ATME College of Engineering

Computer Science and Design

SUBJECT TITLE AND CODE:DATABASE MANAGEMENT SYSTEMS -21CS53

SEMESTER AND SCHEME :V SEMESTER

INSTITUTIONAL MISSION AND VISION

Objectives

- To provide quality education and groom top-notch professionals, entrepreneurs and leaders for different fields of engineering, technology and management.
- To open a Training-R & D-Design-Consultancy cell in each department, gradually introduce doctoral and postdoctoral programs, encourage basic & applied research in areas of social relevance, and develop the institute as a center of excellence.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To cultivate strong community relationships and involve the students and the staff in local community service.
- To constantly enhance the value of the educational inputs with the participation of students, faculty, parents and industry.

Vision

- Development of academically excellent ,culturally vibrant, socially responsible and globally competent human resources.

Mission

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
- To strive to attain ever- higher benchmarks of educational excellence.

Department of Computer Science And Design

ProgramEducationalObjectives(PEO'S):

1. Empower students with a strong basis in the mathematical, scientific and engineering fundamentals to solve computational problems and to prepare them for employment, higher learning andR&D.
2. Gain technical knowledge, skills and awareness of current technologies of computer science engineering and to develop an ability to design and provide novel engineering solutions for software/hardware problems through entrepreneurial skills.
3. Exposure to emerging technologies and work in teams on interdisciplinary projects with effective communication skills and leadership qualities.
4. Ability to function ethically and responsibly in a rapidly changing environment by applying innovative ideas in the latest technology, to become effective professionals in Computer Science to beara life-long career in related areas.

ProgramSpecificOutcomes(PSOs)

PSO1: Abilitytoapplyskillsinthefieldofalgorithms,databasedesign, webdesign,cloudcomputinganddata analytics.

PSO2: Applyknowledgeinthefieldofcomputernetworksforbuildingnetworkandinternet-basedapplications

Course Code	Course Title	Core/Elective	Prerequisite	Contact Hours			Total Hrs/ Sessions
				L	T	P	
21CS53	Database Management System	Core	-	3	2	-	50
Objectives	1.Provide a strong foundation in database concepts, technology, and practice. 2.Practice SQL programming through a variety of database problems. 3.Demonstrate the use of concurrency and transactions in database 4.Design and build database applications for real world problems.						

Topics Covered as Per Syllabus

Module-1: Introduction to Databases: Introduction, Characteristics of database approach, Advantages of using the DBMS approach, History of database applications. **Overview of DatabaseLanguagesandArchitectures:**DataModels,Schemas,andInstances.Threeschema architecture and data independence, database languages, and interfaces, The Database System environment. **Conceptual Data Modelling using Entities and Relationships:** Entity types, Entity sets, attributes, roles, and structural constraints, Weak entity types, ER diagrams, examples, Specialization and Generalization.

Module-2: Relational Model: Relational Model Concepts, Relational Model Constraints and relational database schemas, Update operations, transactions, and dealing with constraint violations. **Relational Algebra:** Unary and Binary relational operations, additional relational operations (aggregate, grouping, etc.) Examples of Queries in relational algebra. **Mapping ConceptualDesignintoaLogicalDesign:**RelationalDatabaseDesignusingER-to-Relational mapping. **SQL:** SQL data definition and data types, specifying constraints in SQL, retrieval queries in SQL, INSERT, DELETE, and UPDATE statements in SQL, Additional features of SQL.

Module-3: SQL : Advances Queries: More complex SQL retrieval queries, Specifying constraints as assertions and action triggers, Views in SQL, Schema change statements in SQL. **Database Application Development:** Accessing databases from applications, Anintroduction to JDBC, JDBC classes and interfaces, SQLJ, Stored procedures, Case study: The internetbookshop.**InternetApplications:**Thethree-Tierapplicationarchitecture,Thepresentationlayer, The Middle Tier

Module-4: Normalization: Database Design Theory – Introduction to Normalization using Functional and Multivalued Dependencies: Informal design guidelines for relation schema, Functional Dependencies, Normal Forms based on Primary Keys, Second and Third Normal Forms, Boyce-Codd Normal Form, Multivalued Dependency and Fourth Normal Form, Join Dependencies and Fifth Normal Form. **Normalization Algorithms:** Inference Rules, Equivalence, and Minimal Cover, Properties of Relational Decompositions, Algorithms for Relational Database Schema Design, Nulls, Dangling tuples, and alternate Relational Designs, Further discussion of Multivalued dependencies and 4NF, Other dependencies and NormalForms.

Module-5:TransactionProcessing: IntroductiontoTransactionProcessing, Transaction andSystem concepts,Desirablepropertiesof Transactions,Characterizingschedulesbasedon recoverability,CharacterizingschedulesbasedonSerializability,TransactionsupportinSQL.

ConcurrencyControlinDatabases: Two-phaselockingtechniquesforConcurrencycontrol, Concurrency control based on Timestamp ordering, Multiversion Concurrency control techniques, Validation Concurrency control techniques, Granularity ofData items and MultipleGranularity Locking. **Introduction to Database Recovery Protocols:** Recovery Concepts, NO- UNDO/REDO recovery based on Deferred update, Recovery techniques based onimmediate update, Shadow paging, Database backup and recovery from catastrophic failures

ListofTextBooks

1. DatabasesystemsModels,Languages,DesignandApplicationProgramming,RamezElmasri and Shamkant B. Navathe, 7th Edition, 2017, Pearson.
2. Databasemanagementsystems,Ramakrishnan,andGehrke,3rdEdition,2014,McGrawHill

ListofReference Books

1. SilberschatzKorthandSudharshan,DatabaseSystemConcepts,6th Edition,Mc-GrawHill, 2013.
2. Coronel,Morris,andRob,DatabasePrinciplesFundamentalsof Design,Implementationand Management, Cengage Learning 2012.

ListofURLs,TextBooks,Notes,MultimediaContent,etc

1. https://www.smartdraw.com/entity-relationship-diagram/
2. https://en.wikipedia.org/wiki/Database_normalization
3. www.databasteknik.se/webbkursen/relalg-lecture
4. https://technet.microsoft.com/en-us/library/bb264565(v=sql.90).aspx
5. pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/.../Ch16_Overview_Xacts.pdf
6. www.c-sharpcorner.com/UploadFile/f0b2ed/transaction-management-in-sql/

Course Outcomes	Thestudents should beableto: 1. Identify,analyzeanddefinedatabaseobjects,enforceintegrityconstraints on a database using RDBMS. 2. UseStructured Query Language(SQL) fordatabasemanipulation. 3. Designandbuildsimpledatabasesystems 4. Developapplicationtointeractwithdatabases.

Module1

Chapter1:Introduction to Databases

Introduction

Databases and database technology have a major impact on the growing use of computers. It is fair to say that databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, genetics, law, education, and library science.

Database

A **database** is a collection of related data.1 By **data**,we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know.

A database has the following implicit properties:

- It represents some aspect of the real world, sometimes called the **mini world** or the **universe of discourse (UoD)**. Changes to the mini world are reflected in the database.
- It is a logically coherent collection of data,to which some meaning can be attached.
- It is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

To summarize: a database has some source(i.e., the mini world)from which data are derived, some degree of interaction with events in the represented mini world and an audience that is interested in using it.

Size/Complexity: A database can be of any size and complexity. For example, the list of names and addresses referred to earlier may consist of only a few hundred records, each with asimple structure. An example of a large commercial database is Amazon.com. It contains data forover 20 million books, CDs, videos, DVDs, games, electronics, apparel, and other
items.

Computerized vs. manual: A database may be generated and maintained manually or it may be computerized. For example, simple database like telephone directory may be created and maintained manually. Huge and complex database may be created and maintained either by a

groupofapplicationprogramswrittenspecificallyforthattaskorbyadatabasemanagement system.

DatabaseManagementSystem(DBMS)

A **database management system** (DBMS) is a collection of programs enabling users to create and maintain a database. More specifically, The DBMS is a general-purpose software system that facilitates the processes of defining, constructing, manipulating, and sharing
Databases among various users and applications.

- **Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the database. The database definition or descriptive information is stored by the DBMS in the form of a database catalog or dictionary; it is called **meta-data**.
- **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS.
- **Manipulating** a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data.
- **Sharing** a database allows multiple users and programs to access the database simultaneously.

Other important functions provided by the DBMS include protecting the database and maintaining it over a long period of time.

- **Protection** includes ***system protection*** against hardware or software malfunction (or crashes) and ***security protection*** against unauthorized or malicious access.
- Atypical large database may have a life cycle of many years, so the DBMS must be able to **maintain** the database system by allowing the system to evolve as requirements change over time.

AdatabasetogetherwiththeDBMS softwareis referredtoas a **database system.**

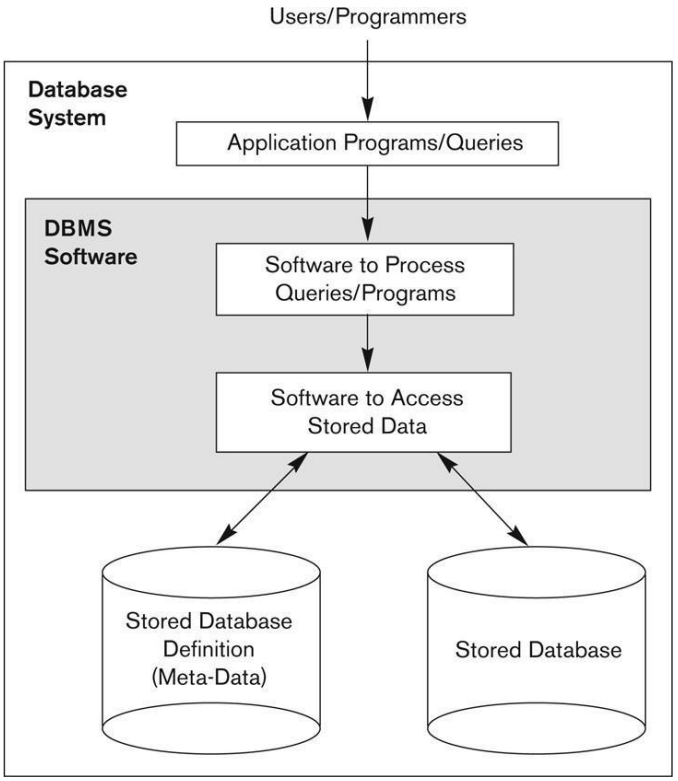


Fig1.1(a):A simplified database system environment

An Example

Consider a UNIVERSITY database for maintaining information concerning students, courses, and grades in a university environment. The database is organized as five files, each of which stores **data records** of the same type.

1. STUDENT file: stores data on each student.
2. COURSEfile:stores data on each course.
3. SECTIONfile:stores data on each section of a course.
4. GRADE_REPORT file:stores the grades that' students receive in the various sections they have completed.
5. PREREQUISITEfile: stores the prerequisites of each course.

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Fig1.1(b):A databasethatstores studentandcourseinformation

DefiningaUNIVERSITYdatabase

- Specify the structure of the records of each file - **data elements** to be stored in each record.For example: each STUDENT record includes data to represent the student's Name,Student_number,ClassMajor.SimilarlyeachCOURSErecordincludesdatato represent the Course_name, Course_number, Credit_hours, and Department.
- Specifyadatatypeforeachdataelementwithinarecord.Forexample:student'sName isa string of alphabetic charactersStudent_number is an integer.

ConstructingtheUNIVERSITYdatabase

- To *construct* the UNIVERSITY database, we store data to represent each student,course, section, grade report, and prerequisite as a record in the appropriate file.
- Records in the various files may be related. For example, the record forSmith in the STUDENT file is related to two records in the GRADE_REPORT file thatspecify Smith's grades in two sections. Similarly, each record in the PREREQUISITEfile relatestwo course records: one representing the course and the other representing the prerequisite.

ManipulatingaUNIVERSITYdatabase

Database *manipulation* involves querying and updating.

Examples of queries are as follows:

- Retrieve the transcript—a list of all courses and grades—of 'Smith'
- Listthenamesofstudentswhotookthesectionofthe'Database'courseofferedinfall 2008 and their grades in that section
- List the prerequisites of the 'Database' course

Examples of updates include the following:

- Change the class of 'Smith' to sophomore
- Create a new section for the 'Database' course for this semester
- Enteragradeof 'A'for'Smith' inthe'Database' sectionoflastsemester

Theseinformalqueriesandupdatesmustbespecifiedpreciselyinthequerylanguage of the DBMS before they can be processed.

As with software in general, design of a new application for an existing database or design of abrand new database starts off with a phase called **requirements specification and analysis**. These requirements are documented in detail and transformed into a**conceptual design** that can be represented and manipulated using some computerizedtools so that it can be easily maintained, modified, and transformed into a databaseimplementation.

Thedesign is then translated to a**logical design** that can be expressed in a data model implemented in a commercialDBMS. The final stage is**physical design**, during which further specifications are provided for storing andaccessing the database. The database design is implemented, populated with actualdata, and continuously maintained to reflect the state of the miniworld.

CharacteristicsoftheDatabaseApproach

Databaseapproachvs.FileProcessingapproach

Consider an organization that is organized as a collection of departments/offices. Each department has certain data processing "needs", many of which are unique to it.

In the file processing approach, each department would control a collection of relevant data files and software applications to manipulate that data. For example, one user, the grade reportingoffice, maykeep files on students and their grades. Programs to print a student's transcript andto enter new grades are implemented as part of the application. A second user, theaccounting office, may keep track of students' fees and their payments. Althoughboth users are interested in data about students, each user maintains separate files—and programs to manipulate these files— because each requires some data not available from the other user's files.This redundancy in defining and storing data resultsin wasted storage space and in redundant efforts to maintain common up-to-datedata.

In the database approach, a single repository maintains data that is defined onceand then accessed by various users. In file systems, each application is free to namedata elements independently. In contrast,inadatabase,thenamesorlabelsofdataaredefined once, andused repeatedlybyqueries, transactions, and applications.

Themaincharacteristicsofthedatabaseapproachversusthefile-processingapproacharethe following:

- Self-describingnatureofadatabase system
- Insulationbetween programsanddata,anddataabstraction
- Supportof multipleviews ofthe data
- Sharingofdata andmultiusertransaction processing

1. Self-DescribingNatureofaDatabaseSystem

A fundamental characteristic of the database approach is that the database system contains not only the database itself but also a complete definition or description of the database structure and constraints.This **meta-data** (i.e., data about data) is stored in the so-called **system catalog**, which contains a description of the structure of each file, the type and storage format of each field, and the various constraints on the data (i.e., conditions that the data must satisfy).

ThesystemcatalogisusednotonlybyusersbutalsobytheDBMSsoftware,whichcertainly needsto"know" howthedataisstructured/organizedinordertointerpretitinamanner consistentwith thatstructure.

RELATIONS

Relation_name	No_of_columns
STUDENT	4
COURSE	4
SECTION	5
GRADE_REPORT	3
PREREQUISITE	2

COLUMNS

Column_name	Data_type	Belongs_to_relation
Name	Character (30)	STUDENT
Student_number	Character (4)	STUDENT
Class	Integer (1)	STUDENT
Major	Major_type	STUDENT
Course_name	Character (10)	COURSE
Course_number	XXXXNNNN	COURSE
….	….	…..
….	….	…..
….	….	…..
Prerequisite_number	XXXXNNNN	PREREQUISITE

Figure 1.2(a): Anexampleofa databasecatalogforthedatabase

2. InsulationbetweenProgramsandData,andDataAbstraction

Program-Data Independence: In traditional file processing, the structure of the data files accessed by an application is "hard-coded" in its source code. If, for some reason, we decideto change the structure of the data ,everyapplication in which a description of that file's structure is hard-coded must be changed!

In contrast, DBMS access programs, in most cases, do not require such changes, because the structure of the data is described separately from the programs that access it and those programs consult the catalog in order to ascertain the structure of the data so that theyinterpret that data properly.

In other words, the DBMS provides a conceptual or logical view of the data to application programs, so that the underlying implementation may be changed without the programs being modified. (This is referred to as *program-data independence*.)

Program-operation independence: In object-oriented and object-relationalsystems , userscan define operations on data as part of the databasedefinitions.An **operation** (also called a *function* or *method*) is specified in two parts.The *interface* (or *signature*) of an operation includes the operation name and thedata types of its arguments (or parameters). The *implementation* (or *method*) of theoperation is specified separately and can bechanged without affecting the interface.User application programs can operate on the data by invoking these operationsthrough their names and arguments, regardless of how the operations are implemented.This may be termed **program-operation independence**.

Data abstraction

The characteristic that allows program-data independence and program-operationindependence is called **data abstraction**. A DBMS provides users with a conceptual representationof data that does not include many of the details of how the data isstored or how the operations are implemented. Informally, a **data model** is a type ofdata abstraction that is used to provide this conceptual representation. The datamodel uses logical concepts, such as objects, their properties, and their interrelationships,that may be easier for most users to understand than computer storageconcepts. Hence, the data model *hides* storage and implementation details that arenot of interest to most database users.

3. SupportofMultiple ViewsoftheData

A database typically has many users, each of whom may require a different perspectiveor **view**of the database. A view may be a subset of the database or it may contain**virtual data** that is derived from the database files but is not explicitly stored. A multiuser DBMS whose users have a variety of distinct applications mustprovide facilities for defining multiple views. For example,oneuserofthedatabaseofFigure1.2maybeinterestedonlyinaccessingandprinting the transcript of eachstudent; the view for this user is shown in Figure 1.2(b)

TRANSCRIPT

Student_name	Student_transcript				
	Course_number	Grade	Semester	Year	Section_id
Smith	CS1310	C	Fall	08	119
	MATH2410	B	Fall	08	112
Brown	MATH2410	A	Fall	07	85
	CS1310	A	Fall	07	92
	CS3320	B	Spring	08	102
	CS3380	A	Fall	08	135

Fig1.2(b):view derived fromthe university database

4. SharingofDataandMultiuserTransaction Processing

AmultiuserDBMS,asits nameimplies,mustallowmultipleuserstoaccessthe database at the same time. This is essential if data for multiple applications is to be integratedandmaintained in a single database. The DBMS must include **concurrencycontrol** software to ensure that several users trying to update the same data do so ina controlled manner so that the result ofthe updates is correct. For example, whenseveral reservation agents try to assign a seat on anairline flight, the DBMS shouldensure that each seat can be accessed by only one agent at a time for assignment to apassenger. These types of applications are generally called **online transaction processing(OLTP)** applications. A fundamental role of multiuser DBMS software istoensure that concurrent transactions operate correctly and efficiently.

Theconceptofa**transaction** hasbecomecentral to manydatabaseapplications.Atransaction isan *executing program* or *process* that includes one or more databaseaccesses, such as reading or updating of database records. The DBMS must enforce several transaction properties. The**isolation** property ensures that each transaction appears to executein isolation from other transactions, even though hundreds of transactions may beexecuting concurrently. The **atomicity** property ensures that either all the databaseoperations in a transaction are executed or none are.

DatabaseUsers

Usersmaybedividedinto

- Thosewhoactuallyuseandcontrolthedatabasecontent,andthosewhodesign,developand maintain database applications called **"Actors on the Scene"**
- ThosewhodesignanddeveloptheDBMSsoftwareandrelatedtools,andthecomputer systems operators called **"Workers Behind the Scene"**

ActorsontheScene

1. **Database Administrator** (DBA): chief administrator, who oversees and manages thedatabase system (including the data and software). Duties include authorizing users to access the database, coordinating/monitoring its use, acquiring hardware/software for upgrades, etc.The DBA is accountable for problems such as securitybreaches and poor system response time.In large organizations, the DBA might have a supportstaff.
2. **Database Designers**: responsible for identifying the data to be stored and for choosing an appropriate way to organize it. Database designers typically interact with each potential groupof users and develop **views** of the database that meet the data andprocessingrequirements of these groups.The final database design must be capable of supportingthe requirements of all user groups.
3. **End Users**: These are persons who access the database for querying, updating,and report generation. Thereare several categories of end users:
    - **Casual end users:** use database occasionally, needing different information each time; use query language to specify their requests; typically middle- or high-level managers.
    - **Naive/Parametric end users**: biggest group of users; frequently query/update the databaseusingstandard**cannedtransactions**thathavebeencarefullyprogrammed and tested in advance. Examples:
        - banktellerscheckaccountbalances,postwithdrawals/deposits
        - reservationclerks for airlines,hotels,etc.,check availabilityof seats/rooms and make reservations.
    - **Sophisticated end users**: include engineers, scientists, business analysts, and others who thoroughly familiarize themselves with the facilities of the DBMS in order to implement their own applications to meet their complex requirements.

- **Stand-alone users**: maintain personal databases by using ready-made programpackages that provide easy-to-use menu-based or graphics-basedinterfaces. Ex:userofataxpackage that storesavarietyofpersonalfinancialdata for tax purposes.

4 **SystemAnalystsandApplicationProgrammers(SoftwareEngineers**)

- **System Analysts**: determine needs of end users, especially naive and parametric users, and develop specifications for canned transactions that meet these needs.
- **Application Programmers**: Implement, test, document, and maintain programs that satisfy the specifications mentioned above.

WorkersbehindtheScene

1. **DBMS system designers and implementers:** design and implement the DBMS modules and interfaces as a software package. A DBMS isa very complex software system that consists of many components, or **modules**, including modules for implementing the catalog, query language processing, interface processing, accessing and buffering data, controlling concurrency, and handling data recovery and security.
2. **Tool developers:** design and implement **tools** that facilitate database modeling and design, database system design, and improved performance.
3. **Operators and maintenance personnel** (system administration personnel) : responsible for the actual running and maintenance of the hardware and software environment for the database system.

AdvantagesofUsingtheDBMSApproach

1. ControllingRedundancy

Data redundancy such as tends to occur in the "file processing" approach leads to **wasted storage space**, **duplication of effort** and a higher likelihood of the introduction of **inconsistency**.

In the database approach, the views of different user groups are integrated during database design. This is known as **data normalization**, and it ensures consistency and saves storage

Space. However, it is sometimes necessary to use **controlled redundancy** to improve the performance of queries. For example, we may store Student_name and Course_number redundantly in a GRADE_REPORT file because whenever we retrieve a GRADE_REPORT record, we want to retrieve the student name and course number along with the grade, student number, and section identifier.

ADBMSshouldprovidethecapabilityto automaticallyenforcethe rulethatnoinconsistencies are introduced when data is updated.

2. **RestrictingUnauthorizedAccess**

When multiple users share a large database, it is likely that most users will not be authorized to access all information in the database. For example, financial data is often considered confidential and only authorized persons are allowed to access such data. In addition, some usersmayonly bepermitted to retrievedata, whereas others are allowed to retrieve and update. Hence, the type of access operation—retrieval or update—must also be controlled. A DBMS should provide a **security and authorization subsystem**, which the DBA uses to create accounts, to specify account restrictions and enforce these restrictions automatically.

3. **ProvidingPersistentStorageforProgramObjects**

The values of program variables or objects are discarded once a program terminates, unlessthe programmer explicitly stores them in permanent files, which often involves convertingthese complex structures into a format suitable for file storage. Object-oriented database systems make it easier for complex runtime objects to be saved in secondary storage so as to survive beyond program termination and to be retrievable at a later time.

Object-oriented database systems are compatible with programming languages such as C++ and Java, and the DBMS software automatically performs any necessary conversions.

4. **ProvidingStorageStructuresandSearchTechniquesforEfficientQueryProcessing**

DBMS maintains indexes that are utilized to improve the execution time of queries andupdates. DBMS has a buffering or caching module that maintains parts of the database inmain memory buffers.The query processing and optimization module is responsible for choosing an efficient query execution plan for each query submitted to the system**.**

5. **ProvidingBackupandRecovery**

The backup and recovery subsystem of the DBMS is responsible for recovery. For example, if the computer system fails in the middle of a complex update transaction, the recovery

subsystem is responsible for making sure that the database is restored to the state it was inbefore the transaction started executing. Disk backup isalso necessary in case of a catastrophic disk failure.

6. **ProvidingMultipleUser Interfaces**

Becausemanytypesofuserswithvaryinglevelsoftechnicalknowledgeuseadatabase,a DBMS should provide a variety of user interfaces. These include

- Querylanguagesforcasual users
- Programminglanguageinterfacesforapplicationprogrammers
- Formsandcommandcodesforparametricusers
- Menu-driveninterfacesandnaturallanguageinterfacesforstandaloneusers.

7. **RepresentingComplexRelationshipsamongData**

Adatabasemayincludenumerous varietiesof datathatareinterrelatedinmany ways.

For example each section record is related to one course record and to a number of GRADE_REPORTrecords—oneforeachstudent whocompletedthatsection.ADBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficiently.

8. **EnforcingIntegrityConstraints**

Most database applications are such that the semantics of the data require that it satisfy certain restrictions in order to make sense.

Thesimplesttypeofintegrity constraintinvolvesspecifyingadatatypefor eachdata item.

For example, in student table we specified that the value of Name must be a string of no more than 30 alphabetic characters.

More complex type of constraint is **referential integrity** involves specifying that a record inone file must be related to records in other files. For example, in university database, we can specify that every section record must be related to a course record.

Another type of constraint specifies uniqueness on data item values, such as every courserecord must have a unique value for Course_number. This is known as a key or **uniqueness constraint**.

It is the responsibility of the database designers to identify integrity constraints during database design.

9. PermittingInferencingandActionsUsingRules

In a **deductive** database system, one may specify *declarative* rules that allow the database to infer new data. For example, figure out which students are on academic probation. Such capabilities would take the place of application programs that would be used to ascertain such information otherwise.

Active database systems go one step further by allowing "active rules" that can be used to initiate actions automatically. In today's relational database systems, it is possible to associate triggers with tables.

10. AdditionalImplicationsofUsingtheDatabaseApproach

- **Potential for Enforcing Standards :** database approach permits the DBA to define and enforce standards among database users in a large organization which facilitates communication and cooperation among various departments, projects, and users within the organization.Standards can be defined for names and formats of data elements, display formats, report structures and so on.
- **Reduced Application Development Time:** once a database is up and running,substantially less time is generally required to create new applications using DBMS facilities. Development time using a DBMS is estimated to be one-sixth to one-fourth of that for a traditional file system.
- **Flexibility:** It may be necessary to change the structure of a database as requirements change. DBMSs allow changes to the structure of the database without affecting the stored data and the existing application programs.
- **AvailabilityofUp-to-DateInformation:**DBMSmakesthedatabaseavailableto allusers. Availability of up-to-date information is essential for many transaction-processing applications, such as reservation systems or banking databases
- **Economies of Scale:** DBMS approach permits consolidation of data and applications, to overlap between activities of data-processing in different projects or departments. This enables the whole organization to invest in more powerful processors, storage devices, or communication gear, rather than having each department purchase its equipment thus reducing overall costs of operation and management.

HistoryofDatabaseApplications

- **EarlyDatabaseApplicationsUsingHierarchicalandNetworkSystems**

  Earlydatabaseapplicationsmaintainedrecordsinlargeorganizationssuchascorporations, universities, hospitals, and banks. In many of these applications, there werelargenumbersofrecordsofsimilarstructure.Therewere alsomanytypesofrecords and many interrelationships among them.

  Problemswiththeearly databasesystems

  - lackofdata abstractionandprogram-dataindependencecapabilities
  - provided only programming language interfaces. This made it time-consuming and expensive to implement new queries and transactions, since new programs had to be written, tested, and debugged.

- **ProvidingData AbstractionandApplicationFlexibilitywithRelational Databases**

  Relational databases were originally proposed to separate the physical storage of data from its conceptual representation and to provide a mathematical foundation for data representation and querying. The relational data model also introduced high-level query languagesthatprovided analternativetoprogramminglanguageinterfaces,makingitmuch faster to write new queries. Hence, data abstraction and program-data independence were much improved when compared to earlier systems.

- **Object-OrientedApplicationsandtheNeedforMoreComplexDatabases**

  Object-oriented databases (OODBs) mainly used in specialized applications, such as engineering design, multimedia publishing, and manufacturing systems. In addition, many object-oriented concepts were incorporated into the newer versions of relational DBMSs, leading to object-relational database management systems, known as ORDBMSs.

- **InterchangingDataon theWebforE-CommerceUsingXML**

  The World Wide Web provides a large network of interconnected computers. Users can create documents using a Web publishing language, such as HyperText Markup Language (HTML), and store these documents on Web servers where other users (clients) can access them. Documents can be linked through **hyperlinks**, which are pointers to otherdocuments.

Currently, eXtended Markup Language (XML) is considered to be the primary standard for interchanging data among various types of databases and Web pages. XML combines concepts from the models used in document systems with database modeling concepts.

- **ExtendingDatabaseCapabilitiesforNew Applications**

The success of database systems in traditional applications encouraged developers of other types of applications to attempt to use them. The following are some examples of these applications:

- **Scientific**applicationsthatstorelarge amountsofdataresultingfrom Scientific experiments in areas such as high-energy physics, the mapping of the human genome, and the discovery of protein structures.
- Storage and retrieval of **images**, including scanned news or personal photographs, satellite photographic images, and images from medicalprocedures such as x-rays and MRIs (magnetic resonance imaging).
- Storageandretrievalof**videos,**suchasmovies,and**videoclips**fromnews or personal digital cameras.
- **Datamining**applicationsthatanalyzelargeamountsofdatasearchingfor the occurrences of specific patterns or relationships, and for identifying unusual patterns in areas such as credit card usage.
- **Spatial** applications that store spatial locations of data, such as weather information, mapsused ingeographical informationsystems, and in automobile navigational systems.
- **Timeseries**applicationsthatstoreinformationsuchaseconomicdataat regular pointsin time,such as daily sales andmonthly gross national product figures.

- **DatabasesversusInformationRetrieval**

Database technology is heavily used in manufacturing, retail, banking, insurance, finance, and health care industries, where structured data is collected through forms, such asinvoices or patient registration documents. An area related to database technology is **Information Retrieval (IR)**, which deals with books, manuscripts, and various forms of library-based articles. Data is indexed, cataloged, and annotated using keywords. IR is concerned with searching for material based on these keywords, and with the many problems dealing with document processing and free-form text processing.

WhenNottoUseaDBMS

- DBMS may involve unnecessary overhead costs that would not be incurred in traditional file processing. The overhead costs of using a DBMS are due to the following:
  - Highinitialinvestment inhardware,software,andtraining
  - ThegeneralitythataDBMSprovidesfordefiningandprocessingdata
  - Overheadforprovidingsecurity,concurrencycontrol,recovery,andintegrity functions
- Therefore,itmaybemoredesirabletouseregularfilesunderthefollowingcircumstances:
  - Simple,well-defineddatabaseapplicationsthatarenotexpectedtochangeatall
  - Stringent,real-timerequirementsforsomeapplicationprogramsthatmaynotbe met because of DBMS overhead
  - Embeddedsystemswithlimitedstoragecapacity,whereageneral-purposeDBMS would not fit
  - Nomultiple-useraccesstodata

Chapter2:OverviewofDatabaseLanguagesandArchitectures

Introduction

The architecture of DBMS packages has evolved from the early monolithic systems, where the whole DBMS software package was one tightly integrated system. Modern DBMS packages are modular in design, with a client/server system architecture. In a basic client/server DBMS architecture, the system functionality is distributed between two types of module.A**client module** is designed to run on a user workstation or personal computer. The
client module handles user interaction and provides the user-friendly interfaces such as forms- or menu-based GUIs. The other kind of module, called a **server module** handles data storage, access, search, and other functions

DataModels,Schemas,andInstances

DataModel

Adatamodel is acollection ofconcepts that can be used to describethestructureofadatabase. By structure of a database we mean the data types, relationships and constraints that apply to the data. Most data models also include a set of basic operations forspecifying retrievals and updates on the database. Data model provides the necessary means to achieve abstraction.

CategoriesofDataModels

Data models can be categorized according to the types of concepts they use to describe the database structure.

1. **High-level** or **conceptual data models:** provide concepts that are close to the way many users perceive data. Conceptual data models use concepts such as entities, attributes, and relationships.
2. **Representational** or **implementation data models**: provide concepts that may be easily understood by end users but that are not too far removed from the way data isorganized in computer storage. Representational data models hide many details of data storage on disk but canbeimplemented on acomputer system directly.Representational orimplementation data models are the models used mostfrequently intraditionalcommercialDBMSs.Theseincludethewidelyused**relational datamodel**,aswellastheso-calledlegacydatamodels—the**network**and**hierarchical**

models. Representational data models represent data by using record structures and hence are sometimes called **record-based data models**.

3. **Low-level** or **physical data models:** provide concepts that describe the details of how data is stored on the computer storage media, typically magnetic disks. Physical data models describe how data is stored as files in the computer by representing informationsuch as record formats, record orderings, and access paths.

Databaseschema

The description of a database is called the **database schema**, which is specified duringdatabase design and is not expected to change frequently.

Schemadiagram

A displayed schema is called a **schema diagram**. A schema diagram displays only some aspects of a schema, such as the names of record types and data items, and some types of constraints.

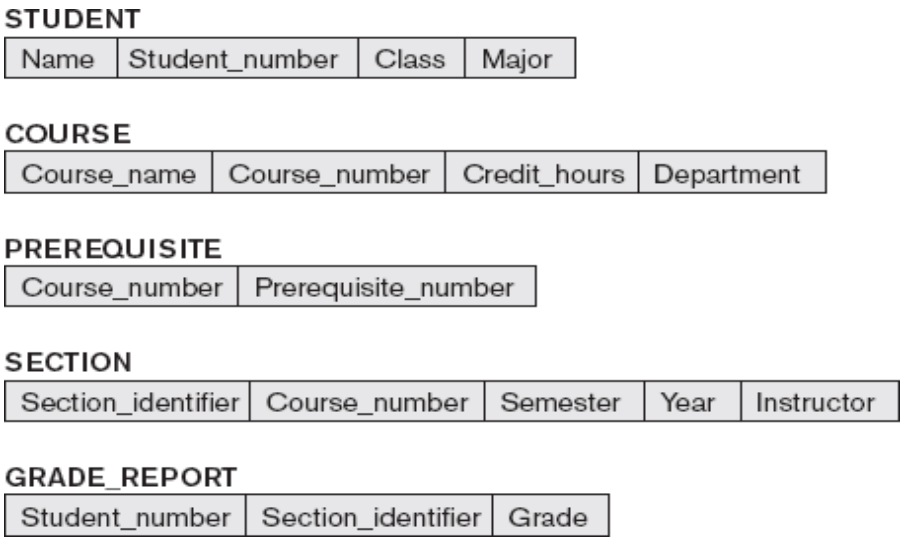


Figure2.1: Schema diagramforthe database

Schemaconstruct

Eachobjectin theschemais calledschema construct.Forexamplestudentor course.

Databasestateor **snapshot**

Thedatainthedatabaseataparticularmomentin timeiscalleda **databasestate**or**snapshot**.It is also called the current set of **occurrences** or **instances** in the database. In a given database state, each schema construct has its own current set of instances; for example, the STUDENT construct will contain the set of individual student entities (records) as its instances.

The distinction between database schema and database state is very important. When we **define** a new database, we specify its database schema only to the DBMS. At this point, the corresponding database state is the *empty state* with no data. We get the *initial state* of the database when the database is first **populated** or **loaded** with the initial data. From then on, every time an update operation is applied to the database, we get another database state. At any point in time, the database has a *current state*.

The DBMS is partly responsible for ensuring that every state of the database is a **valid state**—that is, a state that satisfies the structure and constraints specified in the schema. The

DBMS stores the descriptions of the schema constructs and constraints—also called the **meta-data**—in the DBMS catalog so that DBMS software can refer to the schema whenever it needs to. The schema is sometimes called the **intension**, and a database state is called an **extension** of the schema.

Three-SchemaArchitectureandDataIndependence

TheThree-SchemaArchitecture

Thegoalofthethree-schemaarchitectureistoseparatetheuserapplicationsfromthephysical database. In this architecture, schemas can be defined at the following three levels:

1. The **internal level** has an **internal schema**, which describesthephysicalstorage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
2. The **conceptual level** has a **conceptual schema**, which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. Usually, a representational data model is used to describe the conceptual schema when a database system is implemented.
3. The **external** or **view level** includes a number of **external schemas** or **user views**. Each external schema describes the part of the database that a particular user group is interested inandhidestherestofthedatabasefromthatuser group.Each externalschema istypically

implementedusingarepresentationaldatamodel,possiblybasedonanexternalschema design in a high-level data model.

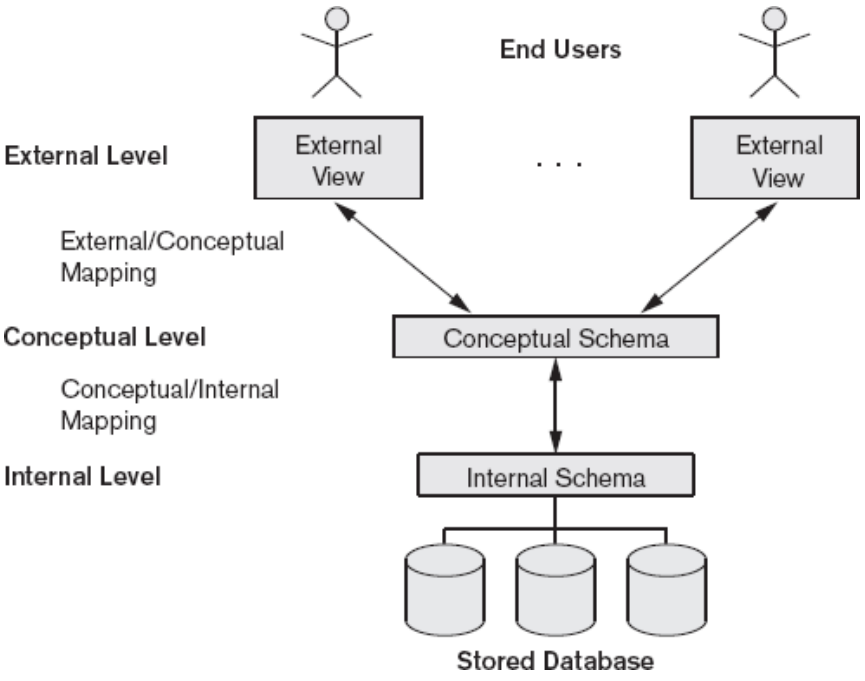


Figure2.2: Thethree-schema architecture.

In a DBMS based on the three-schema architecture, each user group refers to its own external schema.Hence, theDBMS must transform arequest specified on an external schemainto arequest against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is a database retrieval, the data extracted fromthestoreddatabase must bereformattedtomatchtheuser'sexternal view.

Theprocesses oftransforming requestsand resultsbetween levelsarecalled **mappings**.

DataIndependence

Data independence can bedefined as the capacity to changethe schemaat onelevel of adatabase system without having to change the schema at the next higher level. We can define two types of data independence:

1. **Logicaldataindependence**isthecapacitytochangetheconceptualschema without having to change external schemas or application programs. We may change the conceptual schema to expand the database, to change constraints, or to reduce the database. Only the view definition and the mappings need to be changed in a DBMS that supports logical data independence.

2. **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well. Changes to the internal schema may beneeded because some physical files were reorganized—for example, by creating additional access structures—to improve the performance of retrieval or update.

Data independence occurs because when the schema is changed at some level, the schema at the next higher level remains unchanged; only the *mapping* between the two levels is changed.

DatabaseLanguagesandInterfaces

TheDBMSmustprovideappropriatelanguagesandinterfacesforeachcategoryof users.

DBMSLanguages

Once the design of a database is completed and a DBMS is chosen to implement the database,thefirststepistospecifyconceptualandinternalschemasforthedatabaseandany mappings between the two.

DataDefinitionLanguage(DDL)

The **data definition language** (**DDL**) is used by the DBA and by database designers todefine both schemas when no strict separation of levels is maintained . The DBMS will havea DDL compiler whose function is to process DDL statements in order to identify descriptions of the schema constructs and to store the schema description in the DBMS catalog.

StorageDefinitionLanguage (**SDL)**

Storage definition language is used when clear separation is maintained between the conceptual andinternallevels,theDDLisusedto specifytheconceptualschemaonly. The **storage definition language** (**SDL**), is used to specify the internal schema. The mappings between the two schemas may be specified in either one of these languages.

ViewDefinitionLanguage(VDL),

View definition language is usedtospecifyuserviewsandtheirmappingstothe conceptual schema. In relational DBMSs, SQL is used in the role of VDL to define user or application **views** as results of predefined queries.

DataManipulationLanguage(DML)

Data manipulation languages (DML) are used to perform manipulation operation such as retrieval, insertion, deletion, and modification of the data. There are two main types of DMLs :

1. **High-level** or **nonprocedural** DML : can be used on its own to specify complex database operations concisely. Many DBMSs allow high-level DML statements either to beentered interactively from a display monitor or terminal or to be embedded in a general-purpose programming language. In the latter case, DML statements must be identified within the program so that they can be extracted by a precompiler and processed by the DBMS.High-level DMLs, such as SQL, can specify and retrieve many records in a single DML statement; therefore, they are called **set-at-a-time** or **set-oriented** DMLs. A query in a high-level DML often specifies *which* data to retrieve rather than *how* to retrieve it; therefore, such languages are also called **declarative**
2. **Low-level** or **procedural** DML: must be embedded in a general-purpose programming language. This type of DML typically retrieves individual records or objects from the database and processes each separately. language constructs, such as looping, to retrieve and process each record from a set of records. Low-level DMLs are alsocalled **record-at-a-time** DMLs because of this property. DL/1, a DMLdesigned for the hierarchical model, is a low-level DML that uses commands such as GET UNIQUE, GET NEXT, or GET NEXT WITHIN PARENT to navigate from record to record within a hierarchy of records in the database.

Hostlanguage

Whenever DML commands, whether high level or low level, are embedded in a general-purpose programming language, that language is called the **host language** and the DML is called the **data sublanguage**.

Ahigh-levelDMLusedinastandalone interactive manneriscalleda**query language**.

.

DBMSInterfaces

User-friendlyinterfacesprovidedbyaDBMS mayincludethefollowing:

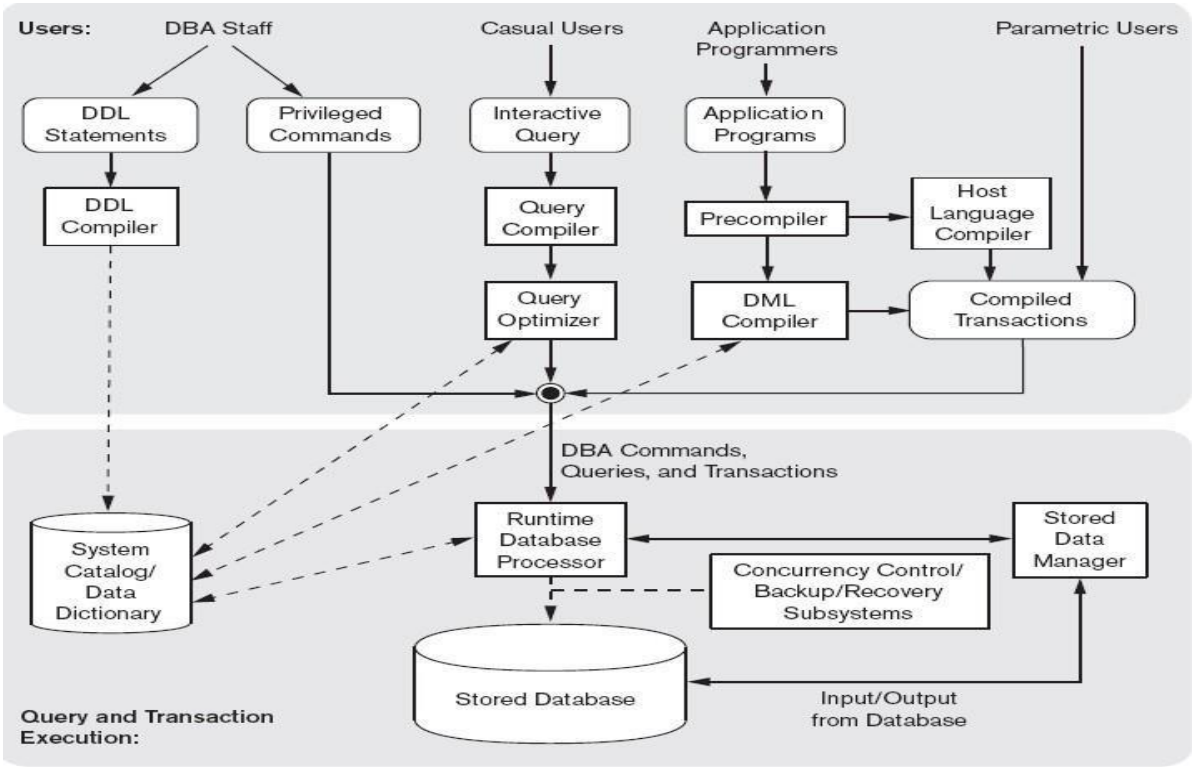
1. **Menu-BasedInterfacesforWebClientsorBrowsing:**Theseinterfacespresent the user with lists of options (called menus) that lead the user through theformulation of a request. There is no need for the user to memorize the specific commands and syntaxofaquerylanguage.Pull-downmenusareaverypopulartechniqueinWeb- based user interfaces.
2. **Forms-Based Interfaces:** A forms-based interface displays a form to each user. Users can fill out all of the form entries to insert new data, or they can fill out only certain entries, in which case the DBMS will retrieve matching data for the remaining entries.Formsareusuallydesignedandprogrammedfornaiveusers asinterfacestocanned transactions.
3. **Graphical User Interfaces:** A GUI typically displays a schema to the user indiagrammatic form. The user then can specify a query by manipulating the diagram. In many cases, GUIs utilize both menus and forms. Most GUIs use a pointing device, suchas amouse, toselect certainparts ofthe displayedschemadiagram.
4. **NaturalLanguageInterfaces:**Theseinterfacesacceptrequestswrittenin English or some other language and attempt to understand them. A natural language interface usually has its own schema, which issimilar to the database conceptual schema, as well as a dictionary of important words. The natural language interface refers to the wordsinitsschema,aswellastothesetofstandardwordsinitsdictionary,tointerpretthe request. If the interpretation is successful, the interface generates a high-level query corresponding to the natural language request and submits itto the DBMS for processing; otherwise, a dialogue is started with the user toclarifythe request.
5. **Speech Input and Output**: Applications with limited vocabularies such as inquiries for telephone directory, flight arrival/departure, and credit card account information are allowing speech for input and output to enable customers to access this information. The speech input is detected using a library of predefined words and used to set up the parameters that are supplied to the queries. For output, a similar conversion from text or numbers into speech takes place.
6. **Interfaces for Parametric Users:** Parametric users, such as bank tellers, often have a small set of operations that they must perform repeatedly. For example, a tellerisabletousesinglefunctionkeysto invokeroutineandrepetitivetransactions suchasaccountdepositsorwithdrawals,orbalanceinquiries.Usuallyasmallsetof abbreviated commands is included, with the goal of minimizing the number of

keystrokesrequiredforeachrequest.

7. **Interfaces for the DBA:** Most database systems contain privileged commands that can be used only by the DBA staff. These include commands for creating accounts,settingsystemparameters,grantingaccountauthorization,changinga schema, and reorganizing the storage structures of a database.

TheDatabaseSystemEnvironment

DBMSComponent Modules



The top part of the figure refers to the various users of the database environment and theirinterfaces.ThelowerpartshowstheinternalsoftheDBMSresponsibleforstorage of data and processing of transactions.

DDL compiler-processes schema definitions, specified in the DDL, and stores descriptions of the schemas (meta-data) in the DBMS catalog.

Interactivequeryinterface:interfaceforCasualusersandpersonswithoccasionalneedfor information from the database.

Querycompiler-validatesforcorrectnessofthequerysyntax,thenamesoffilesanddata elements & compiles them into an internal form.

Query optimizer –concerned with the rearrangement and possible reordering of operations, elimination of redundancies, and use of correct algorithms and indexes during execution. It consults the systemcatalogforstatisticalandotherphysicalinformation about the stored data and generates executable code that performsthe necessary operations for the query and makes calls on the runtime processor.

Precompiler-extractsDMLcommandsfromanapplicationprogramandsendstothe DML compiler for compilation into object code for database access.

Hostlanguagecompiler-restoftheprogramissenttothehostlanguagecompiler.The object codes for the DML commands and the rest of the program are linked, forming a cannedtransaction whoseexecutablecodeincludescallstotheruntimedatabase processor.

Runtimedatabase processor executes:

(1) theprivilegedcommands

(2) theexecutablequeryplans,and

(3) thecannedtransactionswithruntimeparameters.

It works with the system catalog and may update it withstatistics. It also works with the stored data manager, which in turn uses basic operating system services for carrying out low-level input/output (read/write) operations between the disk and main memory. The runtime database processor handles other aspects of data transfer, such as management of buffers in the main memory.

storeddatamanagerusesbasicoperatingsystemservicesforcarryingoutlow-level input/output (read/write) operations between the disk and main memory.

concurrencycontroland**backupandrecoverysystems**integratedintotheworkingofthe runtime database processor for purposes of transaction management.

DatabaseSystem Utilities

DatabaseutilitieshelptheDBAtomanagethedatabasesystem.Commonutilities have the following types of functions:

- **Loading:** used to load existing data files—such as text files or sequential files—into the database.
- **Backup:** creates a backup copy of the database, usually by dumping the entire database onto tape or other massstorage medium. The backup copy can beused to restore the database incase of catastrophic disk failure. Incremental backups are also often used, where only changes since the previous backup are recorded. Incremental backup is more complex, but saves storagespace.

- **Database storage reorganization:** used to reorganize a set of database files into differentfile organizations, and create new access paths to improve performance.
- **Performance monitoring:** monitors database usage and provides statistics to the DBA. The DBA uses the statistics in making decisions such as whether or not to reorganize files or whether to add or drop indexes to improve performance.

Otherutilitiesmaybeavailableforsortingfiles,handlingdatacompression,monitoring access by users, interfacing with the network, and performing other functions.

