of data in a table.
<code>DELETE</code>	Delete rows of data from a table.
<code>REFERENCES</code>	Reference columns of named table in integrity constraints.

USAGE Use domains, collations, character sets, and translations.

It is possible to restrict INSERT/UPDATE/REFERENCES to named columns.

The owner of table must grant other users the necessary privileges using the GRANT statement.

To create a view, users must have the SELECT privilege on all tables that make up the view, and REFERENCES privilege on the named columns.

Refer to the textbook for the format and examples of the GRANT statement.

6.6.2 REVOKE

REVOKE takes away privileges granted with GRANT.

```
REVOKE [GRANT OPTION FOR]
       {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
FROM {AuthorizationIdList | PUBLIC}
     [RESTRICT | CASCADE]
```

ALL PRIVILEGES refers to all privileges granted to a user.

GRANT OPTION FOR allows privileges passed on via WITH GRANT OPTION of GRANT to be revoked separately from the privileges themselves.

REVOKE fails if it results in an abandoned object, such as a view, unless the CASCADE keyword has been specified.

Privileges granted to this user by other users are not affected.

Activity 6.1

- Attempt the *Review Questions* on page 194 of the textbook.
 - Attempt *exercises* 6.7, 6.8, 6.14, 6.15, 6.19 on pages 194–195 of the textbook.
-

Module 7

Query-By-Example & commercial RDBMSs

Learning objectives

On successful completion of this module, you should be able to:

- apply the main features of Query-by-Example (QBE)
- describe the types of queries provided by the Microsoft Access DBMS QBE facility
- use QBE to build queries to select fields and records
- use QBE to target single or multiple tables
- perform calculations using QBE
- use advanced QBE facilities including parameter, find duplicates, find unmatched, crosstab and autolookup queries
- use QBE action queries to change the content of tables
- list two commercial RDBMSs.

Introduction

This module covers chapter 7 and chapter 8 of the textbook. Chapter 7 of the textbook introduces the QBE and illustrates the power of QBE by a commercial RDBMS Microsoft Access.

7.1 Query-by-Example (QBE)

Query-by-Example (QBE) is a visual approach for accessing information in a database through use of query templates. Example values are entered into a template to represent what the access to a database is to achieve, such as the answer to a query.

When the user constructs a QBE – in background, DBMS creates an equivalent SQL statement.

This allows the user to:

- Ask questions about data in one or more tables.
- Specify the fields we want in the answer.
- Select records according to some criteria.
- Perform calculations on the data in tables.
- Insert and delete records.
- Modify values of fields.
- Create new fields and tables.

A good example of a QBE can be shown by Microsoft Access DBMS.

The textbook shows a graphical representation of using QBE on MS Access to introduce the range of facilities available using QBE to students with some prior knowledge of the DBMS software.

Read

- Section 7.1 ‘Introduction to Microsoft Access Queries’
- Section 7.2 ‘Building Select Queries Using QBE’
- Section 7.3 ‘Advanced Queries’
- Section 7.4 ‘Action Queries’.

These sections are quite easy to follow and understand. However, to really appreciate the range of possible operations available using the QBE facility requires ‘hands-on’ experience in a laboratory environment. Therefore, it is important that you use a DBMS that provides a QBE or equivalent facility.

From studying the textbook you should be able to build various types of the queries, such as:

- Building Select Queries Using QBE – Specifying Criteria
- Creating Multi-table Queries
- Calculating Totals
- Using Advanced Queries – Parameter Query
- Using Advanced Queries – Crosstab Query
- Using Advanced Queries – Find Matched Query
- Using Advanced Queries – Find Unmatched Query
- Using Advanced Queries – Autolookup Query
- Changing Content of Tables – Make-Table Action Query
- Changing Content of Tables – Delete Action Query
- Changing Content of Tables – Update Action Query
- Changing Content of Tables – Append Action Query

The textbook demonstrates how to use the facilities of QBE using simple examples. As such, students can work through this chapter with Microsoft Access or an equivalent DBMS to gain ‘hands-on’ experience of QBE. The QBE operations access the tables of the *DreamHome* case study. Therefore, before using chapter 7, create these tables and populate them with the data shown in Figure 3.3 of the textbook.

Activity 7.1

You should attempt exercises 7.1, 7.2 and 7.3 from the textbook using the QBE facility (or similar) of the **available DBMS**.

7.2 Commercial DBMSs: Access and Oracle

Chapter 8 of the textbook introduces two commercial RDBMSs: Access and Oracle.

However, to really appreciate these systems requires ‘hands-on’ experience in a laboratory environment. Therefore, it would be extremely useful if you have access to one (or both) of these systems for experimentation.

It is optional for you to read the chapter regarding these two commercial RDBMSs:

- About Microsoft Access:
 - the DBMS architecture
 - how to create base tables and relationships
 - how to create enterprise constraints
 - how to use forms and reports
 - how to use macros.
- About Oracle:
 - the DBMS architecture
 - how to create base tables and relationships
 - how to create enterprise constraints
 - how to use PL/SQL
 - how to create and use stored procedures and functions
 - how to create and use triggers.

Chapter 8 attempts to give a general overview of the facilities provided by Microsoft Access and Oracle. This is achieved using simple examples to demonstrate some of the functionality.

If you have covered both systems, note the significant differences between the administration and management of database structures in Access and Oracle:

- an Access database essentially consists of a single file and a new database can be easily created by a user
- the administration and management of an Oracle database is a complex task, and in a large organization requires a dedicated team to maintain it.

Review questions 7.1

This set of *Review Questions* is optional. Attempt the *Review Questions* on page 265 of the textbook. Below are listed some helpful guides for certain questions:

- Question 8.1 See Section 8.1.1.
 - Question 8.2 See Section 8.1.2 (Multi-user support).
 - Question 8.3 See Table 8.1.
 - Question 8.4 See start of Section 8.1.3, which lists 5 ways.
 - Question 8.5 See start of Section 8.1.5, which lists 3 ways.
 - Question 8.6 See Section 8.2.1.
 - Question 8.7 Oracle's logical structure consists of tablespaces, schemas, and data blocks and extents/segments (See Section 8.2.2).
 - Question 8.8 Oracle's physical structure consists of datafiles, redo log files, and control files (See Section 8.2.2).
 - Question 8.9 See Section 8.2.2.
 - Question 8.10 See Table 8.3.
 - Question 8.11 Using SQL*Plus or Create Table Wizard (See Section 8.2.3).
 - Question 8.12 See Section 8.2.4, which lists 4 ways.
 - Question 8.13 See Section 8.2.5.
-

Module 8

Entity-Relationship modelling

Learning objectives

On successful completion of this module, you should be able to:

- use Entity-Relationship (ER) modelling to support database design
- understand the basic concepts associated with the Entity-Relationship (ER) model
- represent ER models by traditional diagrams
- apply diagrammatic technique for displaying ER model using Unified Modelling Language (UML)
- identify and resolve problems with ER models called connection traps
- build an ER model from a requirements specification
- discuss the limitations of the basic ER modelling concepts and the requirements to model more complex applications using enhanced data modelling concepts
- implement main concepts associated with the Enhanced Entity-Relationship (EER) model called specialization/generalization and categorization
- display specialization/generalization, aggregation and composition in an EER diagram using UML.

Introduction

This module covers chapter 11 and chapter 12 of the textbook. Chapter 11 and Chapter 12 introduce techniques for conceptual database design: Entity Relationship (ER) modelling. You need to read the textbook for the basic and additional concepts of ER modelling and aim to achieve the goals mentioned above.

8.1 Concepts of the ER model

8.1.1 Entity types

An **entity type** is a set of objects or concepts with the same properties. An entity may be real or abstract, such as an invoice, an organization, a car, or an airplane reservation.

The notation of an entity type is a rectangular box in an Entity-Relationship diagram. An example of this is the ER diagram of the Branch view of *DreamHome* on page 332 of the textbook.

An **entity occurrence** is an instance of an entity type that is uniquely identifiable. For example, an entity type is **Invoice**, and an entity occurrence might be Invoice No 45632.

8.1.2 Relationship types

A relationship type is a set of associations between entity types.

It is different with a **relationship occurrence**.

A **relationship occurrence** is the association between the participating entity occurrences.

See the example on page 334 of the textbook to understand the concepts of relationship types and occurrence.

The **degree of a relationship** is the number of entity types participating in the relationship type.

Relationship of degree:

- one is a recursive relationship where the **same** entity type is involved more than once in **different roles**
- two is a binary relationship
- three is a ternary relationship
- four is a quaternary relationship.

8.1.3 Attribute

An **attribute** is a property of an entity or a relationship type. Attributes have values.

An **attribute domain** is a set of permissible values that may be assigned to one or more attributes.

There are different types of attributes to describe entities and relationships:

- Simple attributes
- Composite attributes
- Single-valued attributes
- Multi-valued attributes
- Derived attributes
- Candidate key:
 - A minimal set of attributes that uniquely identifies each occurrence of an entity type.
- Primary key:
 - A candidate key selected to uniquely identify each entity occurrence.
- Composite key:
 - A candidate key that consists of two or more attributes.

8.1.4 Strong and weak entity type

An entity type is called a strong entity type if it exists by itself.

A weak entity type is dependent on some other entity type for its existence.

A weak entity type does not have a primary key, it must use the primary key of its owner entity to identify the uniqueness.

8.1.5 Attributes on relationship

Sometimes a relationship may need to be described by some attributes to identify the association between entity types.

8.1.6 Structural constraints

There is a main type of constraint on relationships called **multiplicity**.

Multiplicity is described as the number of possible occurrences of an entity type that may be involved in a particular relationship for a single occurrence of an associated entity type.

Multiplicity applied for binary relationships are:

- one-to-one (1:1)

See Figure 11.14 on page 346 of the textbook: the multiplicity of entity type “Staff” is 1:1 with the relationship *Manages* to entity “Branch” with 0:1 multiplicity.

This means that one branch has exactly one manager and a staff may manage zero or one branch.

- one-to-many (1:*)

See the example on page 346–47 of the textbook for the multiplicity and the semantic relationships of Staff : *Oversees*: PropertyForRent (1:*).

- many-to-many (*:*)

See the example on page 348 of the textbook for the multiplicity of Newspaper: *Advertises*: PropertyForRent (*:*) relationship.

Cardinality and participation constraints

Cardinality and participation are two constraints that can be described by multiplicity.

Cardinality describes relationship occurrences and participation determines how many entity occurrences participate in a relationship.

Refer to Figure 11.18 on page 251 of the textbook for a demonstration of the cardinality and participation constraints by the example: Staff *Manages* Branch (1:1) relationship.

8.1.7 Problems with ER models

During designing a conceptual data model some problems such as connection traps may arise.

There are two main types of connection traps called **fan traps** and **chasm traps**.

Fan trap is the problem when the designed conceptual data model cannot answer some questions through the pathway between certain entity occurrences.

It is often the problem of misplacing relationships between entity types (see the example Staff : Division : Branch relationships on page 351 of the textbook).

Chasm trap may be caused by a model which describes an existence of a relationship between entity types, but the indirect relationship does not exist between certain entity occurrences.

Read the textbook for the example of a Fan trap and Chasm trap to understand why the modified models can remove the problems.

Activity 8.1

- Read the *Chapter Summary* and answer *Review Questions: 11.1–11.9* on page 356 of the textbook.
- Attempt *exercises 11.10–11.12* from page 357 of the textbook.

8.2 Enhanced entity-relationship (EER) model

Chapter 12 introduced additional data modelling concepts to the basic ER model, creating the enhanced entity-relationship (EER) model. The concepts are inspired by object oriented techniques. Examples of additional concepts in the EER model are:

- specialization/generalization
- aggregation
- composition.

The additional concepts are very useful for representing semantic requirements of more complex applications during conceptual data modelling.

8.2.1 Specialization / generalization

Specialization is the process of identifying the distinct differences between members of an entity type and classifying them into different subgroups.

Generalization is the process of finding a group of entities by identifying their common features and model a single entity to cover the common properties.

Superclass and subclass are central concepts used in the Specialization / Generalization processes.

A superclass is an entity type that includes distinct subclasses required to be represented in the data model.

A subclass is an entity type that has a distinct role and is also a member of a superclass.

Superclass/subclass relationships can be described as hierarchical structures. Subclasses inherit all attributes of their superclass, while subclasses must have distinct characteristics from the superclass and siblings.

For instance, **Vehicle** is a superclass with properties: registration number, engine and wheel, etc. Classes **Car**, **Truck** and **Bus** can be identified as subclasses of **Vehicle** that inherit all properties of vehicle while they have distinct properties individually.

The textbook illustrated the example of representation of specialization /generalization for the *Staff* entity into subclasses representing job roles.

Constraints on specialization / generalization

There are two constraints that may apply to a specialization/generalization:

1. Participation constraint:

Defines the relationship of superclass and subclass for the participation occurrence. The instances of superclass may be **mandatory** or **optional** members of subclasses. E.g., a vehicle must be one of members of subclasses Car or Truck or Bus. It is participation mandatory.

2. Disjoint constraint:

Describes the relationship between instances of the subclasses, and defines whether instances of a superclass are allowed to overlap and exist as members of more than one subclass.

E.g., a staff member David in superclass *Staff* may be an instance of subclass *Student* and an instance of subclass *Teacher*. They have nondisjoint relationship.

There are four categories of constraints of specialization and generalization:

1. mandatory and disjoint
2. optional and disjoint
3. mandatory and nondisjoint
4. optional and nondisjoint.