Tools,ApplicationEnvironments,andCommunications Facilities

- **Tools**
  - **CASE:**usedinthedesignphaseofdatabase systems
  - **Data dictionary :** In addition to storing catalog information about schemas and constraints, the data dictionary stores other information, such as designdecisions, usage standards, application program descriptions, and user information. Such a systemisalsocalledan**informationrepository**.Thisinformationcanbeaccessed directly by users or the DBA when needed.
- **Applicationdevelopmentenvironments**
  - **PowerBuilder (Sybase) or JBuilder (Borland):** provide an environment for developing database applications including database design, GUIevelopment, querying and updating, and application program development.
  - **Communications software:** allow users at locations remote from the database system site to access the database through computer terminals, workstations, or personal computers. Integrated DBMS and data communications system is called a DB/DC system

CentralizedandClient/ServerArchitecturesforDBMSs

CentralizedDBMSsArchitecture

AllDBMSfunctionality,applicationprogramexecution,anduserinterfaceprocessing carried out on one machine

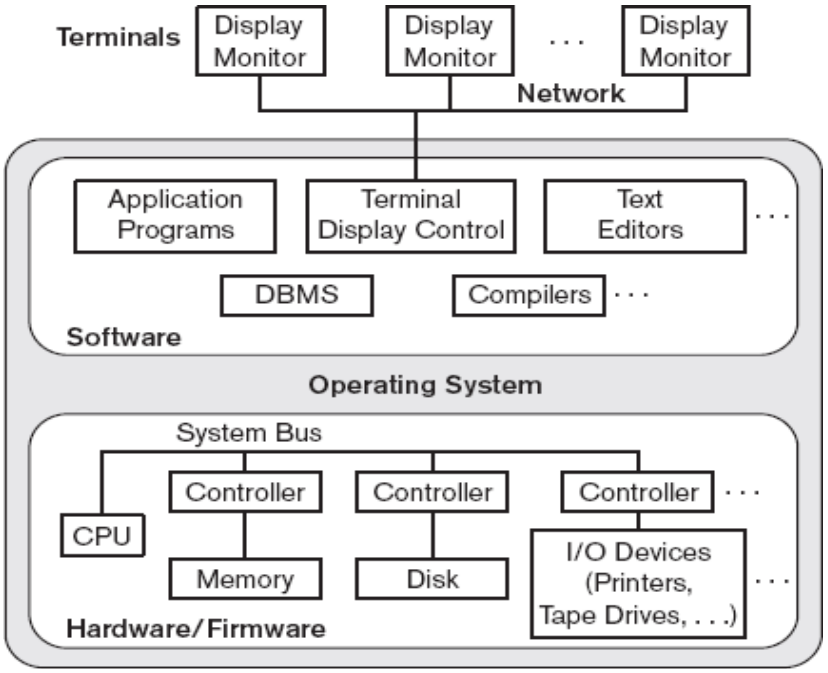


Figure2.5.1: Aphysicalcentralizedarchitecture

- **Disadvantages:**
  - When the central site computer or database system goes down, then everyone is blocked from using the system
  - Communicationcostsfromtheterminalstothecentralsitecanexpensive

BasicClient/ServerArchitectures

The **client/server architecture** was developed to deal with computing environments in which a large number of PCs, workstations, file servers, printers, database servers, Web servers, e-mail servers, and other software and equipment are connected via a network.

- **idea**
  - definespecializedserverswithspecific functionalities.
  - forexamplefileserverthatmaintainsthefilesoftheclient machines
  - Theresourcesprovidedbyspecializedserverscanbeaccessedby many client machines.

- The client machinesprovide the user with the appropriateinterfacesto utilize these servers and local processing power to run local applications

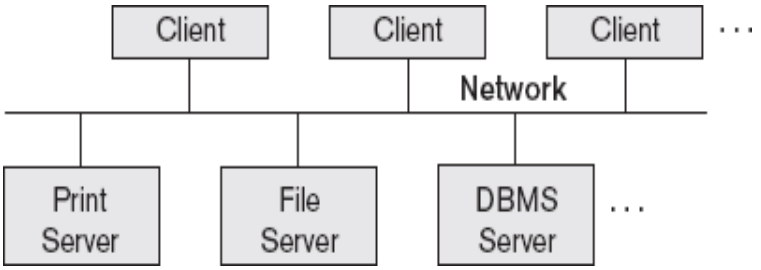


Figure2.5.2(a):Logicaltwo-tierclient/serverarchitecture

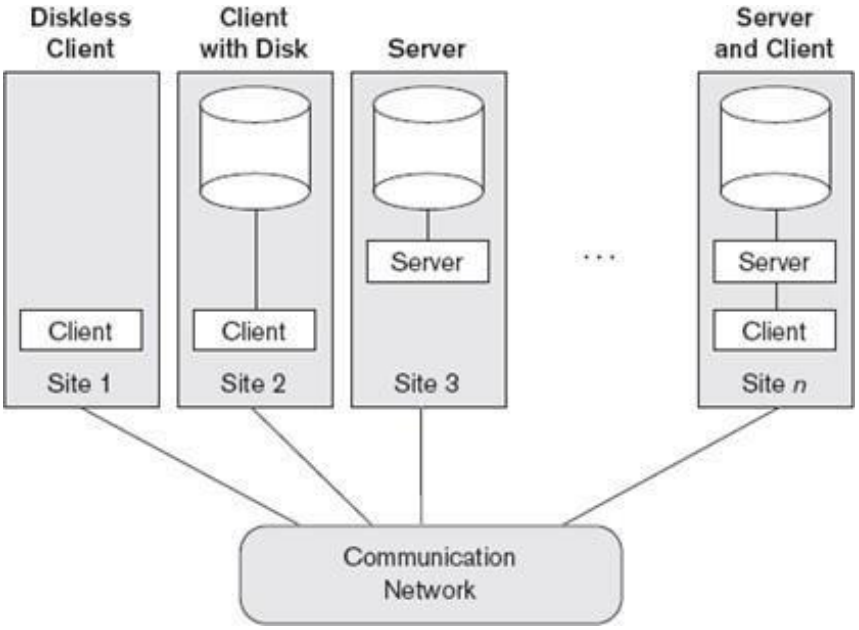


Figure2.5.2(b):Physicaltwo-tierclient/server architecture.

The concept of client/server architecture assumes an underlying framework that consists of many PCs and workstations as well as a smaller number of mainframe machines, connected via LANs and other types of computer networks. A **client** is a user machine that provides user interface capabilities and local processing. When a client requires access to additional functionality such as database access—that does not exist at that machine, it connects to a server that provides the needed functionality. A **server** isa system containing both hardware and software that can provide services to the client machines, such as file access, printing, archiving, or database access.

Two-TierClient/ServerArchitecturesforDBMSs

Thesoftwarecomponentsaredistributedovertwosystems:clientand server

- **Serverhandles**
  - Queryandtransactionfunctionalityrelatedto SQLprocessing
- **Clienthandles**
  - Userinterfaceprogramsandapplicationprograms

The user interface programs and application programs can run on the client side. When DBMS access is required, the program establishes a connection to the DBMS(which is on the server side) once the connection is created, the client program can communicate with the DBMS.

Aclientprogramcanactuallyconnecttoseveral RDBMSsandsendqueryandtransactionrequests using the ODBC API, which are then processed at the server sites. Any query results are sent back to the client program, which can process and display the results as needed. A related standard for the Java programming language, called **JDBC**, has also been defined to allow Java clientprograms to access one or more DBMSs through a standard interface

Object-orientedDBMSs

The different approach to two-tier client/server architecture was taken by some object-oriented DBMSs, where the software modules of the DBMS were divided between client and server in a more integrated way.

- serverlevel

  may include the part of the DBMS software responsible for handling data storage on disk pages, local concurrency control and recovery, buffering and caching of disk pages.
- clientlevel

  may handle the user interface, data dictionary functions, DBMS interactions with programming language compilers, global query optimization, concurrency control, and recovery across multiple servers, structuring of complex objects from the data in thebuffers.

In this approach, the client/server interaction is more tightly coupled and is done internally by the DBMS modules—some of which reside on the client and some on the server—rather than by the users/programmers.

Three-Tierandn-TierArchitecturesforWeb Applications

ManyWebapplicationsuseanarchitecturecalledthethree-tierarchitecture,whichaddsan intermediate layer between the client and the database server

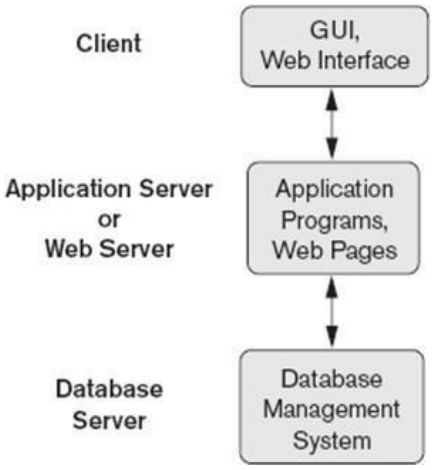


Figure2.5.4(a): Logicalthree-tierclient/serverarchitecture

- **Client**
  - ContainGUIinterfacesand someadditional application-specificbusinessrules
- **ApplicationserverortheWeb server**
  - accepts requests from the client, processes the request and sends database queries and commands to the database server, and then passes processed data from the database server to the clients, where it may be processed further and filtered to be presented to users in GUI format.
  - It can also improve database security by checking a client's credentials before forwarding a request to the database server.

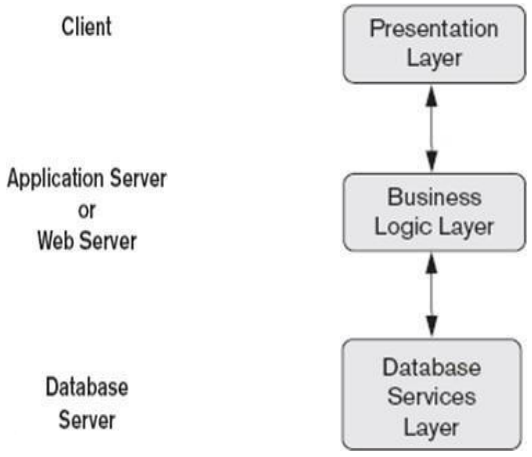


Figure2.5.4(b): Logicalthree-tierclient/serverarchitecture

Figure2.5.4(b)showsanotherarchitectureusedbydatabaseandotherapplicationpackage vendors.

- **Presentationlayer**

- displaysinformation totheuserandallowsdataentry

- **Thebusinesslogiclayer**
  - handles intermediate rules and constraints before data is passed up to the user or down to the DBMS
  - can also act as a Web server, which retrieves query results from the database server and formats them into dynamic Web pages that are viewed by the Web browser at the client side
- **Thebottomlayer**
  - includesall datamanagementservices

N-tier Architecture

It is possible to divide the layers between the user and the stored data further into finercomponents, thereby giving rise to n-tier architectures; where n may be four or five tiers. The business logic layer is divided into multiplelayers

- **Advantage**

  -any onetiercan runon an appropriateprocessor oroperating system platformand can be handled independently.

Vendors of ERP (enterprise resource planning) and CRM (customer relationship management) packages often use a middleware layer, which accounts for the front-end modules (clients) communicating with a number of back-end databases (servers).

ClassificationofDatabaseManagementSystems

Criteriaused toclassifyDBMSs are

1. Datamodelonwhich the DBMSisbased
   - Relational: represents a database as a collection of tables, where each table can be stored as a separate file.
   - Object: defines a database in terms of objects, their properties, and their operations. Objects with the same structure and behavior belong to a **class**, and classes are organized into **hierarchies** (or **acyclic graphs**). The operations of each class are specified in terms of predefined procedures called **methods**.
   - Hierarchical and network (legacy): The **network model** represents data as record types andalsorepresentsalimitedtypeof 1:Nrelationship,calleda**settype**. The

hierarchical model represents data as hierarchical tree structures. Each hierarchy represents a number of related records.

- Native XML DBMS: uses hierarchical tree structures. It combines databaseconcepts with concepts from document representation models. Data is representedas elements; with the use of tags, data can be nested to create complex hierarchical structures.

2. Numberofusers supported by thesystem
   - Single-user:supportonlyoneuseratatimeandaremostlyusedwith PCs.
   - Multiuser:supportconcurrentmultipleusers.
3. Numberofsitesover whichthedatabase is distributed
   - Centralized:dataisstoredatasinglecomputersite
   - Distributed:canhavetheactualdatabaseandDBMSsoftwaredistributedover many sites, connected by a computer network
     - **Homogeneous**DDBMSsusethesame DBMSsoftwareatallthesites
     - **Heterogeneous**DDBMScanusedifferentDBMSsoftwareateachsite
4. Cost
   - Opensource:productslikeMySQLandPostgreSQLthataresupportedbythird- party vendors with additional services.
   - Different types of licensing: Standalone single user versions of some systems like MicrosoftAccessaresoldpercopyorincludedintheoverallconfigurationof a desktop or laptop. In addition, data warehousing and mining features, as well as support for additional data types, are made available at extra cost.
5. Onthebasisof the**types of access pathoptions for**storingfiles

-Onewell-knownfamilyofDBMSsisbasedoninvertedfilestructures.

.6.GeneralpurposeorSpecial purpose

- When performance is a primary consideration, a special-purpose DBMS can be designed and built for a specific application; such a system cannot be used for other applications without major changes. Many airline reservations and telephone directory systems developed in the past are special-purpose DBMSs.

Chapter3:ConceptualDataModellingusingEntitiesandRelationships

Introduction

Conceptual modeling is a very important phase in designing a successful database application. Entity-Relationship (ER) model is a popular high-level conceptual data model. This model and its variations are frequently used for the conceptual design of database applications, and many database design tools employ its concepts.

UsingHigh-LevelConceptualDataModelsforDatabaseDesign

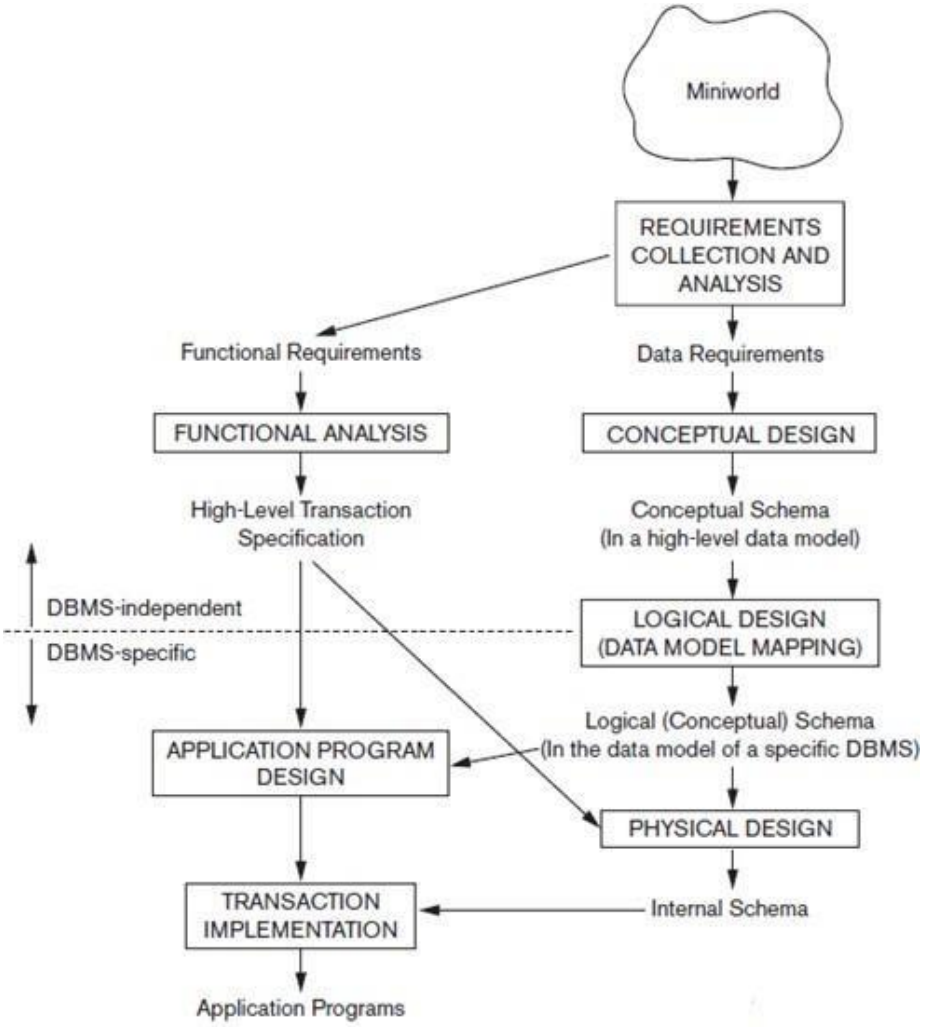


Figure3.1: Asimplifieddiagramtoillustratethe mainphasesofdatabasedesign.

The first step shown is **requirements collection and analysis**. During this step, the database designersinterviewprospectivedatabaseuserstounderstandand documenttheir **data requirements**. The result of this step isa concisely written set of users' requirements. Theserequirementsshouldbespecifiedinasdetailedandcompleteaformaspossible.Inparallel withspecifyingthedatarequirements,itis usefultospecifytheknown**functional requirements**

of the application. These consist of the userdefined **operations** (or **transactions**) that will be applied to the database, including both retrievals and updates.

Once the requirements have been collected and analyzed, the next step is to create a **conceptual schema** for the database, using a high-level conceptual data model. This step is called **conceptual design**. The conceptual schema is a concise description of the data requirements of the users and includes detaileddescriptions oftheentity types, relationships, and constraints; theseareexpressed using the concepts provided by the high-level data model.

The next step in database design is the actual implementation of the database, using a commercial DBMS. Most current commercial DBMSs use an implementation data model—such as the relational or the object-relational database model—so the conceptual schema is transformed from the high-level data model into the implementation data model. This step is called **logical design** or **data model mapping**; its result is a database schema in the implementation data model of the DBMS.

The last step is the **physical design** phase, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified. In parallel with these activities, application programs are designed and implemented as database transactions corresponding to the highlevel transaction specifications.

EntityTypes,EntitySets,Attributes,andKeys

TheERmodeldescribes dataasentities,relationships,andattributes.

Entitiesand Attributes

Entity: a thing in the real world with an independent existence. An entity may be an object with a physical existence (for example, a particular person, car, house, or employee) or it may be an object with a conceptual existence (for instance, a company, a job, or a universitycourse).

Attributes: Particularpropertiesthatdescribeentity.Forexample,anEMPLOYEEentitymay be described by the employee's name, age, address, salary, andjob.

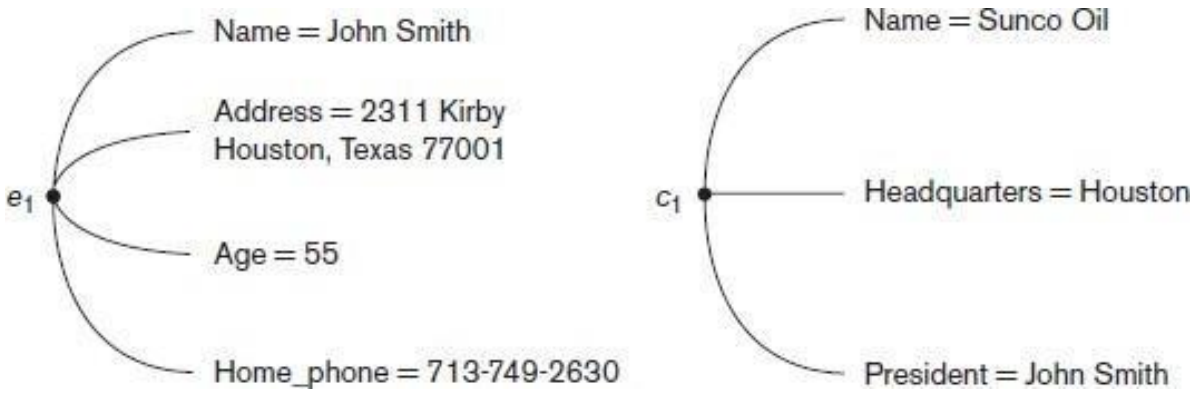


Figure3.2.1(a): Twoentities, EMPLOYEE e1,and COMPANYc1,and their attributes

Typesofattributes:

1.CompositeversusSimple(Atomic)Attributes
2.Single-valued versus multivalued
3. Storedversusderived
4. NULLvalues
5. Complexattributes

1. CompositeversusSimple(Atomic)Attributes

Composite Attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings.For example, the Address attribute of the EMPLOYEE entity can be subdivided into Street_address, City, State, and Zip.

Composite attributes can form a hierarchy. For example, Street_address can be further subdivided into three simple component attributes: Number, Street, andApartment_number.The valueofa compositeattribute istheconcatenation ofthevalues of its component simple attributes.

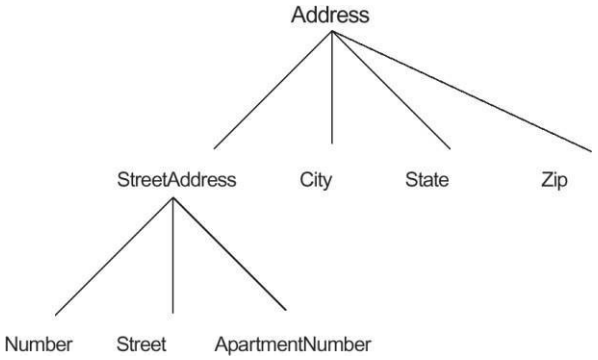


Figure3.2.1(b): Ahierarchyof composite attributes.

Attributes that are not divisible are called **simple** or **atomic attributes**. Example SSN of an employee.

2. Single-ValuedversusMultivaluedAttributes

Attributes that have a single value for a particular entity are called **single-valued**. For example, Age is a single-valued attribute of a person.

Attributes that can have a set of values for a particular entity are called **Multivalued Attributes.** For example Colors attribute for a car, or a College_degrees attribute for a person. A multivalued attribute may have lower and upper bounds to constrain the number of values allowed for each individual entity. For example, the Colors attribute of a car may be restricted to have between one and three values, if we assume that a car can have three colors at most.

3. StoredversusDerivedAttributes

An attribute, which cannot be derived from other attribute are called **stored attribute.** For example, Birth_Date of an employee

Attributes derived from other stored attribute are called **derived attribute.** For example age of an employee can be determined from the current (today's) date and Date of Birth

4. NullValueAttribute(OptionalAttribute)

In some cases, a particular entity may not have an applicable value for an attribute. For example, the Apartment_number attribute of an address applies only to addresses that are in apartment buildings and not to other types of residences, such as single-family homes. Similarly, a College_degrees attribute applies only to people with college degrees. For such situations, a special valuecalled **NULL** is created.Anaddress ofasingle-family homewould haveNULLfor its Apartment_number attribute, and a person with no college degree would have NULL for College_degrees. NULL can also be used if we do not know the value of an attribute for a particular entity

5. ComplexAttributes

If an attribute for an entity, is built using composite and multivalued attributes, then theseattributes are called complex attributes. For example, a person can have more than one residence and each residence can have multiple phones, an addressphone for a person entity can be specified as :

{Addressphone(phone{(AreaCode,PhoneNumber)},

Address(SectorAddress(SectorNumber,HouseNumber),
City, State, Pin))
}

Here {} are used to enclose multivalued attributes and () are used to enclosecomposite attributes with comma separating individual attributes

Entity Types, Entity Sets, Keys, and ValueSets

Entity Types

An **entity type** defines a collection (or set) of entities that have the same attributes. Each entity type in the database is described by its name and attributes. For example, a company employing hundreds of employees may want to store similar information concerning each of the employees. These employee entities share the same attributes, but each entity has its own value(s) for each attribute.

EntitySets

The collection of all entities of a particular entity type in the database at any point in time is called an **entity set**; the entity set is usually referred to using the same name as the entity type. For example, EMPLOYEE refers to both a type of entity as well as the current set of all employee entities in the database.

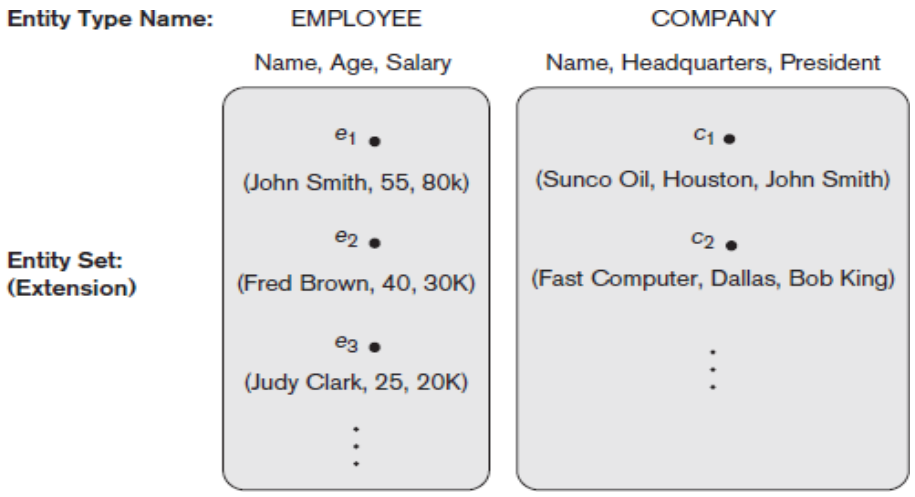


Figure3.2.2(a): Twoentitytypes,EMPLOYEEandCOMPANY,andsomememberentitiesofeach.

An entity type describes the **schema** or **intension** for a set of entities that share the same structure. The collection of entities of a particular entity type is grouped into an entity set, which is also called the **extension** of the entity type.

An **entity type** is represented in ER diagrams a **rectangular box** enclosing the entity type name. **Attribute names** are enclosed in **ovals** and are attached to their entity type by straight lines. **Composite attributes** are attached to their component attributes by straight lines. **Multivalued attributes** are displayed in **double ovals**

KeyAttributesofanEntityType

An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set. Such an attribute is called a key attribute, and its values can be used to identify each entity uniquely. For example, the Name attribute is a key of the COMPANY entity because no two companies are allowed to have the same name.

In ER diagrammatic notation, each key attribute has its name underlined inside the oval.Some entity types have more than one key attribute. For example, each of the Vehicle_id andRegistration attributes of the entity type CAR isa key in its own right

Example: TheCARentitytypewith twokeyattributes,RegistrationandVehicle_id.

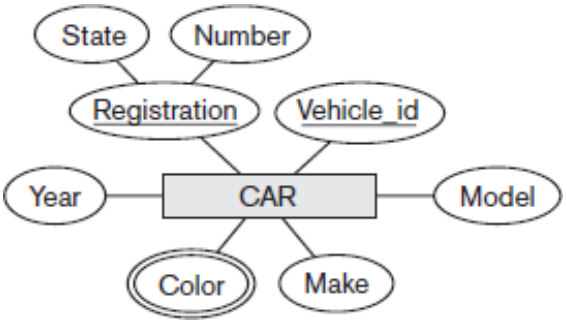


Figure3.2.2(b) :ERdiagramnotation

CAR
Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}

CAR_1
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})

CAR_2
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})

CAR_3
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

Entitysetwiththreeentities.

ValueSets(Domains)ofAttributes

Each simple attribute of an entity type is associated with a **value set** (or **domain** of values), which specifies the set of values that may be assigned to that attribute for each individual entity. For example, if the range of ages allowed for employees is between 16 and 70, we can specify thevalue set of the Age attribute of EMPLOYEE to be the set of integer numbers between 16 and70.

Value sets are not displayed in ER diagrams, and are specified usingthebasic data typesavailable in most programming languages, such as integer, string, Boolean,float, enumerated type, subrange, and so on.

Mathematically, an attribute A of entity set E whose value set is V can be defined as a **function from E to the power set P(V) of V:** A : E → P(V).We refer to the value of attribute A for entity e as A(e).A NULL value is represented by the empty set.

ASampleDatabaseApplication

Miniworld : COMPANY database keeps track of a company's employees, departments, and projects.

- After the requirements collection and analysis phase, the database designers provide the following description of the *miniworld:*
  - Thecompanyisorganizedintodepartments.
  - Each department has a unique name, a unique number, and a particular employee who manages the department.We keep track of the start date when that employee began managing the department. A department may have several locations.
  - A department controls a number of projects, each of which has a unique name, a unique number, and a single location.
  - Westore each employee's name, Social Security number, address, salary,gender, and birth date.
  - An employee is assigned to one department, but may work on several projects, which are not necessarily controlled by the same department.
  - We keep track of the current number of hours per week that an employee works on each project. We also keep track of the direct supervisor of each employee (who is another employee).
  - We want to keep track of the dependents of each employee for insurance purposes. Wekeep each dependent's first name, gender, birth date, and relationship to the employee.

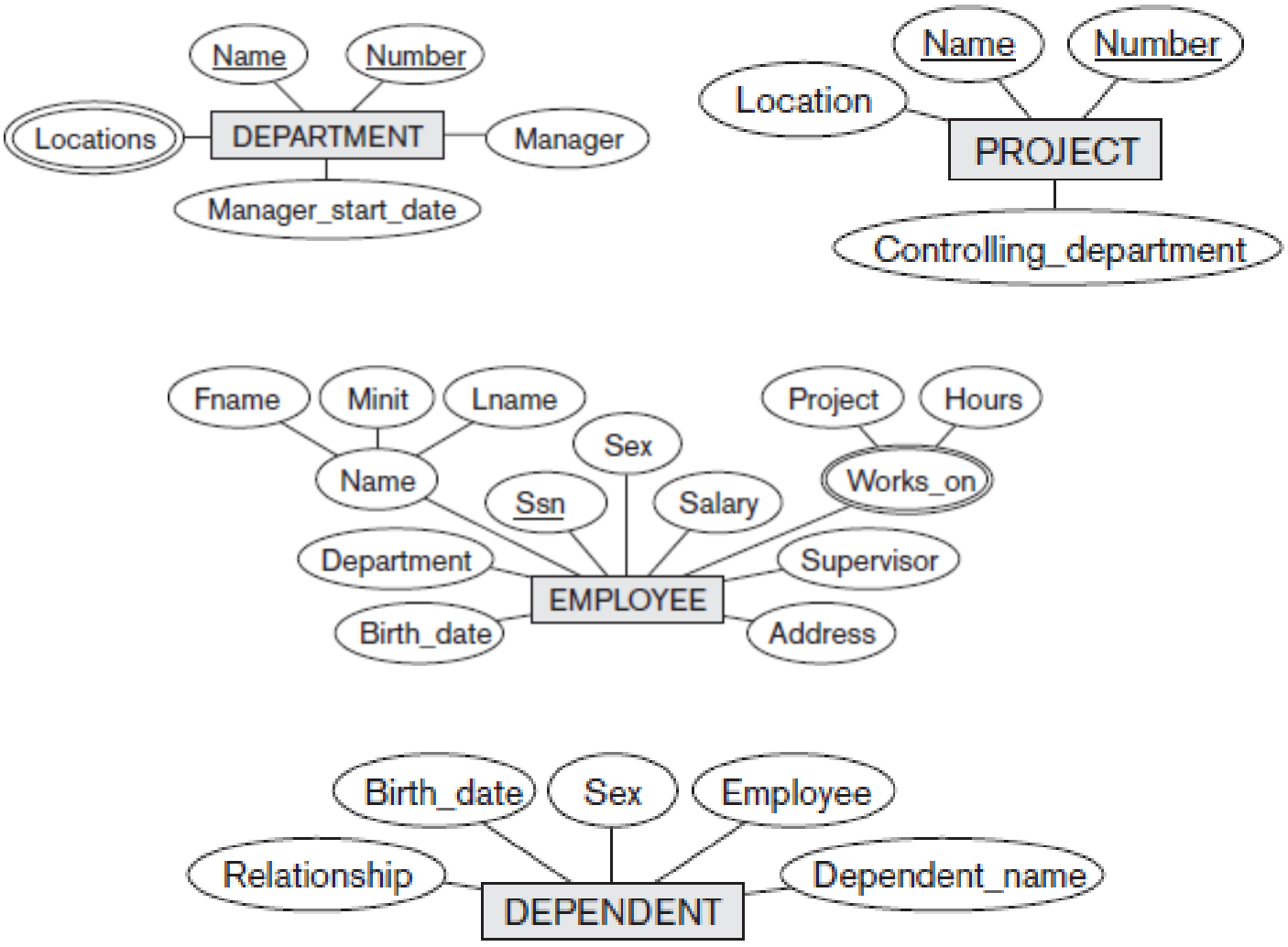


Figure3.3(a): Preliminarydesignofentitytypesforthe COMPANYdatabase.Someoftheshown attributes will be refined into relationships.

RelationshipTypes,RelationshipSets,Roles,andStructuralConstraints

Thereareseveralimplicitrelationshipsamongthevariousentitytypes.Wheneveranattributeofone entity type refers to another entity type, some relationship exists. For example

- TheattributeManagerof DEPARTMENTreferstoan employeewho manages thedepartment
- TheattributeControllingdepartmentofPROJECTreferstothedepartmentthatcontrolsthe project
- TheattributeSupervisorofEMPLOYEEreferstoanotheremployee-theonewhosupervisesthis employee
- TheattributeDepartmentofEMPLOYEEreferstothedepartmentforwhichtheemployeeworks In the ER model,these references should not be represented as attributes but asrelationships

RelationshipTypes,Sets,andInstances

A **relationship type R** among n entity types E_1, E_2, ..., E_n defines a set of associations—or a **relationship set**—among entities from these entity types. Entity types and Entity sets, a Relationship type and its corresponding Relationship set are usually referred to by the same name, R.

Mathematically, the relationship set R is a set of relationship instances r_i , where each r_i associates n individual entities (e_1, e_2, ..., e_n), and each entity e_i in r_i is a member of entity set E_j , $1 \leq j \leq n$. Each of the entity types E_1, E_2, ..., E_n is said to participate in the relationship type R. similarly, each of the individual entities e_1, e_2, ..., e_n is said to participate in the relationship instance $r_i = (e_1, e_2, ..., e_n)$

Informally, each relationship instance r_i in R is an association of entities, where the association includes exactly one entity from each participating entity type. For example, consider a relationship type WORKS_FOR between the two entity types EMPLOYEE and DEPARTMENT, which associates each employee with the department for which the employee works in the corresponding entity set. Each relationship instance in the relationship set WORKS_FOR associates one EMPLOYEE entity and one DEPARTMENT entity.

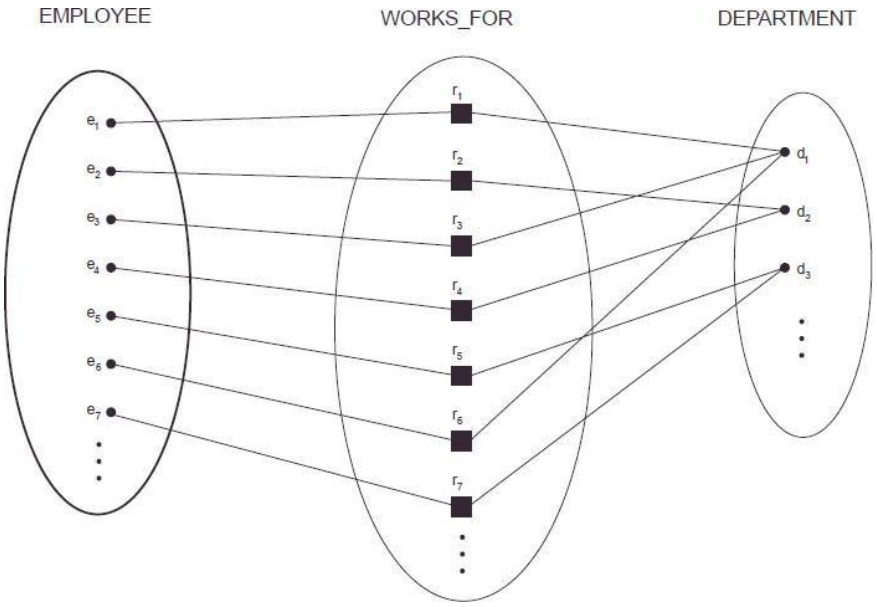


Figure 3.4.1: Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

employeese_1 ,e_3,ande_6 workfordepartmentd_1.employeese_2 ande_4 workfordepartmentd_2 and employees e_5 and e_7 work for department d_3.

In ER diagrams, relationship types are displayed as diamond-shaped boxes, which are connected by straight lines to the rectangular boxes representing the participating entity types. The relationship name is displayed in the diamond-shaped box.

RelationshipDegree,RoleNames,andRecursiveRelationships

DegreeofaRelationshipType

The degree of a relationship type is the number of participating entity types. A relationship type of degree two is called **binary**, and one of degree three is called **ternary** An example of a binary relationship WORKS_FOR and ternary relationship is SUPPLY

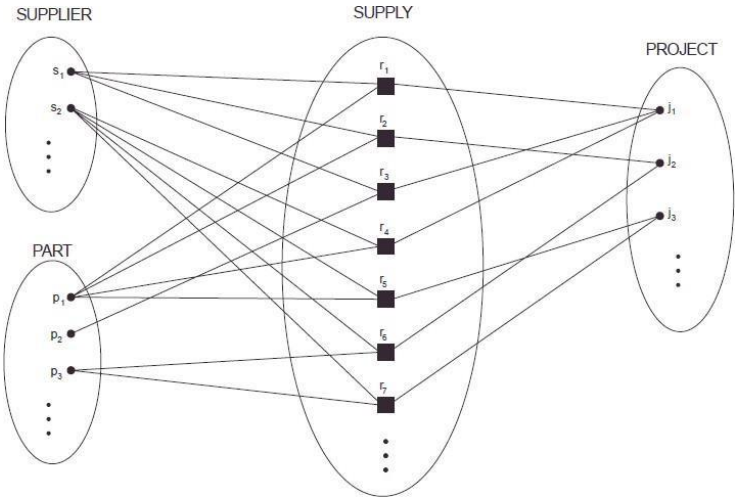


Figure3.4.2(a):SomerelationshipinstancesintheSUPPLYternaryrelationshipset.

Each relationship instance r_i associates three entities—a supplier s, a part p and a project j—whenever s supplies part p to project j.

RelationshipsasAttributes

It is sometimes convenient to think of a binary relationship type in terms of attributes. Consider the WORKS_FOR relationship type. One can think of an attribute called Department of the EMPLOYEE entity type, where the value of Department for each EMPLOYEE entity is a reference to the DEPARTMENT entity for which that employee works. This concept of representing relationship types as attributes is used in a class of data models called **functional data models**.

Inrelationaldatabases,foreignkeysareatypeofreferenceattributeusedtorepresentrelationships.

RoleNamesandRecursiveRelationships

Eachentitytypethatparticipates inarelationshiptypeplaysaparticularroleintherelationship.

The **role name** signifies the role that a participating entity from the entity type plays in each relationship instance,andhelpstoexplainwhattherelationshipmeans.Forexample, intheWORKS_FORrelationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the roleof department or employer.

Role names are not technically necessary in relationship types where all the participating entity types are distinct, since each participating entity type name can be used as the role name. However, in some cases the same entity type participates more than once in a relationship type in different roles.

In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called **recursive relationships.** Example ofrecursive relationships **:** SUPERVISION relationship type

The SUPERVISION relationship type relates an employee to a supervisor, where both employee and supervisor entities are members of the same EMPLOYEE entity set. Hence, the EMPLOYEE entity type participates twice in SUPERVISION: once in the role of supervisor (or boss), and once in the role of supervisee (or subordinate). Each relationship instance r_i in SUPERVISION associates two employee entities e_j and e_k , one of which plays the role of supervisor and the other the role of supervisee.

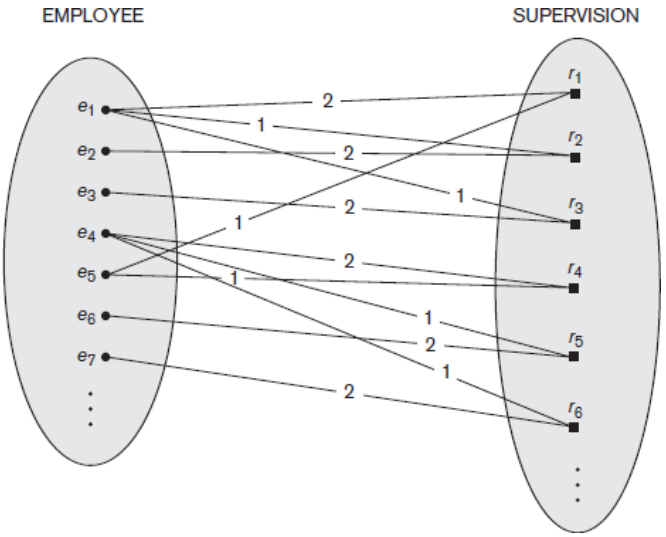


Figure 3.4.2(b): Arecursiverelationship SUPERVISIONbetweenEMPLOYEEinthesupervisorrole(1)andEMPLOYEEinthe subordinaterole (2).

ConstraintsonBinaryRelationshipTypes

Relationshiptypesusuallyhavecertainconstraintsthatlimitthepossiblecombinationsofentitiesthat may participate in the corresponding relationship set. These constraints are determined from the

miniworld situation that the relationships represent.For example, if the company has a rule that each employee must work for exactly one department, then we would like to describe this constraint in the schema. Two main types of binary relationship constraints:

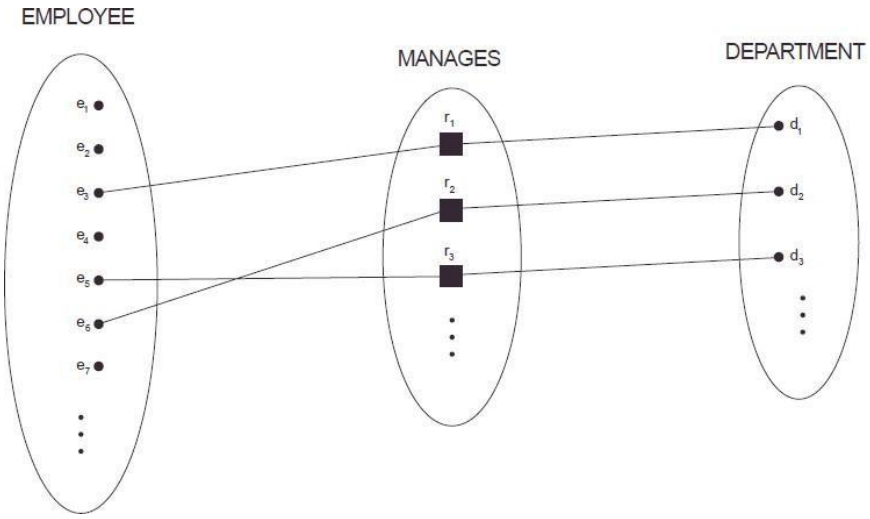
1. cardinalityratio
2. participation.

CardinalityRatiosforBinaryRelationships

Thecardinalityratiofor abinary relationshipspecifiesthe maximumnumberof relationshipinstancesthat an entity can participate in. For example, in the WORKS_FOR binary relationship type, DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N, meaning that each department can be related to any number of employees, but an employee can be related to (work for) only one department. Thepossible cardinality ratios for binary relationship types are 1:1, 1:N, N:1, and M:N.

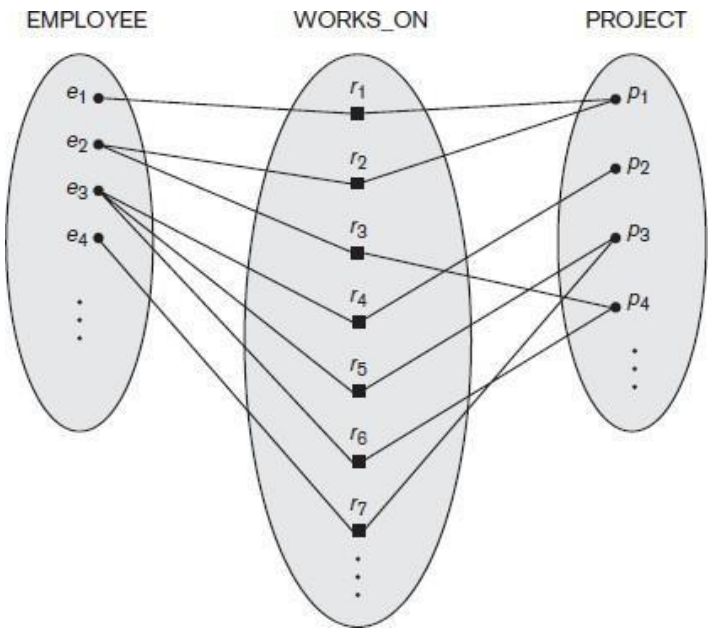
Exampleofa 1:1binary relationship

- MANAGESwhichrelates adepartmententitytotheemployeewho managesthatdepartment
- Thisrepresentstheminiworldconstraintsthat—atanypointintime—anemployeecanmanageone department only and a department can have one manager only



ExampleofaM:Nbinary relationship

- TherelationshiptypeWORKS_ONisofcardinalityratioM:N,becausethemini-worldruleisthat an employee can work on several projects and a project can have severalemployees.
- CardinalityratiosforbinaryrelationshipsarerepresentedonERdiagramsbydisplaying1,M,andN on the diamonds



ParticipationConstraintsandExistenceDependencies

Theparticipationconstraintspecifieswhethertheexistenceofanentitydependsonitsbeingrelatedto another entity via the relationship type. This constraint specifies the minimum number of relationship instances that each entity can participate in, and is sometimes called the minimum cardinality constraint.There are two types of participation constraints:

- Total
- Partial

Total participation

If a company policy states that every employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS_FOR relationship Instance. Thus, the participation of EMPLOYEE in WORKS_FOR is called total participation, meaning that every entity in the total set of employee entities must be related to a department entity via WORKS_FOR. Total participation is also called **existence dependency**

Partial participation

wedonotexpecteveryemployeetomanageadepartment.SotheparticipationofEMPLOYEEinthe MANAGES relationship type is partial, meaning that some or part of the set of employee entities are related to some department entity via MANAGES, but not necessarily all.

InERdiagrams,**totalparticipation**isdisplayedasa**doubleline**connectingtheparticipatingentitytype to the relationship, whereas **partial participation** is represented by a **single line.**

cardinalityratio+participationconstraints=structuralconstraintsofarelationship type.

AttributesofRelationshipTypes

Relationship types can also have attributes, similar to those of entity types. For example, to record the number of hours per week that an employee works on a particular project, we can include an attribute Hours for the WORKS_ON relationship type. Another example is to include the date on which a manager started managing a department via an attribute Start_date for the MANAGES relationship type.
Attributes of 1:1 or 1:N relationship types can be migrated to one of the participating entity types. For a 1:N relationship type, a relationship attribute can be migrated only to the entity type on the N-side of the relationship. For M:N relationship types, some attributes may be determined by the combination of participating entities in a relationship instance, not by any single entity. Such attributes must be specified as relationship attributes.

WeakEntityTypes

Entity types that do not have key attributes of their own are called **weak entity** types. Entities belongingto a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. We call this other entity type the identifying or **owner entity type.** We call the relationship type that relates a weak entity type to its owner the **identifying relationship** of the weak entity type.

Consider the entity type DEPENDENT, related to EMPLOYEE, which is used to keep track of the dependents of each employee via a 1:N relationship. In our example, the attributes of DEPENDENT are Name,Birth_date, gender, and Relationship (to the employee). Two dependents of two distinct employees may, by chance, have the same values for Name, Birth_date, gender, and Relationship, but they are still distinct entities. They are identified as distinct entities only after determining the particular employee

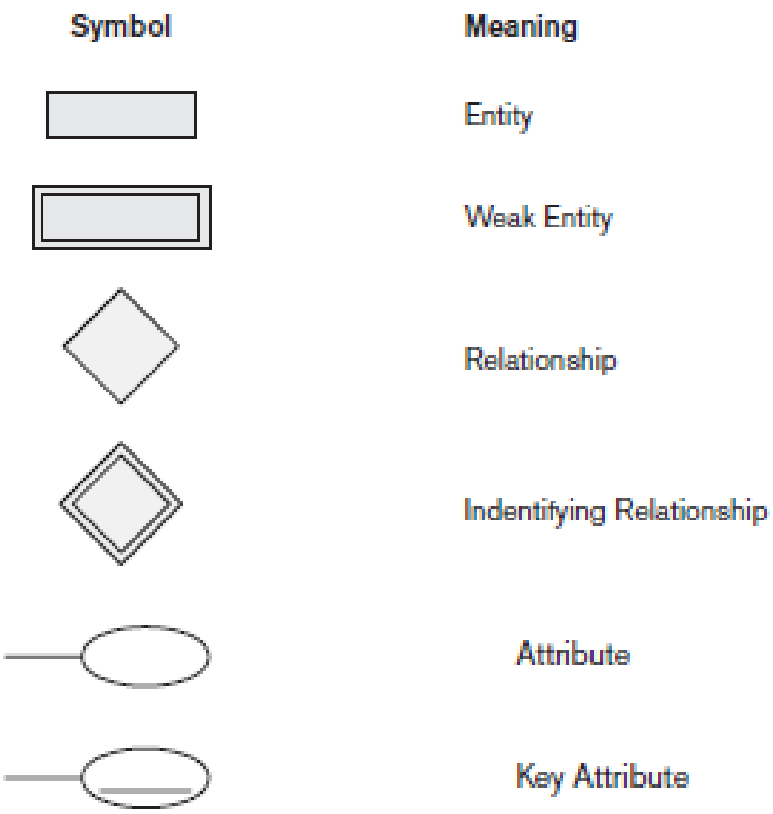
entity to which each dependent is related. Each employee entity is said to own the dependent entities that are related to it.

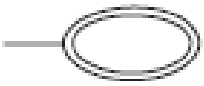
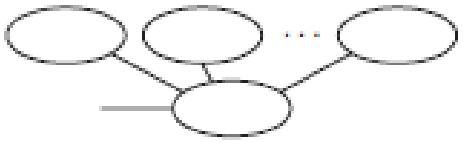
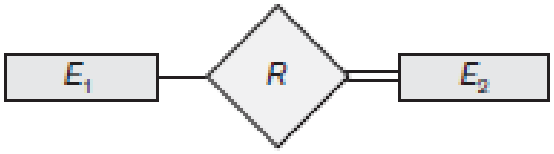
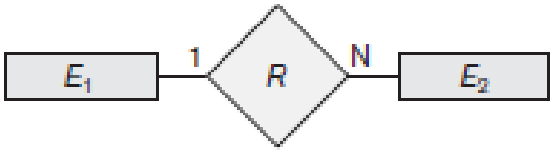
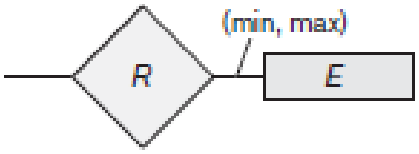
A weak entity type always has a total participation constraint (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity. A weak entity type normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are related to the same owner entity. In our example, if we assume that no two dependents of the same employee ever have the same first name, the attribute Name of DEPENDENT is the partial key.

In ER diagrams, both a weak entity type and its identifying relationship are distinguished by surrounding theirboxesanddiamondswithdoublelines.Thepartialkeyattributeisunderlinedwithadashedordotted line.

ERDiagrams,NamingConventions,andDesignIssues

SummaryofNotationforERDiagrams



Symbol	Meaning
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1: N for E_1:E_2 in R
	Structural Constraint (min, max) on Participation of E in R

Proper Naming of Schema Constructs

- Choose names that convey, asmuch as possible,themeaningsattached tothedifferent constructsintheschema.
- Use *singular names* for entity types,rather than plural ones, because the entity type name applies to each individual entity belonging to that entity type
- In ER diagrams, entity type and relationship type names are uppercase letters, attribute names have their initial letter capitalized, and role names are lowercase letters.
- As a general practice, given a narrative description of the database requirements, the nouns appearing in the narrative tend to give rise to entity type names, and the verbs tend to indicate names of relationship types. Attribute names generally arise from additional nouns thatdescribe the nouns corresponding to entity types.

- Anothernamingconsiderationinvolveschoosingbinaryrelationshipnames to maketheER diagram of the schema readable from left to right andfrom top to bottom.

DesignChoicesforERConceptual Design

In general, the schema design process should be considered an iterative refinement process, where an initial design is created and then iteratively refined until the most suitable design is reached. Some of the refinements that are often used include the following:

- A concept may befirst modeled as an attribute and then refined into a relationship because it is determined that the attribute is a reference to another entity type. It is often the casethat a pair of such attributes that are inverses of one another are refined into a binary relationship.
- Similarly, an attribute that exists in several entity types may be elevated or promoted to an independent entity type. For example, suppose that several entity types in a UNIVERSITY database, such as STUDENT, INSTRUCTOR, and COURSE, each has an attribute Department in the initial design; the designer may then choose to create an entity type DEPARTMENT with a single attribute Dept_name and relate it to the three entity types (STUDENT, INSTRUCTOR, and COURSE) via appropriate relationships.
- An inverse refinement to the previous case may be applied—for example, if an entity type DEPARTMENT exists inthe initial design witha single attribute Dept_name and isrelated to only one other entity type, STUDENT. In this case, DEPARTMENT may be reduced or demoted to an attribute of STUDENT.

AlternativeNotationsforERDiagrams

There are many alternative diagrammatic notations for displaying ER diagrams. One alternativeER notation for specifying structural constraints on relationships, which replaces the cardinality ratio (1:1, 1:N, M:N) and single/double line notation for participation constraints. This notation involves associating a pair of integer numbers (min, max)with each *participation* of an entity type *E* in a relationship type *R*, where $0 \leq \min \leq \max$ and $\max \geq 1$.

The numbers mean that for each entity e in E, e must participate in at least min and at most max relationship instances in R at any point in time. In this method, min= 0 implies partial participation, whereas min > 0 implies total participation.

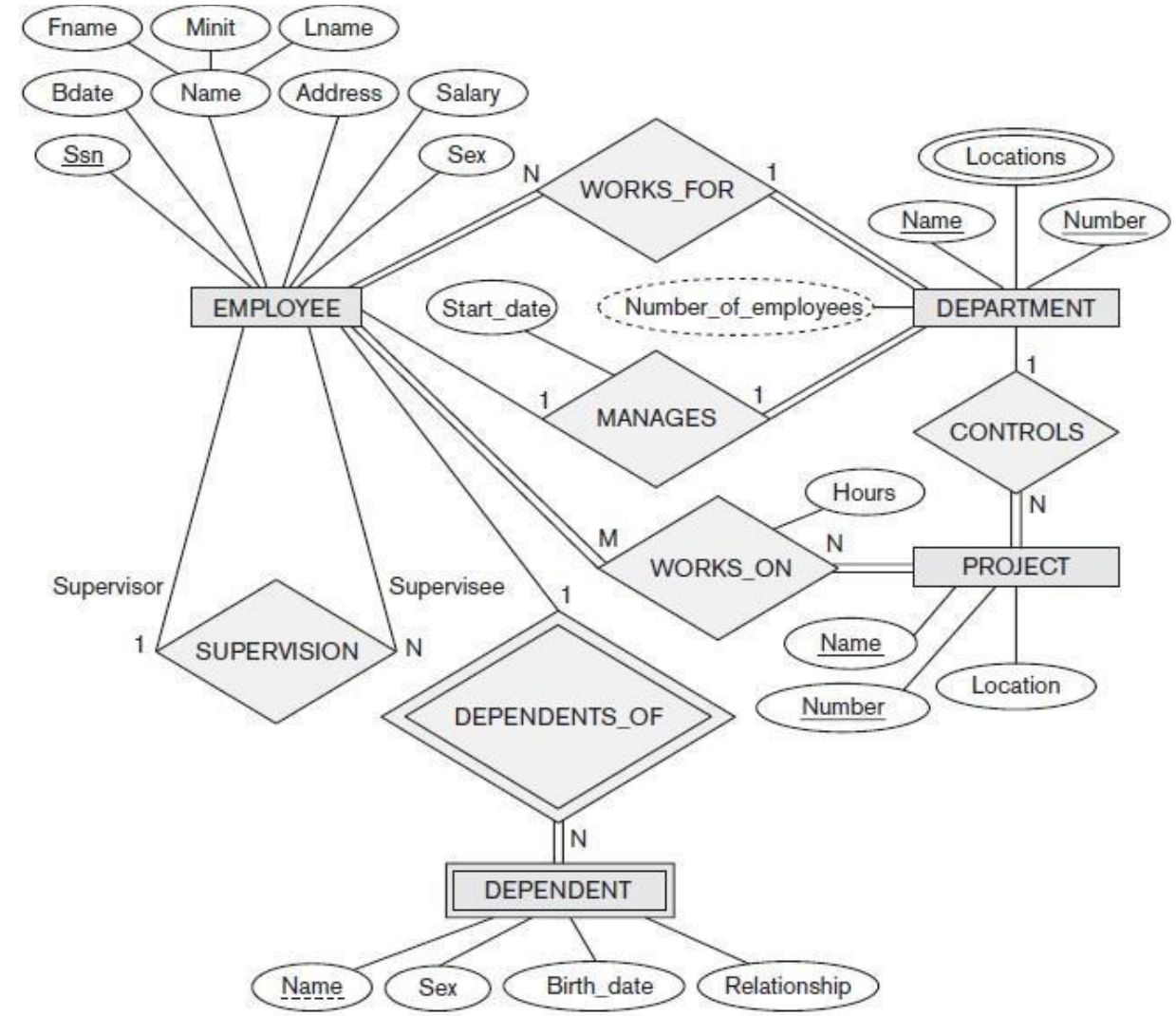


Figure3.6.4 (a): ER diagramforCompany Database

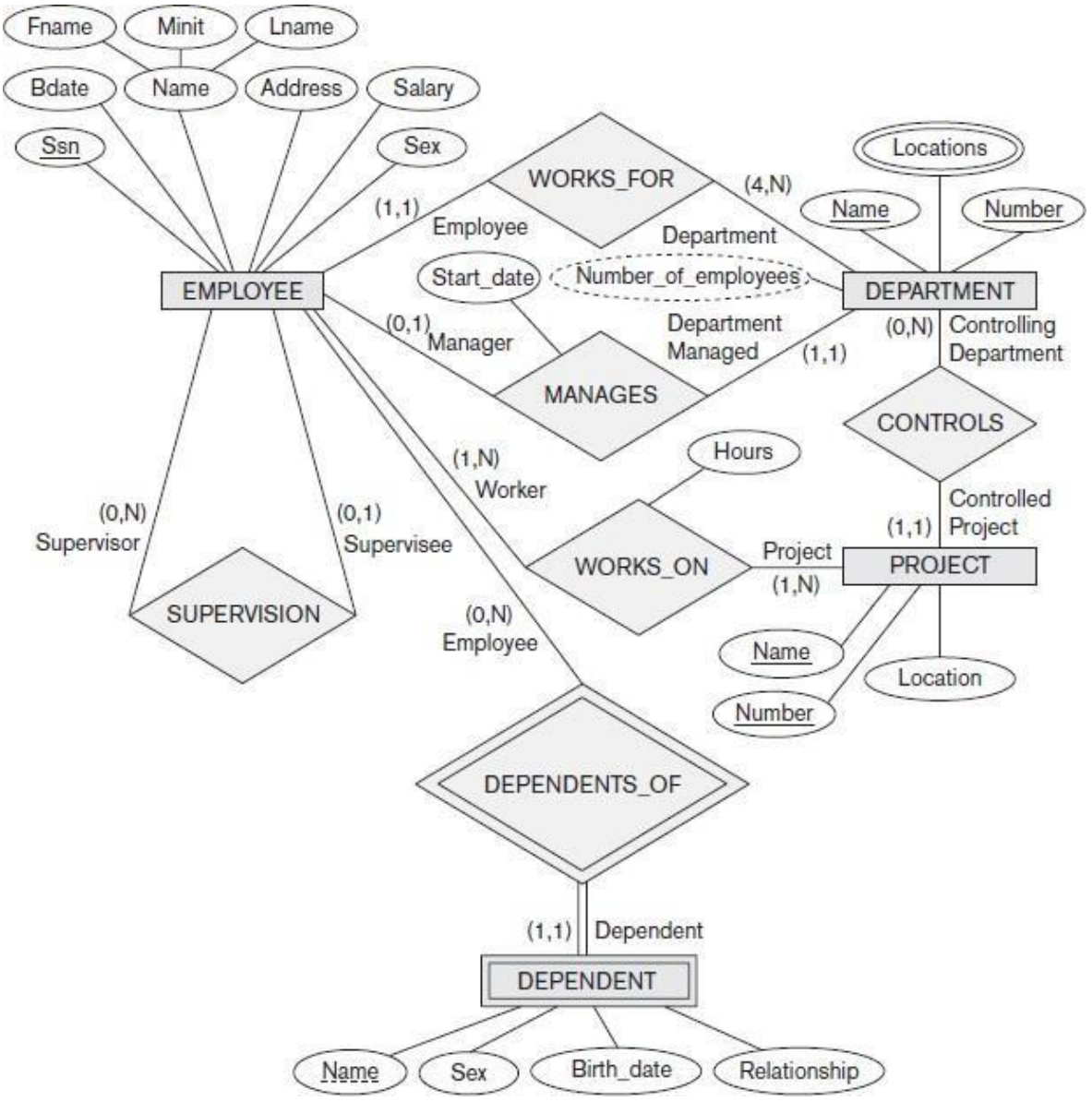


Figure3.6.4(b):ER diagramforCompany Database(usingalternativenotation)

RelationshipTypesofDegreeHigherthanTwo

ChoosingbetweenBinaryandTernary(orHigher-Degree)Relationships A relationship type R of degree n will have n edges inan ER diagram, one connecting R to each participating entity type

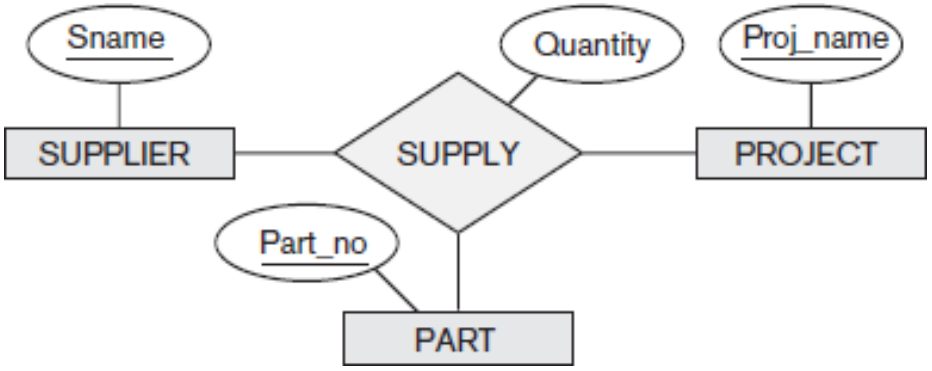


Fig3.7.1(a): TheSUPPLYrelationship

Figure 3.7.1(a0 shows the ER diagram notation for a ternary relationship. SUPPLY is a set of relationship instances (*s*, *j*, *p*), where *s* is a SUPPLIER who is currently supplying a PART *p* to a PROJECT *j*.

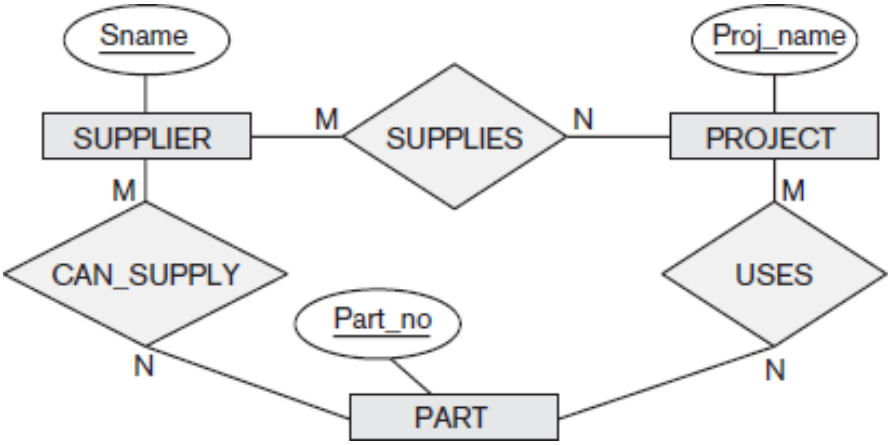


Fig3.7.1(b)ERdiagramforthreebinaryrelationshiptypes CAN_SUPPLY,USES,and SUPPLIES

Figure3.7.1(b)showsanERdiagramforthreebinaryrelationshiptypesCAN_SUPPLY,USES, and SUPPLIES. CAN_SUPPLY between SUPPLIER and PART, includes an instance (s, p)

whenever supplier s can supply part p (to any project). USES between PROJECT and PART, includes an instance (j, p) whenever project j uses part p. SUPPLIES between SUPPLIER and PROJECT, includes an instance (s, j) whenever supplier s supplies some part to project j.

Some database design tools are based on variations of the ER model that permit only binary relationships. In this case, a ternary relationship such as SUPPLY must be represented as a weak entity type, with no partial key and with three identifying relationships.

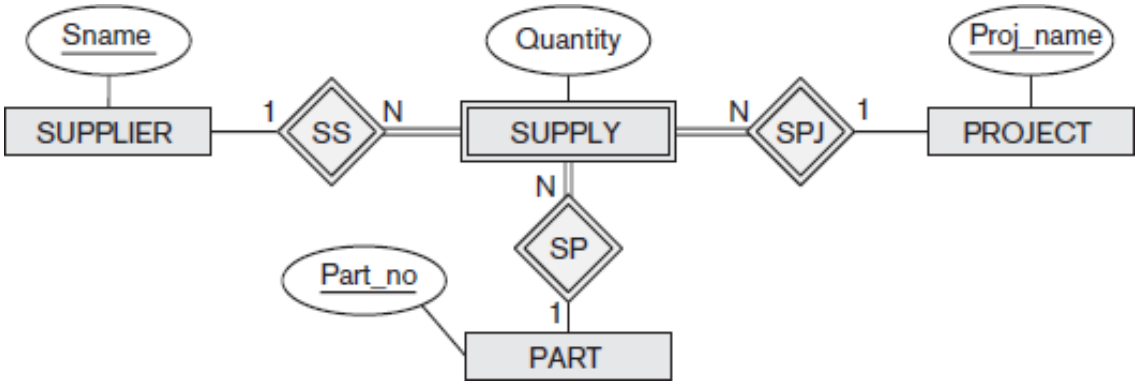


Fig 3.7.1(c): SUPPLY represented as a weakentitytype

The three participating entity types SUPPLIER, PART, and PROJECT are together the owner entity types.Hence, an entity in the weak entity type is identified by the combination of its three owner entities from SUPPLIER, PART, and PROJECT.

ConstraintsonTernary(orHigher-Degree)Relationships

Therearetwonotations forspecifyingstructural constraintsonn-ary relationships

1. basedonthecardinalityrationotationofbinaryrelationshipsdisplayed
   - 1,M,orNisspecifiedoneachparticipationarc(bothMandNsymbolsstandfor many or any number)
2. basedonthe(min,max)notation
   - specifiesthateachentityisrelatedtoatleastminandatmostmaxrelationship instances in the relationship set

SpecializationandGeneralization

Specialization

Specialization is the process of defining a *set of subclasses* of an entity type; this entity type is called the **superclass** of the specialization. The set of subclasses that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass.

For example, the set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of the superclass EMPLOYEE that distinguishes among employee entities based on the job type of each employee entity. We may have several specializations of the same entity type based on different distinguishing characteristics. For example, another specialization of the EMPLOYEE entity type may yield the set of subclasses {SALARIED_EMPLOYEE, HOURLY_EMPLOYEE};thisspecializationdistinguishesamongemployeesbasedonthemethod of pay.

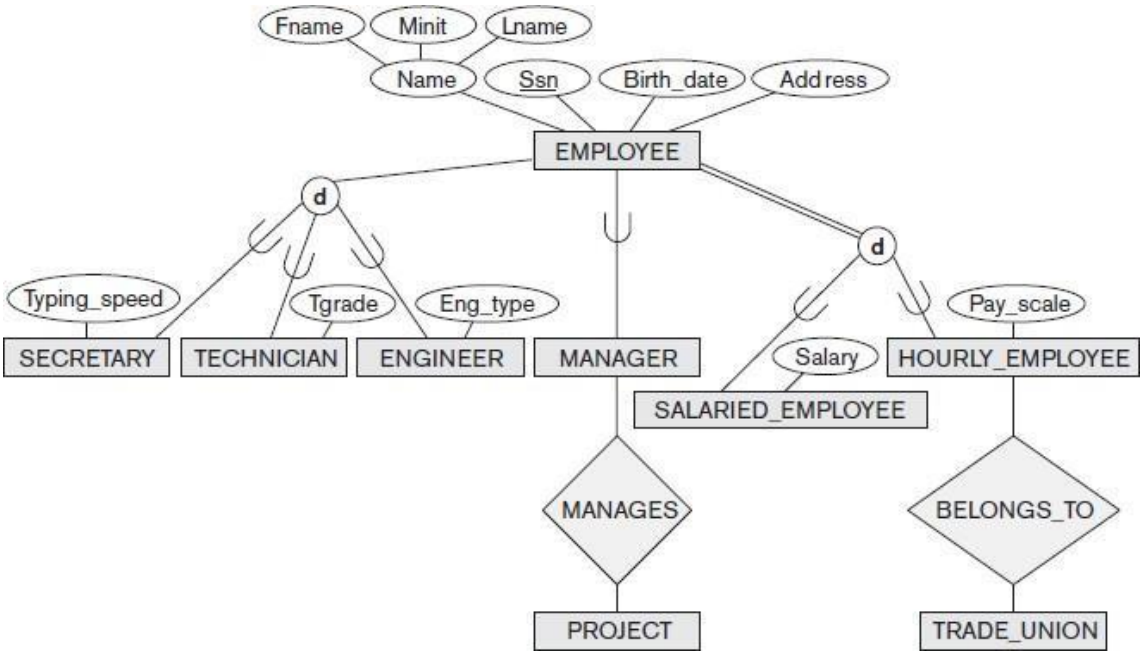


Figure3.8.1(a):EERdiagramnotationtorepresentsubclassesandspecialization.

Figure3.8.1(a)shows howwerepresent aspecialization diagrammatically in an EER diagram. The subclasses that define a specialization are attached by lines to a circle that represents the specialization, which is connected in turn to the superclass. The *subset symbol* on each line connecting a subclass to the circle indicates the direction of the superclass/subclass relationship.5 Attributes that apply only to entities of a particular subclass—such as TypingSpeed of

SECRETARY—are attached to the rectangle representing that subclass. These are called **specific attributes** (or **local attributes**) of the subclass.

Similarly, a subclass can participate in **specific relationship types**, such as the HOURLY_EMPLOYEE subclass participating in the BELONGS_TO relationship inFigureFigure 3.8.1(b).

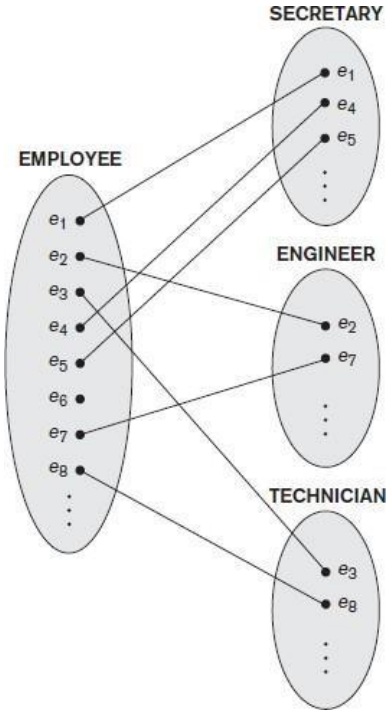


Figure3.8.1(b): Instancesof aspecialization

Figure 3.8.1 (b) shows a few entity instances that belong to subclasses of the {SECRETARY, ENGINEER, TECHNICIAN} specialization. An entity that belongs to a subclass represents the samereal-worldentityastheentityconnectedtoit intheEMPLOYEEsuperclass,eventhoughthe same entity is shown twice; for example, e1 is shown in both EMPLOYEE and SECRETARY. There are two main reasons for including class/subclass relationships and specializations in a data model.

- The firstisthat certain attributes mayapply to some but not all entities of the superclass. A subclass is defined in order to group the entities to which these attributes apply. The members of the subclass may still share the majority of their attributes with the other members of the superclass.
- The second reason for using subclasses is that some relationship types may be participated in only by entities that are members of the subclass. For example, if only HOURLY_EMPLOYEES can belong to a trade union, we can represent that fact by

creatingthesubclassHOURLY_EMPLOYEEofEMPLOYEEandrelatingthesubclassto an entity type TRADE_UNION via the BELONGS_TO relationship type

Insummary,thespecializationprocessallowsustodothefollowing:

- Defineaset ofsubclasses of an entitytype
- Establishadditionalspecificattributeswitheachsubclass
- Establish additional specific relationship types between each subclass and otherentitytypes or other subclasses

Generalization

Generalization process can be viewed as being functionally the inverse of the specialization process. It is a process of defining a generalized entity type from the given entity types. Generalization is the reverse process of abstraction in which we suppress the differences among several entity types, identify their common features, and **generalize** them into a single **superclass**

Forexample,considertheentitytypes CARandTRUCKshowninFigure 3.8.2(a).

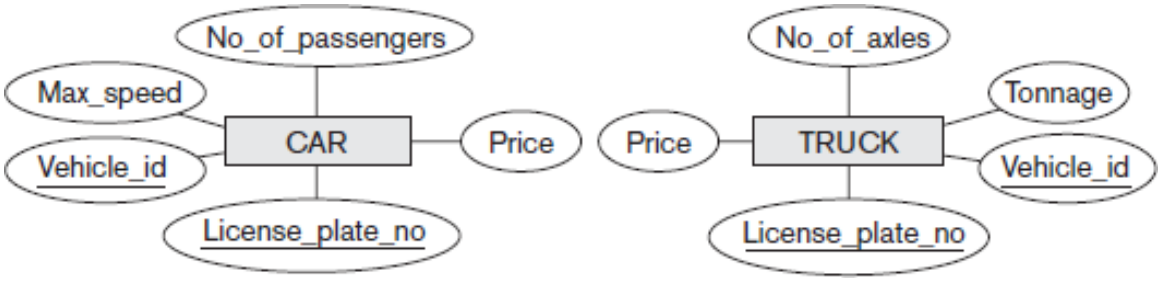


Figure3.8.2(a):Twoentitytypes,CARand TRUCK

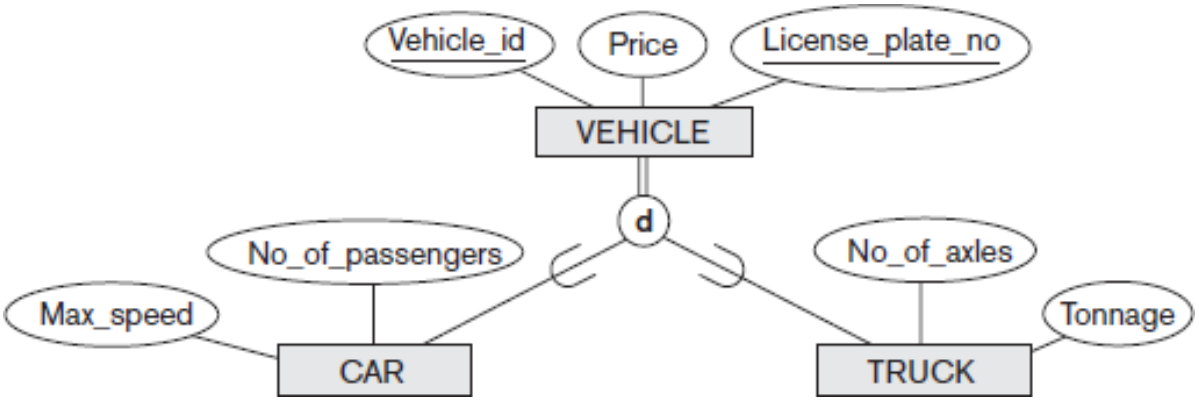


Figure3.8.2(b):Generalizing CARandTRUCK intothesuperclass VEHICLE.

Because they have several common attributes, they can be generalized into the entity type VEHICLE, as shown in Figure 3.8.2(b). Both CAR and TRUCK are now subclasses of the generalized superclass VEHICLE.

A diagrammatic notation to distinguish between generalization and specialization is used in some design methodologies. An arrow pointing to the generalized superclass represents a generalization, whereas arrows pointing to the specialized subclasses represent a specialization.

Question Bank

1. Definethe followingterms:
   i)data ii) database iii) DBMS iv) program-data independence v)Canned transaction
2. Definethedatabaseandbrieflyexplain theimplicitpropertiesofthe database.
3. DiscussthemainCharacteristicsofthedatabaseapproachandhowdoesitdifferfrom Traditional file systems?
4. Whatarethedifferenttypesofdatabaseendusers?Discussthemainactivitiesofeach.
5. BreiflydiscusstheadvantagesofusingtheDBMS.
6. Definethe followingterms:
   i)datamodeii)databaseschema iii)databasestateiv)schemadiagram
7. Describethethree-schemaarchitecture.Whydoweneedmappingsbetweenschema levels?
8. Whatisthedifferencebetweenlogicaldataindependenceandphysicaldataindependence?
9. Whatisthedifferencebetweenproceduralandnonproceduralmls?
10. Discussthevariousdatabaselanguages.
11. Discussthedifferenttypesof user-friendlyinterfacesandthetypesofuserswhotypically use each.
12. Explainthecomponentmodules ofDBMS andtheirinteraction withthehelpof adiagram.
13. Discusssometypes ofdatabaseutilitiesandtools andtheirfunctions.
14. Explaintwo-tierandthree-tierarchitecture.
15. Discusstheclassificationofdatabasemanagementsystems.
16. Explainwith aneat diagram,thephasesofdatabasedesign.
17. Definethe followingterms:
    i)Entityii)attributeiii)entitytypeiv)entitysetv)keyattributevi)valueset v)degreeofarelationshiptypevi) rolenamesvii)cardinalityratioviii)participation constraints
18. Explainthedifferenttypesof attributesthatoccurinanERmodelwith anexample.
19. Whatismeantbyarecursiverelationshiptype?Givesomeexamples of recursive relationship types.
20. Whatisaweak entitytype?Explaintheroleofpartial key inthedesignof weakentity type.
21. Listsymbolsused inERdiagramand theirmeaning.

22. Discussthenaming conventionsusedfor ERschemadiagrams.
23. Explainwithanexamplespecializationandgeneralization.
24. Design an ER diagram for an insurance company. Assume suitable entity types like CUSTOMER, AGENT, BRANCH, POLICY, PAYEMENT and the relationship between them.
25. Designan ER-diagramfortheMovie -databaseconsideringthe following requirements:
    i) EachMovieisidentifiesbyitstitleandyearofrelease, ithaslengthinminutesandcan have zero of more quotes, language.
    ii) Productioncompanies are identified by Name, they have address, and eachproduction company can produce one or more movies.
    iii) Actors are identified by Name and Date of Birth, they can act in one or more movies and
    each actorhasaroleina movie.
    iv) Directorsareidentified by Name and Date of Birth, so each Directorcan direct oneor more movie and each movie can be directed by one or more Directors.
    v) EachmoviebelongstoanyonecategorylikeHorror,action,Drama,etc.
26. Design an Entity Relationship (ER) model for a college database . Say we havethe following statements.

    1. Acollegecontainsmanydepartments
    2. Eachdepartmentcanofferanynumberofcourses
    3. Manyinstructorscanworkinadepartment
    4. Aninstructorcanworkonly inone department
    5. ForeachdepartmentthereisaHead
    6. Aninstructorcanbeheadofonlyone department
    7. Eachinstructor cantake anynumberofcourses
    8. Acourse canbetaken byonly oneinstructor
    9. Astudent can enroll forany numberof courses
    10. Eachcoursecan have any numberofstudents
27. Considerthefollowing set ofrequirements fora UNIVERSITYdatabase that isusedto keep track of students' transcripts

    a. The university keeps track of each student's name, student number, Social Security number, current address and phone number, permanent address and phone number, birth date, sex, class (freshman, sophomore, ..., graduate), major department, minordepartment

(ifany),and degreeprogram (B.A.,B.S., ..., Ph.D.). Someuserapplications need to referto the city, state, and ZIP Code of the student's permanent address and to the student's last name. Both Social Security number and student number have unique values for each student.

b. Eachdepartmentisdescribedbyaname,departmentcode,officenumber,officephone number, and college. Both name and code have unique values for eachdepartment.

c. Each course has a course name, description, course number, number of semester hours, level,andofferingdepartment.Thevalueofthecoursenumberisuniqueforeach course.

d. Eachsectionhasaninstructor,semester,year,course,andsectionnumber. Thesection number distinguishes sections of the same course that are taught during the same semester/year;itsvaluesare1,2,3,...,uptothenumberofsectionstaught duringeach semester.

e. Agrade report hasastudent,section,lettergrade,and numericgrade(0, 1,2, 3,or 4).

DesignanERschemaforthisapplication,anddrawanERdiagramforthe schema.Specify key attributes of each entity type, and structural constraints on each relationship type. Note any unspecified requirements, and make appropriate assumptions to make the specification complete.

28. WriteERdiagramforAirlinereservationandBan database

Module2

Chapter1:TheRelationalDataModel

Introduction

The relational data model was first introduced by Ted Codd of IBM Research in 1970 in a classic paper (Codd 1970), and it attracted immediate attention due to its simplicity and mathematical foundation. The model uses the concept of a mathematical relation—which looks somewhat like a table of values—as its basic building block, and has its theoretical basis in set theory and first-order predicate logic.

The first commercial implementations of the relational model became available in the early 1980s, such as the SQL/DS system on the MVS operating system by IBM and the Oracle DBMS. Sincethen, the model has been implemented in a large number of commercial systems. Current popular relational DBMSs (RDBMSs) include DB2 and Informix Dynamic Server (from IBM), Oracle and Rdb (from Oracle), Sybase DBMS (from Sybase) and SQLServer and Access (from Microsoft). In addition, several open source systems, such as MySQL and PostgreSQL, are available.

RelationalModelConcepts

The relational model represents the database as a collection of relations. Informally, each relation resembles a table of values or, to some extent, a flat file of records. Itis called a **flat file** becauseeach record has a simple linear or flat structure.

When a relation is thought of as a **table** of values, each row in the table represents a collection of related data values. A row represents a fact that typically corresponds to a real-world entity or relationship. The table name and column names are used to help to interpret the meaning of the values in each row.

For example, in STUDENT relation because each row represents facts about a particular student entity. The column names—Name, Student_number, Class, and Major—specify how to interpret the datavaluesin each row, based on thecolumn each valueis in. All valuesin acolumn areofthesame data type.

In the formal relational model terminology, a row is called a tuple, a column header is called an attribute, and the table is called a relation. The data type describing the types of values that canappear in each column is represented by a domain of possible values.

Domains,Attributes,Tuples,andRelations

Domain

A **domain** D is a set of atomic values. By **atomic** we mean that each value in the domain is indivisible as far as the formal relational model is concerned. A common method of specifyinga domain isto specifyadatatype fromwhich the datavaluesformingthe domain are drawn. It is also useful to specify a name for the domain, to help in interpreting itsvalues.

Someexamplesofdomains follow:

- Usa_phone_numbers:Thesetoften-digitphonenumbersvalidintheUnitedStates.
- Social_security_numbers:Thesetofvalidnine-digitSocialSecuritynumbers.
- Names:Thesetofcharacterstringsthatrepresentnamesofpersons.
- Employee_ages.Possibleagesofemployeesinacompany;eachmustbeaninteger value between 15 and 80.

The preceding are called logical definitions of domains. A **data type** or **format** is also specified foreachdomain.Forexample,thedatatypeforthedomainUsa_phone_numberscanbedeclaredasachar acterstringoftheform(ddd)ddddddd,whereeachdis anumeric(decimal) digitandthefirstthreedigitsformavalidtelephoneareacode.Thedatatype for Employee_ages is an integer number between 15 and80.

Attribute

AnattributeA_i isthenameofaroleplayedbysomedomainDintherelationschemaR. Dis called the **domain** of A_i and is denoted by **dom**(A_i).

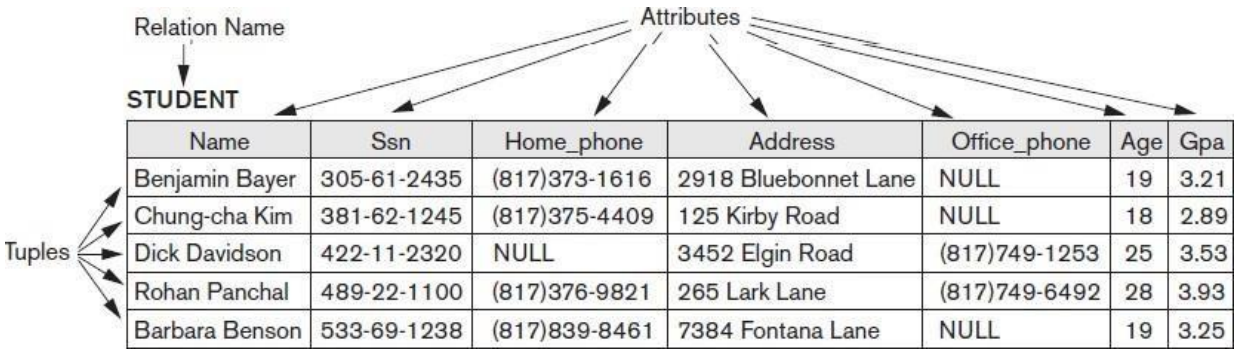
Tuple

Mappingfromattributestovaluesdrawnfromtherespectivedomainsof thoseattributes.Tuples are intended to describe some entity (or relationship between entities) in the miniworld Example: a tuple for a PERSON entity might be

{Name -->“smith", Gender-->Male,Age-->25}

Relation

Anamedset oftuples allof thesameform i.e.,having thesame setof attributes.



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Relation schema

A **relation schema** R, denoted by R(A_1, A_2, ...,A_n), is made up of a relation name Rand alistof attributes A_1, A_2, ...,A_n. Each **attribute** A_i is the name of a role played by some domain D in the relation schema R. D is called the **domain** of A_i and is denoted by **dom**(A_i). A relation schema is used to *describe* a relation; *R* is called the **name** of this relation.

The**degree(orarity)**ofarelationisthenumberofattributesnofitsrelationschema.Arelation of degree seven, which stores information about university students,would contain seven attributes describing each student. as follows:

STUDENT(Name,Ssn,Home_phone,Address, Office_phone,Age,Gpa)

Using the data type of each attribute, the definition is sometimes written as:

STUDENT(Name:string,Ssn:string,Home_phone:string,Address:string,
Office_phone: string, Age: integer, Gpa: real)

Domains for some of the attributes of the STUDENT relation:

dom(Name) = Names; dom(Ssn) =Social_security_numbers;
dom(HomePhone)=USA_phone_numbers,dom(Office_phone)=USA_phone_numbers,

Relation(orrelation state)

A relation (or relation state) r of the relation schema by R(A_1, A_2, ...,A_n), also denoted by r(R),is a set of n-tuples r = {t_1, t_2, ..., t_m}. Each n-tuple t is an ordered list of n values t =<v_1, v_2, ..., v_n>, where each value vi, 1 ≤i<≤n, isan element of dom (A_i) or isa special NULL value. The i^{th} value in tuple t, which corresponds to the attribute A_i, is referred to as t[A_i] or t. A_i .

The terms **relation intension** for the schema R and **relation extension** for a relation state r(R) are also commonly used.

CharacteristicsofRelations

1. **OrderingofTuples in aRelation**

   A relation is defined as a *set* of tuples. Mathematically, elements of a set have *no order*among them; hence, tuples in a relation do not have any particular order. Tuple ordering isnot part of a relation definition because a relation attempts to represent facts at a logical or abstract level. Many tuple orders can be specified on the same relation.

2. **OrderingofValueswithin aTupleandanAlternativeDefinition ofaRelation**

   The order of attributes and their values is *not* that important as long as the correspondence between attributes and values is maintained. An alternative definition of a relation can be given, making the ordering of values in a tuple unnecessary. In this definition A **relation schema**R(A_1,A_2,...,A_n),setofattributesanda **relationstater(R)**isafinitesetofmappings r = {t1, t2, ..., tm}, where each tuple ti is a **mapping** from R to D.

   According to this definition of tuple as a mapping, a **tuple** can be considered as a set of (<attribute>, <value>) pairs, where each pair gives the valueof the mapping from an attribute $A_i$to a value vi from dom(A_i) .The ordering of attributes is not important, because the attribute name appears with its value.

3. **Valuesand NULLsintheTuples**

   Each value in a tuple is atomic. NULL values are used to represent the values of attributesthat may be unknown or may not apply to a tuple. For example some STUDENT tuples have NULL for their office phones because they do not have an office .Another student has a NULL for home phone In general, we can have several meanings for NULL values, such as **value unknown**, **value** exists but is **not available**, or **attribute does not apply** to this tuple (also known as **value undefined**).

4. **Interpretation(Meaning)ofaRelation**

   The relation schema can be interpreted as a declaration or a type of **assertion**. For example, the schema of the STUDENT relation of asserts that, ingeneral, a student entity has a Name, Ssn, Home_phone, Address, Office_phone, Age, and Gpa. Each tuple in the relation can then be interpreted as a particular instance of the assertion.For example, the first tuple asserts the fact that there is a STUDENT whose Name is Benjamin Bayer, Ssn is 305-61-2435, Age is 19, and so on.

   Analternativeinterpretationofarelationschemaisasa **predicate**;inthiscase,thevaluesin each tuple are interpreted as values that *satisfy* the predicate.

RelationalModelNotation

- RelationschemaRof degreen isdenotedbyby $R(A_1,A_2,...,A_n)$
- UppercaselettersQ,R,Sdenoterelationnames
- Lowercaselettersq,r,sdenoterelationstates
- Letterst,u, v denotetuples
- Ingeneral,thenameofarelationschemasuchasSTUDENTalsoindicatesthecurrentsetof tuples in that relation
- AnattributeAcanbequalifiedwiththerelationnameRtowhich itbelongsbyusingthedot notation R.A—for example, STUDENT.Name or STUDENT.Age.
- An *n*-tuple *t* in a relation *r*(*R*) is denoted by $t = <v_1, v_2, ..., v_n>$, where v_i is the value corresponding to attribute A_i. The following notation refers to **component values** oftuples:
- Both$t[A_i]$and$t.A_i$(andsometimes$t[i]$)refertothevaluev_i in$t$forattributeA_i.
- Both $t[A_u, A_w, ..., A_z]$ and $t.(A_u, A_w, ..., A_z)$, where $A_u, A_w, ..., A_z$ is a list of attributes from *R*, refer to the subtuple of values $<v_u, v_w, ..., v_z>$ from *t* corresponding to the attributes specified in the list.

RelationalModelConstraintsandRelationalDatabaseSchemas

Constraints are restrictions on the actual values in a database state. These constraints are derivedfromtherulesintheminiworldthatthedatabaserepresents.Constraintsondatabases can generally be divided into three main categories:

1. Inherentmodel-basedconstraintsorimplicitconstraints
    - Constraintsthatareinherentinthedatamodel.
    - The characteristics of relations are the inherent constraints of the relational model and belong to the first category. For example, the constraint that a relation cannot have duplicate tuples is an inherent constraint.
2. Schema-basedconstraintsorexplicit constraints
    - Constraintsthatcanbedirectlyexpressedinschemasofthedatamodel,typically by specifying them in the DDL.
    - Theschema-basedconstraintsincludedomainconstraints,keyconstraints,constraints on NULLs, entity integrity constraints, and referential integrityconstraints.
3. Application-basedorsemanticconstraintsorbusinessrules
    - Constraintsthat*cannot*bedirectlyexpressedintheschemasof thedatamodel,and hence must be expressed and enforced by the application programs.

- Examples of such constraints are the salary of an employee should not exceed the salary of the employee's supervisor and the maximum number of hours an employee can work on all projects per week is56.

DomainConstraints

DomainConstraintsspecifythatwithineachtuple,thevalueofeachattributeA mustbe an atomic value from the domain dom(A). The data types associated with domains typicallyincludestandardnumericdatatypesforintegers(suchasshortinteger,integer, and long integer) and real numbers (float and doubleprecision float). Characters, Booleans,fixed-lengthstrings,andvariable-lengthstringsarealsoavailable,asaredate, time, timestamp, and money, or other special data types.

KeyConstraintsandConstraintsonNULLValues

All tuples in a relation must also be distinct.This means that no two tuples can have the same combination of values for *all* their attributes.There are other **subsets of attributes** of a relation schema *R* with the property that no two tuples in any relation state *r* of *R* should have the same combination of values for these attributes.

Suppose that we denote one such subset of attributes by SK; then for any two *distinct*tuples *t*1 and *t*2 in a relation state *r* of *R*, we have the constraint that:t_1[SK]≠ t_2[SK].such set of attributes SK is called a **superkey** of the relation schema *R*

superkey

A superkey SK specifies a *uniqueness constraint* that no two distinct tuples in any state *r* of *R* can have the same value for SK. Every relation has at least one default superkey—the set of all its attributes.

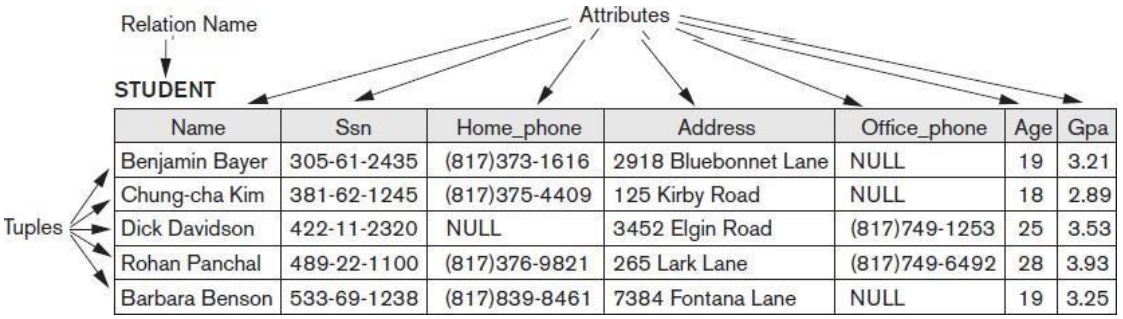
Key

A key *K* of a relation schema *R* is a superkey of *R* with the additional property that removing any attribute *A* from *K* leaves a set of attributes *K*' that is not a superkey of *R* anymore. Hence, a key satisfies two properties:

1. Two distinct tuples in any state of the relation cannot have identical values for (all) the attributes in the key. This first property also applies to a superkey.

2. It is a minimal superkey—that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint in condition 1 hold.This property is not required by a superkey.

Example:ConsidertheSTUDENT relation



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

- Theattributeset{Ssn}isakeyof STUDENTbecausenotwostudenttuplescan have the same value for Ssn
- AnysetofattributesthatincludesSsn—forexample,{Ssn,Name,Age}—isa superkey
- Thesuperkey{Ssn,Name,Age}isnotakeyofSTUDENTbecauseremoving Name or Age or both from the set still leaves us with a superkey

In general, any superkey formed from a single attribute is also a key. A key with multiple attributes must require *all* its attributes together to have the uniqueness property.

Candidate key

A relation schema may have more than one key. In this case, each of the keys is called a **candidate key.** For example, the CAR relation has two candidate keys: License_number and Engine_serial_number

CAR

License_number	Engine_serial_number	Make	Model	Year
Texas ABC-739	A69352	Ford	Mustang	02
Florida TVP-347	B43696	Oldsmobile	Cutlass	05
New York MPO-22	X83554	Oldsmobile	Delta	01
California 432-TFY	C43742	Mercedes	190-D	99
California RSK-629	Y82935	Toyota	Camry	04
Texas RSK-629	U028365	Jaguar	XJS	04

Primarykey

It iscommon to designate one of the candidate keys as the **primary key** of the relation. This is the candidate key whose values are used to *identify* tuples in the relation.We use the convention that the attributes that form the primary key of a relation schema are underlined. Other candidate keys are designated as **unique keys** and are not underlined

Another constraint on attributes specifies whether NULL values are or are not permitted. For example, if every STUDENT tuple must have a valid, non-NULL value for the Name attribute, then Name of STUDENT is constrained to be NOT NULL.

RelationalDatabasesandRelationalDatabaseSchemas

ARelationaldatabaseschema*Sisas*etofrelationschemas$S=\{R_1,R_2,...,R_m\}$andaset of integrity constraints IC.

Exampleofrelationaldatabase schema:

COMPANY={EMPLOYEE,DEPARTMENT,DEPT_LOCATIONS,PROJECT, WORKS_ON, DEPENDENT}

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	gender	Salary	Super_ssn	Dno

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

Dnumber	Dlocation

PROJECT

Pname	Pnumber	Plocation	Dnum

WORKS_ON

Essn	Pno	Hours

DEPENDENT

Essn	Dependent_name	gender	Bdate	Relationship

Figure1.2.3(a): Schema diagramfortheCOMPANYrelationaldatabaseschema.

Theunderlined attributesrepresentprimarykeys

ARelational databasestateisa setofrelationstates$DB=\{r_1, r_2,..., r_m\}$.Each$r_i$isa stateof Randsuchthatther$_i$ relationstatessatisfyintegrityconstraintsspecifiedinIC.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	gender	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	gender	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Figure1.2.3(b): Onepossibledatabasestatefor theCOMPANYrelationaldatabaseschema.

A database state that does not obey all the integrity constraints is called **Invalid state** and a state that satisfies all the constraints in the defined set of integrity constraints IC is called a **Valid state**

Attributes that represent the same real-world concept may or may not have identical names in different relations. For example, the Dnumber attribute in both DEPARTMENT and DEPT_LOCATIONS stands for the same real-world concept—the number given to a department. That same concept is called Dno in EMPLOYEE and Dnum in PROJECT.

Alternatively, attributes that represent different concepts may have the same name in different relations. For example, we could have used the attribute name Name for both Pname of PROJECT and Dname of DEPARTMENT; in this case, we would have two attributes that share the same name but represent different realworld concepts—project names and department names.

Integrity,ReferentialIntegrity,andForeignKeys Entity

integrity constraint

The entity integrity constraint states that no primary key value can be NULL. This is because the primary key value is used to identify individual tuples in a relation. Having NULL values for the primary key implies that we cannot identify some tuples. For example, if two or more tuples had NULL for their primary keys, we may not be able to distinguish them if we try to reference them from other relations.

Keyconstraintsandentityintegrityconstraintsarespecifiedonindividualrelations.

Referentialintegrityconstraint

The referential integrity constraint is specified between two relations and is used to maintain the consistency among tuples in the two relations. Informally, the referential integrity constraint states that a tuple in one relation that refers to another relation must refer to an existing tuple in thatrelation. For example COMPANY database, the attribute Dno of EMPLOYEE gives the department number for which each employee works; hence, its value in every EMPLOYEE tuple must match the Dnumber value of some tuple in the DEPARTMENT relation.

To define referential integrity more formally, first we define the concept of a *foreign key*. The conditions for a foreign key, given below, specify a referential integrity constraint between the two relation schemas R_1 and R_2.

A set of attributes FK in relation schema R_1 is a **foreign key** of R_1 that **references** relation R_2 if it satisfies the following rules:

1. AttributesinFKhavethesamedomain(s)astheprimarykeyattributesPK of R_2;the attributes FK are said to **reference** or **refer to** the relation R_2.
2. AvalueofFK inatuple t_1 ofthecurrent state $r_1(R_1)$eitheroccursas avalue ofPKfor some tuple t_2 in the current state $r_2(R_2)$ *or is NULL*.

In the former case, we have $t_1[FK] = t_2[PK]$, and we say that the tuple t_1 **references** or **refers to** the tuple t_2.

In this definition, $R_1$is called the **referencing relation** and $R_2$is the **referenced relation**. If these two conditions hold, a **referential integrity constraint** from R_1 to R_2 issaid tohold.

OtherTypesofConstraints

Semantic integrity constraints

Semantic integrity constraints can be specified and enforced within the application programs that update the database, or by using a general-purpose constraint specification language**.** Examples of such constraints are the salary of an employee should not exceed the salary of the employee's supervisorandthemaximumnumberofhoursan employeecanworkonallprojectsperweekis56.

Mechanisms called **triggers** and **assertions** can be used. In SQL, CREATE ASSERTION and CREATE TRIGGER statements can be used for this purpose.

Functionaldependencyconstraint

Functionaldependencyconstraintestablishesafunctionalrelationshipamongtwosetsofattributes X and Y. This constraint specifies that the value of X determines a unique value of Y in all states of a relation; itisdenoted as a functional dependency $X \rightarrow Y$. We use functional dependencies and other types of dependencies as tools to analyze the quality of relational designs and to "normalize"relations to improve their quality.

Stateconstraints(static constraints)

Definethe constraints thatavalid stateofthe databasemust satisfy

Transitionconstraints(dynamic constraints)

Definetodealwith state changesinthedatabase

UpdateOperations,Transactions,andDealingwithConstraintViolations

Theoperationsofthe relationalmodelcan becategorizedinto **retrievals**and**updates**

Therearethreebasic operationsthatcan changethestates ofrelationsinthedatabase:

1. Insert-usedtoinsertoneormorenewtuplesinarelation
2. Delete-usedtodeletetuples
3. Update (or Modify)- used to change the values of some attributes in existing tuplesWhenevertheseoperationsareapplied,theintegrityconstraintsspecifiedontherelationaldatabase schema should not be violated.

TheInsertOperation

TheInsertoperationprovidesalistofattributevaluesforanewtuple*t*that istobeinsertedintoa elation *R*. Insert can violate any of the four types of constraints

1. **Domainconstraints:**ifanattributevalueisgiventhatdoesnotappearinthecorresponding domain or is not of the appropriate data type
2. **Keyconstraints:**ifakeyvalueinthenewtuple*t*alreadyexistsinanothertupleinthe relation *r*(*R*)
3. **Entityintegrity:**ifanypartoftheprimarykeyof thenewtuple *t*isNULL
4. **Referentialintegrity:**ifthevalueofanyforeignkeyin*t*refersto atuplethatdoesnotexist in the referenced relation

Examples:

1. Operation:

   Insert <‘Cecilia’, ‘F’, ‘Kolonsky’, NULL, ‘1960-04-05’, ‘6357 Windy Lane,Katy,TX’, F, 28000, NULL, 4>

   Result:Thisinsertionviolatestheentityintegrityconstraint(NULLfortheprimarykey Ssn), so it is rejected

2. Operation:

   Insert<‘Alicia’,‘J’,‘Zelaya’,‘999887777’,‘1960-04-05’,‘6357WindyLane,Katy,TX’, F, 28000, ‘987654321’, 4>

   Result:ThisinsertionviolatesthekeyconstraintbecauseanothertuplewiththesameSsn value already exists in the EMPLOYEE relation, and so it is rejected.

3.Operation:

   Insert<‘Cecilia’,‘F’,‘Kolonsky’,‘677678989’,‘1960-04-05’,‘6357 Windswept, Katy, TX’,F,28000,‘987654321’,7>

Result: This insertion violates the referential integrity constraint specified on Dno in EMPLOYEEbecausenocorrespondingreferencedtupleexistsinDEPARTMENT with Dnumber = 7.

4. Operation:

Insert<‘Cecilia’,‘F’,‘Kolonsky’,‘677678989’, ‘1960-04-05’,‘6357 Windy Lane,Katy, TX’,F,28000,NULL,4>

Result:Thisinsertionsatisfiesallconstraints,soitisacceptable.