Read the *DreamHome* worked example:

- Staff Superclass with Supervisor and Manager subclasses
- Owner Superclass with PrivateOwner and BusinessOwner subclasses
- Person superclass with Staff, PrivateOwner, and Client subclasses

to understand the semantics of the EER diagram of *DreamHome* with specialization /generalization.

8.2.2 Aggregation

An aggregation concept describes the relationship between a larger entity consisting of smaller entities. It is the model with a 'has-a' or 'is-part-of' relationship between entity types.

See the graphical representation of examples of aggregation on page 372 of the textbook.

The notation of aggregation is an empty diamond shape attached to the 'whole' entity with a line to 'parts' entities.

8.2.3 Composition

Composition is a special case of aggregation. It represents an association between entities, where the 'whole' is a strong ownership entity and the existence of 'part' entities depend on the whole entity.

An example of Composition is shown in the textbook.

The notation of aggregation is a solid diamond shape attached to the 'whole' entity with a line to 'parts' entities.

Activity 8.2

- Read the *Chapter Summary* and answer *Review Questions*: 12.1–12.8 on page 356 of the textbook.
 - Attempt *exercises* 12.9–12.10 from page 374 of the textbook.
-

Module 9

Normalization

Learning objectives

On successful completion of this module, you should be able to:

- explain the purpose of normalization
- identify various types of update anomalies such as insertion, deletion, and modification anomalies
- understand the concept of functional dependency
- find the primary key for a relation using functional dependencies
- identify the most commonly used normal forms, such as first (1NF), second (2NF) and third (3NF) normal forms, and Boyce-Codd normal form (BCNF)
- decompose relations into a known normal form using functional dependencies.

Introduction

This module covers chapters 13 and 14 of the textbook.

9.1 The purpose of normalization

The logical data model design of relational database systems is aimed to create an accurate representation of the data with its relationships and constraints. To achieve this objective, we must identify a suitable set of relations. Normal forms are introduced for such purpose. A normal form is a property of a relational database.

There are four most commonly used normal forms: first (1NF), second (2NF) and third (3NF) normal forms, and Boyce-Codd normal form (BCNF). They are introduced in the later sections.

When a relation is non-normalized (that is, does not satisfy a normal form), then it presents redundancies and produces undesirable behaviour during update operations.

Normalization is a procedure used to reduce data redundancy and update anomalies of relations. It transforms non-normalized schemas into new schemas for which the satisfaction of a normal form is guaranteed. This principle can be used to carry out quality analysis and constitutes a useful tool for database design.

9.2 Data redundancy and update anomalies

The textbook discusses the problems caused by data redundancy.

Data redundancy causes problems for database updates. The types of update anomalies include:

- Insertion
- Deletion
- Modification.

The example shown on page 377 of the textbook gives a demonstration of the problems. The relation *StaffBranch* contains redundant data, it cause anomalies for insertion, deletion and modifications.

Read the textbook to understand the problems of insertion, deletion, and modification anomalies that are caused by the data redundancy.

The process of normalization is to decompose a larger relation into two or more smaller relations.

To guarantee the decomposed smaller relations equivalent to the original larger relation, the following two conditions (or properties) must be satisfied by a composition:

1. Lossless-join
2. Dependency preservation.

They can be regarded as the correctness criterion of normalization.

9.2.1 Two important properties of decomposition

1. *Lossless-join property* is to ensure that any instance of the original relation can be found from corresponding instances in the smaller relations. That is, the information is not missing after the process of normalization.
2. *Dependency preservation property* is to guarantee that all constraints on the original relation will be maintained in the smaller relations.

The correctness of normalization will be further discussed in a later section of this module.

9.3 Functional dependency

Functional dependency (FD) is a central concept associated with normalizations. You must understand the meaning of FD in order to carry on the process of normalizations.

The process of normalization is to identify relations based on their keys, and the functional dependencies among their attributes.

A functional dependency (FD) describes the relationship between attributes in a relation.

If A and B are attributes of relation R, B is functionally dependent on A (denoted $A \rightarrow B$), if each value of A in R is associated with exactly one value of B in R.

For example: if FD $\text{BranchNo} \rightarrow \text{bAddress}$ is holding on relation `StaffBranch`, then the values in pair (B005, 22 Deer Rd, London) = (BranchNo, bAddress) must always hold in any tuples of the relation. That is, branch B005 determines its address 22 Deer Rd, London. There are no tuples in a relation with different addresses for branch B005.

For example, given a relation:

A	B	C	D
A1	B1	C1	D1
A1	B2	C1	D1
A2	B1	C2	D1
A2	B2	C2	D1

$A \rightarrow C$ satisfies the above definition, whereas $A \rightarrow B$ does not. Please note that $A \rightarrow D$ satisfies the above definition too. We list more functional dependencies as follows.

$AB \rightarrow C, AB \rightarrow D, AC \rightarrow D, AD \rightarrow C, CD \rightarrow A, \dots$

We also can say that functional dependencies are constraints on a relation schema. In other words, a FD is a statement about all allowable relations. All tuples in a relation instance must satisfy the given FDs. We also can identify FDs on a relation instance.

9.4 Identifying the primary key for a relation using functional dependencies

Note that a key constraint is a special case of functional dependencies.

If a set of attributes K is a candidate key for relation R , it means that $K \rightarrow R$. It also means that $P \rightarrow R$ does not hold, if $P \subset K$. For example, we say CD is a key of $R(ABCDE)$ if $CD \rightarrow ABCDE$ hold and $C \rightarrow ABCDE$ and $D \rightarrow ABCDE$ do not hold.

This is because a candidate key is a minimal set of attributes that uniquely identify a tuple within a relation.

9.5 Inference rules for functional dependencies

The complete set of functional dependencies may be very large and not manageable. To find an approach that can reduce the set of FDs, a set of inference rules are required.

Armstrong's axioms (AA) specify how new functional dependencies can be inferred from given ones.

You need to become familiar with seven inference rules:

- Armstrong's axioms (X, Y, Z are sets of attributes):
 1. **Reflexivity** : If $X \subseteq Y$, then $Y \rightarrow X$
 2. **Augmentation** : If $X \rightarrow Y$, then $XZ \rightarrow YZ$ for any Z
 3. **Transitivity** : If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

These are sound and complete inference rules for FDs!

- Additional rules (that are inferred from AA):
 1. Self-determination: $X \rightarrow X$
 2. Decomposition: If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
 3. Union: If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
 4. Composition: If $X \rightarrow Y$ and $W \rightarrow Z$, then $XW \rightarrow YZ$

9.5.1 Closure of FDs

Given some FDs, we can usually infer additional FDs:

$ssn \rightarrow did, did \rightarrow lot$ implies $ssn \rightarrow lot$ (transitivity rule)

A FD f is **implied** by a set of FDs F if f can be derived by FDs in F .

Closure of F : F^+ : is the set of all FDs that are implied by F . We can say that F is equivalent to F^+ ($F \equiv F^+$)

Example:

Let relation scheme

$R = (A, B, C, G, H, I)$, and FDs

$F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$

We can list some member of F^+ here:

$A \rightarrow H$ since $A \rightarrow B$ and $B \rightarrow H$ (Transitivity)

$CG \rightarrow HI$ since $CG \rightarrow H, CG \rightarrow I$ (Union)

$AG \rightarrow I$ since $AG \rightarrow CG$ and $CG \rightarrow I$ (Augmentation & transitivity)

You can derive more FDs for F^+ !

Computing the closure of a set of FDs can be expensive. (Size of closure is exponential to the number of attributes!)

Typically, we just want to check if a given FD $X \rightarrow Y$ is in the closure of a set of FDs F , without computing all FDs of F^+ . An efficient check is to:

- compute **attribute closure** of X (denoted X^+) in R
- see if Y is in X^+ .

The computation of attribute closure takes linear time only.

9.5.2 Closure of attribute sets

Define the closure of attributes X under FDs F (denoted by X^+) as the set of attributes that are functionally determined by X under F , i.e. $X \rightarrow Y$ is in $F^+ \Leftrightarrow Y \subseteq X^+$.

In other words, An **attribute closure** of X : X^+ is a set of all attributes Y such that $X \rightarrow Y$ is in F^+ .

Check if E is in A^+ under $F = \{A \rightarrow B, B \rightarrow C, C D \rightarrow E\}$:

We need to compute whether $A \rightarrow E$ is true to F . I.e., is $A \rightarrow E$ in the closure F^+ ? Equivalently, is E in A^+ ?

9.5.3 Computing attribute closure of X

The following algorithm from [Silberschatz et. al.](#) is to compute X^+ , the closure of X under F :

```
Result := X; //start with attributes X
while (changes to result) do
  for each  $Z \rightarrow A$  In  $F$  do
    Begin
      if  $Z \subseteq \text{result}$  //if  $Z$  is a subset of  $X$  or  $Z=X$ 
      then  $\text{Result} := \text{Result} \cup A$ ; // add the right side of FD in  $X^+$ 
    end
```

The final Result is X^+

Example — compute $(AG)^+$

Let $R = (A, B, C, G, H, I)$, and:

$F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$

Calculate $(AG)^+$ using the above algorithm:

Result = AG

Result = AGB ($A \rightarrow B$ and $A \subseteq \text{result}(AG)$)

Result = AGBC ($A \rightarrow C$ and $A \subseteq \text{AGB}$)

Result = AGBCH ($B \rightarrow H$ and $B \subseteq \text{AGBC}$) or

Result = AGBCH ($CG \rightarrow H$ and $CG \subseteq \text{AGBC}$)

Result = AGBCHI ($CG \rightarrow I$ and $CG \subseteq \text{AGBCH}$)

Is AG a candidate key?

Since $AG \rightarrow R$ (AGBCHI) all attributes, AG is a key. To determine whether AG is a candidate key of R, two conditions must be satisfied: AG is minimal and $AG \rightarrow R$.

We need to prove that:

1. Does $A^+ \rightarrow R$?
2. Does $G^+ \rightarrow R$?

We need to compute A^+ and G^+ to see all attributes are in A^+ or G^+ . Do it by yourself: to derive A^+ and G^+ .

If none of the above is true, then AG is the minimal left side of FD: $AG \rightarrow R$, AG is a candidate key. That is, if delete any attribute from AG: A or G is not a key of R.

9.5.4 Minimal set of functional dependencies

Sometimes FDs of a relation contain redundant attributes in both sides of FDs.

We want to minimize the FDs by deleting redundancies (or **extraneous attributes**) and **remain the same semantics**. That is, after the deletion of attributes from the original FDs, we have a new set of FDs, while the new set of FDs can drive all original FDs **This is so called the equivalence of two set FDs**.

How do we identify if a set of FDs F is minimal? There are two conditions to be satisfied:

1. if $\alpha \rightarrow \beta$ is a FD of F and A is a subset of α , the FD: $(\alpha - A) \rightarrow \beta$ is not true in F. In other words, α does not contain extraneous attributes.

2. if $\alpha \rightarrow \beta$ is a FD of F and B is any subset of attributes of β , the FD: $\beta - B \rightarrow B$ is not true in F. In other words, there are no transitive dependencies between the attributes in β .

A **minimal set** F_m for F is a set of dependencies such that F logically implies all dependencies in F_m , and F_m logically implies all dependencies in F, and further:

- No functional dependency in F_m contains an extraneous attribute.
- Each left side of a functional dependency in F_m is unique.

Compute a minimal set for F:

Repeat

Use the union rule to replace any dependencies in F:

$A1 \rightarrow B1$ and $A1 \rightarrow B2$ with $A1 \rightarrow B1 B2$

Find a functional dependency $A B$ with an extraneous attribute

For each FD $\alpha \rightarrow \beta$ if $A \in \alpha$ and a FD $(\alpha - A) \rightarrow \alpha$ exists in F then

$(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$

// this is to delete the **extraneous attribute from the** left-hand side if the FD.

For each FD $\alpha \rightarrow \beta$ if $B \in \beta$ and $(\beta - B) \rightarrow B$ exists in F then

$(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - B)\}$

//this is to delete the extraneous attribute from the right-hand side if the FD, until F does not change.

9.5.5 Example of minimal set of functional dependencies

This example is taken from [Silberschatz et. al.](#)

Let us consider to compute F_m for the following set F of functional dependencies on schema $R(A,B,C)$:

$F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$

1. Merge all FDs with the same left side in F:

$A \rightarrow BC$ ($A \rightarrow B$ is merged)

2. Check FDs' left side: FD $AB \rightarrow C$ contains an extraneous attribute A in the left side since there is $B \rightarrow C$ in F.

$$F - \{AB \rightarrow C\} \cup \{B \rightarrow C\} = \{A \rightarrow BC, B \rightarrow C\}$$

3. Check FDs' right side: C is extraneous in $A \rightarrow BC$ since there is a $B \rightarrow C$ in F}

$$F - \{A \rightarrow BC\} \cup \{A \rightarrow B\} = \{A \rightarrow B, B \rightarrow C\}$$

Therefore, $F_m = \{A \rightarrow B, B \rightarrow C\}$.

9.6 The process of normalization

Normalization is based on candidate keys and functional dependencies between attributes of the relation to identify the normal form of the relation.

If a relation does not satisfy a desired normal form, replace it with two or more normalized relations using a process called normalization.

There are a series of steps for the process of normalization. Each step corresponds to a transformation from a lower normal form to a higher normal form.

9.6.1 Decomposition of a relation scheme

Suppose that relation R contains attributes $A_1 \dots A_n$. A **decomposition** of R consists of replacing R by two or more relations such that:

- Each new relation scheme contains a subset of the attributes of R, and
- Every attribute of R appears as an attribute of one of the new relations. (No missing attributes!)