If an insertion violates one or more constraints, the default optionis to reject the insertion.It wouldbe useful if the DBMS could provide a reason to the user as to why the insertion was rejected. Another option is to an attempt to correct the reason for rejecting the insertion

TheDeleteOperation

The Delete operation can violate only referential integrity. This occurs if the tuple being deleted is referenced by foreign keys from other tuples in the database. To specify deletion, a condition on the attributes of the relation selects the tuple (or tuples) to be deleted.

Examples:

1. Operation:

   DeletetheWORKS_ONtuplewithEssn=‘999887777’andPno=10. Result:

   This deletion is acceptable and deletes exactly one tuple.

2. Operation:

   DeletetheEMPLOYEEtuplewithSsn= ‘999887777’.

   Result:Thisdeletionisnotacceptable,becausetherearetuplesinWORKS_ONthatrefer to this tuple. Hence, if the tuple in EMPLOYEE is deleted, referential integrity violations will result.

3. Operation:

   DeletetheEMPLOYEEtuplewithSsn= ‘333445555’

   Result:Thisdeletionwillresultinevenworsereferentialintegrity violations,becausethe tuple involved is referenced by tuples from the EMPLOYEE, DEPARTMENT, WORKS_ON, and DEPENDENT relations.

Severaloptionsare availableifadeletionoperationcausesaviolation

1. restrict-istorejectthedeletion
2. cascade,isto attempttocascade(orpropagate)thedeletionbydeletingtuplesthatreference the tuple that is being deleted

3. Setnullorsetdefault-istomodifythereferencingattributevaluesthatcausetheviolation; each such value is either set to NULL or changed to reference another default valid tuple.

TheUpdateOperation

The Update (or Modify)operation isusedto change the values of one or moreattributes ina tuple (ortuples) ofsome relation *R*. It isnecessarytospecifyaconditiononthe attributesofthe relation to select the tuple (or tuples) to be modified.

Examples:

1. Operation:
   UpdatethesalaryoftheEMPLOYEEtuplewithSsn='999887777'to28000. Result: Acceptable.
2. Operation:
   UpdatetheDnooftheEMPLOYEEtuplewithSsn='999887777'to7. Result: Unacceptable, because it violates referential integrity.
3. Operation:
   Update the Ssn of the EMPLOYEE tuple with Ssn = '999887777' to '987654321'.
   Result:Unacceptable,becauseitviolatesprimarykeyconstraintbyrepeatingavalue that already exists as a primary key in another tuple; it violates referential integrity constraints because there are other relations that refer to the existing value of Ssn

Updating an attribute that is neither part of a primary key nor of a foreign key usually causes no problems; the DBMS need only check to confirm that the new value is of the correct data type and domain.

TheTransactionConcept

A **transaction** isan executing program that includes some database operations, such as reading from the database, or applying insertions, deletions, or updates to the database. At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema A single transaction may involve any number of retrievaloperations and any number of update operations. These retrievals and updates will together form an atomic unit of work against the database.For example, a transaction to apply a bank withdrawal will typically read the user account record, check if there is a sufficient balance, and then update the record by the withdrawal amount.

Chapter2:RelationalAlgebra

Introduction

Relational algebra is the basic set of operations for the relational model. These operations enable a user to specify basic retrieval requests as relational algebra expressions. The result of an operation is a new relation, which may have been formed from one or more input relations.

Therelational algebraisveryimportant forseveral reasons

- First,itprovidesaformalfoundationforrelationalmodeloperations.
- Second, and perhaps more important, it is used as a basis for implementing and optimizing queriesin the query processing and optimization modulesthat are integral parts of relational database management systems (RDBMSs)
- Third, some of its concepts are incorporated into the SQL standard query language for RDBMSs

UnaryRelationalOperations:SELECTandPROJECT

TheSELECTOperation

The SELECT operation denoted by σ (sigma) is used to select a subset of the tuples from a relation based on a selection condition. The selection condition acts as a filter that keeps only those tuplesthat satisfy a qualifying condition. Alternatively, we can consider the SELECT operation to *restrict* the tuples in a relation to only those tuples that satisfy thecondition.

The SELECT operation can also be visualized as a *horizontal partition* of the relation into two setsof tuples—those tuples that satisfy the condition and are selected, and those tuples that donot satisfy the condition and are discarded.

Ingeneral,theselectoperationisdenotedby

$$\mathbf{o}_{\text{<selection condition>}}\mathbf{(R)}$$

where,

- thesymbolσisusedtodenotethe selectoperator
- theselectionconditionisaBoolean(conditional)expressionspecifiedontheattributesof relation R
- tuplesthat makethe conditiontrueareselected
  - ·appearintheresultoftheoperation
- tuplesthatmaketheconditionfalsearefilteredout
  - ·discarded fromtheresultofthe operation

TheBooleanexpressionspecifiedin<selectioncondition>ismadeupofanumberofclausesofthe form:

<attributename><comparisonop><constant value>

or

<attributename><comparison op><attributename>

where

<attributename>is thename ofanattributeof *R*,

<comparisonop>isone of theoperators{=,<, ≤,>,≥,≠}, and

<constantvalue>is aconstantvalue fromtheattribute domain

Clausescanbeconnectedbythestandard Booleanoperators*and*,*or*,and *not*toformageneral selection condition

Examples:

1. SelecttheEMPLOYEEtupleswhosedepartmentnumberis4.

   σ DNO=4(EMPLOYEE)

2. Selecttheemployeetupleswhosesalary isgreater than$30,000.

   σ SALARY>30,000(EMPLOYEE)

3. Selectthetuplesforallemployeeswhoeitherworkindepartment4andmakeover$25,000 per year, or work in department 5 and make over$30,000

   σ(Dno=4ANDSalary>25000)OR(Dno=5ANDSalary>30000)(EMPLOYEE)

TheresultofaSELECT operationcan bedeterminedas follows:

- The<selectioncondition> isappliedindependently toeachindividualtupletinR
- IftheconditionevaluatestoTRUE,thentuple*t*isselected.Alltheselectedtuplesappearin the result of the SELECT operation
- TheBoolean conditionsAND, OR,andNOT havetheirnormalinterpretation, asfollows:

  - (cond1ANDcond2)isTRUEifboth(cond1)and(cond2)areTRUE;otherwise,itis FALSE.

  - (cond1ORcond2)isTRUEifeither(cond1)or(cond2)orbothareTRUE;otherwise,itis FALSE.

  - (NOTcond)isTRUEifcondisFALSE;otherwise,itisFALSE.

The SELECT operator is unary; that is, it is applied to a single relation. The degree of the relation resulting from a SELECT operation is the same as the degree of R.The number of tuples in the resulting relation is always less than or equal to the number of tuples in R. That is,

$|\sigma_c(R)| \leq |R|$ foranycondition C

The fraction of tuples selected by a selection condition is referred to as the selectivity of the condition.

TheSELECT operationis commutative;that is,

$$\sigma_{<cond1>}(\sigma_{<cond2>}(R)) = \sigma_{<cond2>}(\sigma_{<cond1>}(R))$$

Hence, a sequence of SELECTs can be applied in any order.we can always combine a cascade (or sequence) of SELECT operations into a single SELECT operation with a conjunctive (AND) condition; that is,

$$\sigma_{<cond1>}(\sigma_{<cond2>}(\ldots(\sigma_{<condn>}(R))\ \ldots)) = \sigma_{<cond1>}\text{AND}_{<cond2>}\text{AND}\ \ldots\text{AND}_{<condn>}(R)$$

In SQL, the SELECT condition is specified in the WHERE clause of a query.For example, the following operation:

$$\sigma_{Dno=4}\text{AND}_{Salary>25000}(\text{EMPLOYEE})$$

would to thefollowing SQLquery:

```
SELECT*FROMEMPLOYEEWHEREDno=4ANDSalary>25000;
```

ThePROJECT Operation

The PROJECT operation denoted by π (pi) selects certain columns from the table and discards the othercolumns.Usedwhenweareinterestedinonlycertainattributesofarelation.Theresultofthe PROJECT operation can be visualized as a vertical partition of the relation into two relations:

- onehasthe neededcolumns(attributes) andcontainstheresultoftheoperation
- theothercontainsthediscardedcolumns

The general form of the PROJECT operation is

$$\pi_{<attribute\ list>}(R)$$

where

π(pi)-symbolused to representthePROJECT operation,

<attributelist> -desiredsublistofattributesfromtheattributesofrelation R.

TheresultofthePROJECToperationhasonlytheattributesspecifiedin<attributelist> inthesame order as they appear in the list. Hence, its degree is equal to the number of attributes in <attribute list>

Example:

1. Tolisteachemployee'sfirstandlastnameand salarywecanusethePROJECToperationas follows:

$$\pi_{Lname,Fname,Salary}(EMPLOYEE)$$

Iftheattributelistincludesonlynonkeyattributesof R,duplicatetuplesarelikelyto occur. The result of the PROJECT operation is a set of distinct tuples, and hence a valid relation. This is known as **duplicate elimination**.For example, consider the following PROJECT operation:

$$\pi_{gender,Salary}(EMPLOYEE)$$

Lname	Fname	Salary
Smith	John	30000
Wong	Franklin	40000
Zelaya	Alicia	25000
Wallace	Jennifer	43000
Narayan	Ramesh	38000
English	Joyce	25000
Jabbar	Ahmad	25000
Borg	James	55000

Thetuple<'F',25000>appearsonlyonceinresultingrelationeventhough thiscombinationof values appears twice in the EMPLOYEE relation.

Thenumberoftuplesin arelationresultingfromaPROJECToperationisalwayslessthanorequal to the number of tuples in *R*. Commutativity does not hold on PROJECT

$$\pi_{<list1>}(\pi_{<list2>}(R)) = \pi_{<list1>}(R)$$

aslongas<list2>containstheattributesin<list1>;otherwise,theleft-handsideisanincorrect expression. InSQL,thePROJECTattributelistisspecifiedintheSELECTclauseofaquery.Forexample,the following operation:

$$\pi_{gender,Salary}(EMPLOYEE)$$

wouldcorrespondto the followingSQLquery:

SELECTDISTINCTgender,Salary**FROM**EMPLOYEE

SequencesofOperationsandtheRENAMEOperation

For most queries, we need to apply several relational algebra operations one after the other. Either we can write the operations as a single relational algebra expression by nesting the operations, or we can apply one operation at a time and create intermediate result relations. In the latter case, we must give names to the relations that hold the intermediate results.

For example, to retrieve the first name, last name, and salary of all employees who work in department number 5, we must apply a SELECT and a PROJECT operation. We can write a single relational algebra expression, also known as an **in-line expression**, as follows:

$$\pi_{Fname,Lname,Salary}(\sigma_{Dno=5}(EMPLOYEE))$$

Alternatively, we can explicitly showthe sequence of operations, giving aname to each intermediate relation, as follows:

$$DEP5\_EMPS \leftarrow \sigma_{Dno=5}(EMPLOYEE)$$

$$RESULT \leftarrow \pi_{Fname,Lname,Salary}(DEP5\_EMPS)$$

We can also use this technique to **rename** the attributes in the intermediate and result relations. To rename the attributes in a relation, we simply list the new attribute names in parentheses

$$TEMP \leftarrow \sigma_{Dno=5}(EMPLOYEE)$$

$$R(First\_name,Last\_name,Salary) \leftarrow \pi_{Fname,Lname,\ Salary}(TEMP)$$

TEMP

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston,TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston,TX	M	40000	888665555	5
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble,TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

R

First_name	Last_name	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

Ifno renaming is applied, the names of the attributes in the resulting relation of a SELECT operation arethesameasthosein theoriginalrelationandinthesame order.ForaPROJECToperationwith no renaming, the resulting relation has the same attribute names as those in the projection list and in the same order in which they appear in the list.

We can also define a formal RENAME operation—which can rename either the relation name or the attribute names, or both—as a unary operator.

ThegeneralRENAMEoperationwhen appliedtoarelation *R*ofdegree*n*isdenotedbyanyofthe following three forms:

1. $\rho_{S(B1,B2,...,Bn)}(R)$ ρ(rho)–RENAMEoperator
2. $\rho S(R)$ S–newrelationname
3. $\rho_{(B1,B2,.....Bn)}(R)$ $B_1,B_2,.....B_n$-newattributenames

The first expression renames both the relation and its attributes. Second renames the relation only andthethirdrenamestheattributesonly.IftheattributesofRare($A_1,A_2,...,A_n$)inthatorder,then each A_i is renamed as B_i.

RenaminginSQLisaccomplishedbyaliasingusingAS,asinthefollowingexample: SELECT

E.Fname AS First_name,

E.Lname AS Last_name,

E.Salary ASSalary

FROMEMPLOYEEASE

WHERE E.Dno=5,

RelationalAlgebraOperationsfromSetTheory

TheUNION,INTERSECTION,and MINUSOperations

- **UNION:**Theresultofthisoperation,denotedby *R*∪*S*,isarelationthatincludesalltuples that are either in *R* or in *S* or in both *R* and *S*. Duplicate tuples areeliminated.
- **INTERSECTION:**Theresultofthisoperation,denotedby*R*∩*S*,isarelationthatincludes all tuples that are in both *R* and *S*.
- **SETDIFFERENCE(orMINUS):**Theresultofthisoperation,denotedby *R*–*S*,isa relation that includes all tuples that are in *R* but not in *S*.

Example: Considerthethefollowing tworelations: STUDENT&INSTRUCTOR

STUDENT

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR

Fname	Lname
John	Smith
Ricardo	Browne
Susan	Yao
Francis	Johnson
Ramesh	Shah

STUDENT∪INSTRUCTOR

Fn	Ln
Susan	Yao
Ramesh	Shah
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert
John	Smith
Ricardo	Browne
Francis	Johnson

STUDENT∩INSTRUCTOR

Fn	Ln
Susan	Yao
Ramesh	Shah

STUDENT– INSTRUCTOR

Fn	Ln
Johnny	Kohler
Barbara	Jones
Amy	Ford
Jimmy	Wang
Ernest	Gilbert

INSTRUCTOR− STUDENT

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Example: To retrieve the Social Security numbers of all employees who either work in department 5 or directly supervise an employee who works in department 5

DEP5_EMPS ← $\sigma_{Dno=5}$(EMPLOYEE)

RESULT1 ← π_{Ssn}(DEP5_EMPS)

RESULT2(Ssn) ← π_{Super_ssn}(DEP5_EMPS)

RESULT ← RESULT1 ∪ RESULT2

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	gender	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

Singlerelationalalgebra expression:

Result$\leftarrow \pi_{Ssn}(\sigma_{Dno=5}$ **(EMPLOYEE))**$\cup \pi_{Super\_ssn}(\sigma_{Dno=5}$**(EMPLOYEE))**

UNION, INTERSECTION and SET DIFFERENCE are binary operations; that is, each is applied to two sets (of tuples). When these operations are adapted to relational databases, the two relations on which any of these three operations are applied must have the same type of tuples; this condition has been called **union compatibility or type compatibility.**

Two relations R(A_1, A_2, ..., A_n) and S(B_1, B_2, ..., B_n) are said to be union compatible (or type compatible) if they have the same degree n and if dom(A_i) = dom(B_i) for $1 \leq i \leq n$.This means thatthe two relations have the same number of attributes and each corresponding pair of attributes hasthe same domain.

BothUNIONand INTERSECTIONare*commutativeoperations*;thatis,

$R \cup S = S \cup R$ and $R \cap S = S \cap R$

BothUNIONandINTERSECTIONcanbetreatedas*n*-aryoperationsapplicabletoanynumberof relations because both are also *associative operations;* thatis,

$R \cup (S \cup T) = (R \cup S) \cup Tand(R \cap S) \cap T = R \cap (S \cap T)$ The MINUS

operation is *not commutative;* that is, in general,

$R - S \neq S - R$

INTERSECTIONcanbeexpressedintermsofunionandsetdifferenceasfollows: R ∩ S

= ((R ∪ S) − (R − S)) − (S − R)

InSQL,therearethreeoperations—UNION,INTERSECT,andEXCEPT—thatcorrespondtothe set operations

TheCARTESIANPRODUCT(CROSSPRODUCT)Operation

The CARTESIAN PRODUCT operation—also known as CROSS PRODUCT or CROSS JOIN denoted by × is a binary set operation, but the relations on whichit is applied do *not* have tobe union compatible. This set operation produces a new element by combining every member (tuple) from one relation (set) with every member (tuple) from the other relation (set)

In general, the result of $R(A_1, A_2, ..., A_n) \times S(B_1, B_2, ..., B_m)$ is a relation Q with degree $n + m$attributes $Q(A_1, A_2, ..., A_n, B_1, B_2, ..., B_m)$, in that order. The resulting relation Q has one tuple foreach combination of tuples—one from R and one from S. Hence, if R has n_R tuples (denoted as $|R| = n_R$), and S has n_S tuples, then $R \times S$ will have $n_R * n_S$ tuples

Example:supposethatwewanttoretrieve alistofnamesofeachfemaleemployee'sdependents.

$$\text{FEMALE\_EMPS} \leftarrow \sigma_{\text{gender='F'}}(\text{EMPLOYEE})$$

$$\text{EMPNAMES} \leftarrow \pi_{\text{Fname, Lname, Ssn}}(\text{FEMALE\_EMPS})$$

$$\text{EMP\_DEPENDENTS} \leftarrow \text{EMPNAMES} \times \text{DEPENDENT}$$

$$\text{ACTUAL\_DEPENDENTS} \leftarrow \sigma_{\text{Ssn=Essn}}(\text{EMP\_DEPENDENTS})$$

$$\text{RESULT} \leftarrow \pi_{\text{Fname,Lname,Dependent\_name}}(\text{ACTUAL\_DEPENDENTS})$$

FEMALE_EMPS

Fname	Minit	Lname	Ssn	Bdate	Address	gen	Salary	Super_ssn	Dno
Alicia	J	Zelaya	999887777	1968-07-19	3321Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291Berry, Bellaire, TX	F	43000	888665555	4
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5

EMP_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Alicia	Zelaya	999887777	333445555	Alice	F	1986-04-05	...
Alicia	Zelaya	999887777	333445555	Theodore	M	1983-10-25	...
Alicia	Zelaya	999887777	333445555	Joy	F	1958-05-03	...
Alicia	Zelaya	999887777	987654321	Abner	M	1942-02-28	...
Alicia	Zelaya	999887777	123456789	Michael	M	1988-01-04	...
Alicia	Zelaya	999887777	123456789	Alice	F	1988-12-30	...
Alicia	Zelaya	999887777	123456789	Elizabeth	F	1967-05-05	...
Jennifer	Wallace	987654321	333445555	Alice	F	1986-04-05	...
Jennifer	Wallace	987654321	333445555	Theodore	M	1983-10-25	...
Jennifer	Wallace	987654321	333445555	Joy	F	1958-05-03	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...
Jennifer	Wallace	987654321	123456789	Michael	M	1988-01-04	...
Jennifer	Wallace	987654321	123456789	Alice	F	1988-12-30	...
Jennifer	Wallace	987654321	123456789	Elizabeth	F	1967-05-05	...
Joyce	English	453453453	333445555	Alice	F	1986-04-05	...
Joyce	English	453453453	333445555	Theodore	M	1983-10-25	...
Joyce	English	453453453	333445555	Joy	F	1958-05-03	...
Joyce	English	453453453	987654321	Abner	M	1942-02-28	...
Joyce	English	453453453	123456789	Michael	M	1988-01-04	...
Joyce	English	453453453	123456789	Alice	F	1988-12-30	...
Joyce	English	453453453	123456789	Elizabeth	F	1967-05-05	...

EMPNAMES

Fname	Lname	Ssn
Alicia	Zelaya	999887777
Jennifer	Wallace	987654321
Joyce	English	453453453

RESULT

Fname	Lname	Dependent_name
Jennifer	Wallace	Abner

ACTUAL_DEPENDENTS

Fname	Lname	Ssn	Essn	Dependent_name	Sex	Bdate	...
Jennifer	Wallace	987654321	987654321	Abner	M	1942-02-28	...

TheCARTESIANPRODUCTcreatestupleswiththecombinedattributesoftwo relations.We can SELECT related tuples only from the two relations by specifying an appropriate selection condition after the Cartesian product.

InSQL,CARTESIANPRODUCTcanberealizedbyusingtheCROSSJOINoptioninjoined tables

BinaryRelationalOperations:JOINandDIVISION

TheJOIN Operation

TheJOIN operation, denoted by ⋈ isused to combine related tuples from two relations into single"longer"tuples.It allowsus to process relationships among relations.Thegeneral form of a JOIN operation on two relations $R(A_1, A_2, ..., A_n)$ and$S(B_1, B_2, ..., B_m)$ is

$$R \bowtie_{<joinconditio>} S$$

Example: Retrievethe nameofthemanager ofeach department.

Togetthemanager'sname, weneedto combineeachdepartmenttuplewiththeemployeetuple whose Ssn value matches the Mgr_ssn value in the department tuple

$$\text{DEPT\_MGR} \leftarrow \text{DEPARTMENT} \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE}$$
$$\text{RESULT} \leftarrow \pi_{\text{Dname, Lname, Fname}}(\text{DEPT\_MGR})$$

DEPT_MGR

Dname	Dnumber	Mgr_ssn	...	Fname	Minit	Lname	Ssn	...
Research	5	333445555	...	Franklin	T	Wong	333445555	...
Administration	4	987654321	...	Jennifer	S	Wallace	987654321	...
Headquarters	1	888665555	...	James	E	Borg	888665555	...

The result of the JOIN is a relation Q with n + m attributes Q(A_1, A_2, ..., A_n,B_1, B_2, ..., B_m in that order.Qhas onetuplefor each combination oftuples—onefrom R and one from S—wheneverthe combination satisfies the join condition. This is the main difference between CARTESIAN PRODUCT and JOIN. In JOIN, only combinations of tuples satisfying the join condition appearin the result, whereas in the CARTESIAN PRODUCT all combinations of tuples are included in the result. The join condition is specified on attributes from the two relations R and S and is evaluated for each combination of tuples.

EachtuplecombinationforwhichthejoinconditionevaluatestoTRUE isincludedintheresulting relation Q as a single combined tuple. A general join condition is of the form

<condition>**AND**<condition>**AND**...**AND**<condition>

where each <condition> is of the form $A_i\theta\ B_j$, A_i is an attribute of R, B isan attribute of S, Ai andB_j havethesamedomain,andθ(theta)isoneofthecomparisonoperators{=, <,≤,>,≥,≠}.A JOIN operation with such a general join condition is called a **THETA JOIN.** Tuples whose join attributes are NULL or for which the join condition is FALSE do not appear in the result.

VariationsofJOIN:TheEQUIJOINand NATURALJOIN

The most common use of JOIN involves join conditions with equality comparisons only. Such a JOIN, where the only comparison operator used is =, is called an **EQUIJOIN.**In the result of an EQUIJOIN we always have one or more pairs of attributes that have identical values in every tuple.

For example the values of the attributes Mgr_ssn and Ssn are identical in every tuple of DEPT_MGR (the EQUIJOIN result) because the equality join condition specified on these two attributes requires the values to be identical in every tuple in the result.

The standard definition of **NATURAL JOIN** requires that the two join attributes (or each pair ofjoin attributes) have the same name in both relations. If this is not the case, a renaming operation is applied first. Suppose we want to combine each PROJECT tuple with the DEPARTMENT tuple that controls the project.first we rename the Dnumber attribute of DEPARTMENT to Dnum—so that it has the same name as the Dnum attribute in PROJECT—and then we apply NATURAL JOIN:

$$\textbf{PROJ\_DEPT} \leftarrow \textbf{PROJECT} * \rho_{(\textbf{Dname,Dnum,Mgr\_ssn,Mgr\_start\_date})}(\textbf{DEPARTMENT})$$

Thesame querycan bedoneintwo stepsby creatingan intermediatetableDEPTas follows:

$$\textbf{DEPT} \leftarrow \rho_{(\textbf{Dname,Dnum,Mgr\_ssn,Mgr\_start\_date})}(\textbf{DEPARTMENT})$$

PROJ_DEPT← PROJECT* DEPT

TheattributeDnum iscalledthe**joinattribute**fortheNATURALJOINoperation,because itisthe only attribute with the same name in both relations.

PROJ_DEPT

Pname	Pnumber	Plocation	Dnum	Dname	Mgr_ssn	Mgr_start_date
ProductX	1	Bellaire	5	Research	333445555	1988-05-22
ProductY	2	Sugarland	5	Research	333445555	1988-05-22
ProductZ	3	Houston	5	Research	333445555	1988-05-22
Computerization	10	Stafford	4	Administration	987654321	1995-01-01
Reorganization	20	Houston	1	Headquarters	888665555	1981-06-19
Newbenefits	30	Stafford	4	Administration	987654321	1995-01-01

If the attributes on which the natural join is specified already have the same names in both relations, renaming is unnecessary. For example, to apply a natural join on the Dnumber attributes of DEPARTMENT and DEPT_LOCATIONS, it is sufficient to write

DEPT_LOCS←DEPARTMENT* DEPT_LOCATIONS

DEPT_LOCS

Dname	Dnumber	Mgr_ssn	Mgr_start_date	Location
Headquarters	1	888665555	1981-06-19	Houston
Administration	4	987654321	1995-01-01	Stafford
Research	5	333445555	1988-05-22	Bellaire
Research	5	333445555	1988-05-22	Sugarland
Research	5	333445555	1988-05-22	Houston

In general, the join condition for NATURAL JOIN is constructed by equating each pair of join attributes that have the same name in the two relations and combining these conditions with**AND**.Ifnocombinationoftuples satisfies thejoincondition, theresultofa JOIN isan empty relation with zero tuples.

Amoregeneral,butnonstandarddefinitionforNATURALJOIN is

$$Q \leftarrow R *_{(\text{<list1>}),(\text{<list2>})} S$$

where,

<list1>:listof*i*attributesfrom *R*,

<list2>:listof*i*attributesfrom *S*

The lists are used to form equality comparison conditions between pairs of corresponding attributes and thentheconditions arethenANDed together. Only thelist corresponding to attributes ofthefirst relation *R*—<list1>— is kept in the result *Q*.

In general, ifR has $n_R$tuplesand S has $n_S$tuples, theresult of aJOINoperation R⋈ <join condition>S will have between zero and n_R * n_S tuples. The expected size of the join result divided by the maximum size n_R * $n_S$leads to a ratio called join selectivity, which is a property of each join condition. If there is no join condition, all combinations of tuples qualify and the JOIN degenerates into a CARTESIAN PRODUCT, also called CROSS PRODUCT or CROSS JOIN.

A single JOIN operation is used to combine data from two relations so that related information can be presented in a single table. These operations are also known as **inner joins**. Informally, an inner join is a type of match and combine operation defined formally as a combination of CARTESIAN PRODUCT and SELECTION.The NATURAL JOIN or EQUIJOIN operation can also be specified amongmultipletables,leading to ann-wayjoin.Forexample,considerthe followingthree-wayjoin:

$$((\text{PROJECT} \bowtie_{\text{Dnum=Dnumber}} \text{DEPARTMENT}) \bowtie_{\text{Mgr\_ssn=Ssn}} \text{EMPLOYEE})$$

This combines each project tuple with its controlling department tuple into a single tuple, and then combines that tuple with an employee tuple that is the department manager. The net result is a consolidated relation in which each tuple contains this project-department-manager combined information.

InSQL,JOINcanberealizedinseveraldifferentways

-Thefirstmethodistospecifythe<joinconditions>intheWHEREclause,alongwithany other selection conditions.

- Thesecondwayistouseanestedrelation
- Anotherwayistousetheconceptofjoinedtables

ACompleteSetofRelationalAlgebraOperations

Thesetofrelationalalgebraoperations{σ,π,∪,ρ,–,×}isacompleteset; thatis,anyoftheother original relational algebra operations can be expressed as a sequence of operations from this set. For example, the INTERSECTION operation can be expressed by using UNION and MINUS as follows:

$$R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$$

Asanotherexample, aJOINoperationcanbespecifiedasaCARTESIANPRODUCTfollowedbya SELECT operation,

$$R \bowtie_{\text{<condition>}} S \equiv \sigma_{\text{<condition>}}(R \times S)$$

Similarly, a NATURAL JOIN can be specified as a CARTESIAN PRODUCT preceded by RENAME and followed by SELECT and PROJECT operations. Hence, the various JOIN operations are also not strictly necessary for the expressive power of the relational algebra.

TheDIVISIONOperation

The DIVISION operation, denoted by ÷, is useful for a special kind of query that sometimes occursin database applications. An example is Retrieve the names of employees who work on all the projects that 'John Smith' works on. To express this query using the DIVISION operation, proceedas follows.

- First,retrievethelistofprojectnumbersthat'JohnSmith'worksonintheintermediate relation SMITH_PNOS:

$$SSN\_PNOS \leftarrow \pi_{Essn,\ Pno}(WORKS\_ON)$$

- Next,createarelationthatincludesatuple<Pno, Essn>whenevertheemployeewhoseSsnis Essn works on the project whose number is Pno in the intermediate relation SSN_PNOS:

$$SMITH \leftarrow \sigma_{Fname='John'\ \textbf{AND}\ Lname='Smith'}(EMPLOYEE)$$
$$SMITH\_PNOS \leftarrow \pi_{Pno}(WORKS\_ON \bowtie_{Essn=Ssn} SMITH)$$

- Finally,applytheDIVISIONoperationtothetworelations,whichgivesthedesired employees' Social Security numbers:

$$SSNS(Ssn) \leftarrow SSN\_PNOS \div SMITH\_PNOS$$
$$RESULT \leftarrow \pi_{Fname,\ Lname}(SSNS * EMPLOYEE)$$

(a)

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SSNS

Ssn
123456789
453453453

SMITH_PNOS

Pno
1
2

In general, the DIVISION operation is applied to two relations R(Z) ÷ S(X), where the attributes ofR are a subset of the attributes of S; that is, X ⊆ Z.Let Y be the set of attributes of R that are not attributes of S; that is,Y = Z – X (and hence Z = X ∪ Y). The result of DIVISION is a relation T(Y) that includes a tuple t iftuples t_R appear inR with $t_R[Y] = t$, and with $t_R[X] = t_S$ for every tuple $t_S$in S.This means that,foratupletto appear in theresult T oft

Figurebelowillustrates aDIVISIONoperation where $X=\{A\}$,$Y=\{B\}$,and$Z=\{A, B\}$.

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

The tuples (values) *b*1 and *b*4 appear in *R* in combination with all three tuples in *S*; that is why they appear in the resulting relation *T*. All other values of *B* in *R* do not appear with all the tuples in *S* and are not selected: *b*2 does not appear with *a*2, and *b*3 does not appear with *a*1.

TheDIVISIONoperationcanbe expressedasasequenceofπ,×,and–operationsas follows:

$$T1 \leftarrow \pi_Y(R)$$
$$T2 \leftarrow \pi_Y((S \times T1) - R)$$
$$T \leftarrow T1 - T2$$

Table 6.1 Operations of Relational Algebra

OPERATION	PURPOSE	NOTATION
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{<\text{selection condition}>}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{<\text{attribute list}>}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{<\text{join condition}>} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{<\text{join condition}>} R_2$, OR $R_1 \bowtie_{(<\text{join attributes 1}>), (<\text{join attributes 2}>)} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{<\text{join condition}>} R_2$, OR $R_1 *_{(<\text{join attributes 1}>), (<\text{join attributes 2}>)} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

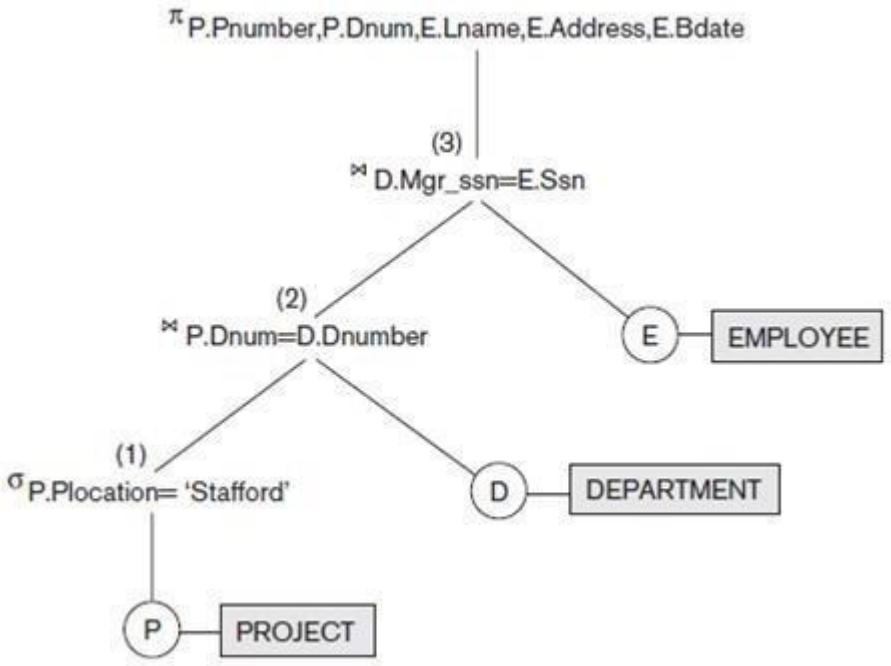
NotationforQueryTrees

Query tree (query evaluation tree or query execution tree) is used in relational systems to represent queries internally. A query tree is a tree data structure that corresponds to a relational algebra expression. It represents the input relations of the query as leaf nodes of the tree, and represents the relational algebra operations as internal nodes.

An execution of the query tree consists of executing an internal node operation whenever itsoperands represented by its child nodes are available, and then replacing that internal node by the relation that results from executing the operation. The execution terminates when the root node is executed and produces the result relation for the query.

Example: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

$$\pi_{Pnumber,\ Dnum,\ Lname,\ Address,\ Bdate}(((\sigma_{Plocation='Stafford'}(PROJECT)) \bowtie_{Dnum=Dnumber}(DEPARTMENT)) \bowtie_{Mgr\_ssn=Ssn}(EMPLOYEE))$$



LeafnodesP,D,andErepresentthethreerelationsPROJECT,DEPARTMENT,andEMPLOYEE. Therelational algebraoperations inthe expression arerepresented by internal treenodes. Thequery tree signifies an explicit order of execution inthe following sense. The node marked (1) mustbegin execution before node (2) because some resulting tuples of operation (1) mustbeavailable before wecanbeginto executeoperation(2).Similarly,node(2)mustbegintoexecuteandproduceresults before node (3) can start execution, and so on.

A query tree gives a good visual representation and understanding of the query in terms of the relational operations it uses and is recommended as an additional means for expressing queries in relational algebra.

AdditionalRelationalOperations

GeneralizedProjection

Thegeneralizedprojectionoperationextendstheprojectionoperationbyallowingfunctionsof attributes to be included in the projection list. The generalized form can be expressed as:

$$\pi_{F1,F2,\ldots,Fn}(R)$$

where$F_1,F_2,\ldots,F_n$ arefunctionsovertheattributesinrelationRandmayinvolvearithmetic operations and constant values.

The generalized projection helpful when developing reports where computed values have to be producedinthecolumnsofaqueryresult.Forexample,considertherelationEMPLOYEE(Ssn, Salary,Deduction, Years_service). A report may be required to show

Net Salary = Salary – Deduction,

Bonus=2000*Years_service,and Tax

= 0.25 * Salary.

generalizedprojectioncombinedwithrenaming:

$$\textbf{REPORT} \leftarrow \rho_{(Ssn,Net\_salary,Bonus,Tax)}(\pi_{Ssn,Salary-Deduction,2000*Years\_service,\ 0.25*Salary}(\textbf{EMPLOYEE})).$$

AggregateFunctionsandGrouping

Aggregate functions are used in simple statistical queries that summarize information from the database tuples.Common functions applied to collections of numeric values include SUM,AVERAGE, MAXIMUM, and MINIMUM.The COUNT function is used for counting tuplesor values. For example, retrieving the average or total salary of all employees or the total number of employee tuples.

Grouping the tuples in a relation by the value of some of their attributes and then applying an aggregatefunctionindependentlytoeachgroup.Forexample,groupEMPLOYEEtuplesbyDno,so that each group includes the tuples for employees working in the same department. We can then list each Dno value along with, say, the average salary of employees within the department, or the number of employees who work in the department.

Aggregate functionoperationcanbedefinedbyusingthesymbolℑ(scriptF) :

$$\langle \text{groupingattributes} \rangle \Im_{\langle \text{functionlist} \rangle}(R)$$

Where,

<groupingattributes>:list ofattributesoftherelationspecifiedin*R*

<functionlist>:listof(<function><attribute>)pairs.

<function>-suchas SUM,AVERAGE,MAXIMUM, MINIMUM,COUNT

<attribute>isan attributeof therelationspecifiedby R

Theresultingrelationhasthegroupingattributesplusoneattributeforeachelementinthefunction list.

Example: Toretrieveeachdepartmentnumber,thenumberofemployees inthedepartment,and their average salary, while renaming the resulting attributes

$$\rho_{R(\text{Dno,No\_of\_employees,Average\_sal})}(_{\text{Dno}}\Im \text{COUNT}_{\text{Ssn,AVERAGESalary}}(\text{EMPLOYEE}))$$

The aggregate function operation.

a. $\rho_{R(\text{Dno, No\_of\_employees, Average\_sal})}(_{\text{Dno}}\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE}))$.

b. $_{\text{Dno}}\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.

c. $\Im_{\text{COUNT Ssn, AVERAGE Salary}}(\text{EMPLOYEE})$.

R

(a)

Dno	No_of_employees	Average_sal
5	4	33250
4	3	31000
1	1	55000

(b)

Dno	Count_ssn	Average_salary
5	4	33250
4	3	31000
1	1	55000

(c)

Count_ssn	Average_salary
8	35125

RecursiveClosureOperations

Recursive closure operation is applied to a recursive relationship between tuples of the same type, such as the relationship between an employee and a supervisor.

Example : Retrieve all supervisees of an employee *e* at all levels—that is, all employees *e*' directly supervised by *e*, all employees *e'*ℑ directly supervised by each employee *e*', all employees *e'''* directly supervised by each employee *e*'' and so on.

$$\text{BORG\_SSN} \leftarrow \pi_{\text{Ssn}}(\sigma_{\text{Fname='James' AND Lname='Borg'}}(\text{EMPLOYEE}))$$
$$\text{SUPERVISION}(\text{Ssn1, Ssn2}) \leftarrow \pi_{\text{Ssn,Super\_ssn}}(\text{EMPLOYEE})$$
$$\text{RESULT1}(\text{Ssn}) \leftarrow \pi_{\text{Ssn1}}(\text{SUPERVISION} \bowtie_{\text{Ssn2=Ssn}} \text{BORG\_SSN})$$

SUPERVISION

(Borg's Ssn is 888665555)

(Ssn)	(Super_ssn)
Ssn1	Ssn2
123456789	333445555
333445555	888665555
999887777	987654321
987654321	888665555
666884444	333445555
453453453	333445555
987987987	987654321
888665555	null

RESULT1

Ssn
333445555
987654321

(Supervised by Borg)

To retrieve all employees supervised by Borg at level 2—that is, all employees e'' supervised by some employee e' who is directly supervised by Borg—we can apply another JOIN to the result of the first query, as follows:

$$\text{RESULT2(Ssn)} \leftarrow \pi_{\text{Ssn1}}(\text{SUPERVISION} \bowtie_{\text{Ssn2=Ssn}} \text{RESULT1})$$

RESULT2

Ssn
123456789
999887777
666884444
453453453
987987987

(Supervised by Borg's subordinates)

To get both sets of employees supervised at levels 1 and 2 by 'James Borg', we can apply the UNION operation to the two results, as follows:

$$\text{RESULT} \leftarrow \text{RESULT2} \cup \text{RESULT1}$$

OUTERJOINOperations

The JOIN operations match tuples that satisfy the join condition. For example, for a NATURAL JOIN operation R * S, only tuples from R that have matching tuples in S—and vice versa—appear in the result. Hence, tuples without a matching (or related) tuple are eliminated from the

JOIN result. Tuples with NULL values in the join attributes are also eliminated.This type of join, where tuples with no match are eliminated, is known as an inner join.

A set of operations, called **outer joins**, were developed for the case where the user wants to keep all the tuples in R, or all those in S, or all those in both relations in the result of the JOIN, regardless of whether or not they have matching tuples in the other relation.

For example, suppose that we want a list of all employee names as well as the name of the departments they manage if they happen to manage a department; if they do not manage one, we can indicate it with a NULL value. We can apply an operation **LEFT OUTER JOIN**, denoted by ⟕ toretrievetheresultasfollows:

$$\text{TEMP} \leftarrow (\text{EMPLOYEE} \overset{}{\bowtie}_{\text{Ssn=Mgr\_ssn}} \text{DEPARTMENT})$$
$$\text{RESULT} \leftarrow \pi_{\text{Fname, Minit, Lname, Dname}}(\text{TEMP})$$

TheLEFT OUTERJOIN operationkeepseverytupleinthefirst,orleft,relationR in R ⟕ S;ifnomatchingtupleisfoundinS,thentheattributesofSinthejoinresultare filled or padded with NULL values.

RESULT

Fname	Minit	Lname	Dname
John	B	Smith	NULL
Franklin	T	Wong	Research
Alicia	J	Zelaya	NULL
Jennifer	S	Wallace	Administration
Ramesh	K	Narayan	NULL
Joyce	A	English	NULL
Ahmad	V	Jabbar	NULL
James	E	Borg	Headquarters

A similar operation, RIGHT OUTER JOIN, denoted by ⟖ keeps every tuplein the *second,* or right, relation *S* in the resultof $R \overset{}{\bowtie} S$.

Athirdoperation,**FULLOUTERJOIN**,denotedby ⟗, keeps all tuples in both the left andtherightrelationswhen nomatchingtuplesarefound,paddingthemwithNULLvaluesas needed.

TheOUTER UNIONOperation

The **OUTER UNION** operation was developed to take the union of tuples from two relationsthathave some common attributes,butare notunion (type)compatible.Thisoperation will take

the UNION of tuples in two relations R(X, Y) and S(X, Z) that are **partially compatible**, meaning that only some of their attributes, say X, are union compatible.

The attributes that are union compatible are represented only once in the result, and those attributes that are not union compatible from either relation are also kept in the result relation T(X, Y, Z). Two tuples $t_1$in *R* and $t_2$in *S* are said to match if $t_1[X]= t_2[X]$. These willbe combined (unioned) into a single tuple in *t*. Tuples in either relation that have no matching tuple in the other relation are padded with NULL values.

Forexample,anOUTER UNIONcan beapplied to tworelationswhoseschemasare:

STUDENT(Name, Ssn, Department, Advisor)

INSTRUCTOR(Name,Ssn,Department, Rank)

Tuples from the two relations are matched based on having the same combination of values ofthe shared attributes—Name, Ssn, Department. All the tuples from both relations are included in the result, but tuples with the same (Name, Ssn, Department) combination will appear only once in the result. Tuples appearing only in STUDENT will have a NULL for the Rank attribute, whereas tuples appearing only in INSTRUCTOR will have a NULL for the Advisor attribute.

A tuple that exists in both relations, which represent a student who is also an instructor, willhave values for all its attributes The resulting relation, STUDENT_OR_INSTRUCTOR, willhave the following attributes:

STUDENT_OR_INSTRUCTOR(Name,Ssn,Department,Advisor, Rank)

ExamplesofQueriesinRelationalAlgebra

Query1. Retrievethename andaddressofallemployeeswhowork forthe ‘Research’ department.

$$RESEARCH\_DEPT \leftarrow \sigma_{Dname='Research'}(DEPARTMENT)$$
$$RESEARCH\_EMPS \leftarrow (RESEARCH\_DEPT \bowtie_{Dnumber=Dno} EMPLOYEE)$$
$$RESULT \leftarrow \pi_{Fname,\ Lname,\ Address}(RESEARCH\_EMPS)$$

As a single in-line expression, this query becomes:

$$\pi_{Fname,\ Lname,\ Address}(\sigma_{Dname='Research'}(DEPARTMENT \bowtie_{Dnumber=Dno}(EMPLOYEE))$$

Query2. Foreveryprojectlocatedin‘Stafford’,listtheprojectnumber,thecontrollingdepartment number, and the department manager’s last name, address, and birth date.

$$STAFFORD\_PROJS \leftarrow \sigma_{Plocation='Stafford'}(PROJECT)$$
$$CONTR\_DEPTS \leftarrow (STAFFORD\_PROJS \bowtie_{Dnum=Dnumber} DEPARTMENT)$$
$$PROJ\_DEPT\_MGRS \leftarrow (CONTR\_DEPTS \bowtie_{Mgr\_ssn=Ssn} EMPLOYEE)$$
$$RESULT \leftarrow \pi_{Pnumber,\ Dnum,\ Lname,\ Address,\ Bdate}(PROJ\_DEPT\_MGRS)$$

Query3.Findthenamesofemployeeswhoworkonalltheprojectscontrolledbydepartment number 5.

$$DEPT5\_PROJS \leftarrow \rho_{(Pno)}(\pi_{Pnumber}(\sigma_{Dnum=5}(PROJECT)))$$
$$EMP\_PROJ \leftarrow \rho_{(Ssn,\ Pno)}(\pi_{Essn,\ Pno}(WORKS\_ON))$$
$$RESULT\_EMP\_SSNS \leftarrow EMP\_PROJ \div DEPT5\_PROJS$$
$$RESULT \leftarrow \pi_{Lname,\ Fname}(RESULT\_EMP\_SSNS * EMPLOYEE)$$

Query4. Makealistof projectnumbersforprojectsthatinvolveanemployeewhoselastnameis 'Smith', either as a worker or as a manager of the department that controls the project.

$$SMITHS(Essn) \leftarrow \pi_{Ssn}\ (\sigma_{Lname='Smith'}(EMPLOYEE))$$
$$SMITH\_WORKER\_PROJS \leftarrow \pi_{Pno}(WORKS\_ON * SMITHS)$$
$$MGRS \leftarrow \pi_{Lname,\ Dnumber}(EMPLOYEE \bowtie_{Ssn=Mgr\_ssn} DEPARTMENT)$$
$$SMITH\_MANAGED\_DEPTS(Dnum) \leftarrow \pi_{Dnumber}\ (\sigma_{Lname='Smith'}(MGRS))$$
$$SMITH\_MGR\_PROJS(Pno) \leftarrow \pi_{Pnumber}(SMITH\_MANAGED\_DEPTS * PROJECT)$$
$$RESULT \leftarrow (SMITH\_WORKER\_PROJS \cup SMITH\_MGR\_PROJS)$$

Query5. List thenames ofallemployees withtwo ormoredependents.

$$T1(Ssn,\ No\_of\_dependents) \leftarrow {}_{Essn}\ \Im\ _{COUNT\ Dependent\_name}(DEPENDENT)$$
$$T2 \leftarrow \sigma_{No\_of\_dependents>2}(T1)$$
$$RESULT \leftarrow \pi_{Lname,\ Fname}(T2 * EMPLOYEE)$$

Query6. Retrievethe names ofemployees whohaveno dependents.

$$ALL\_EMPS \leftarrow \pi_{Ssn}(EMPLOYEE)$$
$$EMPS\_WITH\_DEPS(Ssn) \leftarrow \pi_{Essn}(DEPENDENT)$$
$$EMPS\_WITHOUT\_DEPS \leftarrow (ALL\_EMPS - EMPS\_WITH\_DEPS)$$
$$RESULT \leftarrow \pi_{Lname,\ Fname}(EMPS\_WITHOUT\_DEPS * EMPLOYEE)$$

Query7. List the namesofmanagers whohaveat least one dependent.

$$MGRS(Ssn) \leftarrow \pi_{Mgr\_ssn}(DEPARTMENT)$$
$$EMPS\_WITH\_DEPS(Ssn) \leftarrow \pi_{Essn}(DEPENDENT)$$
$$MGRS\_WITH\_DEPS \leftarrow (MGRS \cap EMPS\_WITH\_DEPS)$$
$$RESULT \leftarrow \pi_{Lname,\ Fname}(MGRS\_WITH\_DEPS * EMPLOYEE)$$

Chapter3:MappingConceptualDesignintoaLogicalDesign

RelationalDatabaseDesignusingER-to-Relationalmapping

Proceduretocreatearelationalschemafromanentity-Relationship(ER)

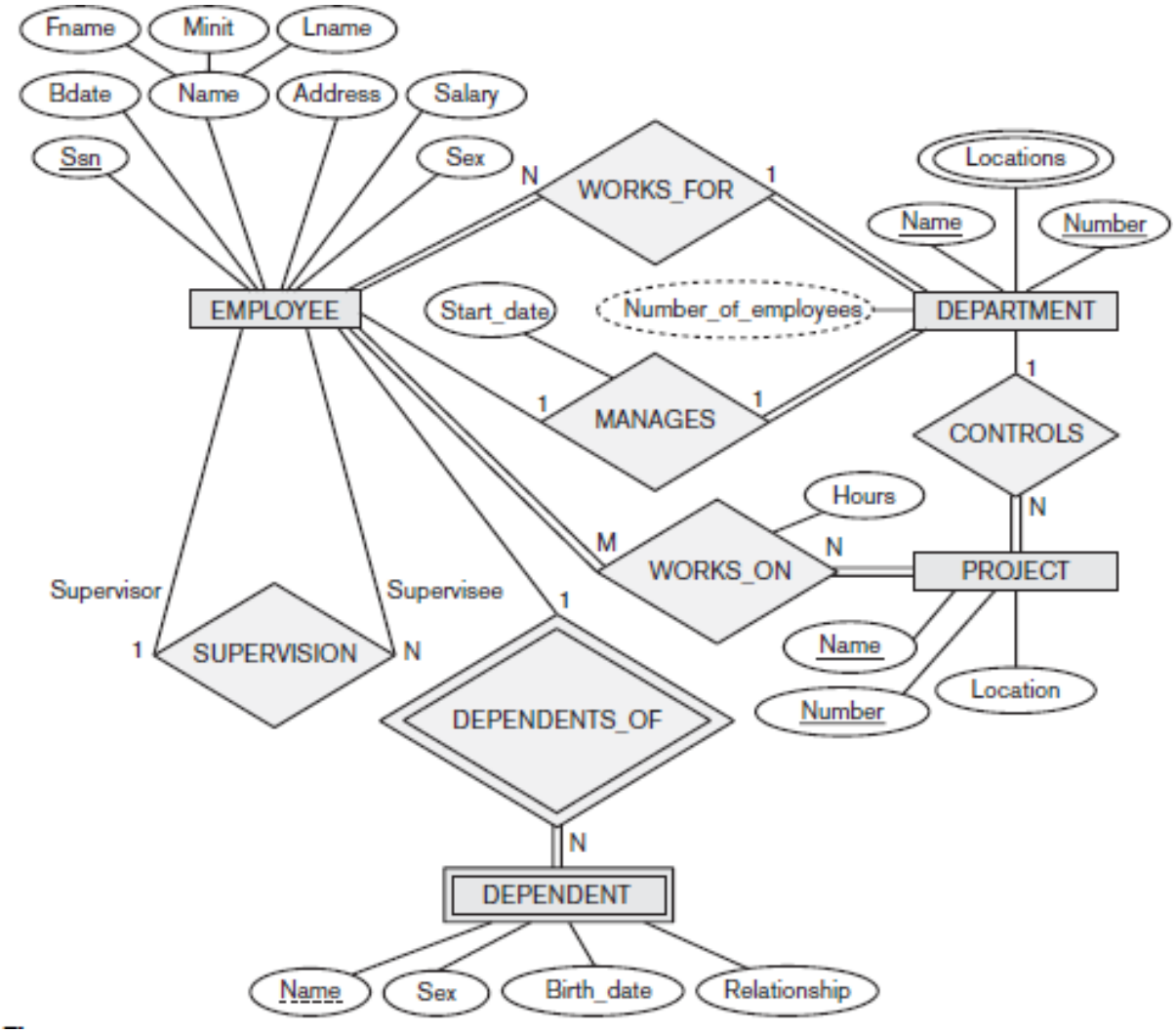


Fig3.1:ERdiagramofcompany database

Step1:MappingofRegularEntityTypes

- Foreachregularentitytype,createarelationRthatincludesallthesimpleattributesofE
- Includeonlythe simplecomponentattributes ofacomposite attribute
- Chooseoneof thekeyattributes of Eas the primarykeyforR
- Ifthechosen keyof Eisacomposite,thentheset ofsimpleattributesthatform itwill together form the primary key of R.