Certainly, the decomposition should **correctly** present the original information.

The purpose of normalization is to reduce redundancy and update/insertion/deletion anomalies

9.7 Relationship between normal forms

There are three types of normal forms called First (1NF), Second (2NF), and Third (3NF) normal forms. A stronger definition of third normal form (3NF) is called the Boyce-Codd Normal Form (BCNF). All these normal forms are dependent on the functional dependencies among the attributes of a relation.

9.7.1 Unnormalized form (UNF)

A relation containing multiple values in one field of one tuple is called an unnormalized form. See the relation ClientRental example in Figure 13.7 of the textbook. The first tuple (or row) of the relation contains two values of fields: pAddress ...oName for one clientNo.

9.7.2 First normal form (1NF)

A relation is in first normal form if it contains one and only one value for each attribute in a tuple.

Steps for UNF to 1NF

1. Nominate an attribute or group of attributes to act as the key for the unnormalized table.
2. Identify repeating group(s) in the unnormalized table which repeat for the key attribute(s).
3. Remove the repeating group by:
 - entering appropriate data into the empty columns of rows containing repeating data ('flattening' the table).
 - Or by
 - placing repeating data along with a copy of the original key attribute(s) into a separate relation.

Refer to the example of the ClientRental table for the steps from UNF to 1NF.

9.7.3 Second normal form (2NF)

2NF is based on concept of full functional dependency. A **functional dependency** $X \rightarrow Y$ is a full functional dependency if removal of any attributes A from X causes the dependency not to be held any more.

For instance, $ABC \rightarrow Y$ holds on relation R, but not any subset of ABC determines Y:

AB not $\rightarrow Y$; BC not $\rightarrow Y$; and AC not $\rightarrow Y$ on R, we say that $ABC \rightarrow Y$ is a fully functional dependency.

A relation is in 2NF if it is

1. in 1NF; and
2. every non-primary-key attribute is fully functionally dependent on the primary key.

Steps for 1NF to 2NF:

1. Identify the primary key for the 1NF relation.
2. Identify functional dependencies in the relation.

3. If partial dependencies exist on the primary key remove them by placing them in a new relation along with a copy of their determinant.

For example, let $R = \{ABCDFGHIN\}$ and

$F = \{AB \rightarrow CD, A \rightarrow N, B \rightarrow FGHI, H \rightarrow I, AC \rightarrow BFDGHI, BC \rightarrow AND\}$

The first step is to find the primary key of R, using the closure of attributes. We can derive

$(AB)^+ \rightarrow ABCDFGHIN = R$ which is a key of R.

Then we need to minimize the key AB: to check $A \rightarrow R?$ or $B \rightarrow R?$ Answers are no. So AB is a candidate key of R. (Note: If there is more than one candidate key, can you prove that AC and BC are candidate keys?)

The second step is to identify FDs in F in regards to partial dependencies on the primary key.

Since $A \rightarrow N$ and $B \rightarrow FGHI$, it means that attributes N and FGHI are not full functional dependencies on the primary key AB.

The third step is to remove the partial dependencies and decompose R into smaller relations as follows:

$R_1 = (A N), R_2 = (BFGHI), R_3 = (ABCD)$

These three relations are in second normal form since they satisfy the two conditions of 2NF.

Note that the FDs of F will be projected on the attributes of R_1, R_2 and R_3 to construct new FDs on individual relations. Some FDs may have less attributes at the right side, but the left side may not be reduced. The final result of the normalization of R is as follows:

$R_1 = (A N), F_1 = \{A \rightarrow N\}$

$R_2 = (BFGHI), F_2 = \{B \rightarrow FGHI, H \rightarrow I\}$

$R_3 = (ABCD), F_3 = \{AB \rightarrow CD, BC \rightarrow AD, AC \rightarrow BD\}$.

Read the example in the textbook for the steps in transforming the relation ClientRental from 1NF to 2NF.

9.7.4 Third normal form (3NF)

3NF is based on concept of **transitive dependency**:

A, B and C are attributes of a relation such that if $A \rightarrow B$ and $B \rightarrow C$, then C is **transitively dependent** on A through B: $A \rightarrow C$.

If we say that a relation is in 3NF if it is:

1. in 1NF and 2NF
2. no non-primary-key attribute is transitively dependent on the primary key.

Another definition of 3NF can be used for the identification of relations. This is from [Atzeni et. al.](#):

A relation R is in 3NF if, for each non trivial FD $X \rightarrow A$ in F^+ , at least one of the following is true:

1. X contains a key for R, or
2. A is part of some key for R.

A trivial FD means $X \rightarrow X$.

The steps transforming 2NF to 3NF

1. Identify the primary key in the 2NF relation.
2. Identify functional dependencies in the relation.
3. If transitive dependencies exist on the primary key remove them and decompose the relation into relations obtained by projections on the attributes corresponding to the functional dependencies.

We can use the result from the previous example 1NF to 2NF:

$$R_1=(A N), F_1=\{A \rightarrow N\}$$

$$R_2=(BFGHI), F_2=\{B \rightarrow FGHI, H \rightarrow I\}$$

$$R_3=(ABCD), F_3=\{AB \rightarrow CD, BC \rightarrow AD, AC \rightarrow BD\}.$$

The relations R_1 and R_3 are in 3NF already since they are in 2NF and have no transitive dependencies.

If relation R_2 has a transitive dependency $H \rightarrow I$, then

- remove attribute **I** from right side of $B \rightarrow FGHI$
- replace the relation R_2 with attributes of $B \rightarrow FGH$ without I into R_{21}
- construct a new relation containing the attributes of transitive dependency FD $H \rightarrow I$.

R_2 is decomposed into two relations as follows:

$$R_{21} = (BFGH)$$

$$R_{22} = (HI)$$

Finally, the following smaller relations are the results for the normalization of 3NF of relation R:

$R_1 = (AN)$

$R_{21} = (BFGH)$

$R_{22} = (HI)$

$R_3 = (ABCD)$

The only condition to guarantee the correctness in the above process is of always maintaining a relation that contains a key of the original relation.

For instance, $R_3 = (ABCD)$ in the above decomposition contains the key of R. The join of R_1 , R_{21} , R_{22} and R_3 will be equal to the original R.

Formal approach for decomposition of third normal form

Before introducing the formal approach for the process of 3NF. We need to understand the term **minimal set of functional dependencies**. There is some discussion in section 13.3.4 of the textbook.

The following algorithm is provided for those students who want to know more, and will not be assessed.

3NF decomposition algorithm

The following is an algorithm ([Silberschatz et. al.](#)) to decompose a relation into 3NF.

```

Input R, F
Calculate minimal set of F:  $F_m$ ;
find all candidates keys;
i := 0;  $R_i = R$ ;
for each functional dependency  $A \rightarrow B$  in F do
  if none of the schemas  $R_j$ ,  $1 \leq j \leq i$  contains A, B
  then begin
    i := i + 1;
     $R_i := AB$ ; //create a new relation  $R_i$  with attributes AB.
  end
  if none of the schemas  $R_j$ ,  $1 \leq j \leq i$  contains a candidate key for R
  then begin
    i := i + 1;
     $R_i :=$  any candidate key for R;
  end
end
return ( $R_1, R_2, \dots, R_i$ )

```

9.7.5 Boyce-Codd normal form (BCNF)

We say a relation is in BCNF, if and only if, every determinant is a candidate key.

You are required to understand all discussions on the textbook, and you are required to know how to transform a relation from UNF to BCNF as shown in 13.10 of the textbook. The following is the supplementary discussion for those students who want to know more and it will not be assessed.

A relation R is in BCNF if, for each FD $X \rightarrow A$ in F^+

X contains a key for R.

In other words, R is in BCNF if the only non-trivial FDs that hold over R are key constraints.

Consider the following example to answer: Is R in BCNF ?

$R = (A, B, C)$ and $F = \{ A \rightarrow C, B \rightarrow C \}$

You need to use the definition of BCNF to identify the relation R:

1. find the keys of R first: A is the only key of R.
2. test every FD in F to see whether the left side is a key:

$A \rightarrow C$ is OK but $B \rightarrow C$ is not since B is not a key at the left side.

then R is not in BCNF.

If R is not in BCNF, the decomposition may be needed

BCNF decomposition algorithm

The following is an algorithm to decompose a relation into BCNF:

```

Result := {R}; compute  $F_M^+$  ;
done := false;
while (not done) do
  if (there is a schema  $R_i$  in result that is not in BCNF)
  then
    begin
      if  $A \rightarrow R_i \notin F_M^+$  and  $A \cap B = \emptyset$  // I.e., A is not a key
      then Result := (Result -  $R_i$ )  $\cup$  ( $R_i - B$ )  $\cup$  (A,B);
      end
    else done := true;

```

Let us look at an example as follows:

$R = (\text{Employee}, \text{Project}, \text{Branch})$

$F = \{\text{Employee} \rightarrow \text{Branch}, \text{Project} \rightarrow \text{Branch}\}$

Employee	Project	Branch
Brown	Mars	Chicago
Green	Jupiter	Birmingham
Green	Venus	Birmingham
Hoskins	Saturn	Birmingham
Hoskins	Venus	Birmingham

The key of R is (Employee Project), R is not in BCNF.

Using the algorithm:

Result := {R}; compute $F_M^+ = F$;

for a nontrivial FD

Employee \rightarrow Branch holds on R

Employee \rightarrow R $\notin F_M^+$ and

Employee \cap Branch = \emptyset

then Result := (Result - R) \cup (R - Branch) \cup (Employee, Branch)

The final result is:

$R_1(\text{Employee}, \text{Project})$

$R_2(\text{Employee}, \text{Branch})$ since R_1 and R_2 are BCNF, it is done.

The decomposed relations are:

Employee	Branch
Brown	Chicago
Green	Birmingham
Hoskins	Birmingham

Employee	Project
Brown	Mars
Green	Jupiter
Green	Venus
Hoskins	Saturn
Hoskins	Venus

Note if we select Project \rightarrow Branch first in the algorithm, we will have a decomposition of R:

$R_1(\text{Employee,Project})$ and $R_2(\text{Project, Branch})$

For each normalization we need to test whether the decomposition is correct or not.

9.8 Correctness criterion of normalization

The following discussion is provided for those students who want to know more, and will not be assessed.

How do we know that the normalization is correct?

The decomposition is correct if the original relation equivalent to the join of decomposed relations and the FDs of the original relation are still true on the new relations.

Saying this more formally, if the following two conditions of the decomposition are true:

1. Lossless-join
2. Dependency preservation.

The normalization is correct.

9.8.1 Lossless join decompositions

Suppose a relation R has a set of attributes A, and X, Y are subsets of A such that $X \cup Y = A$. If the decomposition $R(X)$ and $R(Y)$ of $R(A)$ is **lossless-join**, the following is true:

$$\Pi_X(r) \text{ join } \Pi_Y(r) = r$$

It is always true that $r \subseteq \Pi_X(r) \text{ join } \Pi_Y(r)$. In general, the other direction does not hold! If it does, the decomposition is lossless-join.

It is essential that all decompositions carried out for the purpose of normalization are lossless-joins.

A condition for the lossless-join decomposition

The decomposition of R into R_1 and R_2 is lossless-join if and only if the closure of F contains:

$$R_1 \cap R_2 \rightarrow R_1, \text{ or}$$

$$R_1 \cap R_2 \rightarrow R_2$$

In other words the common attributes of R_1 & R_2 must contain a key for either R_1 or R_2 .

9.8.2 Dependency preservation

A decomposition preserves the dependencies if each of the functional dependencies of the original schema involves attributes that appear all together in one of the decomposed schemas.

It is clearly desirable that a decomposition preserves the dependencies since, in this way, it is possible to ensure, on the decomposed schema, the satisfaction of the same constraints as the original schema.

The formula for the dependency preservation:

$$F^+ = (F_1 \cup F_2 \cup F_n)^+$$

Let us look at the previous decomposition of BCNF again to test its correctness:

Two conditions should be satisfied:

1. Lossless-join
2. Dependency Preservation.

The BCNF decomposition was:

$R_1(\text{Employee, Project})$ and $R_2(\text{Employee, Branch})$

1. Does $R_1 \cap R_2 \rightarrow R_1$, or $R_1 \cap R_2 \rightarrow R_2$

Yes, $R_1 \cap R_2 = \text{Employee}$ which is the key of R_1 , the decomposition is lossless-join.

2. Does $F^+ = (F_1 \cup F_2 \cup F_n)^+$?

No, since $(F_1 \cup F_2)^+ = \{\text{Employee} \rightarrow \text{Branch}\}$ and $F^+ = \{\text{Employee} \rightarrow \text{Branch}, \text{Project} \rightarrow \text{Branch}\}$

The above decomposition is not dependency preservation. It is a lossless decomposition of R , the FD: $\text{Project} \rightarrow \text{Branch}$ is missing.

The relation R can't be decomposed into a BCNF and maintain all semantics of the original relation.

The original relation R is in 3NF according to the definition. The left sides of all FDs are part of the key of R .

9.8.3 Qualities of decompositions

Decompositions should always satisfy the properties of lossless decomposition and dependency preservation:

Lossless decomposition ensures that the information in the original relation can be accurately reconstructed based on the information represented in the decomposed relations.

Dependency preservation ensures that the decomposed relations have the same capacity to represent the integrity constraints as the original relations and thus to reveal illegal updates.

9.9 Comparison of BCNF and 3NF

The **difference** between 3NF and BCNF is that for a functional dependency $A \rightarrow B$, 3NF allows this dependency in a relation if B is a primary-key attribute and A is not a candidate key.

Every relation in BCNF is also in 3NF. However, a relation in 3NF may not be in BCNF.

It is always possible to decompose a relation into relations in 3NF and

1. the decomposition is lossless
and
2. dependencies are preserved.

It is always possible to decompose a relation into relations in BCNF and

1. the decomposition is lossless
but
2. it may not be possible to preserve dependencies

The potential to violate BCNF may occur in a relation that:

- contains two (or more) composite candidate keys
- the candidate keys overlap (i.e. have at least one attribute in common).

9.9.1 Design goals

The goal for a relational database design is: BCNF.

- Lossless join.
- Dependency preservation.

If we cannot achieve BCNF, we accept:

- 3NF
- Lossless join
- Dependency preservation.

9.10 Fourth normal form (4NF)

There is another type of dependency called a multi-valued dependency (MVD), which can also cause data redundancy.

MVD on relation scheme R , where X , Y and Z are all subsets of R , describes the following dependency:

for each value of X there is a set of values for Y and a set of values for Z . However, the set of values for Y and Z are independent of each other.

The notation of MVD between attributes X , Y , and Z in a relation is represented as follows:

$$X \twoheadrightarrow Y$$
$$X \twoheadrightarrow Z$$

Fourth Normal Form (4NF) is a relation that is in BCNF and contains no trivial multi-valued dependency.

4NF is optional.

9.11 Fifth normal form (5NF)

A relation decomposed into two relations must have a lossless-join property, which ensures that no meaningless tuples are generated when relations are reunited through a natural join.

However, there are requirements to decompose a relation into more than two relations.

Although rare, these cases are managed by join dependency and fifth normal form (5NF).

Fifth normal form (5NF) is a relation that contains no join dependency.

The process of normalization up to 5NF is represented diagrammatically in Figure 14.10.

5NF is optional.

Reference list

Atzeni, P, Ceri, S, Paraboschi, S & Torlone, R 1999, *Database Systems, Concepts, Languages & Architectures*, McGraw Hill, Berkshire, England.

Silberschatz, A, Korth, HF & Sudarshan, S 1997, *Database System Concepts*, 3rd edn, McGraw Hill, NY.

Activity 9.1

- Read the *Chapter Summary* and answer *Review Questions*: 13.1–13.15 on page 412 of the textbook.
- Attempt *exercises* 13.16–13.20 from the textbook.

Extra Questions

1. **Suppose we have the following three tuples in a legal instance of relation schema S with three attributes: $R(A, B, C)$:**

r:	A	B	C
	a1	b2	c3
	a4	b2	c3
	a5	b3	c3

- Which of the following dependencies do not hold over R ?
 - (a) $A \rightarrow B$
 - (b) $BC \rightarrow A$
 - (c) $B \rightarrow C$
 - Can you identify those possible dependencies that hold over R ?
2. **Consider a relation R with five attributes $ABCDE$. You are given the following dependencies: $A \rightarrow B$, $BC \rightarrow E$, and $ED \rightarrow A$.**
 - List all the keys for R
 - Is R in 3NF?
 - Is R in BCNF?
-

Feedback

Activity 9.1

2. Semi-Answers:

- All the keys for R are: ACD, BCD and CDE.

In order to find the keys for R, we can take the following two steps:

Step1: Find Super Keys

For the given FDs $F=\{A \rightarrow B, BC \rightarrow E, ED \rightarrow A\}$, we first try to find the A^+ , BC^+ and ED^+ .

$A \rightarrow AB$, so $A^+ = AB$

$BC \rightarrow BCE$, so $BC^+ = BCE$

$ED \rightarrow AED \rightarrow ABED$, so $ED^+ = ABED$

Since $A^+ \neq R$, $BC^+ \neq R$, $ED^+ \neq R$, we know that A, BC and ED are not super keys for R ($R=ABCDE$). However, we know that

$(R - A^+) \cup A \rightarrow R$

$(R - A^+) \cup A = ACDE$ is a super key for R. (you should be able to prove this)

Similarly,

$(R - BC^+) \cup BC = ABCD$ is a super key for R.

$(R - ED^+) \cup ED = CDE$ is a super key for R

In summary, we get three super keys for R

– ACDE

– ABCD

– CDE

Step 2: Find Candidate Keys

For each of the above super keys, we need to see if some attributes can be removed with the closure of the remaining attributes still equalling R. That is, we will minimize each super key to see if any subset of a super key can be a candidate key:

For the super key ACDE, we need to calculate all possible combinations of subsets: $(AC)^+$, $(AD)^+$, $(AE)^+$, $(CD)^+$, $(CE)^+$, $(DE)^+$, $(ACD)^+$, $(ACE)^+$, $(CDE)^+$ and $(ADE)^+$ (**A, BC and ED are not keys in step 1**):

$$(AC)^+ = ABCE \neq R$$

$$(AD)^+ = ABD \neq R$$

$$(AE)^+ = ABE \neq R$$

$$(CD)^+ = CD \neq R$$

$$(CE)^+ = CE \neq R$$

$$(DE)^+ = ABDE \neq R$$

$$(ACD)^+ = ABCDE = R \text{ (e.g. } ACD \rightarrow ABCD \rightarrow ABCDE)$$

$$(ACE)^+ = ABCE \neq R$$

$$(CDE)^+ = ABCDE = R$$

$$(ADE)^+ = ABDE \neq R$$

Therefore, the **ACD** and **CDE** are candidate keys for R.

- Similarly, for the other two super keys ABCD and CDE, you should perform the same derivation as above to obtain the other two candidate keys for R.

At last, we have found the keys for R:

ACD, BCD and CDE.

- Is R in 3NF? Yes. (you state the reason why.)
- Is R in BCNF? No. (you state the reason why.)

Module 10

Database planning, design and
administration

Learning objectives

On successful completion of this module, you should be able to:

- describe the main components of an information system
- identify and describe the main stages of the database application lifecycle
- relate the main phases of database design: conceptual, logical and physical design
- appreciate the benefits of Computer-Aided Software Engineering (CASE) tools
- state the types of criteria used to evaluate a DBMS
- evaluate and select a DBMS
- distinguish between data administration and database administration
- explain the purpose and tasks associated with data administration and database administration.

Introduction

This module covers chapter 9 and chapter 10 of the textbook.

10.1 Database application lifecycle

Sections 9.1 and 9.2 of the textbook describe the information system lifecycle and discuss how this lifecycle relates to the database application lifecycle.

A computer-based information system includes the following **components**: database, database software, application software, computer hardware including storage media, and personnel using and developing the system.

The database is a fundamental component of an information system. The lifecycle of an organizational information system is inherently associated with the lifecycle of the database that supports it.

Figure 9.1 in the textbook shows the main stages of the database application lifecycle. They include:

- database planning
- system definition
- requirements collection and analysis
- database design
- DBMS selection (optional)
- application design
- prototyping (optional)

- implementation
- data conversion and loading
- testing, and operational maintenance.

10.2 Database planning

Database planning aims to manage the stages of the database application to be designed and implemented efficiently and effectively.

Read the textbook and note specifically:

- the **three main issues** involved in formulation of an IS strategy
- the purpose of the **mission statement**
- the importance of the identification of **mission objectives** in database planning.

10.3 System definition

System definition is essential to identify the system environment before attempting to design a database application.

10.3.1 User view

Identifying a user view is an important task in the development of database applications.

A **user view** defines what is required of a database application from the perspective of:

- a particular job role (such as manager or supervisor) or
- an enterprise application area (such as marketing, personnel, or stock control).

Database applications may have one or more user views.

Identifying user views helps ensure that no major users of the database are forgotten when developing requirements for a new application.

User views also help in development of complex database applications, allowing requirements to be broken down into manageable pieces.

10.4 Requirements collection and analysis

Information is gathered for each major user view and is analysed to identify requirements to be included in the new database application.

Three main approaches can be used to manage the requirements of the database:

1. **A centralized approach:**

Requirements for each user view are merged into a single set of requirements. A **global data model** is created based on the merged requirements (which represents all user views).

2. **A view integration approach:**

Requirements for each user view are used to build a separate data model.

A data model representing a single user view is called a **local data model**, composed of diagrams and documentation describing requirements of a particular user view of the database.

Local data models are then merged to produce a **global data model**, which represents all user views for the database.

3. **A combination** of both approaches.

10.5 Database design

Major aims of database design are to:

- represent data and relationships between data required by all major application areas and user groups.
- provide a data model that supports any transactions required on the data.
- specify a minimal design that is appropriately structured to achieve stated performance requirements for the system (such as response times).

There are four types of approaches used in database design:

- Top-down
- Bottom-up
- Inside-out
- Mixed

10.5.1 Data modelling

A data model is built using the information in the users' requirements specification.

Main purposes of data modelling include:

- assisting in understanding the meaning (semantics) of the data

- facilitating communication about the information requirements.

Building a data model requires answering questions about entities, relationships, and attributes.

10.5.2 Three phases of database design

1. Conceptual database design

This is the first phase of database design. It is the activity of modelling the information used in an enterprise, independent of **all** physical considerations.

2. Logical database design

The conceptual data model is refined and mapped on to a logical data model.

It is the activity of refining and mapping the model created from the conceptual design phase to a specific data model (e.g. relational), but independent of a particular DBMS and other physical considerations.

3. Physical database design

This is the process of generating a description of the database implementation on the computer environment.

Physical database design describes storage structures and access methods used to achieve efficient access to data. It is implemented on a specific DBMS system.

10.6 DBMS selection

The processes for selecting a DBMS are:

1. Define 'Terms of reference' of study.
2. Shortlist two or three products.
3. Evaluate products.
4. Recommend selection and produce report.

10.7 Application design

Application design is to create the user interface and application programs that access and process the database.

Application design includes two important activities:

1. Transaction design
2. User interface design.

10.7.1 Transaction design

Transactions are sets of operations to access or update the content of databases.

The purpose of the transaction design is to define and document the data, and functional characteristics of the transactions required on the database.

The design process should ensure that the implemented database satisfies and supports all the required transactions.

There are three main types of transactions that databases should support.

1. Retrieval
2. Update
3. Mixed

10.7.2 User interface design

Read *section 9.8.2* of the textbook for a number of useful user interface design guidelines.

10.8 Prototyping

A prototype is a working model that is used to:

- identify features of a system that work well, or are inadequate.
- suggest improvements or even new features.
- clarify the users' requirements.
- evaluate feasibility of a particular system design.

10.9 Implementation

The database implementation is the stage for the physical realization of the database design.

Database schemas, user views and empty database files are created using DDL.

The application programs are implemented using 3GL or 4GL. Database transactions are created using DML possibly embedded in a host programming language.

10.10 Data conversion and loading

Is only required when new database system is replacing an old system.

10.11 Testing

Testing of the database application is to execute application programs with the intent of finding errors. It is the last stage before the delivery of the product.

The final product should demonstrate that database and application programs **appear** to be working according to requirements.

Testing provides the measurement of software reliability and quality.

10.12 Operational maintenance

Once the database application is installed, the monitoring and maintenance of the system are required.

The activities being:

- Monitoring performance of the system.
- Maintaining and upgrading the database application (when required).
- Incorporating new requirements into the database application.

10.13 CASE tools

Computer-Aided Software Engineering (CASE) tools are efficient and effective ways for software engineering and database development activities.

CASE tools may be divided into three categories:

1. Upper-CASE
2. Lower-CASE
3. Integrated-CASE

From *section 9.14* of the text you should understand what tools should be provided by CASE (see page 294) and what are the benefits of using CASE Tools (see page 296).

10.14 Data administration (DA) and database administration (DBA)

DA is more concerned with early stages of the database design lifecycle, and DBA is more concerned with later stages of database physical realization and management.

Data administration involves:

- management of data resource including:
 - database planning
 - development and maintenance of standards, policies and procedures, and conceptual and logical database design.

Database administration involves:

- management of the physical realization of a database application including:
 - physical database design and implementation
 - setting security and integrity controls
 - monitoring system performance, and reorganizing the database.

Activity 10.1

- Read the *Chapter Summary* and answer the *Review Questions: 9.1–9.12* on page 301 of the textbook.
- Attempt *exercises 9.13–9.15* from page 301 of the textbook.

10.15 Fact-finding techniques

This section covers Chapter 10 of the textbook. It describes a technique that may be used in the database application lifecycle.

Fact-finding techniques is a formal process of using techniques such as interviews and questionnaires to collect facts about systems, requirements, and preferences.

10.15.1 When are fact-finding techniques used?

Fact-finding is crucial to early stages of the database design including database planning, system definition, requirements collection and analysis stages. It enables the developer to learn about the terminology, problems, opportunities, constraints, requirements, and priorities of the organization and the users of the system.

10.15.2 What types of facts are collected?

Read *section 10.2* for the descriptions of data captured and types of documentation produced in each stage of the database application lifecycle. Table 10.1 on page 304 of the textbook lists the items.

Five most commonly used fact-finding techniques:

1. Examining documentation
2. Interviewing
3. Observing organization in operation
4. Research
5. Questionnaires.

1. Examining documentation

The examining documentation is useful:

- to gain some insight as to how the need for a database arose
- to identify the part of the organization associated with the problem
- to understand the current system.

2. Interviewing

An interview enables collection of information from individuals face-to-face. Its objectives include:

- finding out facts
- verifying facts
- clarifying facts
- generating enthusiasm
- getting the end-user involved
- identifying requirements
- gathering ideas and opinions.

There are two types of interviews: unstructured and structured.