- If multiple keys were identified for E during the conceptual design, the information describingtheattributesthatformeachadditional keyiskeptinordertospecifysecondary (unique) keys of relation R
- Inourexample-COMPANYdatabase,wecreatetherelationsEMPLOYEE,DEPARTMENT, and PROJECT
- wechooseSsn,Dnumber,andPnumberasprimarykeysfortherelationsEMPLOYEE, DEPARTMENT, and PROJECT, respectively
- Therelationsthat arecreatedfrom themappingofentitytypesarecalled**entityrelations** becauseeachtuplerepresentsanentityinstance.

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary

DEPARTMENT

Dname	Dnumber

PROJECT

Pname	Pnumber	Plocation

Step2:MappingofWeakEntityTypes

- Foreachweakentitytype, createarelationRand includeallsimpleattributesoftheentity type as attributes of R
- Includeprimarykeyattributeofownerasforeignkeyattributesof R
- Inourexample,wecreatetherelationDEPENDENTinthissteptocorrespondtotheweak entity type DEPENDENT
- WeincludetheprimarykeySsnoftheEMPLOYEErelation—whichcorrespondstothe owner entity type—as a foreign key attribute of DEPENDENT; we rename itasEssn
- TheprimarykeyoftheDEPENDENTrelationisthecombination{Essn,Dependent_name}, because Dependent_name is the partial key of DEPENDENT
- Itiscommontochoosethepropagate(CASCADE)optionforthereferentialtriggeredaction on the foreign key in the relation corresponding to the weak entity type, since a weak entity has an existence dependency on its owner entity.
- ThiscanbeusedforbothONUPDATE andONDELETE.

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

Step3:MappingofBinary1:1Relationship Types

- Foreachbinary1:1relationshiptype*R*intheERschema,identifytherelations *S*and*T*that correspond to the entity types participating in *R*
- Therearethreepossibleapproaches:
  - foreignkeyapproach
  - mergedrelationshipapproach
  - crossreferenceorrelationship relationapproach

1. Theforeign keyapproach

- Chooseoneoftherelations—*S*, say—andincludeas aforeign keyin*S* the primarykey of*T*.
- Itisbettertochooseanentitytypewith*totalparticipation*in*R*intheroleof*S*
- Includeallthesimpleattributes(orsimplecomponentsofcompositeattributes)ofthe1:1 relationship type *R* as attributes of *S*.
- Inourexample,wemapthe1:1relationshiptypebychoosingtheparticipatingentitytype DEPARTMENT to serve in the role of *S* because its participation in the MANAGES relationship type is total
- WeincludetheprimarykeyoftheEMPLOYEErelationasforeignkey inthe DEPARTMENT relation and rename it Mgr_ssn.
- WealsoincludethesimpleattributeStart_dateoftheMANAGESrelationshiptypeinthe DEPARTMENT relation and rename it Mgr_start_date

2. Mergedrelationapproach:

- mergethetwo entitytypesand therelationshipinto asinglerelation
- Thisispossiblewhen*bothparticipationsaretotal*,asthiswouldindicatethatthetwo tables will have the exact same number of tuples at all times.

3. Cross-referenceorrelationshiprelationapproach:

- setupathirdrelationRforthepurposeofcross-referencingtheprimarykeysofthetwo relations S and T representing the entity types.
- requiredforbinaryM:Nrelationships
- TherelationRiscalledarelationshiprelation(orsometimesalookuptable),becauseeach tuple in R represents a relationship instance that relates one tuple from S with one tuple from T
- Therelation Rwill includetheprimarykeyattributesofS andT as foreignkeys to S andT.
- TheprimarykeyofRwill beoneofthetwoforeignkeys,andtheotherforeignkeywillbe a unique key of R.

- Thedrawbackishavinganextrarelation,andrequiringanextrajoinoperationwhen combining related tuples from the tables.

Step4:Mappingof Binary1:NRelationshipTypes

- Foreachregularbinary1:Nrelationshiptype*R*,identifytherelation*S*thatrepresentsthe participating entity type at the *N-side* of the relationship type.
- Includeasforeignkeyin*S*theprimarykeyof therelation*T*thatrepresentstheotherentity type participating in *R*
- Includeanysimpleattributes(orsimplecomponentsofcompositeattributes)ofthe1:N relationship type as attributes of *S*
- Inourexample,wenowmapthe1:NrelationshiptypesWORKS_FOR,CONTROLS,and SUPERVISION
- ForWORKS_FOR weinclude theprimary key Dnumberof theDEPARTMENT relationas foreign key in the EMPLOYEE relation and call it Dno.
- For SUPERVISIONweincludethe primary key of the EMPLOYEE relation as foreignkey in the EMPLOYEE relation itself—because the relationship is recursive—and call it Super_ssn.
- TheCONTROLSrelationshipismappedto theforeignkeyattributeDnumofPROJECT, which references the primary key Dnumber of the DEPARTMENTrelation.

Step5:MappingofBinaryM:NRelationshipTypes

- ForeachbinaryM:Nrelationship type
  - CreateanewrelationS
  - Includeprimarykey ofparticipatingentitytypesasforeignkey attributesinS
  - IncludeanysimpleattributesofM:Nrelationshiptype
- In our example, we map the M:N relationship type WORKS_ON by creating the relation WORKS_ON.WeincludetheprimarykeysofthePROJECTandEMPLOYEErelationsas foreign keys in WORKS_ON and rename them Pno and Essn, respectively.
- WealsoincludeanattributeHoursinWORKS_ONtorepresenttheHoursattributeofthe relationship type.
- TheprimarykeyoftheWORKS_ONrelationisthecombinationoftheforeignkey attributes {Essn, Pno}.

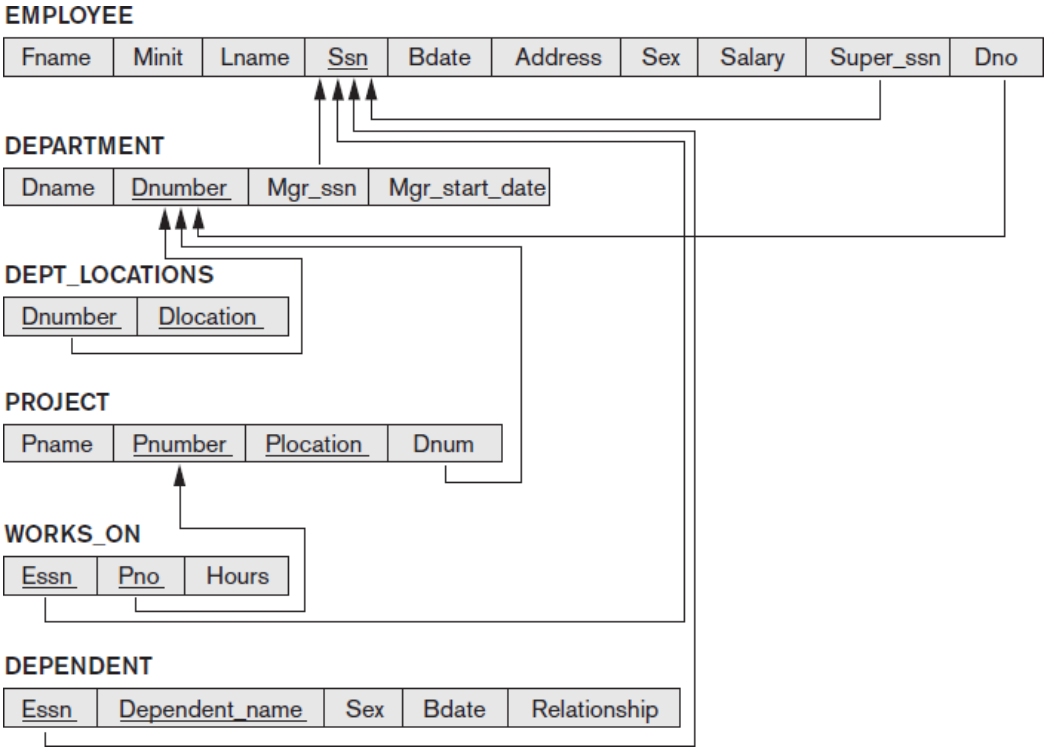
WORKS_ON

Essn	Pno	Hours

- The propagate (CASCADE) option for the referential triggered action should be specified on the foreign keys in the relation corresponding to the relationship R, since each relationship instance has an existence dependency on each of the entitiesit relates. This can be used for both ON UPDATE and ON DELETE.

Step6:MappingofMultivaluedAttributes

- Foreachmultivaluedattribute
  - Createanewrelation
  - PrimarykeyofRisthecombination ofAandK
  - Ifthemultivaluedattributeiscomposite,includeitssimplecomponents
- Inourexample,wecreatearelationDEPT_LOCATIONS
- The attribute Dlocation represents the multivalued attribute LOCATIONS of DEPARTMENT, while Dnumber—as foreign key—represents theprimary key ofthe DEPARTMENT relation.
- TheprimarykeyofDEPT_LOCATIONSisthecombinationof{Dnumber,Dlocation}
- AseparatetuplewillexistinDEPT_LOCATIONSforeachlocationthatadepartmenthas
- The propagate (CASCADE) option for the referential triggered action should be specified ontheforeignkeyintherelationRcorrespondingtothemultivaluedattributeforbothON UPDATE and ON DELETE.



Step7:Mappingof*N*-aryRelationship Types

- Foreach*n*-aryrelationshiptype*R*
  - Createanewrelation*S*torepresent*R*
  - Includeprimarykeysof participatingentitytypesasforeignkeys
  - Includeanysimpleattributesasattributes
- TheprimarykeyofSisusuallyacombinationofalltheforeignkeysthatreferencethe relations representing the participating entity types.
- Forexample,considertherelationshiptypeSUPPLY.Thiscanbe mappedto therelation SUPPLY whose primary key is the combination of the three foreign keys {Sname, Part_no, Proj_name}.

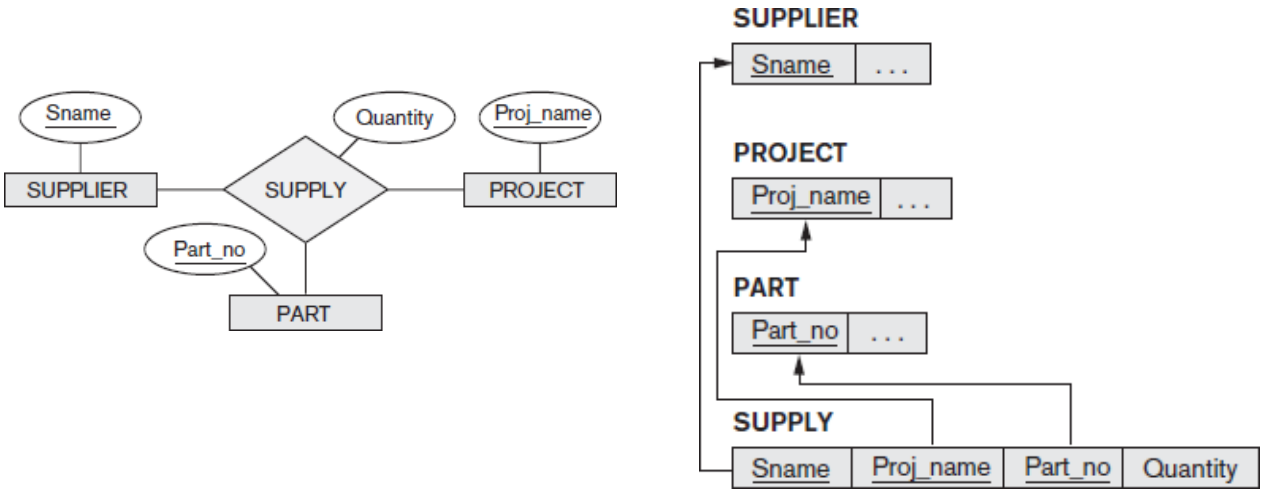


Figure3.2:Mappingthe*n*-aryrelationshiptypeSUPPLY

Chapter4:SQL

Introduction

SQL was called SEQUEL (Structured English Query Language) and was designed and implemented at IBM Research.The SQL language may be considered one of the major reasons for the commercial success of relational databases. SQL is a comprehensive database language. It has statements for data definitions, queries, and updates. Hence, it is both a DDL *and* a DML. In addition, it has facilities for defining views on the database, for specifying security and authorization, for defining integrity constraints, and for specifying transaction controls. It also has rules for embedding SQL statements into a general-purpose programming language such as Java, COBOL, or C/C++.

SQLDataDefinitionandDataTypes

SQL uses the terms table, row, and column for the formal relational model terms relation, tuple, and attribute, respectively. The main SQL command for data definition is the CREATE statement, which can be used to create schemas, tables (relations), domains, views, assertions and triggers.

SchemaandCatalogConceptsinSQL

An SQL schema is identified by a schema name, and includes an authorization identifier to indicate the user or account who owns the schema, as well as descriptors for *each element* in the schema. Schema elements include tables, constraints, views, domains, and other constructs (such as authorization grants) that describe the schema. A schema is created viathe CREATE SCHEMA statement .

For example, the following statement creates a schema called COMPANY, owned by theuser with authorization identifier ‘Jsmith’..

CREATESCHEMACOMPANY**AUTHORIZATION**‘Jsmith’;

In general, not all users are authorized to create schemas and schema elements. The privilege to create schemas, tables, and other constructs must be explicitly granted to the relevant user accounts by the system administrator or DBA.

SQL uses the concept of a **catalog**—a named collection of schemas in an SQL environment. A catalog always contains a special schema called INFORMATION_SCHEMA, which providesinformationonalltheschemasin thecatalogandalltheelementdescriptorsinthese

schemas. Integrity constraints such as referential integrity can be defined between relations only if they exist in schemas within the same catalog. Schemas within the same catalog can also share certain elements, such as domain definitions.

TheCREATETABLECommandinSQL

TheCREATETABLEcommand is used to specify anewrelation by giving it anameand specifying its attributes and initial constraints. The attributes are specified first, and each attribute is given a name, a data type to specify its domain of values, and any attribute constraints, such as NOT NULL. The key, entity integrity, and referential integrity constraints can be specified within the CREATE TABLE statement after the attributes are declared, or they can be added later using the ALTER TABLE command.

Typically, the SQL schema in which the relations are declared is implicitly specified in the environment in which the CREATE TABLE statements are executed. Alternatively, we canexplicitly attach the schema name to the relation name, separated by a period. For example, by writing

CREATETABLECOMPANY.EMPLOYEE...

rather than

CREATETABLEEMPLOYEE...

TherelationsdeclaredthroughCREATETABLEstatementsarecalled **basetables. Examples:**

```
CREATE TABLE EMPLOYEE
        ( Fname          VARCHAR(15)          NOT NULL,
          Minit          CHAR,
          Lname          VARCHAR(15)          NOT NULL,
          Ssn            CHAR(9)              NOT NULL,
          Bdate          DATE,
          Address        VARCHAR(30),
          Sex            CHAR,
          Salary         DECIMAL(10,2),
          Super_ssn      CHAR(9),
          Dno            INT                  NOT NULL,
        PRIMARY KEY (Ssn),
        FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber) );
```

```
CREATE TABLE DEPARTMENT
        ( Dname                 VARCHAR(15)             NOT NULL,
          Dnumber               INT                     NOT NULL,
          Mgr_ssn               CHAR(9)                 NOT NULL,
          Mgr_start_date        DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
CREATE TABLE DEPT_LOCATIONS
        ( Dnumber               INT                     NOT NULL,
          Dlocation             VARCHAR(15)             NOT NULL,
        PRIMARY KEY (Dnumber, Dlocation),
        FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
CREATE TABLE PROJECT
        ( Pname                 VARCHAR(15)             NOT NULL,
          Pnumber               INT                     NOT NULL,
          Plocation             VARCHAR(15),
          Dnum                  INT                     NOT NULL,
        PRIMARY KEY (Pnumber),
        UNIQUE (Pname),
        FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );


CREATE TABLE WORKS_ON
        ( Essn                  CHAR(9)                 NOT NULL,
          Pno                   INT                     NOT NULL,
          Hours                 DECIMAL(3,1)            NOT NULL,
        PRIMARY KEY (Essn, Pno),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
        FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );
CREATE TABLE DEPENDENT
        ( Essn                  CHAR(9)                 NOT NULL,
          Dependent_name        VARCHAR(15)             NOT NULL,
          Sex                   CHAR,
          Bdate                 DATE,
          Relationship          VARCHAR(8),
        PRIMARY KEY (Essn, Dependent_name),
        FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

AttributeDataTypesandDomainsin SQL

Basicdatatypes

1. **Numeric**datatypesincludes
   - integernumbersofvarious sizes(INTEGERor INT,andSMALLINT)
   - floating-point(real)numbersofvariousprecision(FLOATorREAL,and DOUBLE PRECISION).
   - FormattednumberscanbedeclaredbyusingDECIMAL(i,j)—or DEC(i,j) or NUMERIC(i,j)—where
     i-precision,totalnumberofdecimaldigits
     j-scale, numberofdigitsafter thedecimal point

2. **Character-stringdatatypes**
   - fixedlength—CHAR(*n*) orCHARACTER(*n*),where*n*isthenumberofcharacters
   - varyinglength—VARCHAR(*n*)orCHARVARYING(*n*)orCHARACTERVARYING(*n*), where *n* is the maximum number of characters
   - Whenspecifyingaliteralstringvalue,itisplacedbetweensinglequotationmarks (apostrophes), and it is *casesensitive*
   - Forfixedlengthstrings,ashorterstringispaddedwithblankcharacterstotheright
   - Forexample,ifthevalue‘Smith’isforanattributeoftypeCHAR(10),itispaddedwith five blank characters to become ‘Smith ’ if needed
   - Paddedblanksaregenerallyignoredwhenstringsarecompared
   - Anothervariable-lengthstringdatatypecalledCHARACTERLARGEOBJECTorCLOB is also available to specify columns that have large text values, such as documents
   - TheCLOBmaximumlengthcanbespecifiedinkilobytes(K),megabytes(M),orgigabytes (G)
   - Forexample,CLOB(20M)specifiesa maximumlengthof20megabytes.

3. **Bit-string**datatypesare eitherof
   - fixedlength*n*—BIT(*n*)—orvaryinglength—BITVARYING(*n*),where*n*isthemaximum number of bits.
   - Thedefaultfor*n*,thelengthofacharacterstringorbitstring,is1.

- Literalbitstringsareplacedbetweensinglequotesbutprecededbya Btodistinguishthem from character strings; for example, B‘10101’
- Anothervariable-lengthbitstringdatatypecalledBINARYLARGEOBJECTorBLOB is also available to specify columns that have large binary values, such as images.
- ThemaximumlengthofaBLOBcanbespecifiedinkilobits(K),megabits(M),orgigabits (G)
- Forexample,BLOB(30G)specifiesamaximumlengthof30gigabits.

4. **A Boolean** data type has the traditional values of TRUE or FALSE.In SQL, because of the presenceofNULLvalues,athree-valuedlogicisused,soathirdpossiblevalueforaBoolean data type is UNKNOWN
5. The**DATE**datatypehas tenpositions,anditscomponentsareYEAR,MONTH,and DAYin the form YYYY-MM-DD
6. The **TIME data** type has at least eight positions, with the components HOUR, MINUTE, and SECOND in the form HH:MM:SS.
   Onlyvaliddates andtimes shouldbe allowedby theSQLimplementation.
7. **TIME WITH TIME ZONE** data type includes an additional six positions for specifying the displacement from the standard universal time zone, whichis in the range +13:00 to–12:59inunits ofHOURS:MINUTES. If WITH TIME ZONE isnot included, the default isthe local time zone for the SQL session.

Additionaldatatypes

1. **Timestamp** data type (TIMESTAMP) includes the DATE and TIME fields, plus a minimum of six positions for decimal fractions of seconds and an optional WITH TIME ZONE qualifier.
2. **INTERVA**L data type. This specifies an **interval**—a relative value that can be used to increment or decrement an absolute value of a date, time, or timestamp. Intervals arequalified to be either YEAR/MONTH intervals or DAY/TIMEintervals.

It is possible to specify the data type of each attribute directly or a domain can be declared, and the domain name used with the attribute Specification. This makes it easier to change the data type for a domain that is used by numerous attributes in a schema, and improves schema readability. For example, we can create a domain SSN_TYPE by the following statement:

CREATEDOMAINSSN_TYPE**AS**CHAR(9);

WecanuseSSN_TYPEinplaceofCHAR(9)fortheattributesSsnandSuper_ssnofEMPLOYEE, Mgr_ssn of DEPARTMENT, Essn of WORKS_ON, and Essn of DEPENDENT

SpecifyingConstraintsinSQL

BasicconstraintsthatcanbespecifiedinSQLaspartoftablecreation:

- keyandreferentialintegrityconstraints
- RestrictionsonattributedomainsandNULLs
- constraintsonindividualtuples withinarelation

SpecifyingAttributeConstraintsandAttribute Defaults

BecauseSQL allowsNULLsasattributevalues, a constraintNOTNULL maybespecifiedifNULL isnotpermittedforaparticularattribute.Thisisalwaysimplicitly specifiedfortheattributesthatare part of the primary key of each relation, but it can be specified for any other attributes whose values are required not to be NULL.

Itis alsopossible to defineadefault valuefor anattributeby appendingtheclause**DEFAULT** <value>toanattributedefinition.Thedefaultvalueisincludedinanynewtupleifanexplicitvalue is not provided for that attribute.

```
CREATETABLEDEPARTMENT (
. . . ,
    Mgr_ssnCHAR(9)NOT NULLDEFAULT‘888665555’,
    ----------
    ----------
)
```

Anothertypeofconstraintcanrestrictattributeordomainvaluesusingthe **CHECK**clausefollowing an attribute or domain definition . For example, suppose that department numbers are restricted to integer numbers between 1 and 20; then, we can change the attribute declaration of Dnumber in the DEPARTMENT table to the following:

Dnumber INT **NOTNULLCHECK** (Dnumber>0**AND**Dnumber<21);

TheCHECKclausecanalsobeusedinconjunctionwiththeCREATEDOMAINstatement.For example, we can write the following statement:

CREATEDOMAIN D_NUM**AS**INTEGER

CHECK(D_NUM>0**AND**D_NUM <21);

We can then use the created domain D_NUM as the attribute type for all attributes that refer to department number such as Dnumber of DEPARTMENT, Dnum of PROJECT, Dno ofEMPLOYEE, and so on.

SpecifyingKeyandReferentialIntegrityConstraints

The**PRIMARYKEY**clausespecifiesoneormoreattributesthatmakeup theprimarykeyofa relation. If a primary key has a single attribute, the clause can follow the attribute directly. For example, the primary key of DEPARTMENT can be specified as:

DnumberINT**PRIMARYKEY**;

The**UNIQUE**clausecan also bespecifieddirectly forasecondarykeyifthesecondarykeyisa single attribute, as in the following example:

DnameVARCHAR(15)**UNIQUE**;

Referentialintegrityisspecifiedviathe **FOREIGNKEY** clause

FOREIGNKEY (Super_ssn)**REFERENCES**EMPLOYEE(Ssn),
FOREIGNKEY(Dno)**REFERENCES**DEPARTMENT(Dnumber

A referential integrity constraint can be violated when tuples are inserted or deleted, or when a foreign key or primary key attribute value is modified. The default action that SQL takes for an integrity violation is to **reject** the update operation that will cause a violation, which is known as the RESTRICT option.

The schema designer can specify an alternative action to be taken by attaching a **referential triggered action** clause to any foreign key constraint. The options include SET NULL, CASCADE, and SET DEFAULT. An option must be qualified with either ON DELETE or ON UPDATE

- **FOREIGNKEY**(Dno)**REFERENCES**DEPARTMENT(Dnumber)**ONDELETE**SET DEFAULT **ON UPDATE** CASCADE
- **FOREIGNKEY**(Super_ssn)**REFERENCES**EMPLOYEE(Ssn)**ONDELETE**SET NULL **ON UPDATE** CASCADE
- **FOREIGNKEY**(Dnumber)**REFERENCES** DEPARTMENT(Dnumber)**ONDELETE** CASCADE**ONUPDATE** CASCADE

In general, the action taken by the DBMS for SET NULL or SET DEFAULT is the same forboth ON DELETE and ON UPDATE: The value of the affected referencing attributes ischanged to NULL for SET NULL and to the specified default value of the referencing attribute for SET DEFAULT.

The action for CASCADE ON DELETEis todelete all the referencingtuples whereas the actionforCASCADEONUPDATE isto changethevalueofthereferencing foreign key attribute(s)to the updated (new) primary key value for all the referencing tuples . It is the responsibility of the database designer to choose the appropriate action and to specify it in the database schema. As a general rule, the CASCADE option issuitable for"relationship" relations such as WORKS_ON; for relations that represent multivalued attributes, such as DEPT_LOCATIONS; and for relations that represent weak entity types, such as DEPENDENT.

Giving Names toConstraints

Thenames of all constraints within a particularschema must be unique. A constraint name is used to identify a particular constraint in case the constraint must be dropped later and replaced with another constraint.

SpecifyingConstraintsonTuplesUsingCHECK

In addition to key and referential integrity constraints, which are specified by special keywords, other *table constraints* can be specified through additional CHECK clauses at the end of a CREATE TABLE statement. These can be called **tuple-based** constraints because they apply to each tuple individually and are checked whenever a tuple is inserted or modified

Forexample,supposethattheDEPARTMENTtablehadanadditionalattributeDept_create_date, whichstoresthedatewhenthedepartmentwascreated.ThenwecouldaddthefollowingCHECK clause at the end of the CREATE TABLE statement for the DEPARTMENT table to make sure that a manager's start date is later than the department creation date

CHECK(Dept_create_date<=Mgr_start_date);

BasicRetrievalQueriesinSQL

SQLhasonebasicstatementfor retrievinginformation fromadatabase: the**SELECT**statement.

TheSELECT-FROM-WHEREStructureofBasicSQLQueries

The basic form of the SELECT statement, sometimes called a **mapping** or a **select-from-where block**, is formed of the three clauses SELECT, FROM, and WHERE and has the following form:

SELECT<attributelist>**FRO**

M <table list>**WHERE**

<condition>;

Where,

- <attributelist>isalistofattributenameswhosevaluesaretoberetrievedbythe query
- <tablelist>isalistoftherelationnamesrequired toprocessthequery
- <condition>isaconditional(Boolean)expressionthatidentifiesthetuplesto be retrieved by the query.

Examples:

1. Retrievethebirthdateandaddressoftheemployee(s)whosename is‘JohnB.

Smith’.

SELECTBdate, Address

FROMEMPLOYEE

WHEREFname=‘John’**AND**Minit=‘B’**AND**Lname=‘Smith’;

The SELECT clause of SQL specifies the attributes whose values are tobe retrieved, whichare called the **projection attributes.** The WHERE clause specifies the Boolean condition that must be true for any retrieved tuple, which is known as the **selection condition.**

2. Retrievethename andaddressofallemployeeswhoworkforthe‘Research’department.

SELECTFname,Lname,Address

FROMEMPLOYEE,DEPARTMENT

WHEREDname=‘Research’**AND**Dnumber=Dno;

In the WHERE clause, the condition Dname = ‘Research’ is a **selection condition**that chooses the particular tuple of interest in the DEPARTMENT table, because Dname is an attribute of DEPARTMENT. The condition Dnumber = Dno is called a **join condition**, because it combines two tuples: one from DEPARTMENT and one from EMPLOYEE, whenever the value of Dnumber in DEPARTMENT is equal to thevalue of Dno in EMPLOYEE.A query that involves only selection and join conditions plus projectionattributes is known as a **select-project-join** query.

3. For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, address, and birthdate.

SELECTPnumber,Dnum,Lname,Address,Bdate

FROMPROJECT,DEPARTMENT,EMPLOYEE

WHEREDnum=Dnumber**AND**Mgr_ssn=Ssn**AND**Plocation='Stafford';

The join condition Dnum = Dnumber relates a project tuple to its controlling department tuple, whereas the join condition Mgr_ssn = Ssn relates the controlling department tuple to the employee tuple who manages that department. Each tuple in the result will be a *combination* of one project, one department, and one employee that satisfies the joinconditions. The projection attributes are used to choose the attributes to be displayed from each combined tuple.

AmbiguousAttributeNames,Aliasing,Renaming,andTupleVariables

In SQL, the same name can be used for two or more attributes as long as the attributes are in different relations. If this is the case, and a multitable query refers to two or more attributes with the same name, we must **qualify** the attribute name with the relation name to prevent ambiguity. This is done by prefixing the relation name to the attribute name and separating the two by a period.

Example: Retrieve the name and address of all employees who work for the 'Research' department

SELECTFname,EMPLOYEE.Name,Address

FROMEMPLOYEE,DEPARTMENT

WHEREDEPARTMENT.Name='Research'**AND**

DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;

The ambiguity of attribute names also arises in the case of queries that refer to the same relation twice.Forexampleconsiderthequery:Foreachemployee,retrievetheemployee'sfirstandlast name and the first and last name of his or her immediate supervisor.

SELECTE.Fname,E.Lname,S.Fname, S.Lname

FROMEMPLOYEE**AS**E, EMPLOYEE**AS** S

WHEREE.Super_ssn=S.Ssn;

In this case, we are required to declare alternative relation names E and S, called **aliases** or **tuple variables**, for the EMPLOYEE relation. An alias can follow the keyword **AS**, or it can directly follow the relation name—for example, by writing EMPLOYEE E, EMPLOYEE S. It is alsopossible to **rename** the relation attributes within the query in SQL by giving them aliases. For example, if we write

EMPLOYEE**AS**E(Fn,Mi,Ln,Ssn,Bd,Addr,Sex,Sal,Sssn, Dno)

intheFROMclause,FnbecomesanaliasforFname,MiforMinit,LnforLname,andsoon

UnspecifiedWHEREClauseandUseoftheAsterisk

A missing WHERE clause indicates no condition on tuple selection; hence, all tuples of the relation specified in the FROM clause qualify and are selected for the query result.If more than one relationis specified in the FROM clause and there is no WHERE clause, then the CROSS PRODUCT—all possible tuple combinations—of these relations is selected.

Example: Select all EMPLOYEE Ssns and all combinations of EMPLOYEE Ssn and DEPARTMENT Dname in the database.

```
SELECTSsn
FROMEMPLOYEE;
SELECTSsn,Dname
FROMEMPLOYEE,DEPARTMENT;
```

To retrieve all the attribute values of the selected tuples, we do not have to list the attribute names explicitly in SQL; wejust specifyan asterisk (*), which stands for all the attributes. Forexample, the following query retrieves all the attribute values of any EMPLOYEE who works in DEPARTMENT number 5

```
SELECT*FROMEMPLOYEEWHEREDno=5;

SELECT*FROMEMPLOYEE,DEPARTMENTWHERE Dname='Research'
ANDDno=Dnumber;

SELECT*FROMEMPLOYEE,DEPARTMENT;
```

Tablesas Sets inSQL

SQL usually treats a table not as a set but rather as a multiset; duplicate tuples can appear more than once in a table, and in the result of a query. SQL does not automatically eliminate duplicate tuples in the results of queries, for the following reasons:

- Duplicate elimination isan expensive operation. One way to implementitisto sort the tuples first and then eliminate duplicates.
- Theusermaywanttosee duplicatetuplesintheresultofaquery.
- When an aggregate function is applied to tuples, in most cases we do not want to eliminate duplicates.

Ifwedo wantto eliminateduplicate tuples fromtheresult ofan SQLquery, weusethekeyword **DISTINCT**intheSELECTclause,meaningthatonlydistincttuplesshouldremainintheresult.

Example: Retrievethesalaryof everyemployee andalldistinctsalaryvalues

(a) SELECTALL Salary**FROM**EMPLOYEE;

(b) SELECTDISTINCT Salary**FROM**EMPLOYEE;

(a)

Salary
30000
40000
25000
43000
38000
25000
25000
55000

(b)

Salary
30000
40000
25000
43000
38000
55000

SQL has directly incorporated some of the set operations from mathematical *set theory*, which are also part of relational algebra. There are

- setunion(**UNION**)
- setdifference(**EXCEPT)**and
- setintersection(**INTERSECT**)

The relations resulting from these set operations are sets of tuples; that is, duplicate tuples are eliminatedfromtheresult.Thesesetoperationsapplyonly to union-compatiblerelations,sowemust make sure that the two relations on which we apply the operation have the same attributes and that the attributes appear in the same order in both relations.

Example: Make a list of all project numbers for projects that involve an employee whose last nameis 'Smith', either as a worker or as a manager of the department that controls the project

(SELECTDISTINCT Pnumber**FROM**PROJECT,DEPARTMENT,
EMPLOYEE**WHERE**Dnum=Dnumber**AND** Mgr_ssn=Ssn**AND**Lname='Smith')
UNION
(SELECTDISTINCT Pnumber**FROM**PROJECT,WORKS_ON, EMPLOYEE
WHEREPnumber=Pno**AND**Essn=Ssn**AND**Lname='Smith');

SubstringPatternMatchingandArithmeticOperators

Severalmorefeaturesof SQL

Thefirstfeatureallowscomparisonconditionsononlypartsofacharacterstring,usingthe **LIKE** comparison operator. This can be used for string **pattern matching**. Partial strings are specified using two reserved characters:

- %replaces anarbitrary numberof zeroormorecharacters
- _(underscore) replacesa single character

Forexample,considerthefollowingquery:RetrieveallemployeeswhoseaddressisinHouston, Texas

SELECTFname,Lname **FROM**EMPLOYEE**WHERE** Address

LIKE'%Houston,TX%';

Toretrieveallemployeeswhowerebornduring the1950s, wecan useQuery

SELECTFname,Lname **FROM**EMPLOYEE

WHERE Bdate **LIKE** '_ _ 5_____';

If an underscore or % is needed as a literal character in the string, the character should be precededby an *escape character*, which isspecified after the string using the keyword ESCAPE. For example, 'AB_CD\%EF' ESCAPE '\' represents the literal string 'AB_CD%EF' because \ is specified as the escape character.Also, we need a rule to specify apostrophes or single quotation marks (' ') ifthey are to be included in a string because they are used to begin and end strings. If an apostrophe (') is needed, it is represented as two consecutive apostrophes (") so that it will not be interpreted asending the string.

Another feature allows the use of arithmetic in queries.The standard arithmetic operators for addition (+), subtraction (–), multiplication (*), and division (/) can be applied to numeric values or attributes with numeric domains. For example,suppose that we want to see the effect of giving all employees who work on the 'ProductX' project a 10 percent raise; we can issue the following query:

SELECTE.Fname,E.Lname,1.1*E.Salary **AS**Increased_sal

FROMEMPLOYEE**AS**E,WORKS_ON**AS** W,PROJECT **AS** P

WHERE E.Ssn=W.Essn **AND** W.Pno=P.Pnumber **AND** P.Pname='ProductX';

Example:Retrieveallemployeesindepartment5whosesalaryisbetween$30,000and$40,000.

SELECT*FROMEMPLOYEE**WHERE**(Salary**BETWEEN** 30000**AND**

40000)**AND** Dno =5;

The condition (Salary **BETWEEN** 30000 **AND** 40000) is equivalent to the condition((Salary >= 30000) **AND** (Salary <= 40000)).

OrderingofQueryResults

SQLallowstheusertoorderthetuplesintheresultofaquerybythevaluesofoneormoreofthe attributes that appear in the query result, by using the **ORDER BY** clause.

Example:Retrievealistofemployeesandtheprojectstheyareworkingon,orderedbydepartment and, within each department, ordered alphabetically bylast name, then first name.

SELECTD.Dname,E.Lname,E.Fname,P.Pname
FROMDEPARTMENTD,EMPLOYEE E,WORKS_ONW,PROJECTP
WHERED.Dnumber= E.Dno**AND**E.Ssn=W.Essn **AND**W.Pno=P.Pnumber
ORDERBYD.Dname, E.Lname,E.Fname;

The default order is in ascending order of values.We can specify the keyword **DESC** if we want to see the result in a descending order of values. The keyword ASC can be used to specify ascending order explicitly. For example, if we want descending alphabetical order on Dname and ascending order on Lname, Fname, the ORDER BY clause can be written as

ORDERBYD.Dname DESC,E.LnameASC,E.Fname **ASC**

INSERT,DELETE,andUPDATEStatementsinSQL

TheINSERTCommand

INSERT is used to add a single tuple to a relation. We must specify the relation name and a list of values for the tuple. The values should be listed *in the same order* in which the corresponding attributes were specified in the CREATE TABLE command.

Example:INSERTINTOEMPLOYEE**VALUES**(‘Richard’,‘K’,‘Marini’,‘653298653’,‘1962- 12-30’, ’98 Oak Forest, Katy, TX’, ‘M’, 37000, ‘653298653’, 4);

INSERTINTOEMPLOYEE (Fname,Lname,Dno,Ssn)
VALUES(‘Richard’,‘Marini’,4, ‘653298653’);

A second form of the INSERT statement allows the user to specify explicit attribute names that correspondtothevaluesprovidedintheINSERTcommand.Thevaluesmustincludeallattributes withNOTNULLspecificationandnodefaultvalue.AttributeswithNULLallowedorDEFAULT values are the ones that can be left out.

A variation of the INSERT command inserts multiple tuples into a relation in conjunction with creating the relation and loading it with the *result of a query*. For example, to create a temporarytable that has the employee lastname, project name, and hours per week for each employee working on a project, we can write the statements in U3A and U3B:

```
U3A:CREATETABLE WORKS_ON_INFO(
      Emp_name VARCHAR(15),
      Proj_name VARCHAR(15),
      Hours_per_weekDECIMAL(3,1));
U3B: INSERT INTOWORKS_ON_INFO
     (Emp_name,Proj_name,Hours_per_week)
     SELECTE.Lname,P.Pname,W.Hours
     FROMPROJECTP, WORKS_ONW,EMPLOYEE E
     WHEREP.Pnumber=W.PnoANDW.Essn=E.Ssn;
```

AtableWORKS_ON_INFOiscreatedbyU3Aandisloadedwiththejoinedinformationretrieved from the database by the query in U3B. We can now query WORKS_ON_INFO as we would any other relation;

TheDELETECommand

The DELETE command removes tuples from a relation. It includes a WHERE clause, similar to that used in an SQL query, to select the tuples to be deleted. Tuples are explicitly deleted from only one tableat atime.Thedeletion maypropagateto tuples in otherrelations if *referential triggered actions* are specified in the referential integrity constraints of the DDL.

Example:

```
DELETEFROM EMPLOYEEWHERELname='Brown';
```

Depending on the number of tuples selected by the condition in the WHERE clause, zero, one, or several tuples can be deleted by a single DELETE command. A missing WHERE clause specifies that all tuples intherelation are to bedeleted; however,thetableremains inthedatabaseas an empty table.

TheUPDATECommand

The UPDATE command is used to modify attribute values of one or more selected Tuples.An additional SET clause in the UPDATE command specifies the attributes to be modified and theirnew values. For example, to change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively, we use

```
UPDATEPROJECT SETPlocation='Bellaire', Dnum=5 WHERE Pnumber=10;
```

As in the DELETE command, a WHERE clause in the UPDATE command selects the tuples to be modified from a single relation. However, updating a primary key value may propagate tothe foreign key values of tuples in other relations if such a referential triggered action is specified in the referential integrity constraints of the DDL.

Several tuples can be modified with a single UPDATE command. An example is to give all employees in the 'Research' department a 10 percent raisein salary, as shown by the followingquery

```
UPDATEEMPLOYEE
SETSalary=Salary*1.1
WHEREDno =5;
```

EachUPDATEcommandexplicitlyreferstoasinglerelationonly.Tomodifymultiple relations,we must issue several UPDATE commands.

AdditionalFeaturesofSQL

- SQL has various techniques for specifying complex retrieval queries, including nested queries, aggregate functions, grouping, joined tables, outer joins, and recursive queries; SQL views, triggers, and assertions; and commands for schema modification.
- SQL has various techniques for writing programs invarious programminglanguages that include SQL statements to access one or more databases.
- SQL has transaction control commands. These are used to specify units of database processing for concurrency control and recovery purposes.
- SQLhaslanguage constructsforspecifyingthe *grantingandrevokingofprivileges*tousers.
- SQL has language constructs for creating triggers. These are generally referred to as **active database**techniques,sincetheyspecifyactionsthatareautomaticallytriggeredbyeventssuchas database updates.
- SQL has incorporated many features from object-oriented models to have more powerful capabilities, leading to enhanced relational systems known as **object-relational**.
- SQLandrelationaldatabasescaninteractwithnewtechnologiessuchasXML

QuestionBank

1. Definethefollowing terms as they applytothe relational model ofdata:
   i)domainii)attributeiii)n-tupleiv)relationschemav)relationstate
   vi)degreeof arelation vii)relationaldatabaseschemaviii)relationaldatabasestate.
2. Whatisthedifferencebetweenakeyandasuperkey?
3. Discussthevariousreasonsthat leadto theoccurrenceof NULLvaluesinrelations.
4. Discuss thecharacteristicsofrelations
5. Discussthevariousrestrictionsondatathatcanbespecifiedonarelationaldatabaseinthe form of constraints.
6. Suppose that each of the following Update operations isapplied directly tothe company databasestate.Discussallintegrityconstraintsviolatedbyeachoperation, ifany,andthe different ways of enforcing these constraints.
   a. Insert<‘Robert’,‘F’,‘Scott’,‘943775543’,‘1972-06-21’,‘2365NewcastleRd,Bellaire, TX’, M, 58000,
      ‘888665555’,1>into EMPLOYEE.
   b. Insert<‘ProductA’,4,‘Bellaire’,2>intoPROJECT.
   c. Insert<‘Production’,4,‘943775543’,‘2007-10-01’>intoDEPARTMENT.
   d. Insert<‘677678989’,NULL,‘40.0’>intoWORKS_ON.
   e. Insert<‘453453453’,‘John’,‘M’,‘1990-12-12’,‘spouse’>intoDEPENDENT.
   f. Deletethe WORKS_ONtupleswith Essn=‘333445555’.
   g. DeletetheEMPLOYEE tuplewith Ssn=‘987654321’.
   h. DeletethePROJECT tuplewith Pname = ‘ProductX’.
   i. ModifytheMgr_ssnandMgr_start_date oftheDEPARTMENTtuplewithDnumber =5 to ‘123456789’and‘2007-10-01’, respectively.
   j. ModifytheSuper_ssnattributeoftheEMPLOYEEtuplewithSsn=‘999887777’to ‘943775543’.
   k. ModifytheHoursattributeoftheWORKS_ONtuplewithEssn=‘999887777’and Pno = 10 to ‘5.0’.
7. Expainthefollowingunaryoperationswithsyntaxandexample
   i)SELECT ii)PROJECT iii)RENAME

8. Explainthefollowing binaryoperationswith syntaxandexample
   i)UNIONii)INTERSECTIONiii)MINUSiv)CROSSPRODUCTV)DIVISION
9. Whatisunioncompatibility?WhydotheUNION,INTERSECTION,andDIFFERENCE operations require that the relations on which they are applied be union compatible?
10. Discussthevarious types of *join* operations.
11. Discussthenotationusedinrelationalsystemstorepresentqueriesinternally.
12. Illustratewithanexample,significanceofgeneralizedprojection.
13. Illustratewithanexample,Aggregate FunctionsandGrouping
14. Illustratewithanexample,RecursiveClosureOperations
15. HowaretheOUTER JOINoperations differentfromtheINNER JOINoperations?
16. HowistheOUTERUNIONoperationdifferentfromUNION?
17. SpecifythefollowingqueriesontheCOMPANY relationaldatabaseschemausingthe relational operators
    a. Retrievethenamesofallemployeesindepartment5whoworkmorethan10hoursperweek on the ProductX project.
    b. Listthenamesofallemployeeswhohaveadependentwiththesamefirstnameasthemselves.
    c. Findthenamesofallemployeeswhoaredirectlysupervisedby‘FranklinWong’.
    d. Foreachproject,listtheprojectnameandthetotalhoursperweek(byallemployees) spent on that project.
    e. Retrievethenamesofallemployeeswhodonotworkonanyproject.
    f. Retrievethe averagesalaryofall femaleemployees.
18. DiscussthecorrespondencesbetweentheERmodelconstructsandtherelationalmodel constructs. Show how each ER model construct can bemapped to the relationalmodel
19. Discussthedatatypesthat are allowedforSQLattributes
20. WriteSQLupdatestatementstodothefollowingonthedatabaseschemashownin Figure(1)
    a. Insertanewstudent,<‘Johnson’,25,1,‘Math’>, inthedatabase.
    b. Changetheclassofstudent ‘Smith’ to2.
    c. Insertanewcourse,<‘KnowledgeEngineering’,‘CS4390’,3,‘CS’>.
    d. Deletetherecordforthestudentwhosenameis‘Smith’andwhosestudentnumberis17.

STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

STUDENT

Name	Student_number	Class	Major
Smith	17	1	CS
Brown	8	2	CS

COURSE

Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS

GRADE_REPORT

Student_number	Section_identifier	Grade
17	112	B
17	119	C
8	85	A
8	92	A
8	102	B
8	135	A

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	07	King
92	CS1310	Fall	07	Anderson
102	CS3320	Spring	08	Knuth
112	MATH2410	Fall	08	Chang
119	CS1310	Fall	08	Anderson
135	CS3380	Fall	08	Stone

Fig(2):studentschemeanddatabase state

21. Brieflydiscusshow thedifferentupdata operationson arelation deal withconstraint violations?
22. Consider the following schema for a COMPANY database:

    EMPLOYEE(Fname,Lname,Ssn,Address,Super-ssn,Salary,Dno)

    DEPARTMENT (Dname, Dnumber, Mgr-ssn, Mgr-start-date)

    DEPT-LOCATIONS (Dnumber, Dlocation)

    PROJECT(Pname,Pnumber,Plocation,Dnum)

    WORKS-ON (Ess!!, Pno, Hours)

    DEPENDENT(Essn,Dependent-name,Sex,Bdate,Relationship) Write the queries in relational algebra.

    i) Retrievethename andaddressofallemployeeswhoworkfor'Sales'department.

    ii) Findthenamesof employeeswhoworkon allthe projectscontrolledbythedepartment number 3.

    iii) Listthe namesofallemployees with twoormoredependents.

    iv) Retrievethenamesofemployeeswhohave nodependents.

Module3

Chapter1:SQL-AdvancesQueries

MoreComplexSQLRetrievalQueries

Additionalfeaturesallowuserstospecifymorecomplexretrievalsfromdatabase

ComparisonsInvolvingNULLandThree-ValuedLogic

SQLhasvariousrulesfor dealingwithNULLvalues. NULLisusedtorepresent amissingvalue,but that it usually has one of three different interpretations—value

Example

1. **Unknownvalue.**Aperson'sdateofbirthisnot known,soitisrepresentedbyNULL inthe database.
2. **Unavailableorwithheldvalue.** Apersonhasahomephonebutdoesnot wantittobe listed, so it is withheld and represented as NULL in the database.
3. **Notapplicableattribute.** AnattributeCollegeDegreewouldbeNULLforapersonwhohasno college degrees because it does not apply to thatperson.

Each individual NULL value is considered to be different from every other NULL value inthe various database records. When a NULL is involved in a comparison operation, the result is considered to be UNKNOWN (it may be TRUE or it may be FALSE). Hence, SQL uses a three-valued logic with values TRUE, FALSE, and UNKNOWN instead of the standard two-valued (Boolean) logic with values TRUE or FALSE. It is therefore necessary to define the results (or truth values) of three-valued logical expressions when the logical connectives AND, OR, and NOT are used

Table 5.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

The rows and columns represent the values of the results of comparison conditions, which would typically appear in the WHERE clause of an SQLquery.

In select-project-join queries, the general rule is that only those combinations of tuples that evaluate the logicalexpression in the WHERE clause of thequeryto TRUE are selected. Tuple combinations that evaluate to FALSE or UNKNOWN are not selected.

SQLallowsqueriesthatcheckwhetheranattributevalueisNULLusingthecomparisonoperators **IS** or**ISNOT.**

Example: Retrievethenamesofallemployeeswhodonothavesupervisors.

```
SELECTFname,Lname
FROMEMPLOYEE
WHERESuper_ssnISNULL;
```

NestedQueries,Tuples,andSet/MultisetComparisons

Some queries require that existing values in the database be fetched and then used in acomparison condition. Such queries can be conveniently formulated by using **nested queries**,which are complete select-from-where blocks within the WHERE clause of another query.That other query is called the **outer query**

Example1: List the project numbers of projects that have an employee with last name 'Smith' as manager

```
SELECTDISTINCTPnumberFROMPROJECTWHERE
 PnumberIN
(SELECTPnumberFROMPROJECT,DEPARTMENT,EMPLOYEE
 WHEREDnum=DnumberANDMgr_ssn=SsnANDLname='smith');
```

Example2: List the project numbers of projects that have an employee with last name 'Smith' as either manager or as worker.