Interview techniques for questions:

- Open-ended questions allow the interviewee to respond in any way that seems appropriate.
- Closed-ended questions restrict answers to either specific choices or short, direct responses.

3. Observing the organization in operation

This is an effective way to learn about the system if one can possibly participate, or watch a person perform activities, in the real environment.

It is also important when the validity of data collected is in question or when the complexity of certain aspects of a system prevents a clear explanation by end-users.

4. Research

Research application is a useful fact-finding technique.

Using sources such as computer trade journals, reference books, and the Internet (including user groups and bulletin boards) can provide information on how others have solved similar problems.

5. Questionnaires

Conducting surveys through questionnaires is another fact-finding technique.

There are two types of questions:

1. Free-format
2. Fixed-format.

10.16 Using fact-finding techniques – a worked example

Section 10.4 of the textbook demonstrates an example: the *DreamHome* case study uses fact-finding techniques in the early stages of the database application lifecycle.

Read the *DreamHome* case study first to understand the requirement of the database application. Then go through *sections 10.4.2–10.4.4* for the steps in developing database applications.

Section 10.4.2 describes how to accomplish **database planning**. Its activities include:

- creating the mission statement of the *DreamHome* database application
- creating the mission objectives for the *DreamHome* database application.

The fact-finding technique: interviewing is used in this stage.

Section 10.4.3 describes the **system definition** stage. The tasks include:

- defining the systems boundary for the *DreamHome* database application

- identifying the major user views for the *DreamHome* database application.

Section 10.4.4 demonstrates the activities of **requirements collection and analysis** for the *DreamHome* database application. The purpose of this stage is to create:

- a user's requirements specification
- a system specification.

The important activities are:

- Gathering more information on the user views of the *DreamHome* database application.
- Gathering information on the system requirements of the *DreamHome* database application.
- Managing the user views of the *DreamHome* database application.
- Writing the system specification for the *DreamHome* database application.

The documentation created in the above stages are the sources of information for the next stage of the lifecycle called **database design**. Chapters 14–16 will discuss steps of the database design.

Activity 10.2

- Reflect on your understanding of the concepts from the *Chapter Summary*.
 - Answer *Review Questions*: 10.1–10.8 on page 320 of the textbook.
 - Attempt *exercises* 10.9–10.13 from page 320 of the textbook.
-

Case study 10.1

Database Design & Implementation

This is for your exercise ONLY.

1. Database Design

Create and validate the global logical data model for the relational model from the local conceptual data models of the University Accommodation Office case study in assignment 2.

You may go through the process of database design discussed on page 420–22 of the textbook to modify your EER model. You need not write down this process.

You should follow steps in 440–72 to create global data model, and you may skip some steps if not applicable. Results are a set of relations and a global diagram as on pages 470–1. You need to give certain details for the process.

Hints:

You should put much emphasis on 2.1 and 2.2.

You may have only one user's view so you can skip step 3 totally.

You need not give much detail for normalization.

You can use queries in the next question to validate your global model.

You should notice differences between diagrams on page 471 and on page 443.

You should follow strictly UML (Unified Modelling Language) as in the textbook.

2. Implementation

In the answers for this question, you need to show you have implemented a database as required. You can demonstrate this by attaching SQL statements (commands) and outputs. You'd be better off not using Microsoft Access.

1. Briefly describe the purpose of main steps in the physical design methodology in the textbook.
2. Implement the University Accommodation Office database using the target DBMS.

You are required to use SQL language to create and populate the database.

You must define the constraints in each relation, e.g., the primary key and foreign keys etc.

You need to populate tables by some values, and each should have at least five rows.

3. Query University Accommodation Office database:

- List the Manager's name and telephone number for each hall of residence.
 - List the details of all students currently on the waiting list for accommodation, that is, not placed.
 - Display the minimum, maximum, and average monthly rent for rooms in halls of residence.
 - Display the total number of students in each student category.
 - Present a report listing the names and matriculation numbers of students with the details of their lease agreements.
-

Module 11

Methodology – conceptual, logical,
and physical database design

Learning objectives

On successful completion of this module, you should be able to:

- design conceptual and logical databases schemas
- decompose the scope of the design into a specific user's view of an enterprise
- use Entity-Relationship (ER) modelling to build a local conceptual data model for a given user's view of an enterprise
- ensure the resultant conceptual model accurately represents the view of the enterprise
- document the process of conceptual database design
- map a local conceptual model to a local logical data model
- derive relations from a local logical data model
- check a logical data model using the technique of normalization and against the transactions it is required to support
- merge local logical data models based on a specific user view into a global logical data model of the enterprise
- ensure that the resultant global model is a true and accurate representation of the part of the enterprise we are attempting to model
- map the logical database design to a physical database design
- design base relations for the target DBMS
- design enterprise constraints for the target DBMS
- select appropriate file organizations based on analysis of transactions
- use secondary indexes to improve performance
- estimate the size of the database
- design security mechanisms to satisfy user requirements.

Introduction

This module covers chapters 15, 16, and 17 of the textbook. Only a week and a half are set aside for the study of this Module. Therefore, the aim is that you familiarize yourself with the content of these chapters, without going too deeply into details. Only Section 16.1 in the text book needs to be studied thoroughly.

11.1 Conceptual and logical database design

There are many stages of database application development. This module discusses the database design stage only.

Database design methodology consists of three main phases:

1. Conceptual database design
2. Logical database design
3. Physical database design.

Chapter 15 presents a database methodology with the definition of three phases. Read the textbook to understand the three phases and the critical success factors in database design.

11.2 Methodology overview – conceptual database design

Section 15.2 describes the steps of three phases of database design, while *section 15.3* focuses on conceptual database modelling.

Conceptual database design is entirely independent of implementation details such as the target DBMS software, application programs, programming languages, hardware platform, or any other physical considerations. A local conceptual data model is created for each user's view of the enterprise.

Section 15.3 demonstrates how to use the methodology to create a conceptual database design for the *DreamHome* case study.

There are nine steps in conceptual data modelling. There are objectives for each step. The basic techniques you must know are:

- to identify entity types and the relationships between them
- determine candidate and primary key attributes of entity types to produce a visual ER model and the document.

You should apply the steps to list the input source and the output result of each phase of database design methodology.

The conceptual database design:

- input the requirements of the enterprise
- output local conceptual data models for all user views

Activity 11.1

- After studying *chapter 15*, you should review the concepts again by reading the *Chapter Summary* and answering *Review Questions: 15.1–15.12* from the textbook.

- Attempt *exercises* 15.13–15.18 from the textbook.

11.3 Methodology overview – logical database design for relational model

Chapter 16 discusses the second phase of database design – the logical database design methodology for the relational model.

Logical database design maps the local conceptual data models on to the local logical data models of the enterprise, which is influenced by the target data model for the database (for example, the relational data model). But it is independent of the physical database environment.

The logical database design methodology for the relational model:

- Input: a conceptual data model
- Output: a logical data model (a relational schema)

The logical design steps are illustrated by example to create a logical database design for the *DreamHome* case study in the textbook.

The main steps of logical database design methodology for the relational model include:

1. building and validating a local logical data model for each user view (Step 2)
2. building and validating a global logical data model (Step 3).

Each conceptual data model resulting from the first phase of database design needs to be refined and further validated. The activities include having to:

- remove M:N relationships
- remove complex relationships
- remove recursive relationships
- remove relationships with attributes
- remove multi-valued attributes
- re-examine 1:1 relationships
- remove redundant relationships.

Refer to the example in the textbook for explanations.

When the conceptual model or ER model has been validated to accurately represent the enterprise, the mapping from the ER model to the relational model is performed. A set of relation schemas are produced, and this is the logical data model of the enterprise.

The relation schemas can be validated using the technique of normalization discussed in *Chapter 13 and 14*.

- Normalization is used to improve the model so that it satisfies various constraints that avoid unnecessary duplication of data. Normalization ensures that the resultant model is a closer model of the enterprise that it serves, it is consistent, and has minimal redundancy and maximum stability.
- Two possible approaches to ensure that the logical data model supports the required transactions include: checking that all the information (entities, relationships, and their attributes) required by each transaction is provided by the model in documenting a description of each transaction's requirements, and diagrammatically representing the pathway taken by each transaction directly on the ER diagram.
- Integrity constraints are the constraints that we wish to impose in order to protect the database from becoming inconsistent. There are five types of integrity constraints: required data, attribute domain constraints, entity integrity, referential integrity, and enterprise constraints.
- To ensure referential integrity, we specify existence constraints, which define conditions under which a candidate key or foreign key may be inserted, updated, or deleted.
- There are several strategies to consider when there exists a child occurrence referencing the parent occurrence that we are attempting to delete: NO ACTION, CASCADE, SET NULL, SET DEFAULT, and NO CHECK.
- Enterprise constraints are sometimes called business rules. For example, updates to entities may be constrained by enterprise rules governing the 'real world' transactions that are represented by the updates.
- The logical data model is supported by documentation, such as the data dictionary and relational schema, which is produced throughout the development of the model.

11.4 Comparison of logical and physical database design

The output of the logical database design constitutes the input source for the physical design process.

Logical database design is independent of the DBMS software, while physical database design uses a particular database language for the implementation of the database on a target DBMS.

11.5 Methodology overview – physical database design for relational databases

Physical database design is the process of producing a description of the implementation of the database on a target database management system (DBMS).

The physical database design phase allows the designer to make decisions on how the database is to be implemented. Therefore, physical design is tailored to a specific DBMS. There is feedback between physical and logical design, because decisions taken during physical design to improve performance may affect the structure of the logical data model.

There are six main steps (from step 4–step 9) of physical database design in the textbook.

The initial step (Step 4) of physical database design is to produce a relational database schema that can be implemented in the target DBMS from the global logical data model.

To do this, the designer needs to know functionalities of the target DBMS, such as how to create base relations and whether the system supports the definition of:

- PKs, FKs, and AKs
- required data - i.e. whether system supports NOT NULL
- domains
- relational integrity constraints
- enterprise constraints.

At the end of step 4, the base relations and all related constraints are defined in target DBMS language.

The next step (Step 5) designs the file organizations and access methods that will be used to store the base relations. This involves

- analysing the transactions that will run on the database
- choosing suitable file organizations based on this analysis
- adding secondary indexes
- introducing controlled redundancy to improve performance
- finally estimating the disk space that will be required by the implementation.

The textbook illustrates the steps of physical database design using the example of the *DreamHome* case study. Read the textbook to understand the process for each step.

11.6 File organizations

You should be familiar with a number of types of file organization used in database storage:

- Heap;
- Hash;
- Indexed Sequential Access Method (ISAM);
- B+ Tree;
- Clusters.

Please refer to *Appendix C – File Organization and Storage Structure* in the textbook for details.

Heap files are good for inserting a large number of records into the file. They are inappropriate when only selected records are to be retrieved.

Hash files are good when retrieval is based on an exact key match. They are not good when retrieval is based on pattern matching, range of values, part keys, or when retrieval is based on an attribute other than the hash field.

ISAM is a more versatile storage structure than hashing. It supports retrievals based on exact key match, pattern matching, range of values, and part key specification. However, the ISAM index is static, created when the file is created. Thus, the performance of an ISAM file will deteriorate as the relation is updated. Updates also cause the ISAM file to lose the access key sequence, so that retrievals in order of the access key will become slower. These two problems are overcome by the **B+ Tree** file organization, which has a dynamic index. However, unlike B+ Tree, because the index is static, concurrent access to the index can be easily managed. If the relation is not frequently updated or not very large nor likely to be, the ISAM structure may be more efficient as it has one less level of index than the B+ Tree, whose leaf nodes contain record pointers.

Secondary indexes provide a mechanism for specifying an additional key for a base relation that can be used to retrieve data more efficiently. However, there is an overhead involved in the maintenance and use of secondary indexes that has to be balanced against the performance improvement gained when retrieving data.

Activity 11.2

- Answer *Review Questions*: 16.1–16.8 from the textbook.
 - Attempt *exercises* 16.9–16.15 from the textbook.
 - Read the *Chapter Summary* and answer *Review Questions*: 17.1–17.4 on page 517 of the textbook.
 - Attempt *exercises* 17.5–17.14 from the textbook.
-

Module 12

Web Technology and DBMSs

Learning objectives

On successful completion of this module, you should be able to:

- describe the basics of the Internet and the World Wide Web
- list the advantages and disadvantages of the Web as a database platform
- understand the basics of Server-side scripting to integrate a database into the Web environment.

Introduction

This module covers part of chapter 29 of the textbook (fourth edition). We only cover a very introductory part to demonstrate the potential of database applications on the Internet. You will find that the material covered here is truly enabling; everything you have learned about databases in this course can now be put to exciting practical use.

12.1 Introduction to the Internet and the Web

If you haven't taken the course **CSC2406 Web Publishing**, Section 29.1 and the first five sections of 29.2 offer a good introduction to the Internet and the World Wide Web. It is especially important that you understand the basic architecture behind the web: web servers such as Apache provide web pages written in HTML stored on a network computer to clients, also called browsers, using the HTTP protocol.

12.2 Dynamic Websites and Databases

Typically, very basic personal World Wide Web home pages of your friends or family members are **static**; i.e., the content is encoded in an HTML file which needs to be changed when new information should be presented in the website. In contrast, almost all serious websites of large companies are **dynamic**: content changes all the time, but the underlying web pages are not modified. Instead, the changing data is present in a database, and the web page displays the data stored in the DBMS.