```
SELECTDISTINCTPnumberFROMPROJECTWHERE
 PnumberIN
(SELECTPnumberFROMPROJECT,DEPARTMENT,EMPLOYEE
WHEREDnum=DnumberANDMgr_ssn=SsnANDLname='smith')
OR
PnumberIN
(SELECTPnoFROMWORKS_ON,EMPLOYEEWHEREEssn=SsnAND
 Lname='smith');
```

We makeuseofcomparisonoperator**IN**,whichcomparesavalue*v*withaset(ormultiset)of values *V* and evaluates to **TRUE** if *v* is one of the elements in *V*.

The first nested query selects theprojectnumbers of projectsthat have an employee with last name ‘Smith’ involved as manager. The second nested query selects the project numbers of projects that have an employee with last name ‘Smith’ involved as worker. In the outer query, we use the **OR** logical connective to retrievea PROJECT tupleif the PNUMBER valueof that tupleis in the resultof either nested query.

SQL allows the use of **tuples** of values in comparisons by placing them within parentheses. For example, the following query will select the Essns of all employees who work the same (project, hours) combination on some project that employee ‘John Smith’ (whose Ssn = ‘123456789’) works on

```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     (Pno, Hours) IN ( SELECT    Pno, Hours
                            FROM      WORKS_ON
                            WHERE     Essn='123456789' );
```

In this example, the IN operator compares the subtuple of values in parentheses (Pno,Hours) within each tuple in WORKS_ON with the set of type-compatible tuples produced by the nested query.

NestedQueries::ComparisonOperators

Other comparison operators can be used to compare a single value v to a set or multiset *V.* The = ANY (or = SOME) operator returns TRUE if the value *v* is equal to *some value* in the set *V* and is hence equivalent to IN. The two keywords ANY and SOME have the same effect. The keyword ALL can also be combined with each of these operators. For example, the comparison condition(*v*> ALL *V*) returns TRUE if the value *v* is greater than *all* the values in the set (or multiset) *V*. For example is the following query, which returns the names of employees whose salary is greater than the salary of all the employees in department 5:

```
SELECTLname,Fname
FROMEMPLOYEE
WHERESalary>ALL(SELECTSalary
FROMEMPLOYEE
WHEREDno=5);
```

In general, we canhaveseveral levels of nestedqueries. We canonceagain befaced withpossible ambiguity among attribute names if attributes of the same name exist—one in a relation in the FROM clause of the *outer query,* and another in a relation in the FROM clause of the *nested query.* The rule is that a reference to an *unqualified attribute* refers to the relation declared in the**innermost nested query**.

To avoid potential errorsand ambiguities, create tuple variables (aliases) for all tables referenced in SQL query

Example: Retrievethename of eachemployeewhohas adependentwiththesamefirstnameand is the same sex as the employee

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE
ASEWHEREE.SsnIN(SELECTEs
sn FROM DEPENDENT AS D
WHEREE.Fname=D.Dependent_name
ANDE.Sex=D.Sex);
```

Intheabovenestedquery, wemust qualifyE.SexbecauseitreferstotheSexattributeof EMPLOYEE from the outer query, and DEPENDENT also has an attribute called Sex.

CorrelatedNestedQueries

WheneveraconditionintheWHEREclauseofanestedqueryreferencessome attributeofa relation declared in the outer query, the two queries are said to be **correlated**.

Example:

```
SELECT E.Fname, E.Lname
FROM EMPLOYEE
ASEWHEREE.SsnIN(SELECTEs
sn FROM DEPENDENT AS D
WHEREE.Fname=D.Dependent_name
ANDE.Sex=D.Sex);
```

The nested query is evaluated once for each tuple (or combination of tuples) in the outer query. we can think of query in above example as follows: For each EMPLOYEE tuple, evaluate the nested query, which retrieves the Essn values for all DEPENDENT tuples with the same sex and name as that EMPLOYEE tuple; if the Ssn value of the EMPLOYEE tuple is in the result of the nested query, then select that EMPLOYEE tuple.

TheEXISTSandUNIQUEFunctionsinSQL

EXISTS Functions

TheEXISTSfunctioninSQLisusedtocheckwhethertheresultofacorrelatednestedqueryis *empty*(containsnotuples)ornot.TheresultofEXISTSisaBooleanvalue

- **TRUE**ifthenestedqueryresultcontainsatleastonetuple,or
- **FALSE**ifthenestedqueryresultcontainsnotuples.

For example,thequerytoretrievethenameofeachemployeewhohasadependent withthesame first name and is the same sex as the employee can be written using EXISTS functions as follows:

```
SELECTE.Fname,E.Lname
```

```
FROM EMPLOYEE AS E
WHEREEXISTS(SELECT*
FROM DEPENDENT AS D
WHEREE.Ssn=D.EssnANDE.Sex=D.Sex
ANDE.Fname=D.Dependent_name);
```

Example: Listthenamesofmanagerswhohaveatleastonedependent

```
SELECTFname,Lname
FROMEMPLOYEE
WHEREEXISTS(SELECT*
FROMDEPENDENT
WHERESsn=Essn)
AND
EXISTS(SELECT*
FROMDEPARTMENT
WHERESsn=Mgr_ssn);
```

Ingeneral,EXISTS(Q)returns**TRUE**ifthereis at least onetupleintheresult ofthenestedqueryQ, and it returns **FALSE** otherwise.

NOTEXISTSFunctions

NOTEXISTS(Q)returns**TRUE**iftherearenotuplesintheresultofnestedqueryQ,anditreturns **FALSE**otherwise.

Example:Retrievethenamesofemployeeswhohavenodependents.

```
SELECTFname,Lname
FROMEMPLOYEE
WHERENOTEXISTS(SELECT *
FROMDEPENDENT
WHERESsn=Essn);
```

For each EMPLOYEE tuple, the correlated nested query selects all DEPENDENT tuples whose Essn value matches the EMPLOYEE Ssn; if the result is empty, no dependents are related to the employee, so we select that EMPLOYEE tuple and retrieve its Fname and Lname.

Example: Retrievethenameof eachemployeewhoworksonalltheprojectscontrolled by department number 5

```
SELECTFname,Lname
```

```
FROMEMPLOYEE
WHERENOTEXISTS((SELECTPnumber
FROM PROJECT
WHERE Dnum=5)
EXCEPT(SELECTPno
FROM WORKS_ON
WHERESsn=Essn));
```

UNIQUEFunctions

UNIQUE(Q) returns TRUE if there are no duplicate tuples in the result of query Q; otherwise, it returnsFALSE.Thiscanbeusedtotestwhethertheresult ofanestedqueryisaset oramultiset.

ExplicitSetsandRenamingofAttributesinSQL

INSQLitis possibletouseanexplicitsetof valuesintheWHEREclause,ratherthananested query. Such a set is enclosed in parentheses.

Example: RetrievetheSocialSecuritynumbersof allemployeeswhoworkonprojectnumbers1,2, or 3.

```
SELECTDISTINCTEssn
FROM WORKS_ON
WHERE Pno IN (1, 2, 3);
```

InSQL,it ispossibletorename anyattributethatappears intheresultofaquerybyaddingthe qualifier AS followed by the desired new name

Example: Retrievethelastnameofeachemployeeandhisorhersupervisor

```
SELECTE.LnameASEmployee_name,
S.Lname AS Supervisor_name
FROMEMPLOYEEASE,
EMPLOYEE AS S
WHEREE.Super_ssn=S.Ssn;
```

JoinedTablesinSQLandOuterJoins

An SQL join clause combines records from two or more tables in a database. It creates a set that can be saved as a table or used as is. A JOIN is a means for combining fields from two tables by using values common to each. SQL specifies four types of JOIN

1. INNER,
2. OUTER
3. EQUIJOINand
4. NATURALJOIN

INNERJOIN

An inner join is the most common join operation used in applications and can be regarded as the default join-type. Inner join creates a new result table by combining column values of two tables (A and B) based upon the join- predicate (the condition). The result of the join can be defined as the outcome of first taking the Cartesian product (or Cross join) of all records in the tables (combining every record in table A with every record in table B)—then return all records which satisfy the join predicate

Example:SELECT*FROM*employee

INNERJOINdepartment**ON**

employee.dno=department.dnumber;

EQUIJOINandNATURALJOIN

An **EQUIJOIN** is a specific type ofcomparator-based join that uses only equality comparisonsin the join-predicate. Using other comparison operators (such as <) disqualifies a join as an equijoin.

NATURAL JOIN is a type of EQUIJOIN where the join predicate arises implicitly by comparing all columns in both tables that have the same column-names in the joined tables. The resulting joined table contains only one column for each pair of equally named columns.

```
SELECT    Fname, Lname, Address
FROM      EMPLOYEE NATURAL JOIN
          DEPARTMENT
WHERE     Dname='Research';
```

If the names of thejoin attributes are not the same in the base relations, it is possible to rename the attributes so that they match, and then to apply NATURAL JOIN. In this case, the AS construct can be used to rename a relation and all its attributes in the FROM clause.

CROSS JOIN returns the Cartesian product of rows from tables in the join. In other words, it will produce rows which combine each row from the first table with each row from the second table.

OUTER JOIN

Anouterjoindoesnotrequireeachrecordinthetwojoinedtablestohaveamatchingrecord. The joined table retains each record-even if no other matching record exists**.** Outer joins subdivide further into

- Leftouterjoins
- Rightouterjoins
- Fullouterjoins

Noimplicitjoin-notationforouterjoinsexistsinstandardSQL.

LEFT OUTER JOIN

- Every tuple in left table must appear in result
- If no matching tuple
  - Padded with NULL values for attributes of right table

Query Retieve the names of employees and their supervisors

```
Q8A:  SELECT  E.Lname AS Employee_name, S.Lname AS Supervisor_name
      FROM    EMPLOYEE AS E, EMPLOYEE AS S
      WHERE   E.Super_ssn=S.Ssn;
```

Implicit inner join

only employees who have a supervisor are included in the result; an EMPLOYEE tuple whose value for Super_ssn is NULL is excluded.

If the user requires that all employees be included, an OUTER JOIN must be used explicitly

```
Q8B:  SELECT  E.Lname AS Employee_name,
              S.Lname AS Supervisor_name
      FROM    (EMPLOYEE AS E LEFT OUTER JOIN EMPLOYEE AS S
              ON E.Super_ssn=S.Ssn);
```

- RIGHT OUTER JOIN
  - Every tuple in right table must appear in result
  - If no matching tuple
    - Padded with NULL values for the attributes of left table
- FULL OUTER JOIN
  - a full outer join combines the effect of applying both left and right outer joins.
  - Where records in the FULL OUTER JOINed tables do not match, the result set will have NULL values for every column of the table that lacks a matching row.
  - For those records that do match, a single row will be produced in the result set (containing fields populated from both tables).

- Not all SQL implementations have implemented the new syntax of joined tables.
- In some systems, a different syntax was used to specify outer joins by using the comparison operators +=, =+, and +=+ for left, right, and full outer join, respectively
- For example, this syntax is available in Oracle. To specify the left outer join in Q8B using this syntax, we could write the query Q8C as follows:

```
Q8C:    SELECT    E.Lname, S.Lname
        FROM      EMPLOYEE E, EMPLOYEE S
        WHERE     E.Super_ssn += S.Ssn;
```

MULTIWAYJOIN

It is also possible to *nest* join specifications; that is, one of the tables in a join may itself be a joined table. This allows the specification of the join of three or more tables as a single joined table, which is called a **multiway join**.

Example: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name,address, and birth date.

```
SELECTPnumber,Dnum,Lname,Address,Bdate
FROM((PROJECTJOINDEPARTMENTONDnum=Dnumber)
JOINEMPLOYEEONMgr_ssn=Ssn)
WHEREPlocation='Stafford';
```

AggregateFunctionsinSQL

Aggregate functions are used to summarize information from multiple tuples into a single-tuple summary. A number of built-in aggregate functions exist: **COUNT**, **SUM**, **MAX**, **MIN**, and **AVG.** The COUNT function returns the number of tuples or values as specified in aquery. The functions SUM, MAX, MIN, and AVG can be applied to a set or multiset of numeric values and return, respectively, the sum, maximum value, minimum value, and average (mean) of those values. These functionscan be used inthe SELECTclause or in a HAVING clause (which we introduce later). The functions MAX and MIN can also be used with attributes that have nonnumeric domains if the domain values have a total ordering among one another.

Examples

1. Find the sumofthesalaries of all employees,themaximum salary,theminimumsalary,andthe average salary.

```
SELECTSUM(Salary),MAX(Salary),MIN(Salary),AVG(Salary)
FROMEMPLOYEE;
```

2. Findthesumofthesalariesof allemployeesofthe'Research'department, aswellas the maximum salary, the minimum salary, and the average salary in thisdepartment.

```
SELECTSUM(Salary),MAX(Salary),MIN(Salary),AVG(Salary)
FROM(EMPLOYEEJOINDEPARTMENTONDno=Dnumber)
WHEREDname='Research';
```

3. Countthenumberofdistinctsalaryvaluesinthedatabase.

```
SELECTCOUNT(DISTINCTSalary)
FROMEMPLOYEE;
```

4. Toretrievethenamesofallemployeeswhohavetwoormoredependents

```
SELECTLname,Fname
FROMEMPLOYEE
WHERE(SELECTCOUNT(*)
FROMDEPENDENT
WHERESsn=Essn)>= 2;
```

Grouping:TheGROUPBYandHAVINGClauses

Grouping is used to create subgroups of tuples before summarization. For example, we may want to find the average salary of employees *in each department* or the number of employees who work *on each project*. In these cases we need to **partition** the relation into non overlapping subsets (or **groups**) of tuples. Eachgroup (partition) will consist of thetuples that have the same value of some attribute(s), called the **grouping attribute(s)**.

SQL has a **GROUP BY** clause for this purpose. The GROUP BY clause specifies the grouping attributes,whichshould*also appearintheSELECT clause,*sothat thevalue resultingfromapplying each aggregate function to a group of tuples appears along with the value of the grouping attribute(s).

Example: Foreach department,retrieve thedepartmentnumber,thenumberof employeesinthe department, and their average salary.

```
SELECTDno,COUNT(*),AVG(Salary)
FROMEMPLOYEE
GROUPBYDno;
```

Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Grouping EMPLOYEE tuples by the value of Dno

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

Result of Q24

If NULLs exist in the grouping attribute, then a **separate group** is created for all tuples with a NULL value in the grouping attribute. For example, if the EMPLOYEE table had some tuples that hadNULL for the grouping attribute Dno, there would be a separate group for those tuples in the resultof query

Example: For each project, retrieve the project number, the project name, and the number of employees who work on that project.

```
SELECTPnumber,Pname,COUNT(*)
FROMPROJECT,WORKS_ON
WHEREPnumber=Pno
GROUPBYPnumber,Pname;
```

Above query shows how we can use a join condition in conjunction with GROUP BY. In this case, the grouping and functions are applied *after* the joining of the two relations.

HAVING provides a condition on the summary information regarding the group of tuples associated with each value of the grouping attributes. Only the groups that satisfy the condition are retrieved in the result of the query.

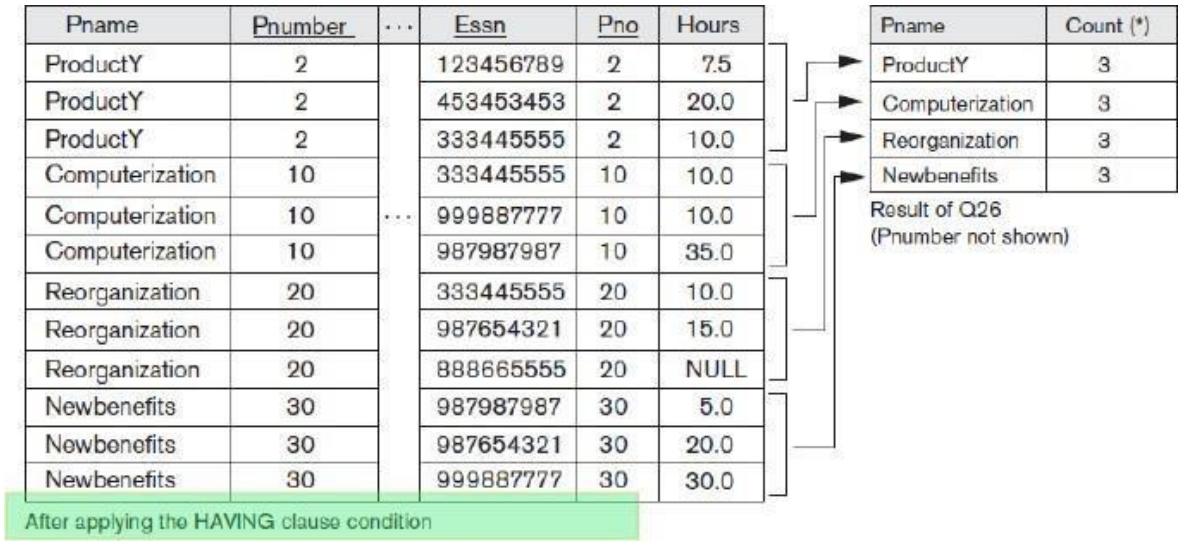
Example: For each project on which more than two employees work, retrieve the project number, the project name, and the number of employees who work on the project.

```
SELECTPnumber,Pname,COUNT(*)
FROMPROJECT,WORKS_ON
WHERE Pnumber=Pno
GROUPBYPnumber,Pname
HAVING COUNT (*) > 2;
```

Pname	Pnumber	...	Essn	Pno	Hours
ProductX	1		123456789	1	32.5
ProductX	1		453453453	1	20.0
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
ProductZ	3		666884444	3	40.0
ProductZ	3		333445555	3	10.0
Computerization	10	...	333445555	10	10.0
Computerization	10		999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

These groups are not selected by the HAVING condition of Q26.

After applying the WHERE clause but before applying HAVING



Pname	Pnumber	...	Essn	Pno	Hours
ProductY	2		123456789	2	7.5
ProductY	2		453453453	2	20.0
ProductY	2		333445555	2	10.0
Computerization	10		333445555	10	10.0
Computerization	10	...	999887777	10	10.0
Computerization	10		987987987	10	35.0
Reorganization	20		333445555	20	10.0
Reorganization	20		987654321	20	15.0
Reorganization	20		888665555	20	NULL
Newbenefits	30		987987987	30	5.0
Newbenefits	30		987654321	30	20.0
Newbenefits	30		999887777	30	30.0

After applying the HAVING clause condition

Pname	Count (*)
ProductY	3
Computerization	3
Reorganization	3
Newbenefits	3

Result of Q26
(Pnumber not shown)

Example: For each project, retrieve the project number, the project name, and the number of employees from department 5 who work on the project.

```
SELECTPnumber,Pname,COUNT(*)
FROMPROJECT,WORKS_ON,EMPLOYEE
WHEREPnumber=PnoANDSsn=EssnANDDno=5
GROUPBYPnumber,Pname;
```

Example: For each department that has morethan five employees, retrieve the department number and the number of its employees who are making more than $40,000.

```
SELECTDnumber,COUNT(*)
FROMDEPARTMENT,EMPLOYEE
WHEREDnumber=DnoANDSalary>40000AND
( SELECT Dno
FROMEMPLOYEE
GROUP BY Dno
HAVINGCOUNT(*)>5);
```

DiscussionandSummaryofSQLQueries

A retrieval query in SQL can consist of up to six clauses, but only the first two—SELECT and FROM—are mandatory.The query can span several lines, and is ended by a semicolon. Query terms are separated by spaces, and parentheses can be used to group relevant parts of a query in the standard way.The clauses are specified in the following order, with the clauses between square brackets [...] being optional:

SELECT <attribute and function list>
FROM <table list>
[**WHERE** <condition>]
[**GROUP BY** <grouping attribute(s)>]
[**HAVING** <group condition>]
[**ORDER BY** <attribute list>];

The **SELECT** clause lists the attributes or functions to be retrieved. The **FROM** clause specifies all relations (tables) needed in the query, including joined relations, but not those in nested queries.The **WHERE** clause specifies the conditions for selecting the tuples from these relations, including join conditions if needed. **GROUP** BY specifies grouping attributes, whereas **HAVING** specifies a condition on the groups being selected rather than on the individual tuples. Finally, **ORDER BY** specifies an order for displaying the result of a query.

A query is evaluated conceptually by first applying the FROM clause to identify all tables involved in the query or to materialize any joined tables followed by the WHERE clause to select and jointuples, and then by GROUP BY and HAVING. ORDER BY is applied at the end to sort the query result Each DBMS has special query optimization routines to decide on an execution plan that is efficient to execute

In general, there are numerous ways to specify the same query in SQL.This flexibility in specifying queries has advantages and disadvantages.

- The main advantage is that users can choose the technique with which they are most comfortable when specifying a query. For example, many queries may be specified with join conditions in the WHERE clause, or by using joined relations in the FROM clause, or with some form of nested queries and the IN comparison. From the programmer's and the system's pointofviewregarding queryoptimization, it is generally preferable to write aquery with as little nesting and implied ordering as possible.
- The disadvantage of having numerous ways of specifying the same query is that this may confuse the user, who may not know which technique to use to specify particular types of queries. Another problem is that it may be more efficient to execute a query specified in one way than the same query specified in an alternative way

SpecifyingConstraintsasAssertionsandActionsasTriggers

SpecifyingGeneralConstraintsasAssertionsinSQL

Assertions are used to specify additional types of constraints outside scope of built-in relational model constraints. In SQL, users can specify general constraints via declarative assertions, using the **CREATE ASSERTION** statement of the DDL.Each assertion is given a constraint name and is specified via a condition similar to the WHERE clause of an SQL query.

Generalform:

CREATEASSERTION<Name_of_assertion>**CHECK**(<cond>)

Fortheassertiontobesatisfied,theconditionspecifiedafterCHECKclausemustreturntrue.

For example, to specify the constraint that the salary of an employee must not be greater than the salary of the manager of the department that the employee works for in SQL, we can write the following assertion:

CREATEASSERTIONSALARY_CONSTRAINT
CHECK(**NOTEXISTS**(**SELECT***FROM**EMPLOYEEE,EMPLOYEEM,
DEPARTMENT D **WHERE** E.Salary>M.Salary **AND**
E.Dno=D.Dnumber**AND**D.Mgr_ssn=M.Ssn));

The constraint name SALARY_CONSTRAINTis followedby thekeywordCHECK, which is followed by a **condition** in parentheses that must hold true on every database state for the assertion to be satisfied. The constraint name canbe used later to refer totheconstraintor tomodifyor dropit. Any WHEREclauseconditioncanbeused, butmanyconstraintscanbespecifiedusingtheEXISTSand NOT EXISTS style of SQL conditions.

By including this query inside a NOT EXISTS clause, the assertion will specify that the result ofthis query must beemptysothat thecondition will always beTRUE. Thus, theassertionisviolated if the result of the query is not empty

Example:considerthebankdatabasewiththefollowingtables

- *branch (branch_name, branch_city, assets)*
- *customer (customer_name, customer_street, customer_city)*
- *account (account_number, branch_name, balance)*
- *loan (loan_number, branch_name, amount)*
- *depositor (customer_name, account_number)*
- *borrower (customer_name, loan_number)*

1. WriteanassertiontospecifytheconstraintthattheSumofloanstakenbyacustomer doesnot exceed 100,000

```
CREATEASSERTIONsumofloans
CHECK(100000>=ALL
SELECTcustomer_name,sum(amount)
FROMborrowerb,loanl
WHEREb.loan_number=l.loan_number
GROUPBYcustomer_name);
```

2. WriteanassertiontospecifytheconstraintthattheNumberofaccountsforeachcustomerina given branch is at most two

```
CREATEASSERTIONNumAccounts
CHECK(2>=ALL
SELECTcustomer_name,branch_name,count(*)
FROM    accountA,depositorD
WHEREA.account_number=D.account_number
GROUPBYcustomer_name,branch_name);
```

IntroductiontoTriggersinSQL

A trigger is a procedure that runs automatically when a certain event occurs in the DBMS. In many cases it is convenient to specify the type of action to be taken when certain events occur and when certain conditions are satisfied. The CREATE TRIGGER statement is used to implement such actions in SQL.

Generalform:

```
CREATETRIGGER<name>
BEFORE|AFTER|<events>
FOREACHROW|FOREACHSTATEMENT
WHEN(<condition>)
<action>
```

Atriggerhasthreecomponents

1. **Event:**Whenthiseventhappens,thetriggerisactivated
   - Threeeventtypes:Insert,Update,Delete
   - Twotriggeringtimes:Beforetheevent
     Aftertheevent

2. **Condition(optional):**Iftheconditionistrue,thetriggerexecutes,otherwise skipped
3. **Action:**Theactionsperformedbythetrigger

Whenthe**Event**occursand**Condition**istrue,executethe**Action**

```
CreateTriggerABC
BeforeInsertOn
Students
```

Thistriggerisactivatedwhenaninsertstatement is issued, but before the new record is inserted

```
CreateTriggerXYZ
AfterUpdateOnStudents
….
```

This trigger is activated when an updatestatementisissuedandaftertheupdateis executed

Doesthetriggerexecuteforeachupdatedordeletedrecord,oronceforthe entire statement ?. We define such granularity as follows:

```
CreateTrigger<name>
Before|After        Insert|Update|Delete

ForEachRow|ForEach Statement
….
```

Thisistheevent

Thisisthegranularity

```
CreateTriggerXYZ
AfterUpdateON<tablename>
Foreachstatement
….
```

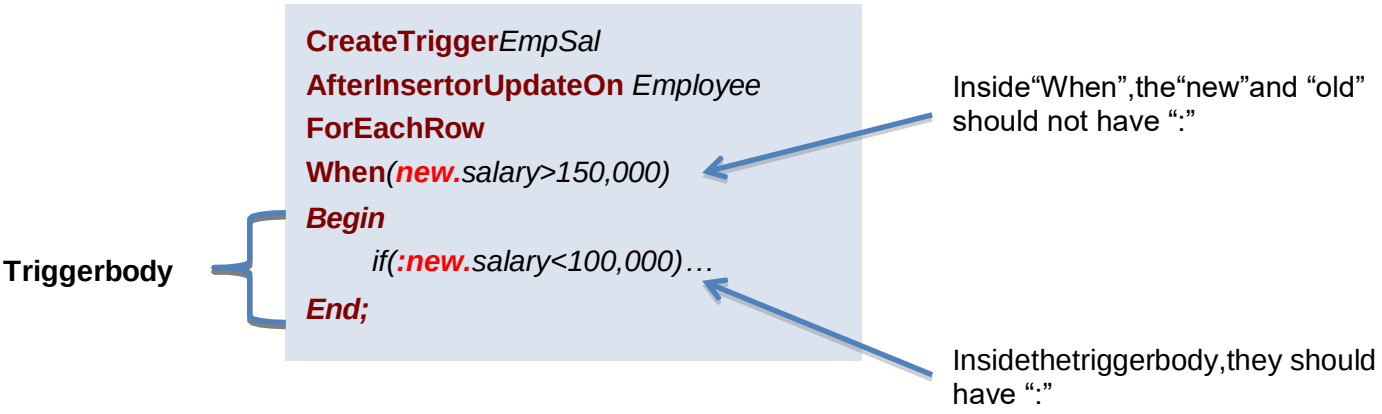
Thistriggerisactivatedonce(perUPDATE statement) after all records are updated

```
CreateTriggerXYZ
BeforeDeleteON<tablename> For each row
….
```

Thistriggerisactivatedbeforedeletingeach record

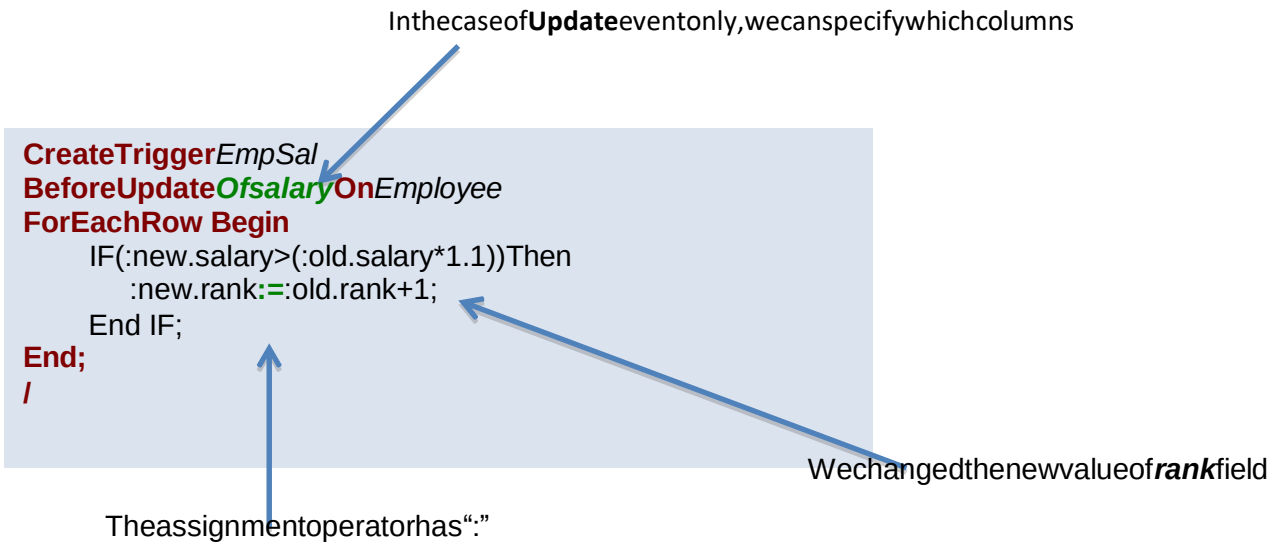
Intheaction, youmaywanttoreference:

- Thenewvaluesofinsertedorupdatedrecords***(:new)***
- Theoldvaluesofdeletedorupdatedrecords***(:old)***

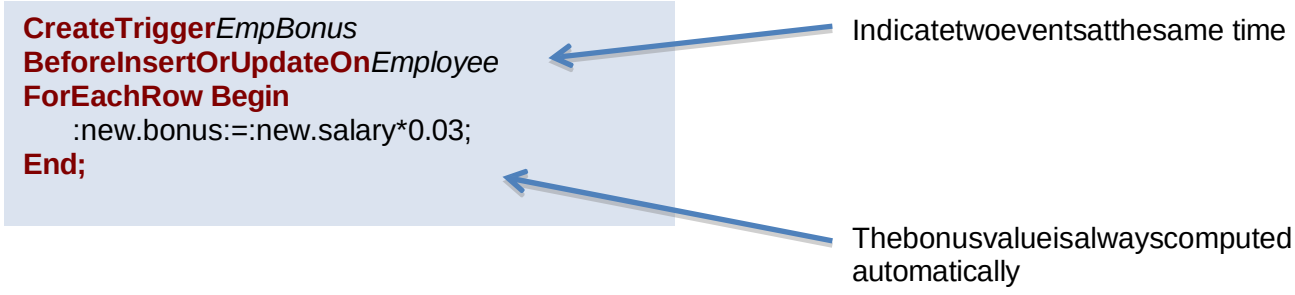


Examples:

1) Iftheemployeesalaryincreasedbymorethan10%,thenincrementtherankfieldby1.



2) KeepthebonusattributeinEmployeetablealways3%ofthesalaryattribute



3. Supposewewanttocheckwheneveranemployee'ssalaryisgreaterthanthesalaryofhisor her direct supervisor in the COMPANY database
   - Severaleventscantriggerthisrule:
     - insertinganewemployeerecord
     - changinganemployee'ssalaryor
     - changinganemployee'ssupervisor

   - Supposethattheactiontotakewouldbetocallanexternalstoredprocedure SALARY_VIOLATION which will notify the supervisor

```
CREATETRIGGERSALARY_VIOLATION
BEFOREINSERTORUPDATEOFSALARY,SUPERVISOR_SSN
ON EMPLOYEE
FOREACHROW
WHEN(NEW.SALARY>(SELECTSALARYFROMEMPLOYEE
WHERESSN=NEW.SUPERVISOR_SSN))
INFORM_SUPERVISOR(NEW.Supervisor_ssn,NEW.Ssn);
```

   - ThetriggerisgiventhenameSALARY_VIOLATION,whichcanbeusedtoremoveor deactivate the trigger later
   - Inthisexampletheeventsare:insertinganewemployeerecord,changinganemployee's salary, or changing an employee's supervisor
   - TheactionistoexecutethestoredprocedureINFORM_SUPERVISOR

Triggerscanbeusedinvariousapplications,suchasmaintainingdatabaseconsistency,monitoring database updates.

Assertionsvs.Triggers

- Assertionsdonotmodifythedata,theyonlycheck certainconditions. Triggersaremore powerful because the can check conditions and also modify thedata
- Assertionsarenotlinkedtospecifictablesinthedatabaseandnotlinkedtospecific events. Triggers are linked to specific tables and specific events
- Allassertionscanbeimplementedastriggers(oneormore). Not alltriggerscanbe implemented as assertions

Example:Triggervs.Assertion

Allnewcustomersopeninganaccountmusthaveopeningbalance>=$100.However,once the account is opened their balance can fall below that amount.

Weneedtriggers,assertionscannotbeused

TriggerEvent:BeforeInsert

```
Create Trigger OpeningBal
BeforeInsertOnCustomer
For Each Row
Begin
   IF (:new.balance is null or :new.balance < 100) Then
   RAISE_APPLICATION_ERROR(-20004,'Balanceshouldbe>=$100');
   End IF;
End;
```

Views(VirtualTables)inSQL

ConceptofaViewinSQL

A view in SQL terminology is a single table that is derived from other tables. other tables can be base tables or previously defined views. A view does not necessarily exist in physical form; it is considered to be a virtual table, in contrast to base tables, whose tuples are always physicallystored in the database. Thislimitsthe possibleupdateoperationsthatcanbe applied to views,butit does not provide anylimitations onquerying aview. We can think of aview as a way of specifying a table that we need to reference frequently, even though it may not existphysically.

For example, referring to the COMPANY database, we may frequently issue queries that retrievethe employee name and the project names that the employee works on. Rather than having to specify the join of the three tables EMPLOYEE,WORKS_ON, and PROJECT every time we issue this query, we can define a view that is specified as the result of these joins. Then we can issue queries on the view, which are specified as single table retrievals rather than as retrievals involving two joinson threetables. We call the EMPLOYEE,WORKS_ON, and PROJECT tables the **defining tables** of the view.

SpecificationofViewsinSQL

In SQL, the command to specify a view is **CREATE VIEW.** The view is given a (virtual) table name (or view name), a list of attribute names, and a query to specify the contents of the view. If none of the view attributes results from applying functions or arithmetic operations, we do not have tospecify new attribute names for the view, since they would be the same as the names of the attributes of the defining tables in the default case.

Example1:

```
CREATEVIEWWORKS_ON1
 ASSELECTFname,Lname,Pname,Hours
 FROMEMPLOYEE,PROJECT,WORKS_ON
 WHERESsn=EssnANDPno=Pnumber;
```

Example2:

```
CREATEVIEW     DEPT_INFO(Dept_name,No_of_emps,Total_sal)
ASSELECTDname,COUNT(*),SUM(Salary)
FROMDEPARTMENT,EMPLOYEE
WHEREDnumber=Dno
GROUPBYDname;
```

In example 1, we did not specify any new attribute names for the view WORKS_ON1. In this case,WORKS_ON1 *inherits* the names of the view attributes from the defining tables EMPLOYEE, PROJECT, and WORKS_ON.

Example 2 explicitly specifies new attribute names for the view DEPT_INFO, using a one-to-one correspondence between the attributes specified in the CREATE VIEW clause and those specifiedin the SELECT clause of the query that defines the view.

WORKS_ON1

Fname	Lname	Pname	Hours

DEPT_INFO

Dept_name	No_of_emps	Total_sal

We cannowspecify SQLqueries onaview—orvirtualtable—inthesameway wespecifyqueries involving base tables.

Forexample,toretrievethelastnameandfirst nameof allemployeeswhoworkonthe‘ProductX’ project, we can utilize the WORKS_ON1 view and specify the query as :

```
SELECT Fname,Lname
FROM WORKS_ON1
WHERE Pname='ProductX';
```

The same query would require the specification of two joins if specified on the base relationsdirectly. one of the main advantages of a view is to simplify the specification of certain queries. Views are also used as a security and authorizationmechanism.

A view is supposed to be always up-to-date; if we modify the tuples in the base tables on which the view is defined, the view must automatically reflect these changes. Hence, the view is not realizedor materialized at the time of view definition but rather at the time when we specify a query on the view. It is the responsibility of the DBMS and not the user to make sure that the view is kept up-to- date.
If we do not need a view any more, we can use the **DROP VIEW** command to dispose of it. For example : **DROP VIEW** WORKS_ON1;

ViewImplementation,ViewUpdateandInline Views

Theproblemofefficientlyimplementingaviewforqueryingiscomplex.Twomainapproacheshave been suggested.

- Onestrategy,called**querymodification**,involvesmodifyingortransformingtheviewquery (submittedbytheuser)intoaqueryontheunderlyingbasetables.Forexample,thequery

```
SELECT Fname,Lname
FROM WORKS_ON1
WHERE Pname='ProductX';
```

wouldbeautomaticallymodifiedtothefollowingquerybytheDBMS:

```
SELECT Fname,Lname
FROM EMPLOYEE,PROJECT,WORKS_ON
WHERE Ssn=Essn AND Pno=Pnumber
AND Pname='ProductX';
```

Thedisadvantageofthisapproachisthatit isinefficientforviews definedviacomplexqueriesthat are time-consuming to execute, especially if multiple queries are going to be applied to the same view within a short period of time.

- The second strategy, called **view materialization**, involves physically creating a temporary view table when the view is first queried and keeping that table on the assumption that other querieson the view will follow. In this case, an efficient strategy for automatically updating the view table when the base tables are updated must be developed in order to keep the view up-to-date.

Techniques using the concept of **incremental update** have been developed for this purpose, where the DBMS can determine what new tuples must be inserted, deleted, or modified in a materialized view table when a database update is applied to one of the defining base tables.

The viewis generallykeptasamaterialized(physically stored)table aslong asit isbeingqueried.If the view is not queried for a certain period of time, the system may then automatically remove the physical table and recompute it from scratch when future queries reference the view.

Updating of views is complicated and can be ambiguous. In general, an update on a view definedon a single table without any aggregate functions can be mapped to an update on the underlying base table under certainconditions. For a viewinvolving joins, an updateoperation may be mapped to update operations on the underlying base relations in multiple ways. Hence, it is often notpossible for the DBMS to determine which of the updates is intended.

To illustrate potential problems with updating a view defined on multiple tables, consider the WORKS_ON1 view, and suppose that we issue the command to update the PNAME attribute of 'John Smith' from 'ProductX' to 'ProductY'. This view update is shown in UV1:

```
UV1: UPDATEWORKS_ON1
     SET Pname='ProductY'
     WHERELname='Smith'ANDFname='John'
     AND Pname='ProductX';
```

This query can be mapped into several updates on the base relations to give the desired update effect on the view. In addition, some of these updates will create additional side effects that affect the result of other queries.

For example, here are two possible updates, (a) and (b), on the base relations corresponding to the view update operation in UV1:

```
(a) :UPDATEWORKS_ON
    SETPno=(SELECTPnumber
    FROMPROJECT
    WHERE Pname='ProductY' )
    WHEREEssnIN(SELECTSsn
    FROM EMPLOYEE
    WHERELname='Smith'ANDFname='John')
    AND
    Pno=(SELECTPnumber
    FROMPROJECT
    WHEREPname='ProductX');
```

(b) :UPDATEPROJECT**SET**Pname='ProductY'
WHEREPname='ProductX';

Update (a) relates 'John Smith' to the 'ProductY' PROJECT tuple instead of the 'ProductX' PROJECT tuple and is the most likely desired update. However, (b) would also give the desired update effect on the view, but it accomplishes this by changing the name of the 'ProductX' tuple in the PROJECT relation to 'ProductY'.

It is quite unlikely that the user who specified the view update UV1 wants the update to be interpreted as in (b), since it also has the side effect of changing all the view tuples with Pname = 'ProductX'.

Some view updatesmaynot makemuchsense;for example, modifyingthe Total_sal attributeofthe DEPT_INFO viewdoesnot makesensebecauseTotal_sal isdefined tobe thesum of theindividual employee salaries. This request is shown as UV2:

UV2: UPDATEDEPT_INFO
SETTotal_sal=100000
WHEREDname='Research';

Alargenumberofupdatesontheunderlyingbaserelationscansatisfythisviewupdate.

Generally, a view update is feasible when only one possible update on the base relations can accomplish the desired update effect on the view. Whenever an update on the view can be mapped to more than one update on the underlying base relations, we must have a certain procedure for choosing one of the possible updates as the most likely one.

Insummary,wecanmakethefollowingobservations:

- Aviewwithasingledefiningtableisupdatable iftheviewattributescontaintheprimarykeyofthe base relation, as well as all attributes with the NOT NULL constraint *that do not have* default values specified.
- Viewsdefinedonmultipletablesusingjoinsaregenerallynotupdatable.
- Viewsdefinedusinggroupingandaggregatefunctionsarenotupdatable.

In SQL, the clause **WITH CHECK OPTION** must be added at the end of the view definition if a view *is to be updated*. This allows the system to check for view updatability and to plan an execution strategy for view updates. It is also possible to define a view table in the **FROM clause** of an SQL query. This is known as an **in-line view**. In this case, the view is defined within the query itself.

SchemaChangeStatementsinSQL

Schema evolution commands available in SQL can be used to alter a schema by adding or dropping tables, attributes, constraints, and other schema elements. This can be done while the database is operational and does not require recompilation of the database schema.

The DROPCommand

The DROP command can be used to drop named schema elements, such as tables, domains, or constraints. One can also drop a schema. For example, if a whole schema is no longer needed, the DROP SCHEMA command can be used.

There are two drop behavior options: **CASCADE** and **RESTRICT**. For example, to remove the COMPANY databaseschema andall itstables, domains,and otherelements, theCASCADE option is used as follows:

DROPSCHEMACOMPANY**CASCADE**;

If the **RESTRICT** option is chosen in place of **CASCADE**, the schema is dropped only if it has no elements in it; otherwise, the DROP command will not be executed. To use the RESTRICT option, the user must first individually drop each element in the schema, then drop the schema itself.

If a base relation within a schema is no longer needed, the relation and its definition can be deleted by using the DROP TABLE command. For example, if we no longer wish to keep track of dependents of employees in the COMPANY database, , we can get rid of the DEPENDENT relation by issuing the following command:

DROPTABLEDEPENDENT**CASCADE**;

If the RESTRICT option is chosen instead of CASCADE, a table is dropped only if it is not referencedin anyconstraints(forexample, by foreign key definitionsinanother relation)orviews or by any other elements. With the CASCADE option, all such constraints, views, and otherelements that reference the table being dropped are also dropped automatically from the schema, along with the table itself.

The DROP TABLE command not only deletes all the records in the table if successful, but also removes the table definition from the catalog. If it is desired to delete only the records but to leave the table definition for future use, then the DELETE command should be used instead of DROP TABLE.

The DROP command can also be used to drop other types of named schema elements, such as constraints or domains.

TheALTERCommand

The definition of a base table or of other named schema elements can be changed by using the ALTER command. For base tables, the possible **alter table actions** include adding or dropping a column (attribute), changing a column definition, and adding or dropping table constraints.

For example, to add an attribute for keeping track of jobs of employees to the EMPLOYEE base relation in the COMPANY schema , we can use the command:

ALTERTABLECOMPANY.EMPLOYEE**ADDCOLUMN**JobVARCHAR(12);

We must still enter a value for the new attribute Job for each individual EMPLOYEE tuple. This can be done either by specifying a default clause or by using the UPDATE command individually oneach tuple. Ifno default clause is specified, the new attribute will have NULLs in all the tuples of the relation immediatelyafterthecommandisexecuted; hence,theNOTNULL constraintisnot allowed in this case.

To dropacolumn, wemustchooseeither **CASCADE** or **RESTRICT** for drop behavior. If **CASCADE** is chosen, all constraints and views that reference the column are dropped automatically from the schema, alongwiththecolumn. If **RESTRICT** ischosen, thecommandissuccessfulonlyif noviews or constraints (or other schema elements) reference the column.

For example, the following command removes the attribute Address from the EMPLOYEE base table:

ALTERTABLECOMPANY.EMPLOYEE**DROPCOLUMN**Address**CASCADE**;

Itisalsopossibletoalteracolumndefinitionbydroppinganexistingdefault clauseorbydefininga new default clause. The following examples illustrate this clause:

ALTERTABLECOMPANY.DEPARTMENT**ALTERCOLUMN**Mgr_ssn**DROPDEFAULT**;

ALTER TABLE COMPANY.DEPARTMENT **ALTER COLUMN** Mgr_ssn **SET DEFAULT**

'333445555';

AlterTable-Alter/Modify Column

Tochangethedatatypeofacolumninatable,usethefollowing syntax:

ALTERTABLEtable_name

MODIFYcolumn_namedatatype;

For examplewecanchangethedatatypeofthecolumn named"DateOfBirth"fromdatetoyear in the "Persons" table using the following SQL statement:

ALTERTABLEPersons

ALTERCOLUMNDateOfBirthyear;

Noticethatthe"DateOfBirth"columnis nowoftypeyearandisgoingtoholda yearinatwo-or four-digit format.

Chapter2:DatabaseApplicationDevelopment

Introduction

We often encounter a situations in which we need the greater flexibility of a general-purpose programming language in addition to the data manipulation facilities provided by SQL.For example, we may want to integrate a database applications with GUI or we may want to integrate with other existing applications.

AccessingDatabasesfromapplications

SQL commands can be executed from within a program in a host language such as C or Java. A language to which SQL queries are embedded are called Host language.

EmbeddedSQL

The use of SQL commands within a host language is called **Embedded SQL**. Conceptually, embedding SQL commands ina host language program is straight forward. SQL statements can be used wherever a statement in the host language is allowed. SQL statements must be clearlymarked so that a preprocessor can deal with them before invoking the compiler for the host language. Any host language variable used to pass arguments into an SQL command must be declared in SQL.
Therearetwocomplications:

1. DatatypesrecognizedbySQLmaynotberecognizedbythehostlanguageandviceversa
   - This mismatch is addressed by casting data values appropriately before passing them to or from SQL commands.
2. SQLisset-oriented
   -Addressedusingcursors

DeclaringVariablesandExceptions

SQL statements can refer to variables defined in the host program. Such host language variables must be prefixed by a colon(:) in SQL statements and be declared between the commands

EXECSQLBEGINDECLARESECTIONandEXECSQLENDDECLARESECTION

The declarations are similar to C, are separated by semicolons. For example, we can declare variables c_sname, c_sid, c_rating, and c_age (with the initial c used as a naming convention to emphasize that these are host language variables) as follows:

EXECSQLBEGINDECLARESECTION

charc_sname[20];
long c_sid;
shortc_rating;
float c_age;

EXECSQLENDDECLARESECTION

The first question that arises is which SQL types correspond to the various C types, since we have just declared a collection of C variables whose values are intended to be read (and possibly set) in anSQLrun-timeenvironmentwhenanSQLstatementthatreferstothem is executed.TheSQL-92 standard defines such a correspondence between the host language types and SQL types for a numberofhostlanguages.In our example,c_sname hasthetype CHARACTER(20) whenreferred to in an SQL statement, c_sid has the type INTEGER, crating has the type SMALLINT, and c_age has the type REAL.

We also need some way for SQL to report what went wrong if an error condition arises when executing an SQL statement. The SQL-92 standard recognizes two special variables for reporting errors, **SQLCODE** and **SQLSTATE**.

- **SQLCODE** is the older of the two and is defined to return some negative value when an error condition arises, without specifying further just what error a particular negative integer denotes.
- **SQLSTATE**, introduced in the SQL-92 standard for the first time, associates predefined values with several common error conditions, thereby introducing some uniformity to how errors are reported.

Oneofthesetwovariablesmust bedeclared. TheappropriateCtypeforSQLCODEis longandthe appropriate C type for SQLSTATE is char [6] , that is, a character string five characters long.

EmbeddingSQL statements

All SQL statements embedded within a host program must be clearly marked with the details dependent on the host language. In C, SQL statements must be prefixed by **EXEC SQL.** An SQL statement can essentially appear in any place in the host language program where a host language statement can appear.

Example: The following embedded SQL statement inserts a row, whose column values are based on the values of the host language variables contained in it, into the sailors relation

EXECSQLINSERTINTOsailorsVALUES(:c_sname,:c_sid,:c_rating,:c_age);

The **SQLSTATE** variable should be checked for errors and exceptions after each Embedded SQL statement.SQL provides the **WHENEVER** command to simplify this task:

EXECSQLWHENEVER[SQLERROR|NOTFOUND][CONTINUE|GOTOstmt]

If **SQLERROR** is specified and the value of SQLSTATE indicates an exception, control istransferred to stmt, which is presumably responsible for error and exception handling. Control isalso transferred to stmt if NOT FOUND is specified and the value of SQLSTATE is 02000, which denotes NO DATA.

Cursors

A major problem in embedding SQL statements in a host language like C is that an impedance mismatch occurs because SQL operates on sets of records, whereas languages like C do not cleanly support a set-of-records abstraction. The solution is to essentially provide a mechanism that allows us to retrieve rows one at a time from a relation- this mechanism is called a **cursor**

WecandeclareacursoronanyrelationoronanySQLquery.Onceacursorisdeclared,wecan

- **open**it(positionsthecursorjustbeforethefirstrow)
- **Fetch**thenextrow
- **Move**thecursor(tothenextrow,totherowafterthenext n,tothefirstrowor previousrow etc by specifying additional parameters for the fetchcommand)
- **Close**thecursor

Cursor allows ustoretrievetherowsinatablebypositioningthecursor ataparticularrowand reading its contents.

BasicCursorDefinitionandUsage

Cursorsenableustoexamine, inthehostlanguageprogram, acollectionofrows computedbyan Embedded SQL statement:

- We usuallyneedtoopenacursoriftheembeddedstatementisaSELECT. wecanavoid opening a cursor if the answer contains a single row
- INSERT,DELETEandUPDATEstatementsrequirenocursor.somevariantsofDELETE and UPDATE use a cursor.

Examples:

i) Findthenameandageof asailor,specifiedbyassigningavaluetothehost variablec_sid, declared earlier

```
EXECSQLSELECTs.sname,s.age
INTO:c_sname,:c_age
FROM Sailaor s
WHERE s.sid=:c.sid;
```

The **INTO** clause allows us assign the columns of the single answer row to the host variable c_sname and c_age. Therefore, we do not need a cursor to embed this query in a host language program.

ii) Computethenameandagesofallsailors witharatinggreaterthanthecurrent valueofthehost variable c_minrating

SELECTs.sname,s.age

FROMsailorss**WHERE**s.rating>:c_minrating;

The query returns a collection of rows. The INTO clause is inadequate. The solution is to use a cursor:

DECLAREsinfo**CURSORFOR**

SELECTs.sname,s.age

FROMsailorss

WHEREs.rating>:c_minrating;

ThiscodecanbeincludedinaCprogramandonceitisexecuted,thecursorsinfoisdefined. We can open the cursor by using the syntax:

OPEN sinfo;

A cursor canbethought of as'pointing' toarow in thecollection of answers tothe queryassociated with it.When the cursor is opened, it is positioned just before the first row.

WecanusetheFETCHcommandtoreadthefirstrowofcursorsinfointohostlanguagevariables:

FETCHsinfoINTO:c_sname,:c_age;

When the FETCH statement is executed, the cursor is positioned to point at the next row and the column values in the row are copied into the corresponding host variables. By repeatedly executing this FETCH statement, we can read all the rows computed by the query, one row at time.

Whenwearedonewithacursor,wecancloseit:

CLOSEsinfo;

iii) Toretrievethename,addressandsalaryofanemployeespecifiedbythevariablessn

```
   //Program Segment E1:
0) loop = 1 ;
1) while (loop) {
2)   prompt("Enter a Social Security Number: ", ssn) ;
3)   EXEC SQL
4)     SELECT Fname, Minit, Lname, Address, Salary
5)     INTO :fname, :minit, :lname, :address, :salary
6)     FROM EMPLOYEE WHERE Ssn = :ssn ;
7)   if (SQLCODE = = 0) printf(fname, minit, lname, address, salary)
8)     else printf("Social Security Number does not exist: ", ssn) ;
9)   prompt("More Social Security Numbers (enter 1 for Yes, 0 for No): ", loop) ;
10)  }
```

PropertiesofCursors

Thegeneralformofacursordeclarationis:

DECLARE*cursorname***[INSENSITIVE][SCROLL]CURSOR [WITH HOLD]**
FOR*somequery*
[ORDERBYorder-item-list**]**
[FORREADONLYIFORUPDATE]

A cursor can be declared to be a read-only cursor (FOR READ ONLY) or updatable cursor (FOR UPDATE).If it is updatable, simple variants of the UPDATE and DELETE commands allow us to update or delete the row on which the cursor is positioned. For example, if sinfo is an updatable cursor and open, we can execute the following statement:

UPDATESailorsS
SETS.rating=S.rating-1
WHERECURRENTofsinfo;

A cursor is updatable bydefault unless it is ascrollable or insensitive cursor in which case it isread-only by default.