Read sections 29.2.6 – 29.2.8 in the text book to get an appreciation of the potential and drawbacks of integrating the Web and databases. Then read section 29.3 about scripting languages. A very popular approach, and the one we will use in practicals, is to use the LAMP platform: Linux Apache MySQL PHP. Hence, the web server Apache runs on a Linux machine. The web pages are created using the PHP scripting language, querying a MySQL database that resides on the same Linux server.

Lamp is now also available for Windows, and is therefore sometimes called XAMPP to indicate platform independence. If you're taking this course as an external student, you are encouraged (but not required) to download and install XAMPP on your personal computer, and try the lab exercises posted on the course web site. You will need to read XAMPP's documentation to get everything working.

We do not discuss the remaining sections of Chapter 29. Some of its topics are discussed in the course “Web Publishing”.

Module 13

Solutions

Introduction

Note that the question numbers and chapter numbers are following the textbook not this study book.

13.1 Chapter 4 Relational Algebra and Relational Calculus

Solutions and hints of some exercises

For the following exercises, use the Hotel schema defined at the start of the Exercises at the end of Chapter 3.

4.8 Describe the relations that would be produced by the following relational algebra operations:

a) $\Pi_{\text{hotelNo}} (\sigma_{\text{price} > 50} (\text{Room}))$

This will produce a relation with a single attribute (hotelNo) giving the number of those hotels with a room price greater than £50.

b) $\sigma_{\text{Hotel.hotelNo} = \text{Room.hotelNo}} (\text{Hotel} \times \text{Room})$

This will produce a join of the Hotel and Room relations containing all the attributes of both Hotel and Room (there will be two copies of the hotelNo attribute). Essentially this will produce a relation containing all rooms at all hotels.

c) $\Pi_{\text{hotelName}} (\text{Hotel} \bowtie_{\text{Hotel.hotelNo} = \text{Room.hotelNo}} (\sigma_{\text{price} > 50} (\text{Room})))$

This will produce a join of Hotel and those tuples of Room with a price greater than £50. Essentially this will produce a relation containing all hotel names with a room price above £50.

d) $\text{Guest} \bowtie (\sigma_{\text{dateTo} \geq '1-Jan-2002'} (\text{Booking}))$

This will produce a (left outer) join of Guest and those tuples of Booking with an end date (dateTo) greater than or equal to 1-Jan-2002. All guests who don't have a booking with such a date will still be included in the join. Essentially this will produce a relation containing all guests and show the details of any bookings they have beyond 1-Jan-2002.

e) $\text{Hotel} \triangleright_{\text{Hotel.hotelNo} = \text{Room.hotelNo}} (\sigma_{\text{price} > 50} (\text{Room}))$

This will produce a (semi) join of Hotel and those tuples of Room with a price greater than £50. Only those Hotel attributes will be listed. Essentially this will produce a relation containing all the details of all hotels with a room price above £50.

f) $\Pi_{\text{guestName, hotelNo}} (\text{Booking} \bowtie_{\text{Booking.guestNo} = \text{Guest.guestNo}} \text{Guest}) \div \Pi_{\text{hotelNo}} (\sigma_{\text{city} = 'London'} (\text{Hotel}))$

This will produce a relation containing the names of all guest who have booked all hotels in London.

4.9 Provide the equivalent tuple relational calculus and domain relational calculus expressions for each of the relational algebra queries given in Exercise 4.8.

a) $\Pi_{\text{hotelNo}} (\sigma_{\text{price} > 50} (\text{Room}))$

TRC: $\{R.\text{hotelNo} \mid \text{Room}(R) \wedge R.\text{price} > 50\}$

DRC: $\{\text{hotelNo} \mid (\exists r\text{No}, \text{typ}, \text{prce}) (\text{Room}(r\text{No}, \text{hotelNo}, \text{typ}, \text{prce}) \wedge \text{prce} > 50)\}$

b) $\sigma_{\text{Hotel.hotelNo} = \text{Room.hotelNo}} (\text{Hotel} \times \text{Room})$

TRC: $\{H, R \mid \text{Hotel}(H) \wedge (\exists R) (\text{Room}(R) \wedge (H.\text{hotelNo} = R.\text{hotelNo}))\}$

DRC: $\{h\text{No}, h\text{Name}, \text{cty}, r\text{No}, h\text{No1}, \text{typ}, \text{prce} \mid (\text{Hotel}(h\text{No}, h\text{Name}, \text{cty}) \wedge \text{Room}(r\text{No}, h\text{No1}, \text{typ}, \text{prce}) \wedge (h\text{No} = h\text{No1}))\}$

c) $\Pi_{\text{hotelName}} (\text{Hotel}_{\text{Hotel.hotelNo} = \text{Room.hotelNo}} (\sigma_{\text{price} > 50} (\text{Room})))$

TRC: $\{H.\text{hotelName} \mid \text{Hotel}(H) \wedge (\exists R) (\text{Room}(R) \wedge (H.\text{hotelNo} = R.\text{hotelNo}) \wedge (R.\text{price} > 50))\}$

DRC: $\{\text{hotelName} \mid (\exists h\text{No}, \text{cty}, r\text{No}, h\text{No1}, \text{typ}, \text{prce}) (\text{Hotel}(h\text{No}, \text{hotelName}, \text{cty}) \wedge \text{Room}(r\text{No}, h\text{No1}, \text{typ}, \text{prce}) \wedge (h\text{No} = h\text{No1}) \wedge (\text{prce} > 50))\}$

d) $\text{Guest} \times (\sigma_{\text{dateTo} \geq '1\text{-Jan-2002}'} (\text{Booking}))$

TRC: $\{G.\text{guestNo}, G.\text{guestName}, G.\text{guestAddress}, B.\text{hotelNo}, B.\text{dateFrom}, B.\text{dateTo}, B.\text{roomNo} \mid \text{Guest}(G) \vee (\exists B) (\text{Booking}(B) \wedge (G.\text{guestNo} = B.\text{guestNo}) \wedge (B.\text{dateTo} > '1\text{-Jan-2002}'))\}$

DRC: $\{\text{guestNo}, \text{guestName}, \text{guestAddress}, \text{hotelNo}, \text{dateFrom}, \text{dateTo}, \text{roomNo} \mid (\exists g\text{No1}) (\text{Guest}(\text{guestNo}, \text{guestName}, \text{guestAddress}) \vee (\text{Booking}(\text{hotelNo}, g\text{No1}, \text{dateFrom}, \text{dateTo}, \text{roomNo}) \wedge (\text{guestNo} = g\text{No1}) \wedge (\text{dateTo} \checkmark '1\text{-Jan-2002}')))\}$

e) $\text{Hotel} \triangleright_{\text{Hotel.hotelNo} = \text{Room.hotelNo}} (\sigma_{\text{price} > 50} (\text{Room}))$

TRC: $\{H.\text{hotelNo}, H.\text{hotelName}, H.\text{city} \mid \text{Hotel}(H) \wedge (\exists R) (\text{Room}(R) \wedge (H.\text{hotelNo} = R.\text{hotelNo}) \wedge (R.\text{price} > 50))\}$

DRC: $\{\text{hotelNo}, \text{hotelName}, \text{city} \mid (\exists r\text{No}, h\text{No1}, \text{typ}, \text{prce}) (\text{Hotel}(\text{hotelNo}, \text{hotelName}, \text{city}) \wedge \text{Room}(r\text{No}, h\text{No1}, \text{typ}, \text{prce}) \wedge (\text{hotelNo} = h\text{No1}) \wedge (\text{prce} > 50))\}$

f) $\Pi_{\text{guestName}, \text{hotelNo}} (\text{Booking} \bowtie_{\text{Booking.guestNo} = \text{Guest.guestNo}} \text{Guest}) \div \Pi_{\text{hotelNo}} (\sigma_{\text{city} = 'London'} (\text{Hotel}))$

TRC: $\{G.\text{guestName} \mid \text{Guest}(G) \wedge (\sim (\exists H) (\text{Hotel}(H) \wedge (H.\text{city} = \text{'London'}) \wedge (\sim (\exists B) (\text{Booking}(B) \wedge G.\text{guestNo} = B.\text{guestNo} \wedge H.\text{hotelNo} = B.\text{hotelNo}))))\}$

DRC: $\{\text{guestName} \mid (\exists g\text{No}, g\text{Name}, g\text{Address}, h\text{No}, g\text{No1}, d\text{From}, d\text{To}, r\text{No}, h\text{Name}, \text{cty}, h\text{No1}, \text{typ}, \text{prce}) (\sim (\text{Hotel}(h\text{No}, h\text{Name}, \text{cty}) \wedge (\text{cty} = \text{'London'}) \wedge \text{Guest}(g\text{No}, g\text{Name}, g\text{Address}) \wedge \text{Booking}(h\text{No1}, g\text{No1}, d\text{From}, d\text{To}, r\text{No}) \wedge (g\text{No} = g\text{No1}) \wedge (h\text{No} = h\text{No1}))))\}$

4.11 Provide the equivalent domain relational calculus and relational algebra expressions for each of the tuple relational calculus expressions given in Exercise 4.10.

(a) $\{H.\text{hotelName} \mid \text{Hotel}(H) \wedge H.\text{city} = \text{'London'}\}$

DRC: $\{\text{hotelName} \mid (\exists h\text{No}, \text{cty}) (\text{Hotel}(h\text{No}, \text{hotelName}, \text{cty}) \wedge \text{cty} = \text{'London'})\}$

RA: $\Pi_{\text{hotelName}} (\sigma_{\text{city} = \text{'London'}} (\text{Hotel}))$

(b) $\{H.\text{hotelName} \mid \text{Hotel}(H) \wedge (\exists R) (\text{Room}(R) \wedge H.\text{hotelNo} = R.\text{hotelNo} \wedge R.\text{price} > 50)\}$

DRC: $\{\text{hotelName} \mid (\exists h\text{No}, \text{cty}, r\text{No}, h\text{No1}, \text{typ}, \text{prce}) (\text{Hotel}(h\text{No}, \text{hotelName}, \text{cty}) \wedge \text{Room}(r\text{No}, h\text{No1}, \text{typ}, \text{prce}) \wedge (h\text{No} = h\text{No1}) \wedge (\text{prce} > 50))\}$

RA: $\Pi_{\text{hotelName}} (\text{Hotel} \bowtie_{\text{Hotel.hotelNo} = \text{Room.hotelNo}} (\sigma_{\text{price} > 50} (\text{Room})))$

RA: $\Pi_{\text{hotelName}} (\sigma_{\text{guestName} = \text{'John Smith'}} (\text{Guest}) \bowtie_{\text{Guest.guestNo} = \text{guestNo}} (\text{Booking} \bowtie_{\text{Booking.hotelNo} = \text{Hotel.hotelNo}} \text{Hotel}))$

(c) $\{H.\text{hotelName}, G.\text{guestName}, B1.\text{dateFrom}, B2.\text{dateFrom} \mid \text{Hotel}(H) \wedge \text{Guest}(G) \wedge \text{Booking}(B1) \wedge \text{Booking}(B2) \wedge H.\text{hotelNo} = B1.\text{hotelNo} \wedge G.\text{guestNo} = B1.\text{guestNo} \wedge B2.\text{hotelNo} = B1.\text{hotelNo} \wedge B2.\text{guestNo} = B1.\text{guestNo} \wedge B2.\text{dateFrom} \neq B1.\text{dateFrom}\}$

DRC: $\{\text{hotelName}, \text{guestName}, \text{dateFrom1}, \text{dateFrom2} \mid (\exists h\text{No}, \text{cty}, g\text{No}, g\text{Address}, h\text{No1}, g\text{No1}, d\text{To1}, r\text{No1}, h\text{No2}, g\text{No2}, d\text{To2}, r\text{No2}) (\text{Hotel}(h\text{No}, \text{hotelName}, \text{cty}) \wedge \text{Guest}(g\text{No}, \text{guestName}, g\text{Address}) \wedge \text{Booking}(h\text{No1}, g\text{No1}, \text{dateFrom1}, d\text{To1}, r\text{No1}) \wedge \text{Booking}(h\text{No2}, g\text{No2}, \text{dateFrom2}, d\text{To2}, r\text{No2}) \wedge (h\text{No} = h\text{No1}) \wedge (g\text{No} = g\text{No1}) \wedge (h\text{No2} = h\text{No1}) \wedge (g\text{No2} = g\text{No1}) \wedge (\text{dateFrom1} \neq \text{dateFrom2}))\}$

RA: $\text{Booking2}(\text{hotelNo}, \text{guestNo}, \text{dateFrom2}, \text{dateTo2}, \text{roomNo2}) \leftarrow \Pi_{\text{hotelNo}, \text{guestNo}, \text{dateFrom}, \text{dateTo}, \text{roomNo}} (\text{Booking}) \Pi_{\text{hotelName}, \text{guestName}, \text{dateFrom}, \text{dateFrom2}} (\text{Hotel} \bowtie_{\text{Hotel.hotelNo} = \text{hotelNo}} (\text{Guest} \bowtie_{\text{Guest.guestNo} = \text{guestNo}} (\text{Booking} \bowtie_{\text{Booking.hotelNo} = \text{Booking2.hotelNo} \wedge \text{Booking.guestNo} = \text{Booking2.guestNo} \wedge \text{dateFrom} \neq \text{dateFrom2}} \text{Booking2})))$

4.12 Generate the relational algebra, tuple relational calculus, and domain relational calculus expressions for the following queries:

(a) List all hotels.

RA: Hotel

TRC: {H | Hotel(H)}

DRC: {hotelNo, hotelName, city | Hotel(hotelNo, hotelName, city)}

(b) List all single rooms with a price below £20 per night.