If the keyword **SCROLL** is specified, the cursor is scrollable, which means that variants of the FETCH command can be used to position the cursor in very flexible ways; otherwise, only the basic FETCH command, which retrieves the next row, is allowed

If the keyword **INSENSITIVE** is specified, the cursor behaves as if it is ranging over a private copyof the collection of answer rows. Otherwise, and by default, other actions of some transaction could modify these rows, creating unpredictable behavior.

A holdable cursor is specified using the WITH **HOLD** clause, and is not closed when the transaction is committed.

Optional**ORDERBY**clausecanbeusedtospecify asort order.Theorder-item-list isalistoforder- items. An order-item is a column name, optionally followed by one of the keywords ASC or DESC Every column mentioned in the **ORDER BY** clause must also appear in the select-list of the query associated with the cursor; otherwise it is not clear what columns we should sort on

ORDERBYminage**ASC**,rating**DESC**

Theanswerissortedfirst inascendingorderbyminage, andifseveralrowshavethesameminage value, these rows are sorted further in descending order by rating

Rating	minage
8	25.5
3	25.5
7	35.0

DynamicSQL

Dynamic SQL Allow construction of SQL statements on-the-fly. Consider an application such as a spreadsheet or a graphical front-end that needs to access data from a DBMS. Such an application must accept commands from a user and, based on what the user needs, generate appropriate SQL statementstoretrievethenecessarydata.Insuchsituations,wemaynotbeabletopredictin advance just what SQL statements need to be executed. SQL provides some facilities to deal with such situations; these are referred to as **Dynamic SQL**.

Example:

```
charc_sqlstring[]={"DELETEFROMSailorsWHERErating>5"};
EXECSQLPREPAREreadytogoFROM:csqlstring;
EXECSQLEXECUTEreadytogo;
```

- Thefirststatement declarestheCvariable *c_sqlstring*andinitializesitsvaluetothestring representation of an SQL command
- ThesecondstatementresultsinthisstringbeingparsedandcompiledasanSQLcommand, with the resulting executable bound to the SQL variable *readytogo*
- Thethirdstatementexecutesthecommand

AnIntroductiontoJDBC

Embedded SQL enables the integration of SQL with a general-purpose programming language. A DBMS-specific preprocessor transforms the Embedded SQL statements into function calls in the host language. The details of this translation vary across DBMSs, and therefore even though the sourcecodecanbecompiledtowork withdifferent DBMSs, thefinalexecutableworks onlywithone specific DBMS.

ODBCandJDBC,shortforOpenDataBaseConnectivityandJavaDataBaseConnectivity, also enable the integration of SQL with a general-purpose programming language.

- IncontrasttoEmbeddedSQL,ODBCandJDBCallowasingleexecutabletoaccess different DBMSs *Without recompilation.*

- While Embedded SQL is DBMS-independent only at the source code level, applications usingODBCorJDBCareDBMS-independentatthesourcecodelevelandatthelevelof the executable
- In addition, usingODBCor JDBC, an applicationcan accessnot just oneDBMS but several different ones simultaneously
- ODBC and JDBC achieve portability at the level of the executable by introducing an extra level of indirection
- AlldirectinteractionwithaspecificDBMShappensthroughaDBMS-specificdriver.

A driver is a software program that translates the ODBC or JDBC calls into DBMS-specific calls. Drivers are loaded dynamically on demand since the DBMSs the application is going to access are known only at run-time. Available drivers are registered with a driver manager a driver does not necessarilyneed to interact with a DBMS that understands SQL. It is sufficient that thedriver translates the SQL commands from the application into equivalent commands that the DBMS understands.

An application that interacts with a data source through ODBC or JDBC selects a data source, dynamically loads the corresponding driver, and establishes a connection with the data source. There is no limit on the number of open connections. An application can have several open connections to different data sources. Each connection has transaction semantics; that is, changes from one connection are visible to other connections only after the connection has committed its changes. While a connection is open, transactions are executed by submittingSQL statements, retrieving results, processing errors, and finally committing or rolling back. The application disconnects from the data source to terminate theinteraction.

Architecture

ThearchitectureofJDBChasfourmaincomponents:

- Application
- Drivermanager
- Drivers
- Datasources

Application

- initiatesandterminatestheconnectionwithadatasource
- setstransactionboundaries,submitsSQLstatementsandretrievestheresults

Drivermanager

- LoadJDBCdriversandpassJDBCfunctioncallsfromtheapplicationtothecorrectdriver
- HandlesJDBCinitializationandinformationcallsfromtheapplicationsandcanlogall function calls
- Performssomerudimentaryerrorchecking

Drivers

- Establishestheconnectionwiththedatasource
- Submitsrequestsandreturnsrequestresults
- Translatesdata, errorformats, anderrorcodesfromaformthatisspecifictothedatasource into the JDBC standard

Datasources

- Processescommandsfromthedriverandreturns theresults

DriversinJDBCareclassifiedintofourtypesdependingonthearchitecturalrelationshipbetween the application and the data source:

TypeIBridges:

- ThistypeofdrivertranslatesJDBCfunctioncallsintofunctioncallsofanotherAPIthatisnot native to the DBMS.
- AnexampleisaJDBC-ODBCbridge; anapplicationcanuseJDBCcallstoaccessan ODBC compliant data source. The application loads only one driver, thebridge.
- Advantage:
  - it iseasytopiggybacktheapplicationontoan existinginstallation,andnonew drivers have to be installed.
- Drawbacks:
  - Theincreasednumberoflayersbetweendatasourceandapplicationaffects performance
  - theuserislimitedtothefunctionalitythattheODBCdriversupports.

TypeII DirectTranslationtotheNativeAPIviaNon-Java Driver:

- ThistypeofdrivertranslatesJDBCfunctioncallsdirectly intomethodinvocationsoftheAPI of one specific data source.
- Thedriverisusually,writtenusingacombinationofC++andJava; itisdynamically linked and specific to the data source.
- Advantage
  - Thisarchitectureperformssignificantlybetterthanа JDBC-ODBCbridge.
- Disadvantage
  - ThedatabasedriverthatimplementstheAPIneedstobeinstalledoneach computer that runs the application.

TypeIII~~NetworkBridges:

- Thedrivertalksover anetworktoamiddleware serverthattranslatestheJDBCrequests into DBMS-specific method invocations.
- Inthiscase,thedriverontheclientsiteisnotDBMS-specific.
- TheJDBCdriverloadedbytheapplicationcanbequitesmall, astheonlyfunctionalityit needs to implement is sending of SQL statements to the middlewareserver.
- ThemiddlewareservercanthenuseaTypeIIJDBCdrivertoconnecttothedatasource.

TypeIV-DirectTranslationtotheNativeAPIviaJavaDriver:

- InsteadofcallingtheDBMSAPIdirectly,thedrivercommunicateswiththe DBMS through Java sockets
- Inthiscase,thedriverontheclient sideiswritten inJava,butitisDBMS-specific.It translates JDBC calls into the native API of the database system.
- Thissolutiondoesnotrequireanintermediatelayer,andsincetheimplementationisall Java, its performance is usually quite good.

JDBCCLASSESANDINTERFACES

JDBC is a collection of Java classes and interfaces that enables database access from programs written in the Java language. It contains methods for connecting to a remote data source, executing SQL statements, examining sets of results from SQL statements, transaction management, and exception handling.

The classes and interfaces are part of the java.sql package. JDBC 2.0 also includes the javax.sql package, the JDBC Optional Package. The package javax.sql adds, among other things, the capability of connection pooling and the Row-Set interface.

JDBCDriverManagement

In JDBC, data source drivers are managed by the Drivermanager class, which maintains a list of all currentlyloadeddrivers.TheDrivermanagerclasshasmethodsregisterDriver,deregisterDriver,and getDrivers to enable dynamic addition and deletion of drivers.

The first stepin connecting to a data source is toload the corresponding JDBC driver. Thefollowing Java example code explicitly loads a JDBC driver:

Class.forName("oracle/jdbc.driver.OracleDriver");

There are two other ways ofregistering a driver. We can include the driver with -Djdbc. drivers=oracle/jdbc. driver at thecommandline when westart theJava application. Alternatively, we can explicitly instantiate a driver, but this method is usedonly rarely, as the name of the driver hasto bespecified inthe application code, andthusthe application becomessensitivetochangesatthe driver level.

Afterregisteringthedriver,weconnecttothedatasource.

Connections

Asessionwithadatasourceisstartedthroughcreationofa Connectionobject; Connectionsare specified through a JDBC URL, a URL that uses the jdbc protocol. Such a URL has the form

jdbc:<subprotocol>:<otherParameters>

```
Stringuri=..jdbc:oracle:www.bookstore.com:3083..
Connection connection;
try
 {
        Connection connection =
        DriverManager.getConnection(url,userId,password);
    }
catch(SQLExceptionexcpt)
{
        System.out.println(excpt.getMessageO);
        return;
}
```

Programcode:EstablishingaConnectionwithJDBC

InJDBC,connectionscanhavedifferentproperties. Forexample,aconnectioncanspecifythe granularityoftransactions.If**autocommit**issetforaconnection,theneachSQLstatement is

consideredtobeitsowntransaction.If**autocommit**isoff,thenaseriesof statementsthatcompose a transaction can be committed using the commit() method of the Connection class, or aborted using the rollback() method. The Connection class has methods to set theautocommit mode (Connection. setAutoCommit) and to retrieve the current autocommit mode (getAutoCommit). The following methods are part of the Connection interface and permit setting and getting other properties:

- public int getTransactionIsolation() throws SQLException and publicvoidsetTransactionIsolation(int1)throwsSQLException.
  - These two functionsget andsetthecurrentlevelofisolationfortransactions handled in the current connection. All five SQL levels of isolation are possible, and argument *I* can be set as follows:
    - TRANSACTION_NONE
    - TRANSACTION_READ_UNCOMMITTED
    - TRANSACTION_READ_COMMITTED
    - TRANSACTION_REPEATABLE_READ
    - TRANSACTION_SERIALIZABLE
- publicbooleangetReadOnlyOthrowsSQLExceptionand publicvoidsetReadOnly(booleanreadOnly)throwsSQLException.
  - Thesetwofunctionsallowtheusertospecifywhetherthetransactionsexecuteclthrough this connection are rcad only.
- publicbooleanisClosed()throwsSQLException.
  - Checkswhetherthecurrentconnectionhasalreadybeenclosed.
- setAutoCommitandgetAutoCommit.

In case an application establishes many different connections from different parties (such as a Web server), connections are often **pooled** to avoid this overhead. A **connection pool** is a set of established connections to a data source. Whenever a new connection is needed, one of the connections from the pool is used, instead of creating a new connection to the data source.

ExecutingSQLStatements

JDBCsupportsthreedifferentwaysofexecutingstatements:

- Statement
- PreparedStatement,and
- CallableStatement.

The **Statement**classis thebaseclassfortheothertwostatementclasses.Itallows ustoquerythe data source with any static or dynamically generated SQL query.

The **PreparedStatement** class dynamically generates precompiled SQL statements that can be used several times; these SQL statements can have parameters, but their structure is fixed whenthe PreparedStatement object is created.

```
//initialquantityisalwayszero
Stringsql="INSERTINTOBooksVALUES('?,7,'?,?,0,7)";
PreparedStatementpstmt=con.prepareStatement(sql);
//nowinstantiatetheparameterswithvalues
//a,ssumethatisbn,title,etc.areJavavariablesthat
//containthevaluestobeinserted
pstmt.clearParameters() ;
pstmt.setString(l, isbn);
pstmt.setString(2, title);
pstmt.setString(3, author);
pstmt.setFloat(5, price);
pstmt.setInt(6, year);
intnumRows=pstmt.executeUpdate();
```

programcode:SQLUpdateUsingaPreparedStatementObject

The SQL query specifies the query string, but uses ''?' for the values of the parameters, which are setlaterusingmethodssetString,setFloat,andsetInt.The"?"placeholderscanbeusedanywhere in SQL statements where they canbe replaced with a value. Examples of places wherethey can appear include the WHERE clause (e.g., 'WHERE author=?'), or in SQL UPDATE and INSERT statements. The method setString is one way to set a parameter value; analogous methods are available for int, float, and date. It is good style to always use clearParameters() before setting parameter values in order to remove any old data.

There are different ways of submitting the query string to the data source. In the example, we used the **executeUpdate** command, which is used if we know that the SQL statement does not returnany records (SQL UPDATE, INSERT,ALTER, and DELETE statements). The executeUpdate method returns

- anintegerindicatingthenumberofrowstheSQLstatementmodified;
- 0forsuccessfulexecutionwithoutmodifyinganyrows.

The executeQuery method is used if the SQL statement returns data, such as in a regular SELECT query. JDBC has its own cursor mechanism in the form of a ResultSet object.

ResultSets

ResultSet cursors in JDBC 2.0 are very powerful; they allow forward and reverse scrolling and in-place editing and insertions. In its most basic form, the **ResultSet** object allows us to read one row of the output of the query at a time.Initially, the **ResultSet** is positioned before the first row, and we havetoretrievethefirstrowwithanexplicit calltothe **next()**method.Thenextmethodreturnsfalse if there are nomorerowsin thequeryanswer,and true other\vise.The code fragment shown below illustrates the basic usage of a ResultSet object:

```
ResultSetrs=stmt.executeQuery(sqlQuery);
//rs isnowa cursor
//firstcalltors.nextOmovestothefirstrecord
//rs.nextOmovestothenextrow
String sqlQuery;
ResultSetrs=stmt.executeQuery(sqlQuery) while
(rs.next())
 {
     // processthedata
 }
```

While next () allows us to retrieve the logically next row in the query answer, we can move about in the query answer in other ways too:

- **previous()**movesbackonerow.
- **absolute(intnum)**movestotherowwiththespecifiednumber.
- **relative(intnum)**movesforwardorbackward(ifnumisnegative)relativetothecurrent position. **relative (-1)** has the same effect as previous.
- **first()**movestothefirstrow,and**last()**movestothelastrow.

MatchingJavaandSQLDataTypes

In consideringthe interaction of anapplication with adatasource,the issues we encountered inthe context of Embedded SQL (e.g., passing information between the application and the data source through shared variables) arise again. To deal with such issues, JDBC provides special data types and specifies their relationship to corresponding SQL data types. Table 2.4.4 shows the accessor methods in a ResultSet object for the most common SQL datatypes.

With these accessor methods, we can retrieve values from the current row of the query result referenced by the ResultSet object. There are two forms for each accessor method. One method retrieves values by column index, starting at one, and the other retrieves values by column name.

Thefollowingexampleshowshowtoaccessfieldsofthecurrent ResultSetrowusingaccesssor methods.

```
ResultSetrs=stmt.executeQuery(sqlQuery);

StringsqlQuerYi
ResultSetrs=stmt.executeQuery(sqlQuery) while
(rs.nextO)
{
    isbn= rs.getString(l);
    title=rs.getString("TITLE");
    // processisbnand title
}
```

SQLType	Javaclass	ResultSetgetmethod
BIT	Boolean	getBoolean()
CHAR	String	getString()
VARCHAR	String	getString()
DOUBLE	Double	getDouble()
FLOAT	Double	getDouble()
INTEGER	Integer	getInt()
REAL	Double	getFloat()
DATE	java.sql.Date	getDate()
TIME	java.sql.Time	getTime()
TIMESTAMP	java.sql.TimeStamp	getTimestamp()

Table2.4.4:ReadingSQLDatatypesfromaResultSetObject

ExceptionsandWarnings

Similar to the SQLSTATE variable, most of the methods in java. sql can throw an exception of the type SQLException if an error occurs. The information includes SQLState, a string that describesthe error (e.g., whether the statement contained an SQL syntax error). In addition to the standard getMessage() method inherited from Throwable, SQLException has two additional methods that provide further information, and a method to get (or chain) additionalexceptions:

- public String getSQLState() returns an SQLState identifier based on the SQL:1999 specification
- publicintgetErrorCode()retrievesavendor-specificerrorcode.

- publicSQLExceptiongetNextExceptionOgetsthenextexceptioninachainofexceptions associated with the current SQLException object.

An SQLWarning is a subclass of SQLException. Warnings are not as severe as errors and the program can usually proceed without special handling of warnings. Warnings are not thrown like other exceptions, andthey are not caught as part of the try-catch block around ajava.sql statement. We need to specifically test whether warnings exist. **Connection**, **Statement**, and **ResultSet**objects all have a **getWarnings()** method with which we can retrieve SQL warnings if they exist. Duplicate retrieval of warnings can be avoided through **clearWarnings(). Statement** objects clear warnings automatically on execution of the next statement; **ResultSet** objects clear warnings every time a new tuple is accessed.

TypicalcodeforobtainingSQLWarningslookssimilartothecodeshownbelow: try

```
{
        stmt=con.createStatement();
        warning = con.getWarnings();
        while( warning != null)
         {
                / / handleSQLWarnings / / code to process warning
                warning=warning.getNextWarningO;//getnextwarning
        }
        con.clear\Varnings() ;
        stmt.executeUpdate(queryString);
        warning = stmt.getWarnings();
        while( warning != null)
         {
                / / handleSQLWarnings / / code to process warning
                warning=warning.getNextWarningO;//getnextwarning
        }
} //endtry
catch(SQLExceptionSQLe)
 {
        // codetohandleexception
} //end catch
```

ExaminingDatabaseMetadata

Wecan use the DatabaseMetaData object to obtain information about the database system itself,as well as information from the database catalog. For example, the following code fragment shows how to obtain the name and driver version of the JDBC driver:

```
Databa..seMetaData md = con.getMetaD<Lta():
System.out.println("Driver Information:");
System.out.println("Name:"+md.getDriverNameO
        +";version:"+mcl.getDriverVersion());
```

The DatabaseMetaDataobjecthasmanymoremethods(inJDBC2.0,exactly 134).Someofthe methods are:

- **public ResultSet getCatalogs() throws SqLException**. This function returns a ResultSet that can be used to iterate over all **public int getMaxConnections() throws SqLException** the catalog relations.This function returns the maximum number of connections possible.

Example:codefragmentthatexaminesalldatabasemetadata

```
DatabaseMetaData dmd = con.getMetaDataO;
ResultSettablesRS=dmd.getTables(null,null,null,null);
string tableName;
while(tablesRS.next())
 {
        tableNarne=tablesRS.getString("TABLE_NAME");
        / / print out the attributes of this table
        System.out.println("Theattributesoftable"
                +tableName+"are:");
        ResultSetcolumnsRS=dmd.getColums(null,null,tableName,null);
        while (columnsRS.next())
        {
                System.out.print(columnsRS.getString("COLUMN_NAME")
                +"");
        }
        / / print out the primary keys of this table
        System.out.println("The keys of table" + tableName + " are:");
        ResultSetkeysRS=dmd.getPrimaryKeys(null,null,tableName);
        while (keysRS. next ())
        {
                 System.out.print(keysRS.getStringC'COLUMN_NAME")+"");
        }
```

}

7stepsforjdbc:

1. Importthepackage
   --importjava.sql.*;
2. Loadandregisterthedriver
   --class.forname();
3. Establishtheconnection
   --Connectioncon;
4. CreateaStatementobject
   --Statementst;
5. Executeaquery
   --st.execute();
6. Processstheresult
7. Closetheconnection

Step 2: load the corresponding JDBC driver

Class.forName("oracle/jdbc.driver.OracleDriver");

Step3:createasessionwithdatasourcethroughcreationofConnectionobject.

Connectionconnection=DriverManager.getConnection(database_url,
userId,password);

EX:Connectioncon=DriverManager.getConnection
("jdbc:oracle:thin:@localhost:1521:xesid","system","ambika");

Step4:createastatementobject

- JDBCsupportsthreedifferentwaysofexecutingstatements:
- Statement
- PreparedStatementand
- CallableStatement.

- TheStatement classisthebaseclassfortheothertwostatementclasses.Itallowsusto query the data source with any static or dynamically generated SQLquery.
- ThePreparedStatementclassdynamicallygeneratesprecompiledSQLstatementsthat can be used several times
- CallableStatementareusedtocallstoredproceduresfromJDBC.CallableStatementisa subclass of PreparedStatement and provides the same functionality.
- Example:

Statementst=con.createStatement();

Step5:executinga query

Stringquery="select*fromstudentswhereusn='4VV15CS001'";

ResultSet rs=st.executeQuery(query);

Step6: processtheresult

Stringsname=rs.getString(2);

System.out.println(sname);

Step7: closetheconnection

con.close();

```
import java.sql.*;
publicclassDemo{
publicstaticvoidmain(String[]args){ try
    {
      Stringquery="select*fromstudentswhereusn='4VV15CS001'";
      Class.forName("oracle/jdbc.driver.OracleDriver");
      Connectioncon=DriverManager.getConnection
                    ("jdbc:oracle:thin:@localhost:1521:xesid","system","ambika");
      Statement st=con.createStatement();
      ResultSetrs=st.executeQuery(query);
     String s=rs.getString(1);
     System.out.println(s);
      con.close();
    }
    catch(Exceptione)
    {
     }
  }
```

SQLJ:SQL-JAVA

SQLJenablesapplicationsprogrammerstoembedSQLstatementsinJavacodeinawaythat is compatible with the Java design philosophy

Example:SQLJcodefragmentthatselectsrecordsfromtheBookstablethatmatchagivenauthor.

```
Stringtitle;Floatprice;Stringauthor;
#sqliteratorBooks(Stringtitle,Floatprice); Books
books;
#sqlbooks={
        SELECT title, price INTO :title, :price
        FROMBooksWHEREauthor=:author
    };
while(books.next()){
        System.out.println(books.title()+","+books.price());
    }
books.close();
```

AllSQLJstatements havethespecialprefix #sql.InSQLJ,weretrievetheresultsofSQLqueries with iterator objects, which are basically cursors. An iterator is an instance of an iterator class. UsageofaniteratorinSQLJgoesthroughfivesteps:

1. DeclaretheIteratorClass:Inthepreceedingcode,thishappenedthroughthestatement #sql iterator Books (String title, Float price);
   ThisstatementcreatesanewJavaclassthatwecanusetoinstantiateobjects.
2. Instantiate an Iterator Object from the New Iterator Class:

   WeinstantiatedouriteratorinthestatementBooksbooks;.

3. InitializetheIteratorUsingaSQLStatement:
   Inourexample,thishappensthroughthestatement#sqlbooks=....
4. Iteratively,ReadtheRowsFromtheIteratorObject:
   ThisstepisverysimilartoreadingrowsthroughaResultSetobjectinJDBC.
5. ClosetheIteratorObject.

Therearetwotypesofiteratorclasses:

- namediterators
- positionaliterators

For named iterators, we specify both the variable type and the name of each column of the iterator. This allows us to retrieve individual columns by name. This method is used in our example.

For positional iterators, we need to specify only the variable type for each column of the iterator. To access the individual columns of the iterator, we use a FETCH ... INTO construct, similar to Embedded SQL

Wecanmaketheiteratorapositionaliteratorthroughthefollowingstatement: #sql

iterator Books (String, Float);

Wethenretrievetheindividualrowsfromtheiteratorasfollows:

```
while(true)
  {
    #sql{FETCH:booksINTO:title,:price,}; if
   (books.endFetch())
   {break:}
  // processthebook
}
```

STOREDPROCEDURES

Storedprocedureisaset of logicalgroupofSQLstatementswhicharegroupedtoperforma specific task.

Benefits:

- reducestheamountofinformationtransferbetweenclientanddatabaseserver
- Compilation step is required only once when the stored procedure is created. Then after it does not require recompilation before executing unless it is modified and reutilizes the same execution plan whereas the SQL statements need to be compiled every time whenever it is sent for execution even if we send the same SQL statement everytime
- It helps in re usability of the SQL code because it can be used by multiple users and by multiple clients since we need to just call thestoredprocedureinsteadofwritingthe same SQL statement every time. It helps in reducing the development time

Syntax:

Createorreplaceprocedure<procedureName>[(arg1datatype,arg2datatype)]

Is/As

<declaration>

Begin

<SQL Statement>

Exception

Endprocedurename;

CreatingaSimpleStoredProcedure

Considerthefollowingschema:

Student(usn:string,sname:string)

Let usnowwriteastoredproceduretoretrievethecountofstudentswithsname‘Akshay’ create

or replace procedure ss

is

stu_cntint;

begin

selectcount(*)intostu_cntfromstudentswheresname='AKSHAY';

dbms_output.put_line('the count of student is :' || stu_cnt);

endss;

Storedprocedurescanalsohaveparameters. TheseparametershavetobevalidSQLtypes,and have one of three different modes: IN, OUT, or INOUT.

- INparametersareargumentstothestoredprocedure
- OUTparametersarereturnedfromthestoredprocedure; itassignsvaluestoallOUT parameters that the user can process
- INOUTparameterscombinethepropertiesofINandOUTparameters:Theycontainvalues to be passed to the stored procedures, and the stored procedure can set their values as return values

Example:

```
CREATEPROCEDUREAddInventory(
IN book_isbn CHAR(lO),
INaddedQtyINTEGER)
UPDATEBooksSETqty_in_stock=qtyjn_stock+addedQty WHERE
bookjsbn = isbn
```

InEmbeddedSQL,theargumentstoastoredprocedureareusuallyvariablesinthehostlanguage. For example, the stored procedure AddInventory would be called as follows:

```
EXECSQLBEGINDECLARESECTION
  char isbn[lO];
  long qty;
EXECSQLENDDECLARESECTION
// set isbnandqtytosome values
EXECSQLCALL AddInventory(:isbn,:qty);
```

Storedproceduresenforcestricttypeconformance:IfaparameterisoftypeINTEGER, itcannot be called with an argument of type VARCHAR.

Procedureswithoutparametersarecalled**staticprocedures**andwithparametersarecalled **dynamicprocedures.**

Example:storedprocedurewithparameter

```
createorreplaceprocedureemp(Essnint) as
        eNamevarchar(20);
 begin
         selectfnameintoeNamefromemployeewheressn=Essnanddno=5;
         dbms_output.put_line(' the employee name is :'||Essn ||eName);
end emp;
```

CallingStored Procedures

StoredprocedurescanbecalledininteractiveSQLwiththeCALLstatement: CALL

```
storedProcedureName(argl, arg2, .. ,argN);
```

CallingStoredProceduresfromJDBC

We can call stored procedures from JDBC using the CallableStatment class.A stored procedure could containmultiple SQL statementsoraseries of SQLstatements-thus, theresult couldbemany different ResultSet objects.We illustrate the case when the stored procedure result is a single ResultSet.

```
CallableStatementcstmt=con.prepareCall("{callShowNumberOfOrders}");
ResultSet rs = cstmt.executeQuery();
while(rs.next())
```

CallingStoredProceduresfromSQLJ

ThestoredprocedureShowNumberOfOrdersiscalledasfollowsusingSQLJ:

```
//createthecursorclass
    #sqlIteratorCustomerInfo(intcid,Stringcname,intcount);
// createthecursor
    CustomerInfocustomerinfo;
    // callthestoredprocedure
    #sqlcustomerinfo={CALLShowNumberOfOrders};
    while (customerinfo.next()
    {
        System.out.println(customerinfo.cid()+","+
        customerinfo.count()) ;
    }
```

SQL/PSM

SQL/PersistentStoredModulesisanISOstandardmainlydefininganextensionofSQLwith procedural language for use in stored procedures.

InSQL/PSM,wedeclareastoredprocedureasfollows:

```
CREATEPROCEDUREname(parameter1,...,parameterN)
local variable declarations
procedurecode;
```

Wecandeclareafunctionsimilarlyasfollows:

CREATEFUNCTIONname(parameterl,...,parameterN)

RETURNS sqIDataType

localvariabledeclarations

function code;

Example:

CREATEFUNCTIONRateCustomer(INcustIdINTEGER,INyearINTEGER)

RETURNS INTEGER

DECLARERatingINTEGER;

DECLAREnumOrdersINTEGER;

SETnumOrders=(SELECTCOUNT(*)FROMOrders0WHEREO.tid=custId); IF

(numOrders> 10) THEN rating=2;

ELSEIF(numOrders>5)THENrating=1;

ELSE rating=O;

ENDIF;

RETURNrating;

- We candeclarelocalvariablesusingtheDECLAREstatement.Inourexample,wedeclaretwo local variables: 'rating', and 'numOrders'.
- PSM/SQLfunctionsreturnvaluesviatheRETURNstatement.Inour example,wereturnthe value of the local variable 'rating'.
- We canassignvaluestovariableswiththeSETstatement.Inourexample,weassignedthe return value of a query to the variable 'numOrders'.
- SQL/PSMhasbranchesandloops.Brancheshavethefollowingform: IF

  (condition) THEN statements;

  ELSEIFstatements;

  ELSEIFstatements;

  ELSEstatements;

  ENDIF

  - Loopsareoftheform

LOOP

statements:

ENDLOOP

Queries can be used as part of expressions in branches; queries that return a single value can be assigned to variables.We can use the same cursor statements as in Embedded SQL (OPEN, FETCH, CLOSE), but we do not need the EXEC SQL constructs, and variables do not have to be prefixed by a colon ':'.

Chapter3:InternetApplications

Introduction

Data-intensive is used to describe applications with a need to process large volumes of data. The volume of datathat is processed can be in the size of terabytes and petabytesandthis type of data is also referred as big data. Data-intensive computing is used in many applications ranging from social networking to computational science where a large amount of data needs to be accessed, stored, indexed and analyzed. It is more challenging as the amount of data keeps on accumulating over time and the rate at which the data is generating also increases

THETHREE-TIERAPPLICATIONARCHITECTURE

Data-intensiveInternet applicationscanbeunderstoodintermsofthreedifferentfunctional components:

1. Datamanagement
2. Applicationlogic
3. Presentation

ThecomponentthathandlesdatamanagementusuallyutilizesaDBMSfordatastorage,but application logic and presentation involve much more than just the DBMS itself.

Single-Tier

Initially, data-intensive applications were combined into a single tier, including the DBMS,application logic, and user interface. The application typically ran on a mainframe, and users accessed it through dumb terminals that could perform only data input and display.

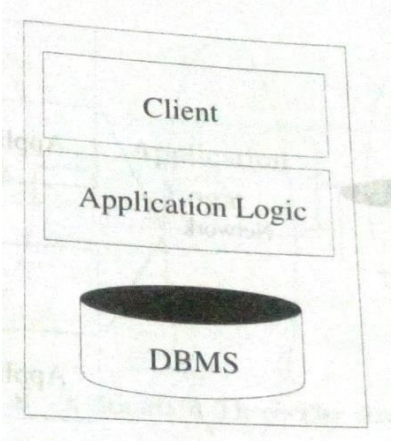


Figure3.2.1:ASingle-TierArchitecture

- **Benefit**
  - easilymaintainedbyacentraladministrator
- **Drawback:**
  - Usersexpectgraphicalinterfacesthatrequiremuchmorecomputationalpowerthan simple dumb terminals.
  - Donotscaletothousandsofusers

Two-tierarchitectures

Two-tier architectures, often also referred to as client-server architectures, consist of a client computer and a server computer, which interact through a well-defined protocol. What part of the functionality the client implements, and what part is left to the server, can vary.

In the traditional client server architecture, the client implements just the graphical user interface - suchclientsareoftencalled**thinclients**theserverimplementsboththe businesslogicandthedata management.

Other divisions are possible, such as more powerful clients that implement both user interface and business logic, or clients that implement user interface and part of the business logic, with the remaining part being implemented at the server level; such clients are often called **thick clients**.

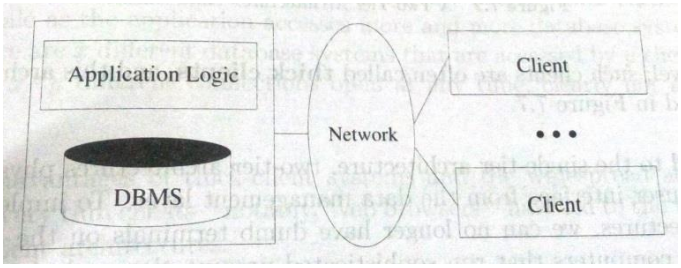


Figure3.2.2(a):ATwo-ServerArchitecture:thinclient

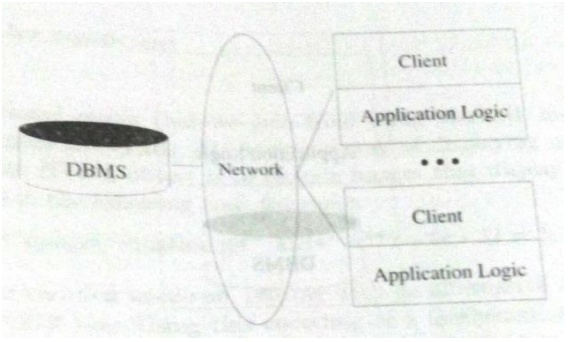


Figure3.2.2(a):ATwo-ServerArchitecture:thickclient

Thethick-clientmodelhasseveraldisadvantageswhencomparedtothethinclientmodel

1. Thereis no central placeto update and maintainthe business logic,sincethe application code runs at many client sites.
2. A large amount of trust is required between the server and the clients. As an example, theDBMS of abankhastotrustthe applicationexecuting at anATM machinetoleavethedatabase in a consistent state.
3. Thick-client architecturedoes not scale with the number of clients; it typically cannot handle more than a few hundred clients. The application logic at the client issues SQL queries to the server and the server returns the query result to the client, where further processing takes place. Large query results might be transferred between client and server.

Single-tierarchitecturev/sTwo-tierarchitectures

- Compared to the single-tier architecture, two-tier architectures physically separate the user interface from the data management layer
- To implement two tier architectures, we can no longer have dumb terminals on the client side, we require computers that run sophisticated presentation code and possibly,application logic

Three-TierArchitectures

The thin-client two-tier architecture essentially separates presentation issues from the rest of the application. The three-tier architecture goes one step further, and also separates application logic from data management:

- PresentationTier
- MiddleTier
- DataManagementTier

Differenttechnologieshavebeendevelopedtoenabledistributionofthethreetiersof an application across multiple hardware platforms and different physical sites

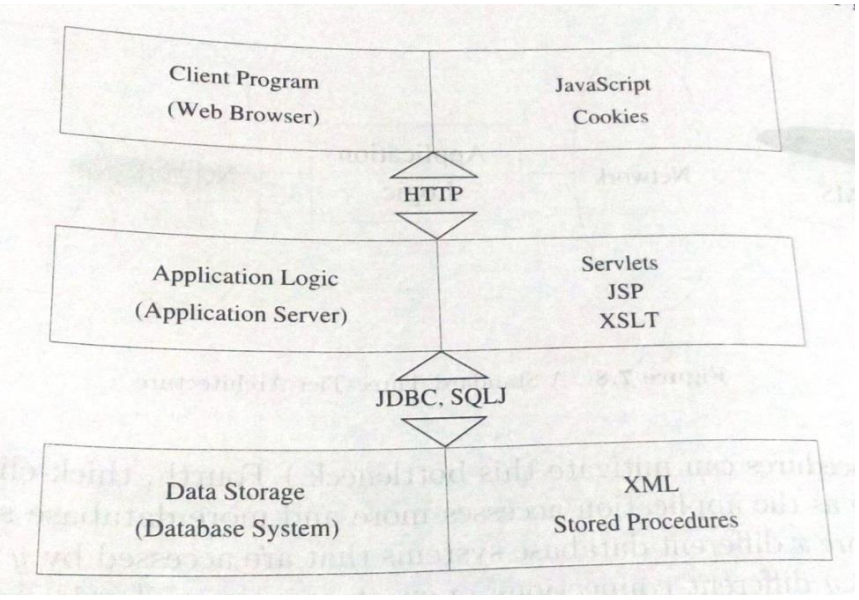


Figure3.2.3:TechnologiesfortheThree Tiers

OverviewofthePresentation Tier

At the presentation layer, we need to provide forms through which the user can issue requests, and display responses that the middle tier generates. It is important that this layer of code be easy to adapt to different display devices and formats; for example, regular desktops versus handheld devices versus cell phones. This adaptivity can be achieved either at the middle tier through generation of different pages for different types of client, or directly at theclient through style sheets that specify how the data should be presented. The hypertext markup language (HTML) is the basic data presentation language.

Technologiesfortheclientsideofthethree-tier architecture

HTMLForms

HTMLformsareacommonwayof communicatingdatafromtheclienttiertothemiddletier. The general format of a form :

```
<FORMACTION="page.jsp"METHOD="GET"NAME="LoginForm">
.....-
</FORM>
```

- ACTION:SpecifiestheURI ofthepagetowhichtheformcontentsaresubmitted. Ifthe ACTION attribute is absent, then the URI of the current page is used
- METHOD:TheHTTP/1.0methodusedtosubmittheuserinputfromthefilled-outformto the webserver. There are two choices: GET and POST
- NAME:Thisattributegivestheformaname

AsingleHTMLdocument cancontainmorethanoneform.InsideanHTMLform, wecanhaveany HTML tags except another FORM element

PassingArgumentstoServer-SideScripts

There are two different ways to submit HTML Form data to the webserver. If the method GET is used, thenthecontentsof theform are assembled into aquery URI (asdiscussed next) andsent to the server. If the method POST is used, then the contents of the form are encoded as in the GET method, butthecontentsaresentinaseparatedata blockinsteadofappendingthemdirectlytothe URI. Thus, in the GET method the form contents are directly visible to the user as the constructed URI, whereas in the POST method, the form contents are sent inside the HTTP request message body and are not visible to the user.

JavaScript

JavaScript is a scripting language at the client tier with which we can add programs to webpages that run directly at the client. JavaScript is often used for the following types of computation at the client:

- **BrowserDetection:** JavaScript canbeusedtodetectthebrowsertypeand loadabrowser-specific page.
- **FormValidation:**JavaScriptisusedtoperformsimpleconsistencychecksonformfields
- **Browser Control:** This includes opening pages in customized windows; examples include the annoying pop-up advertisements that you see at many websites, which are programmed using JavaScript.

JavaScriptisusuallyembeddedintoanHTMLdocumentwithaspecialtag,theSCRIPTtag

<SCRIPTLANGUAGE="JavaScript"SRC="validateForm.js"></SCRIPT>

The SCRIPT tag has the attribute LANGUAGE, which indicates the language in which the script is written. For JavaScript, we setthe language attribute to JavaScript. Another attribute ofthe SCRIPT tag is the SRC attribute, which specifies an external file with JavaScript code that is automatically embedded into the HTML document. Usually JavaScript source code files use a '.js' extension.

Style Sheets

A style sheet is a method to adapt the same document contents to different presentation formats. A style sheet contains instructions that tell a web how to translate the data of a document into a presentation that is suitable for the client's display. The use of style sheets has many advantages:

- we can reuse the same document many times and display it differently depending on the context
- we can tailor the display to the reader's preference such as font size, color style, and even level of detail.
- we can deal with different output formats, such as different output devices (laptops versus cell phones), different display sizes (letter versus legal paper), and different display media (paper versus digital display)
- we can standardize the display format within a corporation and thus apply style sheet conventions to documents at any time.
- changes and improvements to these display conventions can be managed at a centralplace.

Therearetwostylesheet languages:

- XSL
- CSS

CascadingStyleSheets(CSS)

- CSSwascreatedforHTMLwiththegoalofseparatingthedisplaycharacteristicsof different formatting tags from the tags themselves
- CSSdefineshowtodisplayHTMLelements.
- Stylesarenormallystoredinstylesheets,whicharefilesthatcontainstyledefinitions.
- ManydifferentHTMLdocuments,suchasalldocumentsinawebsite,canrefertothesame CSS.
- Thus,wecanchangetheformatofawebsitebychangingasinglefile.
- EachlineinaCSSsheetconsistsofthreeparts;aselector,aproperty,andavalue.Theyare syntactically arranged in the following way:

  selector{property:value}
- Theselectoristheelementortagwhoseformatwearedefining.
- Thepropertyindicatesthetag'sattributewhosevaluewewanttosetinthestylesheet
- Example:BODY{BACKGROUND-COLOR:yellow}

  P {MARGIN-LEFT: 50px; COLOR: red}

XSL

- XSLisalanguageforexpressingstylesheets
- An XSL style sheet is, like CSS,a file that describes how to display an XML document of a given type.
- XSL contains the XSL Transformation language, or XSLT, a language that allows us to transform the input XML document into a XML document with anotherstructure
- For example, with XSLT we can change the order of elements that we are displaying (e.g.; by sorting them), process elements more than once, suppress elements in one place and present them in another, and add generated text to the presentation

OverviewoftheMiddleTier

The middle layer runs code that implements the business logic of the application. The middle tier code is responsible for supporting all the different roles involved in the application. For example, in an Internet shopping site implementation, we would like

- customerstobeabletobrowsethecatalogandmakepurchases
- administratorstobeabletoinspectcurrentinventory,and
- dataanalyststoasksummaryqueriesaboutpurchasehistories
- Eachoftheserolescanrequiresupportforseveralcomplexactions

Thefirstgenerationofmiddle-tierapplicationswasstand-aloneprogramswritteninageneral- purpose programminglanguagesuch asC,C++,andPerl. Programmersquickly realizedthat

interaction with a stand-alone application was quite costly. The overheads include starting the application every time it is invoked and switching processes between the webserver and the application. Therefore, such interactions do not scale to large numbers of concurrent users. Most of today's large-scale websites use an application server to run application code at the middle tier. Application server provides the run-time for several technologies that can be used to program middle-tier application components.

CGI:TheCommonGateway Interface

TheCommonGatewayInterfaceconnectsHTMLformswithapplicationprograms.

- It is a protocol that defines how arguments from forms are passed to programs at the server side
- CGI is the part of the Web server that can communicate with other programs running on the server
- With CGI, the Web server can call up a program, while passing user-specific data to the program(suchaswhat hosttheuserisconnectingfrom, orinputtheuser has suppliedusing HTML form syntax)
- The program then processes that data and the server passes the program's response back to the Web browser.

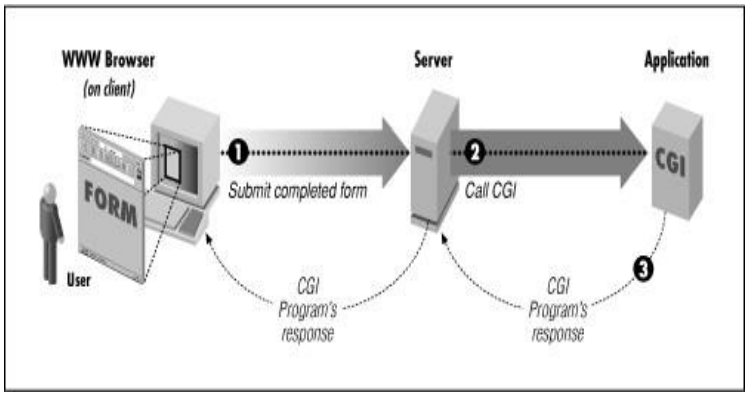


Figure:SimplediagramofCGI

```
<HTML><HEAD><TITLE>TheDatabaseBookstore</TITLE></HEAD>
<BODY>
<FORMACTION="find_books.cgiIIMETHOD=POST>
  Type an author name:
<INPUTTYPE="textIINAME=lauthorName"
SIZE=30 MAXLENGTH=50>
<INPUTTYPE="submitilvalue="Sendit">
<INPUTTYPE=lreset"VALUE="ClearformII>
```

```
</FORM>
</BODY></HTML>
```

Programfragment:ASample'webPageWhereFormInputIsSenttoaCGIScript

ApplicationServers

Application logic can be enforced through server-side programs that are invoked using the CGI protocol. However, since each page request results in the creation of a new process, this solution does not scale well to a large number of simultaneous requests. An application server maintains a pool of threads or processes and uses these to execute requests. Thus, it avoids the startup cost of creating a new process for each request. They facilitate concurrent access to severalheterogeneous data sources (e.g., by providing JDBC drivers), and provide session management services.

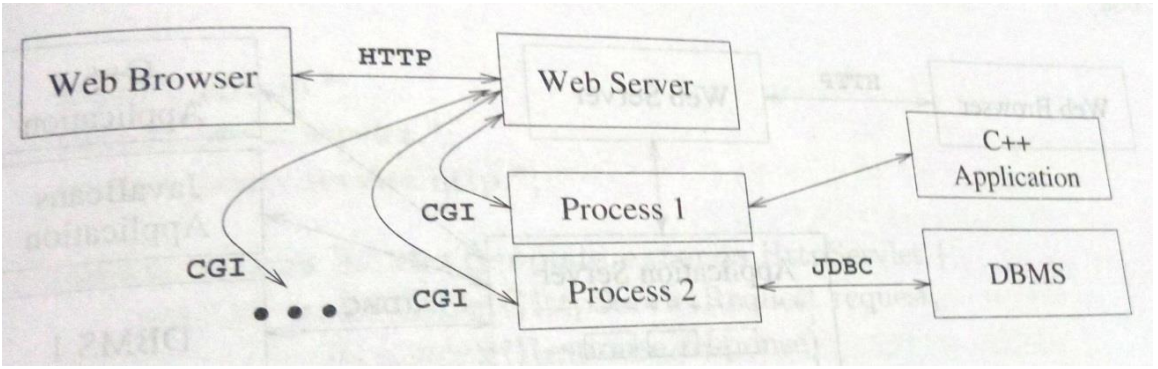


Fig:ProcessStructurewithCGIScripts

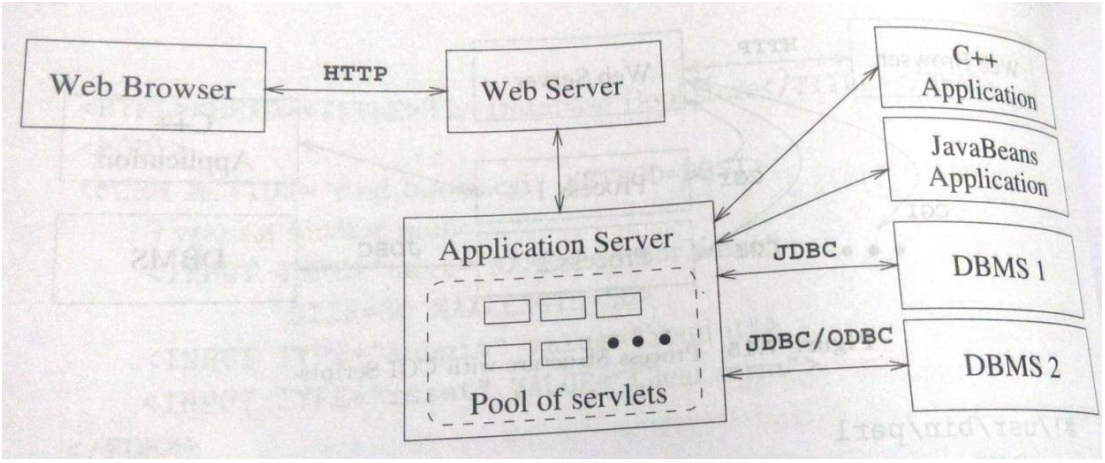


Fig:ProcessStructureintheApplicationServerArchitecture

Servlets

Java servlets are pieces of Java codethat run on the middle tier, ineither webservers or application servers. Servlets can build webpages, access databases, and maintain state.Servlets usuallyhandle requests from HTML forms and maintain state between the client and theserver.

Servlets are compiled Java classes executed and maintained by a servlet container. The servlet container manages the lifespan of individual servlets by creating and destroying them. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by webservers.

JavaServerPages

Java Server Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. JSP have access tothe entire family of Java APIs, including the JDBC API to access enterprisedatabases

JavaServer pages (.JSPs) interchange the roles of output amI application logic. JavaServer pages are writteninHTMLwithservlet-likecodeembedded inspecial HT1VILtags. Thus, incomparisonto servlets, JavaServer pages are better suited to quickly building interfaces that have some logic inside, whereas servlets are better suited for complex application logic.

MaintainingState

There is a need to maintain a user's state across different pages. As an example, consider a user who wants to make a purchase at the Barnes and Nobblewebsite. The user must first add itemsinto her shopping basket, which persists while she navigates through the site Thus, we use the notion of state mainly to remember information as the user navigates through the site.

TheHTTPprotocol is stateless.Wecall aninteractionwith a webserver statelessif noinformation is retained from one request to the next request. We call an interaction with a webserver stateful, or we say that state is maintained, if some memory is stored between requests to the server, and different actions are taken depending on the contents stored.

SincewecannotmaintainstateintheHTTPprotocol, whereshouldwemaintainstate?Thereare basically two choices:

- We canmaintainstateinthemiddletier,bystoringinformation inthelocalmainmemory of the application logic, or even in a database system
- Alternatively,wecanmaintainstateontheclientsidebystoringdataintheformofa*cookie.*

MaintainingStateattheMiddleTier

Atthemiddletier,wehaveseveralchoicesasto*where*wemaintainstate.

- First, we could store the state at the bottom tier, in the database server. The state survives crashes of the system, but a database access is required to query or update the state, a potential performance bottleneck
- An alternative is to store state in main memory at the middle tier. The drawbacks are that this information is volatile and that it might take up a lot of main memory
- We can also store state in local files at the middle tier, as a compromise between the first two approaches.

MaintainingStateatthePresentationTier:Cookies

A **cookie** is a collection of *(name,* value)pairs that can be manipulated at the presentation and middle tiers. Cookies are easy to use in Java servlets and Java server Pages. They survive several client sessions because they persist in the browser cache even after the browser is closed. One disadvantage of cookies is that they are often perceived as as being invasive, and many users disable cookies in their Web browser; browsers allow users to prevent cookies from being saved on their machines. Another disadvantage is that the data in a cookie is currently limited to 4KB, but for most applications this is not a bad limit.

AdvantagesoftheThree-TierArchitecture

Thethree-tierarchitecturehasthefollowingadvantages:

- **Heterogeneous Systems:** Applications can utilize the strengths of different platforms and different softwarecomponentsatthedifferenttiers.Itiseasyto modifyorreplacethecodeat any tier without affecting the other tiers.
- **Thin Clients:** Clients only need enough computation power for the presentation layer. Typically, clients are Web browsers.
- **Integrated Data Access:** In many applications, the data must be accessed from several sources. This can be handled transparently at the middle tier, where we can centrally manage connections to all database systems involved.
- **Scalability to Many Clients:** Each client is lightweight and all access to the system is through the middle tier. The middle tier can share database connections across clients, andif the middle tier becomes the bottle-neck, we can deploy several servers executing the middle tier code; clients can connect to anyone of these servers, if the logic is designed appropriately.
- **Software Development Benefits:** By dividing the application cleanly intoparts that address presentation, data access, and business logic, we gain many advantages. The businesslogic is centralized, and is therefore easy to maintain, debug, and change. Interaction between tiers occurs through well-defined, standardized APIs.Therefore, each application

tiercanbebuiltoutofreusablecomponentsthatcanbeindividuallydeveloped, debugged, and tested.