RA: $\sigma_{\text{type}='S' \wedge \text{price} < 20}(\text{Room})$

TRC: {R | Room(R) \wedge R.type = 'S' \wedge R.price < 20}

DRC: {roomNo, hotelNo, type, price | (Room(roomNo, hotelNo, type, price) \wedge type = 'S' \wedge price < 20)}

(c) List the names and cities of all guests.

RA: $\Pi_{\text{guestName, guestAddress}}(\text{Guest})$

TRC: {G.guestName, G.guestAddress | Guest(G)}

DRC: {guestName, guestAddress | (\exists guestNo) (Guest(guestNo, guestName, guestAddress))}

(d) List the price and type of all rooms at the Grosvenor Hotel.

RA: $\Pi_{\text{price, type}}(\text{Room} \bowtie_{\text{hotelNo}} (\sigma_{\text{hotelName}='Grosvenor Hotel'}(\text{Hotel})))$

TRC: {R.price, R.type | Room(R) \wedge (\exists H) (Hotel(H) \wedge (R.hotelNo = H.hotelNo) \wedge (H.hotelName = 'Grosvenor Hotel'))}

DRC: {price, type | (\exists roomNo, hotelNo, hotelNo1, hotelName, city) (Room(roomNo, hotelNo, type, price) \wedge Hotel(hotelNo1, hotelName, city) \wedge (hotelNo = hotelNo1) \wedge (hotelName = 'Grosvenor Hotel'))}

4.13 $\Pi_{\text{roomNo, hotelNo, type}}(\text{Room} \bowtie_{\text{hotelNo}} (\sigma_{\text{hotelName}='Grosvenor Hotel'}(\text{Hotel})))$

Security – hides the price details from people who should not see it.

Reduced complexity – a query against this view is simpler than a query against the two underlying base relations.

13.2 Chapter 5 SQL: Data Manipulation

Solutions and hints of some exercises

Simple Queries

5.9 `SELECT guestName, guestAddress FROM Guest
WHERE address LIKE '%London%' ORDER BY guestName;`

Strictly speaking, this would also find rows with an address like: '10 London Avenue, New York'.

5.10 `SELECT * FROM Room WHERE price < 40 AND type IN ('D', 'F')
ORDER BY price;`

(Note, ASC is the default setting).

5.11 `SELECT * FROM Booking WHERE dateTo IS NULL;`

Aggregate Functions

5.12 `SELECT COUNT(*) FROM Hotel;`

5.13 `SELECT AVG(price) FROM Room;`

5.15 `SELECT COUNT(DISTINCT guestNo) FROM Booking
WHERE (dateFrom <= DATE'2001-08-01' AND dateTo >= DATE'2001-08-01')
OR (dateFrom >= DATE'2001-08-01' AND dateFrom <= DATE'2001-08-31');`

Subqueries and Joins

5.16 `SELECT price, type
FROM Room
WHERE hotelNo =
(SELECT hotelNo
FROM Hotel
WHERE hotelName = 'Grosvenor Hotel');`

5.18 `SELECT r.* FROM Room r LEFT JOIN
(SELECT g.guestName, h.hotelNo, b.roomNo
FROM Guest g, Booking b, Hotel h
WHERE g.guestNo = b.guestNo AND
b.hotelNo = h.hotelNo AND
hotelName= 'Grosvenor Hotel' AND
dateFrom <= CURRENT_DATE AND
dateTo >= CURRENT_DATE) AS XXX
ON r.hotelNo = XXX.hotelNo AND r.roomNo = XXX.roomNo;`

5.21 `SELECT SUM(price)
FROM Room r
WHERE roomNo NOT IN
(SELECT roomNo FROM Booking b, Hotel h
WHERE (dateFrom <= CURRENT_DATE AND dateTo >= CURRENT_DATE) AND
b.hotelNo = h.hotelNo AND hotelName = 'Grosvenor Hotel');`

Grouping

5.22 `SELECT hotelNo, COUNT(roomNo) AS count FROM Room GROUP BY hotelNo;`

5.24 `SELECT AVG(X)
FROM (SELECT hotelNo, COUNT(hotelNo) AS X
FROM Booking b
WHERE (dateFrom <= DATE'2001-08-01' AND
dateTo >= DATE'2001-08-01') OR
(dateFrom >= DATE'2001-08-01' AND
dateFrom <= DATE'2001-08-31'))
GROUP BY hotelNo);`

Yes - this is legal in SQL-92!

5.26 `SELECT hotelNo, SUM(price) FROM Room r
WHERE roomNo NOT IN
(SELECT roomNo FROM Booking b, Hotel h
WHERE (dateFrom <= CURRENT_DATE AND
dateTo >= CURRENT_DATE) AND
b.hotelNo = h.hotelNo)
GROUP BY hotelNo;`

Creating and Populating Tables

5.27 `INSERT INTO Hotel
VALUES ('H111', 'Grosvenor Hotel', 'London');
INSERT INTO Room
VALUES ('1', 'H111', 'S', 72.00);
INSERT INTO Guest
VALUES ('G111', 'John Smith', 'London');
INSERT INTO Booking
VALUES ('H111', 'G111', DATE'2001-01-01', DATE'2001-01-02', '1');`

5.28 `UPDATE Room SET price = price*1.05;`

5.30 *Show that a query using the HAVING clause has an equivalent formulation without a HAVING clause.*

Hint: Allow the students to show that the restricted groups could have been restricted earlier with a WHERE clause.

5.31 *Show that SQL is relationally complete.*

Hint: Allow the students to show that each of the relational algebra operations can be expressed in SQL.

13.3 Chapter 6 SQL: Data Definition

Solutions and hints of some exercises

```
6.7 CREATE TABLE Hotel(
    hotelNo CHAR(4) NOT NULL,
    hotelName VARCHAR(20) NOT NULL,
    city VARCHAR(50) NOT NULL,
    PRIMARY KEY (hotelNo));
```

Or

```
CREATE DOMAIN HotelNumber AS CHAR(4);
```

```
CREATE TABLE Hotel(
    hotelNo HotelNumber NOT NULL,
    hotelName VARCHAR(20) NOT NULL,
    city VARCHAR(50) NOT NULL,
    PRIMARY KEY (hotelNo));
```

6.8 *Now create the Room, Booking, and Guest tables using the integrity enhancement features of SQL with the following constraints:*

- (a) Type must be one of Single, Double, or Family.
- (b) Price must be between £10 and £100.
- (c) roomNo must be between 1 and 100.
- (d) dateFrom and dateTo must be greater than today's date.
- (e) The same room cannot be double booked.
- (f) The same guest cannot have overlapping bookings.

```
CREATE TABLE Room(
    roomNo VARCHAR(4) NOT NULL
        CHECK(VALUE BETWEEN '1' AND '100'),
    hotelNo CHAR(4) NOT NULL
        CHECK(VALUE IN (SELECT hotelNo FROM Hotel)),
    type CHAR(1) NOT NULL DEFAULT 'S'
        CHECK(VALUE IN ('S', 'F', 'D')),
    price NUMERIC(5, 2) NOT NULL
        CHECK(VALUE BETWEEN 10 AND 100),
    PRIMARY KEY (roomNo, hotelNo),
    FOREIGN KEY (hotelNo) REFERENCES Hotel
    ON DELETE CASCADE ON UPDATE CASCADE);
```

```
CREATE TABLE Guest(
    guestNo CHAR(4) NOT NULL,
    guestName VARCHAR(20) NOT NULL,
    guestAddress VARCHAR(50) NOT NULL);
```

```

CREATE TABLE Booking(
  hotelNo    CHAR(4)    NOT NULL
             CHECK(VALUE IN (SELECT hotelNo FROM Hotel)),
  guestNo    CHAR(4)    NOT NULL,
  dateFrom   DATETIME   NOT NULL
             CHECK(VALUE > CURRENT_DATE),
  dateTo     DATETIME   NULL
             CHECK(VALUE > CURRENT_DATE),
  roomNo     VARCHAR(4) NOT NULL
             CHECK(VALUE BETWEEN '1' AND '100'),
  PRIMARY KEY (hotelNo, guestNo, dateFrom),
  FOREIGN KEY (hotelNo) REFERENCES Hotel
    ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (guestNo) REFERENCES Guest
    ON DELETE NO ACTION ON UPDATE CASCADE,
  FOREIGN KEY (hotelNo, roomNo) REFERENCES Room
    ON DELETE NO ACTION ON UPDATE CASCADE,
  CONSTRAINT RoomBooked CHECK (NOT EXISTS (
    SELECT * FROM Booking b
    WHERE b.dateTo > Booking.dateFrom AND
          b.dateFrom < Booking.dateTo AND
          b.roomNo = Booking.roomNo AND
          b.hotelNo = Booking.hotelNo)),
  CONSTRAINT GuestBooked CHECK (NOT EXISTS (
    SELECT * FROM Booking b
    WHERE b.dateTo > Booking.dateFrom AND
          b.dateFrom < Booking.dateTo AND
          b.guestNo = Booking.guestNo)));

```

Or

```

CREATE DOMAIN RoomType AS CHAR(1)
  CHECK(VALUE IN ('S', 'F', 'D'));
CREATE DOMAIN HotelNumbers AS HotelNumber
  CHECK(VALUE IN (SELECT hotelNo FROM Hotel));
CREATE DOMAIN RoomPrice AS DECIMAL(5, 2)
  CHECK(VALUE BETWEEN 10 AND 100);
CREATE DOMAIN RoomNumber AS VARCHAR(4)
  CHECK(VALUE BETWEEN '1' AND '100');

```

```

CREATE TABLE Room(
  roomNo    RoomNumber    NOT NULL,
  hotelNo   HotelNumbers  NOT NULL,
  type      RoomType      NOT NULL DEFAULT 'S',
  price     RoomPrice     NOT NULL,
  PRIMARY KEY (roomNo, hotelNo),
  FOREIGN KEY (hotelNo) REFERENCES Hotel
    ON DELETE CASCADE ON UPDATE CASCADE);

```

```

CREATE DOMAIN GuestNumber AS CHAR(4);

CREATE TABLE Guest(
  guestNo      GuestNumber NOT NULL,
  guestName    VARCHAR(20) NOT NULL,
  guestAddress VARCHAR(50) NOT NULL);

CREATE DOMAIN GuestNumbers AS GuestNumber
  CHECK(VALUE IN (SELECT guestNo FROM Guest));
CREATE DOMAIN BookingDate AS DATETIME
  CHECK(VALUE > CURRENT_DATE);

CREATE TABLE Booking(
  hotelNo      HotelNumbers NOT NULL,
  guestNo      GuestNumbers NOT NULL,
  dateFrom     BookingDate  NOT NULL,
  dateTo       BookingDate  NULL,
  roomNo       RoomNumber   NOT NULL,
  PRIMARY KEY (hotelNo, guestNo, dateFrom),
  FOREIGN KEY (hotelNo) REFERENCES Hotel
    ON DELETE CASCADE ON UPDATE CASCADE,
  FOREIGN KEY (guestNo) REFERENCES Guest
    ON DELETE NO ACTION ON UPDATE CASCADE,
  FOREIGN KEY (hotelNo, roomNo) REFERENCES Room
    ON DELETE NO ACTION ON UPDATE CASCADE,
  CONSTRAINT RoomBooked CHECK (NOT EXISTS (
    SELECT * FROM Booking b
    WHERE b.dateTo > Booking.dateFrom AND
          b.dateFrom < Booking.dateTo AND
          b.roomNo = Booking.roomNo AND
          b.hotelNo = Booking.hotelNo)),
  CONSTRAINT GuestBooked CHECK (NOT EXISTS (
    SELECT * FROM Booking b
    WHERE b.dateTo > Booking.dateFrom AND
          b.dateFrom < Booking.dateTo AND
          b.guestNo = Booking.guestNo)));

```

```

6.11 CREATE VIEW BookingOutToday AS
      SELECT g.guestNo,g.guestName,g.guestAddress,r.price*(b.dateTo- b.dateFrom)
      FROM Guest g, Booking b, Hotel h, Room r
      WHERE g.guestNo = b.guestNo AND
            r.roomNo = b.roomNo AND
            b.hotelNo = h.hotelNo AND
            h.hotelName = 'Grosvenor Hotel' AND
            b.dateTo = CURRENT_DATE;

```

```
6.13 GRANT SELECT ON HotelData TO Accounts;
GRANT SELECT ON BookingOutToday TO Accounts;

REVOKE SELECT ON HotelData FROM Accounts;
REVOKE SELECT ON BookingOutToday FROM Accounts;
```

General

6.15 *Consider the following table: Part (partNo, contract, partCost) which represents the cost negotiated under each contract for a part (a part may have a different price under each contract).*

Now consider the following view ExpensiveParts, which contains the distinct part numbers for parts that cost more than £1000:

```
CREATE VIEW ExpensiveParts (partNo)
AS SELECT DISTINCT partNo
FROM Part
WHERE partCost > 1000;
```

Discuss how you would maintain this as a materialized view and under what circumstances you would be able to maintain the view without having to access the underlying base table Part.

If a row is inserted into Part with a partCost less than or equal to £1000, the view would not have to be updated. If a partNo is inserted into Part that is already in the view, no new record has to be inserted into the view (because of the DISTINCT keyword). Similarly for update. If a partNo is deleted from Part have to access the underlying base table to check if there is another partNo with same value, to determine whether row should be deleted from the view.

13.4 Chapter 9 Database Planning, Design, and Administration

Solutions and hints of some exercises

- 9.14 The student should follow the approach to DBMS selection described in Section 9.7 and produce a report that identifies a suitable DBMS product that meets the requirements of the organization. The selection should be fully justified and any assumptions made should be highlighted.
- 9.15 The student should follow the approach to DBMS selection described in Section 9.7 and produce a report that identifies a suitable DBMS product that meets the requirements of each organization described in Appendix B. The selection should be fully justified and any assumptions made about the case study should be highlighted.

- 9.16 The student should investigate the organization and identify whether data administration and database administration exist as distinct functional areas. However, the student should be careful to note that these functions may be named differently, merged as a single function, or included as part of a larger IT/IS function. If identified, the student should compile a report documenting the organization, responsibilities, and tasks.

13.5 Chapter 10 Fact-Finding Techniques

Solutions and hints of some exercises

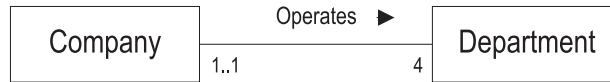
- 10.10 The student could come at this from two directions: first, these are case studies and individuals within the organizations cannot be met or interviewed; second approach is more open, and the student assumes these are real organizations and staff can be interviewed, if necessary. This may involve some form of role play to help the student perform interviews and clarify the requirements or elicit additional information.

13.6 Chapter 11 Entity-Relationship Modeling

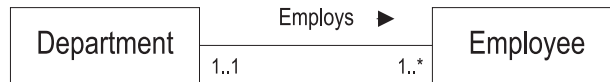
Solutions and hints of some exercises

11.10 Create an ER diagram for each of the following descriptions:

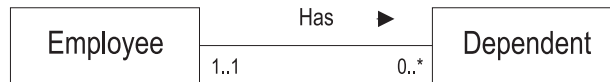
- (a) Each company operates four departments, and each department belongs to one company.



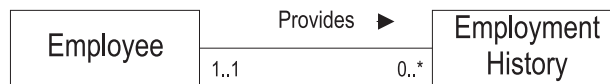
- (b) Each department in part (a) employs one or more employees, and each employee works for one department.



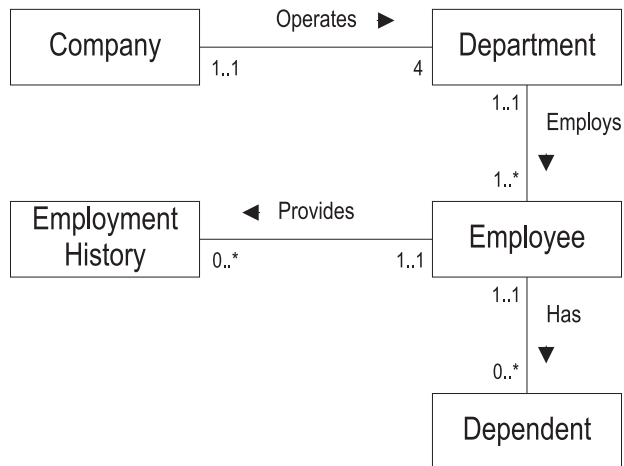
- (c) Each of the employees in part (b) may or may not have one or more dependants, and each dependant belongs to one employee.



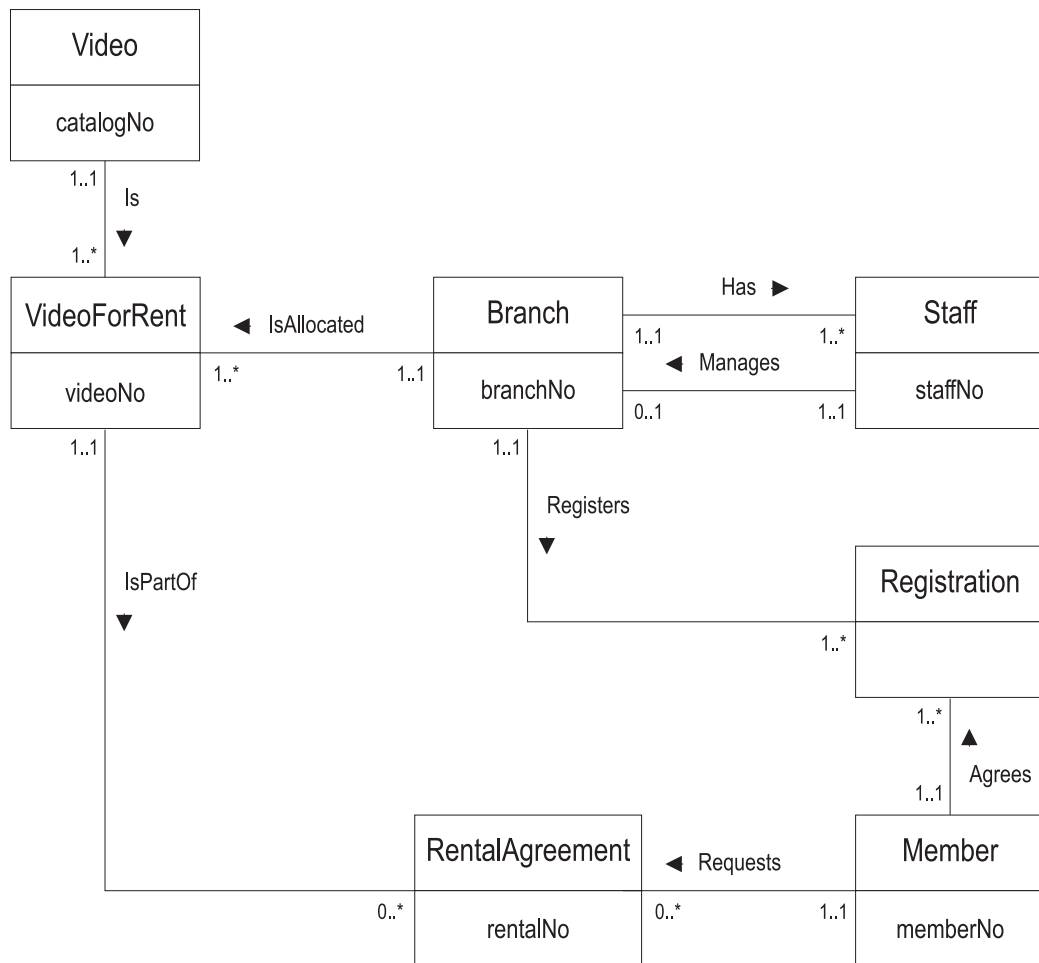
- (d) Each employee in part (c) may or may not have an employment history.



- (e) Represent all the ER diagrams described in (a), (b), (c), and (d) as a single ER diagram.



11.12



13.7 Chapter 12 Enhanced Entity-Relationship Modeling

Solutions and hints of some exercises

- 12.10 Could consider Manager as a specialization of the Staff entity. This would move the Manages relationship from Staff to the Manager subclass. However, the attributes for both entities would be the same and there would, therefore, seem to be no obvious advantage to introducing the Manager specialization.

13.8 Chapter 13 Normalization

Exercises

- 13.16 (a) *Identify the functional dependencies represented by the data shown in the form in Figure 13.25.*

Patient No → Full Name

Ward No → Ward Name

Drug No → Name, Description, Dosage, Method of Admin

Patient No, Drug No, Start Date → Units per Day, Finish date

The functional dependencies for Bed No are unclear. If Bed No was a unique number for the entire hospital, then could say that Bed No → Ward No. However, from further examination of the requirements specification, we can observe that Bed No is to do with the allocation of patients on the waiting list to beds.

- (b) Describe and illustrate the process of normalizing the data shown in Figure 13.25 to first (1NF), second (2NF), third (3NF), and BCNF.

First Normal Form

Patient No, Drug No, Start Date, Full Name, Ward No, Ward Name, Bed No, Name, Description, Dosage, Method of Admin, Units per Day, Finish Date

Second Normal Form

Patient No, Drug No, Start Date, Ward No, Ward Name, Bed No, Units per Day, Finish Date

Drug No, Name, Description, Dosage, Method of Admin

Patient No, Full Name

Third Normal Form/BCNF

Patient No, Drug No, Start Date, Ward No, Bed No, Units per Day, Finish Date

Drug No, Name, Description, Dosage, Method of Admin

Patient No, Full Name

Ward No, Ward Name

- (c) *Identify the primary, alternate, and foreign keys in your BCNF relations.*

Patient No(FK), Drug No(FK), Start Date, Ward No(FK), Bed No, Units per Day, Finish Date

Drug No, Name, Description, Dosage, Method of Admin

Patient No, Full Name

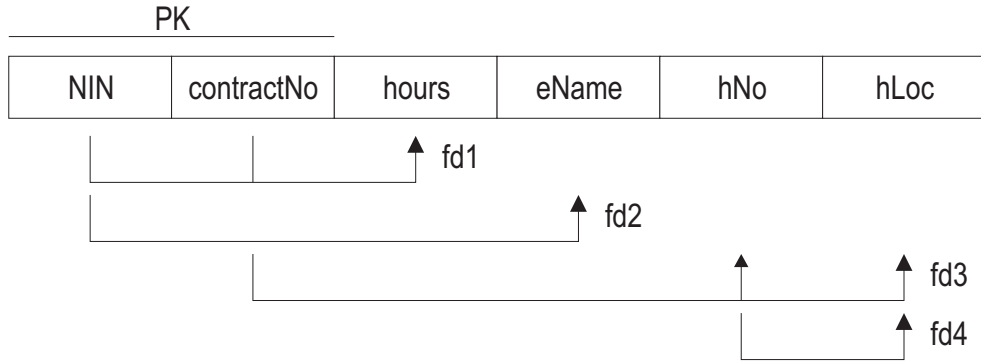
Ward No, Ward Name

Primary keys underlined.

The student should state any assumptions made about the data shown in the table. For example, we may assume that a hotel may be associated with one or more contracts.

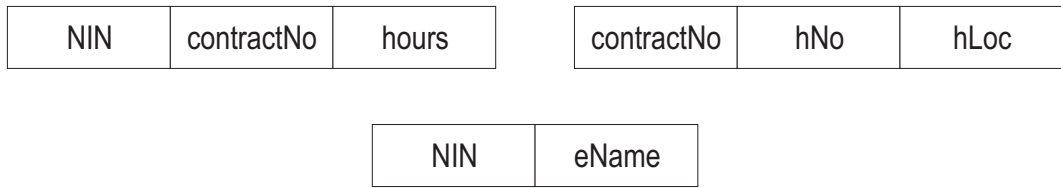
The following is only a diagrammatic illustration of the solution, but in your assignment you need to give more details, for example, function dependencies and explanation.

1NF



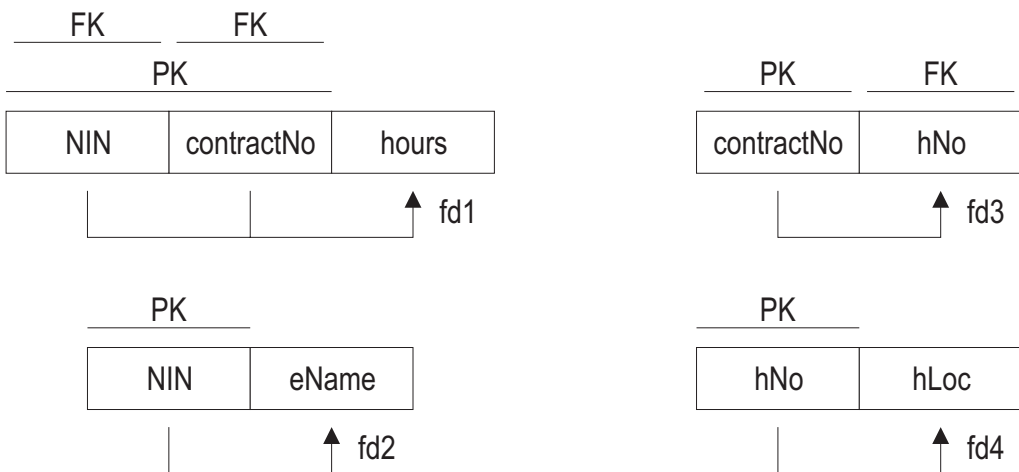
↓ fd2 and fd3 violates 2NF

2NF



↓ fd4 violates 3NF

3NF / BCNF



- 13.19 (a) *Describe why the relation shown in Figure 13.28 is in BCNF and not in 4NF.*

wardName —»» staffName

wardName —»» patientName

Relation is in BCNF but there is a nontrivial multi-valued dependency in the relation, so relation is not in 4NF.

- (b) *The relation shown in Figure 13.28 is susceptible to update anomalies. Provide examples of insertion, deletion, and update anomalies.*

If we wanted to insert a new patient name, would have to add two records, one for each member of staff.

If we wanted to update the name of patient Claire Johnson, we would have to update two records.

If we wanted to delete the record corresponding to patient Claire Johnson, we would have to delete two records.

- (c) *Describe and illustrate the process of normalizing the relation shown in Figure 13.28 to 4NF.*

To remove the MVD, we create two new relations:

WardStaff (wardName, staffName) WardPatient(wardName, patientName)

- 13.20 (a) *Describe why the relation shown in Figure 13.29 is not in 5NF.*

This relation has a join dependency JD(hospitalName, itemDescription, supplierNo) among the three projections: R1(hospitalName, itemDescription), R2(hospitalName, supplierNo), and R3(itemDescription, supplierNo) of HospitalItemSupplier.

- (b) *Describe and illustrate the process of normalizing the relation shown in Figure 13.29 to 5NF.*

To remove the join dependency, we create the following 5NF relations:

HospitalItem(hospitalName, itemDescription) HospitalSupplier(hospitalName, supplierNo) ItemSupplier(itemDescription, supplierNo).