QuestionBank

1. Discuss how NULLs are treated in comparison operators in SQL. How are NULLs treated when aggregate functions are applied in an SQL query? How are NULLs treated if they exist ingrouping attributes?
2. Describe the six clauses in the syntax of an SQL retrieval query. Show what type of constructs can be specified in each of the six clauses. Which of the six clauses are required and which are optional?
3. Describe conceptually how an SQL retrieval query will be executed by specifying the conceptual order of executing each of the six clauses.
4. Explainhow theGROUPBY clause works. What isthe differencebetweentheWHEREand HAVING clause?
5. Explaininsert,deleteandupdatestatementsinSQLandgiveexampleforeach.
6. Writeanoteon:
   i) ViewsinSQL
   ii) AggregatefunctionsinSQL
7. ExplainDROPcommandwithan example.
8. Howisviewcreatedanddropped?Whatproblemsareassociatedwithupdatingviews?
9. HowaretriggersandassertionsdefinedinSQL?Explain.
10. ConsiderthefollowingschemaforaCOMPANYdatabase:

    EMPLOYEE(Fname,Lname,Ssn,Address,Super-ssn,Salary,Dno) DEPARTMENT
    (Dname, Dnumber, Mgr-ssn, Mgr-start-date)
    DEPT-LOCATIONS (Dnumber, Dlocation)
    PROJECT(Pname,Pnumber,Plocation,Dnum)
    WORKS-ON (Ess!!, Pno, Hours)
    DEPENDENT(Essn,Dependent-name,Sex,Bdate,Relationship)
    write the SQL query for the following:

    i) Listthenamesofmanagerswhohaveatleastonedependent.
    ii) Retrievethelistofemployeesandtheprojectstheyareworkingon, orderedbydepartment and, within each department, ordered alphabetically by last name, first name.
    iii) Foreachproject,retrievetheprojectnumber,theproject name, andthenumberof employees who work on that project.
    iv) For each project on which more than two employees work, retrieve the project number,the projectname, andthenumber ofemployees who workontheproject.
    v) Foreachproject,retrievetheprojectnumber,theproject name, and thenumberof employees from department 4 who work on theproject.

11. Consider thefollowingtables:

Works(Pname,Cname,Salary)

Lives(Pname,Street,City)

Located-in(Cname,City)

Manager(Pname,mgrname)

writetheSQLqueryforthe following:

i) Findthenamesofallpersonswholiveinthecity 'Mumbai';

ii) RetrievethenamesofallpersonofInfosys'ehosesalaryis betweenRs.30,000and Rs.50,000.

iii) Findthenamesofallpersonswholiveandworkinthesamecity.

iv) Listthenamesofthepeoplewhoworkfor'Wipro'alongwiththecitiestheylivein.

v) Findtheaveragesalaryofall'Infosyians'.

12. Considerthefollowingschema

Sailors(sid,sname,rating,age)

Boats(bid,bname,color)

Reserves(sid,bid,day)

writetheSQLqueryforthe following:

i) Retrieve the sailors name who have reserved red and green boats.

ii)Retrievethesailorsnameswithageover20yearsandreservedblackboat.

iii) Retrievethenumberofboatswhicharenotreserved.

iv) RetrievethesailorsnameswhohavereservedgreenboatonMonday.

v) Retrievethesailorsnameswhoisoldestsailorwithrating10.

13. Consider the following schema and write the SQL queries:

STUDENT-ID,SNAME,MAJOR,GPA)

FACULTY(FACULTY_ID,FNAME,DEPT,DESIGNATION,SALARY)

COURSE(COURSE_ID,CNAME,FACULTY_ID)

ENROLL(COURSE_ID,STUDENT_ID,GRADE)

i) Retrievethestudentnamewhoisstudyingunderfacultiesof"Mechanicaldept".

ii) Retrievethestudent namewhohaveenrolledunderanyofthecourses inwhich'kumar' has enrolled.

iii) Retrievethefacultynamewhoearnsalarywhichis greaterthantheaveragesalaryofall the faculties.

iv) Retrievethesnamewhoarenotbeetaughtbyfaculty'kumar'.

v) RetrievethefacultynameswhoareassistantprofessorsofCSEdept.

14. HowdoweuseSQLstatementswithinahostlangl.lage?Howdowecheckforerrorsin statement execution?

15. Definecursor.whatpropertiescancursors have?
16. WhatisDynamicSQLandhowisitdifferentfromEmbeddedSQL?
17. WhatisJDBCandwhatareitsadvantages?
18. WhatarethecomponentsoftheJDBCarchitecture?Describefourdifferent architectural alternatives for JDBC drivers.
19. Withanexample,explainSQLJ?
20. Illustratewithanexamplestoredprocedure.Mentionitsbenefits.
21. Whatisathree-tierarchitecture?'What advantagesdoesit offeroversingletierandtwo-tier architectures? Give a short overview of the functionality at each of the threetiers.

Module4

Chapter1:DatabaseDesignTheory

Introduction
Objective
IntroductiontoDB design
InformalDesignGuidelinesforRelationSchemas
ImpartingClearSemantics toAttributesin Relations
RedundantInformationinTuplesandUpdateAnomalies
NULLValuesinTuples
GenerationofSpuriousTuples
FunctionalDependencies
NormalizationofRelations
PracticalUseofNormalForms
Definitions ofKeysandAttributesParticipating inKeys
FirstNormalForm
SecondNormalForm
ThirdNormalForm
GeneralDefinitionofSecondandThirdNormalForm
Boyce-CoddNormalForm
MultivaluedDependency andFourthNormalForm
FormalDefinitionofMultivaluedDependency
JoinDependenciesand Fifth NormalForm
InferenceRulesforFunctionalDependencies
Equivalenceof SetsofFunctionalDependencies
SetsofFunctionalDependencies
PropertiesofRelationalDecompositions
AlgorithmsforRelationalDatabaseSchemaDesign
Dependency-PreservingandNonadditive(Lossless)JoinDecompositioninto3NF Schemas
NonadditiveJoinDecompositionintoBCNF Schemas
Dependency-PreservingandNonadditive(Lossless)JoinDecompositioninto3NF Schemas
AboutNulls,DanglingTuples,andAlternativeRelationalDesigns
ProblemswithNULLValuesandDangling Tuples
OtherDependenciesand NormalForms
InclusionDependencies
TemplateDependencies
FunctionalDependencies BasedonArithmeticFunctionsandProcedures
Domain-KeyNormalForm
AssignmentQuestions
ExpectedOutcome
FurtherReading

Introduction

Database Normalization is a technique of organizing the data in the database.Normalization is a systematic approach of decomposing tables to eliminate dataredundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies.It is a multi-step process that puts data into tabular form by removing duplicated data from the relation tables. This module discuss the basic and higher normal forms.

Objectives

- Tostudytheprocessofnormalization andrefinethedatabasedesign
- Tonormalizethe tablesupto 4NFand5NF
- Tostudy lossless and lossy joinoperations
- Tostudyinferencerules
- TostudyotherdependenciesandNormalForms.

IntroductiontoDBdesign

Each relation schema consists of a number of attributes, and the relational database schema consists of a number of relation schemas. So far, we have assumed that attributes are grouped to form a relation schema by using the common sense of the database designer or by mappinga database schema design from a conceptual data model such as the ER or Enhanced-ER (EER) data model. These models make the designer identify entity types and relationship types and their respective attributes, whichleadstoa naturaland logical grouping oftheattributes into relations.

Database Design deals with coming up with a 'good' schema. There are two levels at which we can discuss the goodness of relation schemas:

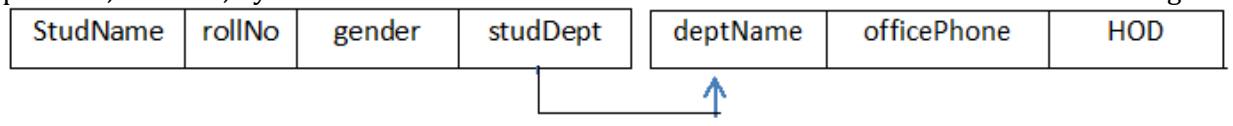
1. Thelogical(orconceptual)level—howusersinterprettherelationschemasandthe meaning of their attributes.
2. Theimplementation(orphysicalstorage)level—howthetuplesinabaserelationare stored and updated. This level applies only to schemas of baserelations

AnExample

- STUDENTrelationwithattributes:studName,rollNo,gender,studDept
- DEPARTMENTrelationwithattributes:deptName,officePhone,hod
- Severalstudentsbelongtoadepartment
- studDeptgivesthenameofthestudent's department

Correctschema:

Student Department



Incorrectschema:

Studdept

StudName	rollNo	gender	deptName	officePhone	HOD

Problemswithbadschema

- **Redundantstorageofdata:**
  - OfficePhone&HODinfo-storedredundantlyoncewitheachstudentthat belongs to the department
  - wastageofdiskspace
- **AprogramthatupdatesOfficePhoneofadepartment**
  - mustchangeitatseveralplaces
  - morerunningtime
  - error-prone

InformalDesignGuidelinesforRelationSchemas

- Fourinformalguidelines thatmaybeusedasmeasurestodeterminethequalityof relation schema design:
  1. Makingsurethatthesemanticsoftheattributesisclearintheschema
  2. Reducingtheredundantinformationintuples
  3. ReducingtheNULLvaluesintuples
  4. Disallowingthepossibilityofgeneratingspurioustuples
- Thesemeasuresarenotalwaysindependentofoneanother

ImpartingClearSemanticstoAttributesinRelations

- **semantics** of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple
- Whenever we group attributes to form a relation schema, we assume that attributes belonging to one relation have certain real-world meaning and a proper interpretation associated with them
- The easier it is to explain the semantics of the relation, the better the relation schema design will be

Guideline1

- Designarelationschemasothatitiseasytoexplainitsmeaning
- Do not combine attributes from multiple entity types and relationship types into a single relation

- if arelationschemacorrespondstooneentitytypeoronerelationshiptype, itis straightforward to interpret and to explain its meaning
- iftherelationcorrespondstoamixtureofmultipleentitiesandrelationships, semanticambiguitieswill result andtherelationcannot beeasilyexplained.

ExamplesofViolatingGuideline1

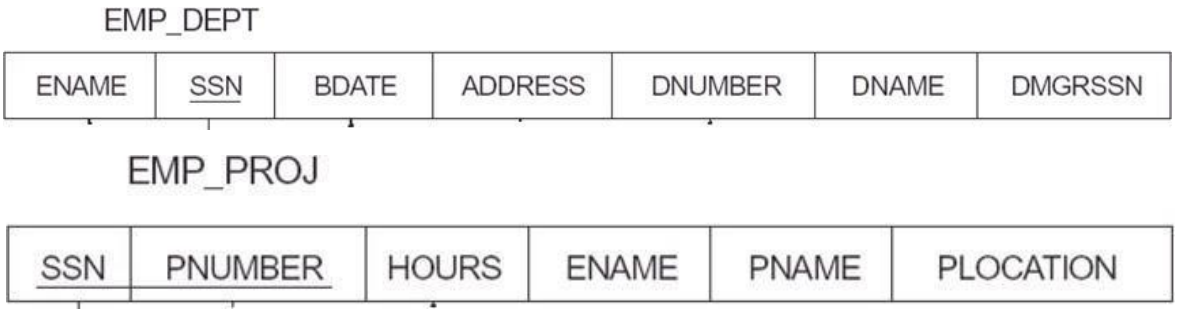


Fig:schemadiagramforcompanydatabase

- Boththerelationschemashaveclearsemantics
- A tuple in the EMP_DEPT relation schema represents a single employee but includes additional information— the name (Dname) of the department for which the employee works and the Social Security number (Dmgr_ssn) of the department manager.
- A tuple in the EMP_PROJ relates an employee to a project but also includes the employee name (Ename), project name (Pname), and project location(Plocation)
- logically correct but they violate Guideline 1 by mixing attributes from distinct real-world entities:
  - EMP_DEPTmixesattributesofemployeesanddepartments
  - EMP_PROJmixesattributesofemployeesandprojectsandtheWORKS_ON relationship
- Theymaybeusedasviews,buttheycauseproblemswhenusedasbaserelations

RedundantInformationinTuplesandUpdateAnomalies

- Onegoalofschemadesignistominimizethestoragespaceusedbythebaserelations
- Groupingattributesintorelationschemashasasignificanteffectonstoragespace
- Forexample,comparethespaceusedbythetwobaserelationsEMPLOYEEand DEPARTMENT with that for an EMP_DEPT baserelation
- In EMP_DEPT, the attribute values pertaining to a particular department (Dnumber, Dname, Dmgr_ssn) are repeated for every employee who works for thatdepartment

- In contrast, each department's information appears only once in the DEPARTMENT relation. Only the department number Dnumber is repeated in the EMPLOYEE relation for each employee who works in that department as a foreignkey

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	gender	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

Figure1: OnepossibledatabasestatefortheCOMPANYrelationaldatabaseschema

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

DEPENDENT

Essn	Dependent_name	gender	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Figure1: One possible database state for the COMPANY relational database schema

Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

Redundancy Redundancy

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

Fig: Sample states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 1

- Storingnaturaljoinsof baserelationsleadstoanadditionalproblemreferredtoas **update anomalies**. These can be classified into:
  - insertionanomalies
  - deletionanomalies,
  - modificationanomalies

Insertion Anomalies

- Insertionanomaliescanbedifferentiatedintotwotypes,illustratedbythefollowing examples based on the EMP_DEPT relation:

1. To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for the department that the employee works for, orNULLs
   - For example, to insert a new tuple for an employee who works in department number 5, we must enter all the attribute values of department 5 correctly so that they are *consistent* with the corresponding values for department 5 in other tuples in EMP_DEPT
   - In the design of Employee in fig 1, we do not have to worry about this consistency problem because we enter only the department number in the employee tuple; all other attribute values of department 5 are recorded only once in the database, as a single tuple in the DEPARTMENT relation
2. It is difficult to insert a new department that has no employees as yet in the EMP_DEPT relation. The only way to do this is to place NULL values in the attributes foremployee
   - ThisviolatestheentityintegrityforEMP_DEPTbecauseSsnisitsprimarykey
   - This problem does not occur in the design of Figure 1 because a department is entered in the DEPARTMENT relation whether or not any employees work for it, andwheneveranemployeeisassignedtothatdepartment,acorrespondingtuple is inserted in EMPLOYEE.

DeletionAnomalies

- Theproblemofdeletionanomaliesisrelatedtothesecondinsertionanomalysituation just discussed
  - Ifwe delete from EMP_DEPT an employee tuple that happens to represent the lastemployeeworkingforaparticulardepartment,theinformationconcerningthat department is lost from the database
  - ThisproblemdoesnotoccurinthedatabaseofFigure2becauseDEPARTMENT tuples are stored separately.

ModificationAnomalies

- In EMP_DEPT, if we change the value of one of the attributes of a particular department—say, the manager of department 5—we must update the tuples of *all* employees who work in that department; otherwise, the database will become inconsistent
- If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would bewrong

Guideline2

- Designthebaserelationschemassothatnoinsertion,deletion,ormodification anomalies are present in the relations
- Ifanyanomaliesarepresent,notethemclearlyandmakesurethattheprogramsthat update the database will operate correctly
- Thesecondguidelineisconsistentwithand,inaway,arestatementofthefirstguideline
- Theseguidelinesmaysometimeshavetobeviolated inordertoimprovethe performance of certain queries.

NULLValuesinTuples

- Ifmanyoftheattributesdonotapplytoalltuplesintherelation,weendupwithmany NULLs in those tuples
  - thiscanwastespaceatthestoragelevel
  - mayleadtoproblemswithunderstandingthemeaningoftheattributes
  - mayalsoleadtoproblemswithspecifyingJOINoperations
  - howtoaccountforthemwhenaggregateoperationssuchasCOUNTorSUMare applied
- SELECTandJOINoperationsinvolvecomparisons; ifNULLvaluesarepresent,the results may become unpredictable.
- Moreover,NULLscanhavemultipleinterpretations,suchasthefollowing:
  - Theattribute*doesnotapply*tothistuple.Forexample,Visa_statusmaynotapply to U.S. students.
  - Theattributevalueforthistupleis*unknown*.Forexample,theDate_of_birthmay be unknown for an employee.
  - Thevalueis*knownbutabsent*;thatis,ithasnotbeenrecordedyet.Forexample, the Home_Phone_Number for an employee may exist, but may not be available and recorded yet.

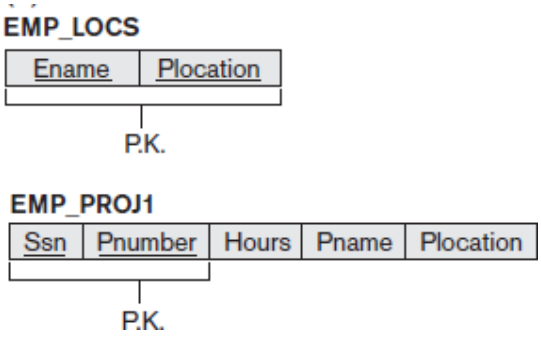
Guideline 3

- As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL
- If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation
- Using space efficiently and avoiding joins with NULL values are the two overridingcriteriathat determine whetherto include thecolumnsthat may have NULLs in a relation or to have a separate relation for those columns with the appropriate keycolumns

- For example, if only 15 percent of employees have individual offices,there is little justification for including an attribute Office_number in the EMPLOYEE relation; rather,a relation EMP_OFFICES(Essn, Office_number) can be created to include tuples for only the employees with individual offices.

GenerationofSpuriousTuples

- Consider the two relation schemas EMP_LOCS and EMP_PROJ1 which can be used instead of the single EMP_PROJ



- A tuple in EMP_LOCSmeans that the employeewhose name is Enameworks on *some project* whose location is Plocation
- A tuple in EMP_PROJ1 refers to the fact that the employee whose Social Security number is Ssn works Hours per week on the project whose name, number, and location are Pname, Pnumber, and Plocation.

EMP_LOCS

Ename	Plocation
Smith, John B.	Bellaire
Smith, John B.	Sugarland
Narayan, Ramesh K.	Houston
English, Joyce A.	Bellaire
English, Joyce A.	Sugarland
Wong, Franklin T.	Sugarland
Wong, Franklin T.	Houston
Wong, Franklin T.	Stafford
Zelaya, Alicia J.	Stafford
Jabbar, Ahmad V.	Stafford
Wallace, Jennifer S.	Stafford
Wallace, Jennifer S.	Houston
Borg, James E.	Houston

EMP_PROJ1

Ssn	Pnumber	Hours	Pname	Plocation
123456789	1	32.5	ProductX	Bellaire
123456789	2	7.5	ProductY	Sugarland
666884444	3	40.0	ProductZ	Houston
453453453	1	20.0	ProductX	Bellaire
453453453	2	20.0	ProductY	Sugarland
333445555	2	10.0	ProductY	Sugarland
333445555	3	10.0	ProductZ	Houston
333445555	10	10.0	Computerization	Stafford
333445555	20	10.0	Reorganization	Houston
999887777	30	30.0	Newbenefits	Stafford
999887777	10	10.0	Computerization	Stafford
987987987	10	35.0	Computerization	Stafford
987987987	30	5.0	Newbenefits	Stafford
987654321	30	20.0	Newbenefits	Stafford
987654321	20	15.0	Reorganization	Houston
888665555	20	NULL	Reorganization	Houston

- Supposethat weusedEMP_PROJ1andEMP_LOCSasthebaserelationsinsteadof EMP_PROJ. This produces a particularly bad schema design because we cannot recover the information that was originally in EMP_PROJ from EMP_PROJ1 and EMP_LOCS
- IfweattemptaNATURALJOINoperationonEMP_PROJ1andEMP_LOCS,theresult produces many more tuples than the original set of tuples inEMP_PROJ
- Additionaltuplesthat werenot inEMP_PROJarecalled **spurioustuples**becausethey represent spurious information that is not valid.
- Thespurioustuplesaremarkedbyasterisks(*)

	Ssn	Pnumber	Hours	Pname	Plocation	Ename
	123456789	1	32.5	ProductX	Bellaire	Smith, John B.
*	123456789	1	32.5	ProductX	Bellaire	English, Joyce A.
	123456789	2	7.5	ProductY	Sugarland	Smith, John B.
*	123456789	2	7.5	ProductY	Sugarland	English, Joyce A.
*	123456789	2	7.5	ProductY	Sugarland	Wong, Franklin T.
	666884444	3	40.0	ProductZ	Houston	Narayan, Ramesh K.
*	666884444	3	40.0	ProductZ	Houston	Wong, Franklin T.
*	453453453	1	20.0	ProductX	Bellaire	Smith, John B.
	453453453	1	20.0	ProductX	Bellaire	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Smith, John B.
	453453453	2	20.0	ProductY	Sugarland	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	2	10.0	ProductY	Sugarland	Smith, John B.
*	333445555	2	10.0	ProductY	Sugarland	English, Joyce A.
	333445555	2	10.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	3	10.0	ProductZ	Houston	Narayan, Ramesh K.
	333445555	3	10.0	ProductZ	Houston	Wong, Franklin T.
	333445555	10	10.0	Computerization	Stafford	Wong, Franklin T.
*	333445555	20	10.0	Reorganization	Houston	Narayan, Ramesh K.
	333445555	20	10.0	Reorganization	Houston	Wong, Franklin T.

⋮

- Decomposing EMP_PROJ into EMP_LOCS and EMP_PROJ1 is undesirable because when we JOIN them back using NATURAL JOIN, we do not get the correct original information

- This is because in this case Plocation is the attribute that relates EMP_LOCS and EMP_PROJ1, and Plocation is neither a primary key nor a foreign key in either EMP_LOCS or EMP_PROJ1.

Guideline4

- Design relation schemas sothat they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated
- Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurioustuples.

FunctionalDependencies

- Formaltool for analysis of relational schemasthatenables usto detect and describe some of the problems in precise terms

DefinitionofFunctional Dependency

- A functional dependency is a constraint between two sets of attributes from thedatabase.
- Given a relation R, a set of attributes X in R is said to **functionally determine** another attribute Y, also in R, (written X → Y) if and only if each X value is associated with at most one Y value.
- X is the determinant set and Y is the dependent attribute. Thus, given a tuple and the values of the attributes in X, one can determine the corresponding value of the Y attribute.
- The abbreviation for functional dependency is FD or f.d. The set of attributes X is called the left-hand side of the FD, and Y is called the right-handside.
- Afunctionaldependencyisapropertyofthesemanticsormeaningoftheattributes.
- The database designers will use their understanding of the semantics of the attributes of R to specify the functional dependencies that should hold on all relation states (extensions) r of R.

- ConsidertherelationschemaEMP_PROJ;

EMP_PROJ

SSN	PNUMBER	HOURS	ENAME	PNAME	PLOCATION

- From the semantics of the attributes and the relation, we know that the following functional dependencies should hold:

a. Ssn→Ename

b. Pnumber→{Pname,Plocation}

c. {Ssn,Pnumber}→Hours

- Thesefunctionaldependenciesspecifythat

  (a) thevalueofanemployee'sSocialSecuritynumber(Ssn)uniquely determines the employee name (Ename)

  (b) thevalueofaproject'snumber(Pnumber)uniquelydetermines the project name (Pname) and location (Plocation),and

  (c) acombinationofSsnandPnumbervaluesuniquelydetermines the number of hours the employee currently works on the project per week (Hours).

- Alternatively,wesaythatEnameisfunctionallydeterminedby(orfunctionallydependent on) Ssn, or *given a value of Ssn, we know the value of Ename,* and soon.
- Relationextensionsr(R)thatsatisfythefunctionaldependencyconstraintsarecalled **legalrelationstates**(or**legalextensions**)ofR
- Afunctionaldependencyisapropertyoftherelationschema R,notofaparticularlegal relation state r of R
- Therefore,anFDcannotbeinferredautomaticallyfromagivenrelationextensionrbut must be defined explicitly by someone who knows the semantics of the attributes ofR

DiagrammaticnotationfordisplayingFDs

- EachFDisdisplayedasahorizontalline
- The left-hand-sideattributes ofthe FDareconnected byverticallines totheline representing the FD
- Theright-hand-sideattributesareconnectedbythelineswitharrowspointingtowardthe attributes.

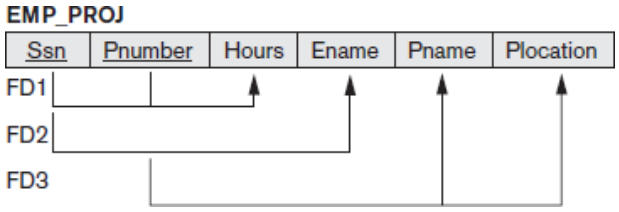


Fig:diagrammaticnotationfordisplayingFDs

Example:

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

- ThefollowingFDs*mayhold*becausethefourtuplesinthecurrent extensionhaveno violation of these constraints:
  - *B* →*C*
  - *C* →*B*
  - {*A, B*} →*C*
  - {*A, B*} →*D*
  - {*C, D*}→*B.*
- Thefollowing*donot* holdbecausewealreadyhaveviolationsoftheminthegiven extension*:*
  - *A*→*B*(tuples1and2violatethisconstraint)
  - *B*→*A*(tuples2and3violatethisconstraint)
  - *D*→*C*(tuples3and4violateit)

NormalFormsBasedonPrimaryKeys

Weassumethata

- Setoffunctionaldependenciesisgivenforeachrelation
- Eachrelationhasadesignatedprimarykey
- Thisinformationcombinedwiththetests(conditions)fornormalformsdrivesthe normalization process for relational schema design
- Firstthreenormalformsforrelationtakesintoaccountallcandidatekeysof a relation rather than the primary key

NormalizationofRelations

- Thenormalizationprocess, asfirstproposedbyCodd(1972a),takesarelationschema through a series of tests to *certify* whether it satisfies a certain **normal form**.
- Initially, Coddproposedthreenormalforms, whichhecalledfirst,second,andthird normal form
- Allthesenormalformsarebasedonasingleanalyticaltool:thefunctionaldependencies among the attributes of a relation
- Afourthnormalform(4NF) andafifthnormalform(5NF) wereproposed,basedonthe concepts of multivalued dependencies and join dependencies, respectively
- **Normalization of data** can be considered a process of analyzing the given relation schemasbasedontheirFDsandprimarykeystoachievethedesirablepropertiesof (1) minimizingredundancyand (2) minimizingtheinsertion,deletion,andupdateanomalies

- It can be considered as a “filtering” or “purification” process to make the design have successively better quality
- Unsatisfactory relation schemas that do not meet certain conditions—the **normal form tests**—are decomposed into smaller relation schemas that meet the tests and hence possess the desirable properties.
- Thus,thenormalizationprocedureprovidesdatabasedesignerswiththefollowing:
  - Aformalframework for analyzingrelation schemas basedontheir keysand on the functional dependencies among their attributes
  - A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desireddegree
- **Definition:** The normalform of a relation refers to the highest normalform condition that it meets, and hence indicates the degree to which it has been normalized

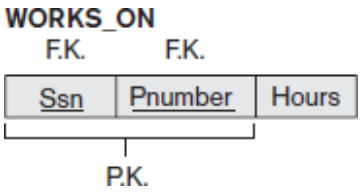
PracticalUseofNormalForms

- Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties
- Database design as practiced inindustry today pays particular attention to normalization only up to 3NF, BCNF, or at most 4NF.
- Thedatabasedesigners*neednot*normalizetothehighestpossiblenormalform
- Relations may be left in a lower normalization status, such as 2NF, for performance reasons
- **Definition: Denormalization** is the process of storing the join of higher normal formrelations as a base relation, which is in a lower normalform.

DefinitionsofKeysandAttributesParticipatinginKeys

- **Superkey:** specifies a uniqueness constraint that notwo distinct tuplesinanystater of R can have the same value
- **key**Kisasuperkeywiththeadditionalpropertythatremovalofanyattributefrom Kwill cause K not to be a superkey any more
- Example:
  - Theattributeset {Ssn} isakeybecausenotwoemployeestuplescan have thesame value for Ssn

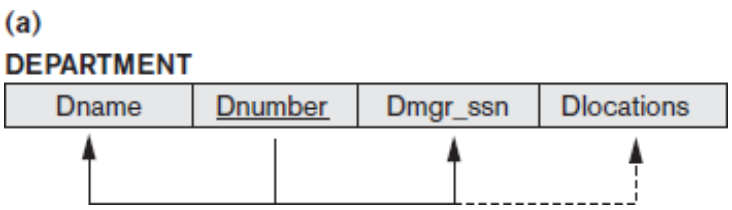
- Anysetofattributesthatincludes Ssn—for example,{Ssn, Name,Address}—isa superkey
- Ifarelationschemahasmorethanonekey,eachiscalleda**candidatekey**
- Oneofthecandidatekeysisarbitrarilydesignatedtobethe **primarykey**,andtheothers are called **secondary keys**
- Inapracticalrelationaldatabase,eachrelationschemamusthaveaprimarykey
- If nocandidatekeyis known for arelation, theentirerelationcan betreated asa default superkey
- For example {Ssn} is the only candidate key for EMPLOYEE, so it is also the primarykey
- **Definition.** An attribute of relation schema *R* is called a **prime attribute** of *R* if it is a member of *some candidate key* of *R*. An attribute is called **nonprime** if it is not a prime attribute—that is, if it is not a member of any candidate key



- In WORKS_ON relation Both Ssn and Pnumber are prime attributes whereas other attributes are nonprime.

FirstNormalForm

- Definedtodisallowmultivaluedattributes,compositeattributes,andtheircombinations
- It states that the domain of an attribute must include only atomic (simple, indivisible) values and that the value of any attribute in a tuple must be a single value from the domain of that attribute
- 1NFdisallowsrelationswithinrelationsorrelationsasattributevalueswithintuples
- Theonlyattributevaluespermittedby1NFaresingle**atomic**(or**indivisible**)**values**.
- ConsidertheDEPARTMENTrelationschemashowninFigure below



- PrimarykeyisDnumber
- Weassumethateachdepartmentcanhaveanumberoflocations

- TheDEPARTMENTschemaandasamplerelationstateareshowninFigurebelow

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

- As we can see, this is not in 1NF because Dlocations is not an atomic attribute, as illustrated by the first tuple in Figure
- TherearetwowayswecanlookattheDlocationsattribute:
  - The domain of Dlocations contains atomic values, but some tuples can have a set of these values. In this case, Dlocations is not functionally dependent on the primary key Dnumber
  - The domain of Dlocations contains sets of values and hence is nonatomic. In thiscase, Dnumber→Dlocations because each set is considered a single member of the attribute domain
- Ineithercase,theDEPARTMENTrelationisnotin 1NF

Therearethreemaintechniquestoachievefirstnormalformforsucharelation:

1. Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key Dnumber of DEPARTMENT. The primary key of this relation is the combination {Dnumber, Dlocation}. A distinct tuple in DEPT_LOCATIONS exists for *each location* of a department. This decomposes the non-1NF relation into two 1NF relations.
2. Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT. In this case, the primary keybecomesthe combination {Dnumber, Dlocation}.This solution hasthe disadvantage of introducing *redundancy* in the relation

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

3. If a maximum number of values is known for the attribute—for example, if it is known that at most three locations can exist for a department—replace the Dlocations attribute by three atomic attributes: Dlocation1, Dlocation2, and Dlocation3.ThissolutionhasthedisadvantageofintroducingNULLvaluesif most departments have fewer than three locations. Querying on this attribute becomes more difficult; forexample, consider how you would write the query: Listthe departments that have 'Bellaire' as one of their locations in thisdesign.

- Of the three solutions, the first is generally considered best because it does not suffer from redundancy and it is completely general, having no limit placed on a maximum number of values
- Firstnormalformalsodisallowsmultivaluedattributesthatarethemselvescomposite.
- Thesearecalled**nestedrelations**becauseeachtuplecanhavearelationwithinit.

(a)

EMP_PROJ

		Projs	
Ssn	Ename	Pnumber	Hours

- FigureaboveshowshowtheEMP_PROJrelationcouldappearifnestingisallowed
- Each tuple represents an employee entity, and a relation PROJS(Pnumber, Hours) *within each tuple* represents the employee's projects and the hours per week that employee works on each project.
- TheschemaofthisEMP_PROJrelationcanberepresentedasfollows:

  EMP_PROJ(Ssn, Ename, {PROJS(Pnumber, Hours)})
- Ssn is the primary key of the EMP_PROJ relation and Pnumber is the **partial** key of the nested relation; that is, within each tuple, the nested relation must have unique values of Pnumber
- To normalize this into 1NF, we remove the nested relation attributes into a new relation and *propagate the primary key* into it; the primary key of the new relation will combinethe partial key with the primary key of the original relation
- Decomposition and primary key propagation yield the schemas EMP_PROJ1 and EMP_PROJ2,

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

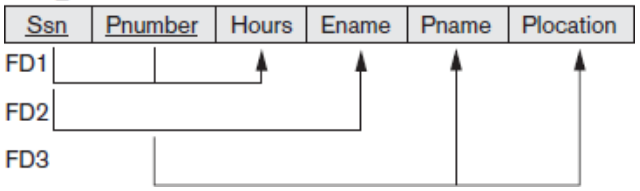
EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, Alicia J.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

SecondNormalForm

- **Secondnormalform(2NF)**isbasedontheconceptoffullfunctionaldependency
- A functional dependency X → Y is a **full functional dependency** if removal of any attribute A from X means that the dependency does not hold any more; that is, for any attribute A ε X, (X – {A}) does not functionally determineY
- A functional dependency X→Y is a **partial dependency** if some attribute A ε X can be removed from X andthe dependency still holds; that is, for some A ε X, (X – {A}) →Y

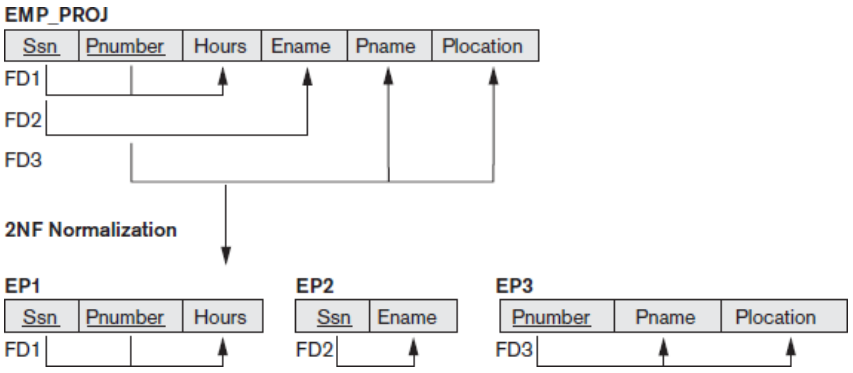
EMP_PROJ



- Intheabovefigure,{Ssn, Pnumber}→Hoursisafulldependency(neither Ssn→ Hours nor Pnumber→Hours holds)
- {Ssn,Pnumber}→EnameispartialbecauseSsn→Enameholds
- **Definition.** ArelationschemaRisin2NF ifeverynonprimeattributeAinRisfully functionally dependent on the primary key of R

- Thetestfor2NFinvolvestestingforfunctionaldependencieswhoseleft-handside attributes are part of the primary key
- Iftheprimarykeycontainsasingleattribute,thetestneednotbeappliedatall

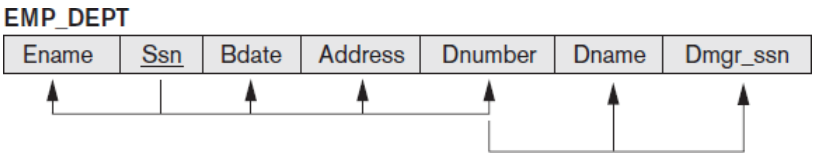
- TheEMP_PROJrelationisin1NFbutisnotin2NF.
- The nonprime attribute Ename violates 2NF because of FD2, as do the nonprime attributes Pname and Plocation because of FD3
- The functional dependencies FD2 and FD3 make Ename, Pname, and Plocationpartially dependent onthe primary key{Ssn,Pnumber} of EMP_PROJ, thus violating the 2NF test.
- If a relation schema is not in2NF, it can be *second normalized* or *2NF normalized* into a number of 2NF relations in which **nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent.**
- Therefore, the functional dependencies FD1, FD2, and FD3 lead to the decomposition of EMP_PROJ into the three relation schemas EP1, EP2, and EP3 shown inFigure below, each of which is in 2NF.



ThirdNormalForm

- **Transitivefunctionaldependency**

  A functional dependency *X→Y* in a relation schema *R* is a **transitive dependency** if there exists a set of attribute Z that are neither a primary nor a subset of any key of *R(*candidate key) and both X →Z and Y →Z holds
- **Example:**

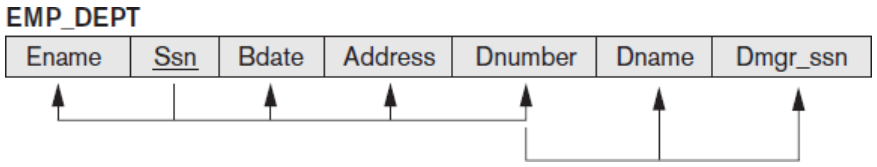


- **SSN→DMGRSSNisatransitiveFD**sinceSSN→DNUMBERandDNUMBER →DMGRSSNhold

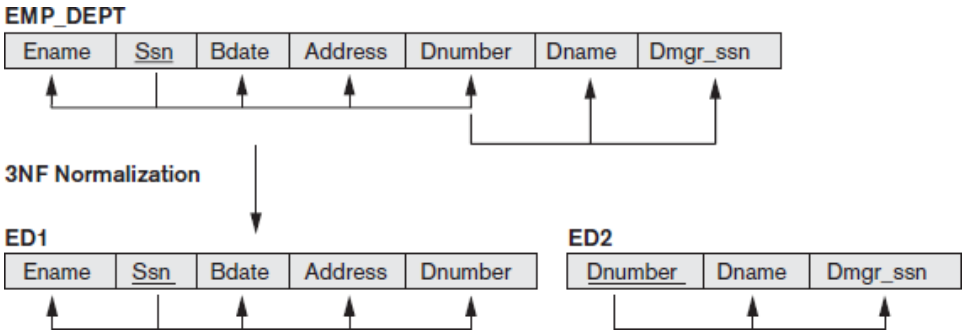
Dnumberisneitherakeyitselfnorasubsetofthekeyof EMP_DEPT

- **SSN→ENAMEisnon-transitive**sincethereisnosetofattributesXwhere SSN →X and X →ENAME

- **Definition: A relation schema R is in third normal form (3NF) if it is in 2NF *and* no non-prime attribute A in R is transitively dependent on the primary key**
- The relation schema EMP_DEPT is in 2NF, since no partial dependencies on a keyexist. However, EMP_DEPT is not in 3NF because of the transitive dependency of Dmgr_ssn (and also Dname) on Ssn via Dnumber



- We can normalize EMP_DEPT by decomposing it into the two 3NF relation schemas ED1 and ED2



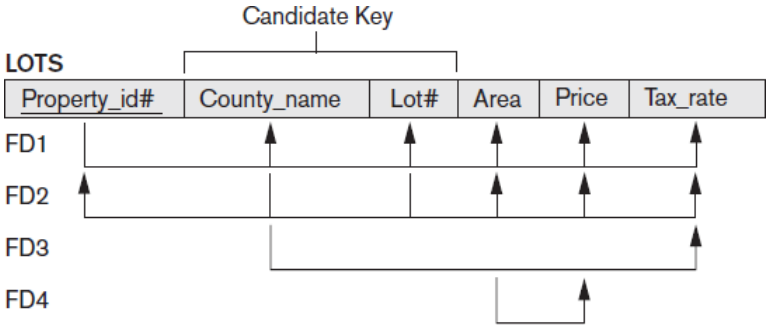
- ED1andED2representindependententityfactsaboutemployeesanddepartments
- ANATURALJOINoperationonED1andED2willrecovertheoriginalrelation EMP_DEPT without generating spurious tuples
- ProblematicFD
  - Left-handsideispartofprimarykey
  - Left-handsideisanon-keyattribute
- 2NF and 3NF normalization remove these problem FDs by decomposing the original relation into new relations
- In general, we want to design our relation schemas so that they have neither partial nor transitive dependencies because these types of dependencies cause the update anomalies

Table 15.1 Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

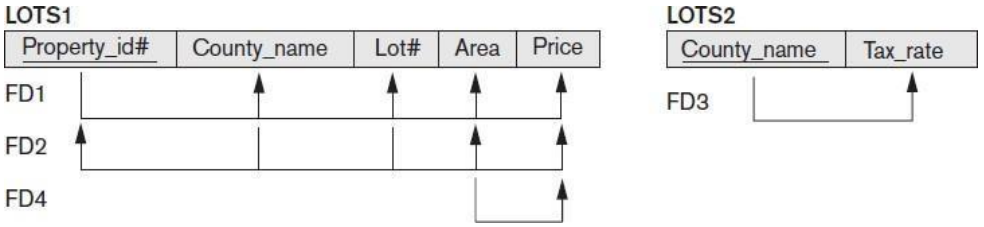
Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

GeneralDefinitionofSecondandThirdNormalForm

- Takesintoaccountallcandidatekeysofarelationintoaccount
- **Definition of 2NF:** A relation schema R is in **second normal form (2NF)** if every nonprime attribute A in R is not partially dependent on any key of R
- Consider the relation schema LOTS which describes parcels of land for sale in various counties of a state
- Suppose that there are two candidate keys: Property_id# and {County_name, Lot#}; that is, lot numbers are unique only within each county, but Property_id# numbers are unique across counties for the entire state.

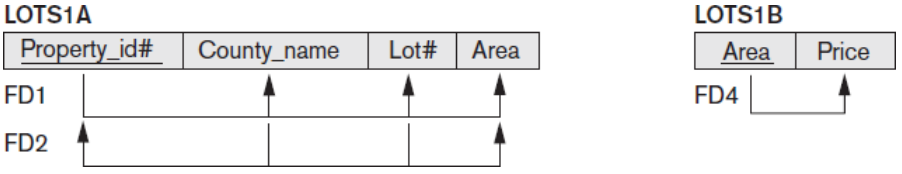


- Basedonthetwocandidatekeys Property_id# and {County_name,Lot#},thefunctional dependencies FD1 and FD2 hold
  - FD1:Property_id→{County_name,Lot#,Area,Price,Tax_rate}
  - FD2:{County_name,Lot#}→{Property_id,Area,Price,Tax_rate}
  - FD3:County_name→Tax_rate
  - FD4:Area→Price
- We chooseProperty_id#astheprimarykey,butnospecialconsiderationwillbegivento this key over the other candidate key
- FD3saysthatthetaxrateisfixedforagivencounty(doesnotvarylotbylotwithinthe same county)
- FD4saysthatthepriceof alotisdeterminedby itsareregardlessofwhich countyitis in.
- TheLOTSrelationschemaviolatesthegeneraldefinitionof2NFbecauseTax_rateis partially dependent on the candidate key {County_name, Lot#}, due to FD3
- TonormalizeLOTSinto2NF,wedecomposeitintothetworelationsLOTS1andLOTS2



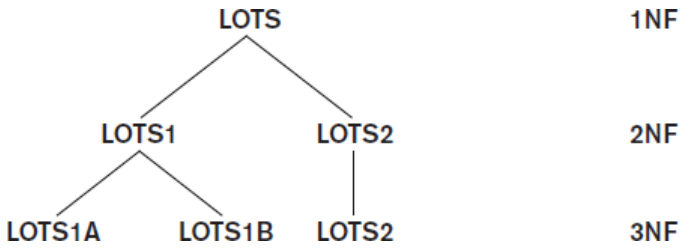
- Weconstruct LOTS1 byremovingthe attribute Tax_ratethat violates 2NFfromLOTS and placing it with County_name (the left-hand side of FD3 that causes the partial dependency) into another relation LOTS2.
- BothLOTS1andLOTS2arein2NF.

- **Definition of 3NF:** A relation schema R is in **third normal form (3NF)** if, whenever a nontrivial functional dependency X→A holds in R,either (a)X is a superkey of R, or (b)A is a prime attribute of R
- Accordingtothisdefinition,LOTS2isin3NF
- FD4 in LOTS1 violates 3NF because Area is not a superkey and Price is not a prime attribute in LOTS1
- To normalize LOTS1 into 3NF, wedecompose it into the relation schemas LOTS1A and LOTS1B



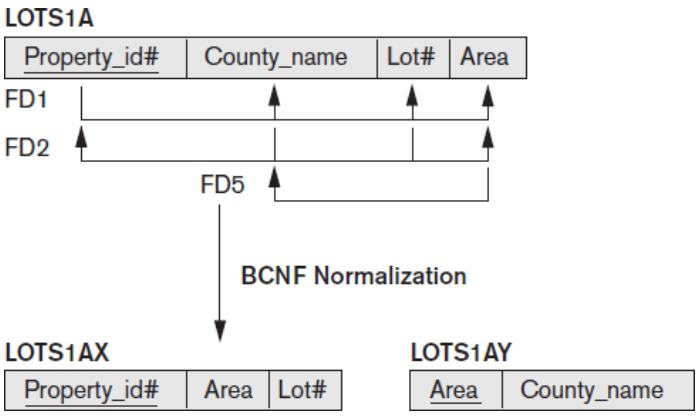
- Weconstruct LOTS1A by removing the attributePrice that violates 3NFfrom LOTS1and placing it with Area (the lefthand side of FD4 that causes the transitive dependency) into another relation LOTS1B.

- BothLOTS1AandLOTS1Barein3NF



Boyce-CoddNormalForm

- **Boyce-Codd normal form (BCNF)** was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF
- Every relation in BCNF is also in 3NF; however, a relation in 3NF is not necessarily in BCNF
- **Definition.** A relation schema R is in **BCNF** if whenever a nontrivial functional dependency X→A holds in R, then X is a superkey of R
- The formal definition of BCNF differs from the definition of 3NF in that condition (b) of 3NF, which allows *A* to be prime, is absent from BCNF. That makes BCNF a stronger normal form compared to 3NF
- In our example, FD5 violates BCNF in LOTS1A because AREA is not a superkey of LOTS1A
- FD5satisfies3NFinLOTS1AbecauseCounty_nameisaprimeattribute(conditionb), but this condition does not exist in the definition ofBCNF
- We can decompose LOTS1A into two BCNF relations LOTS1AX and LOTS1AY. This decomposition loses the functional dependency FD2 because its attributes no longer coexist in the same relation after decomposition.



- Inpractice,mostrelationschemasthatarein3NFarealsoin BCNF
- Onlyif*X*→*A*holdsinarelationschema*R*with*X*notbeingasuperkey *and**A*beinga prime attribute will *R* be in 3NF but not in BCNF
- Example:considertherelationTEACHwiththefollowingdependencies:

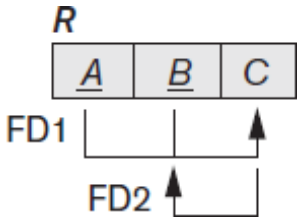
TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

FD1:{Student,Course}→Instructor

FD2:Instructor→Course--meansthateachinstructorteachesonecourse

- {Student,Course}isacandidatekeyforthisrelation
- ThedependenciesshownfollowthepatterninFigurebelowwithStudent as *A*,Course as *B*, and Instructor as *C*



- Hencethisrelationisin3NFbutnotBCNF
- Decompositionofthisrelationschemaintotwoschemasisnotstraightforwardbecauseit may be decomposed into one of the three following possible pairs:
  1. R1(Student,Instructor)andR2(Student,Course)
  2. R1(Course,Instructor)andR2(Course,Student)
  3. R1(Instructor,Course)andR2(Instructor,Student)
- It is generally not sufficient to check separately that each relation schema in the database is, say, in BCNF or 3NF
- Rather, the process of normalization through decomposition must also confirm the existence of additional properties that the relational schemas, taken together, should possess. These would include two properties:
  - The **nonadditivejoinorlosslessjoin property**,whichguaranteesthatthespurious tuple generation problem does not occur with respect to the relation schemascreated after decomposition.
  - The **dependency preservation property**, which ensures that each functional dependency is represented in some individual relation resulting afterdecomposition.

- We are not able to meet the functional dependency preservation ,but we must meet the non additive join property
- **NonadditiveJoinTestforBinary Decomposition:**

  AdecompositionD={R_1,R_2}ofRhasthelosslessjoinpropertywithrespecttoaset of functional dependencies F on R if and only if either

  - TheFD((R_1∩R_2)→(R_1-R_2)isinF^+or
  - TheFD ((R_1∩R_2) →(R_2-R_1)isinF^+
- Thethirddecompositionmeetsthetest

  R_1∩R_2 is Instructor

  R_1-R_2isCourse
- Hence,theproper decompositionof TEACHintoBCNFrelationsis:

  TEACH1(Instructor,Course) and TEACH2(Instructor,Student)
- In general, a relation R not in BCNF can be decomposed so as to meet the nonadditive join prorperty by the following procedure. It decomposes R successively into set of relations that are in BCNF:

  Let R be the relation not in BCNF, let X R, and let X → A be the FD that causes violation of BCNF. R may be decomposed into two relations:

  R–A XA

  IfeitherR-AorXAisnotinBCNF,repeatthe process

MultivaluedDependencyandFourthNormalForm

- Forexample,considertherelationEMPshowninFigurebelow:

EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

- A tuple in this EMP relation represents the fact that an employee whose name is Ename works on the project whose name is Pname and has a dependent whose name is Dname
- Anemployeemayworkonseveralprojectsandmayhaveseveraldependents
- Theemployee'sprojectsanddependentsareindependentofoneanother

- To keep the relation state consistent, andto avoid any spurious relationship between the two independent attributes, we must have a separate tuple to represent every combination of an employee's dependent and an employee'sproject
- In therelationstate showntheEMP, the employeeSmithworks on twoprojects 'X' and 'Y' and has two dependents 'John' and 'Anna' and therefore there are 4tuples to represent these facts together
- The relation EMP is an **all-key relation** (with key made up of all attributes) and therefore no f.d's and as such qualifies to be a BCNF relation
- There is a redundancy in the relation EMP-the dependent information is repeated for every project and project information is repeated for everydependent
- To address this situation, the concept of multivalued dependency(MVD) was proposed and based on this dependency, the fourth normal form wasdefined
- **Multivalued dependencies** are a consequence of 1NF which disallows an attribute in a tuple to have a set of values, and the accompanying process of converting an unnormalized relation into 1NF
- Informally, whenever two independent 1:N relationships are mixed in the same relation, R(A, B, C), an MVD may arise

FormalDefinitionofMultivaluedDependency

Definition. A multivalued dependency X→→Y specified on relation schema R, where X and Y are bothsubsets of R, specifies thefollowing constraint on anyrelation state r of R: If twotuples t1 andt2exist in r suchthat t1[X] =t2[X], thentwo tuplest3 andt4 shouldalso exist in r withthe following properties where we use Z to denote (R – (X ∪ Y))

- t3[X]=t4[X]=t1[X]=t2[X].
- t3[Y]=t1[Y]andt4[Y]=t2[Y].
- t3[Z]=t2[Z] andt4[Z]=t1[Z].

EMP

Ename	Pname	Dname
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

Let X= Ename, Y=Pname
t1[Ename]=t2[ename]=Smith
Z=(EMP-(EnameuPname))
=Dname

- t3(Ename)=t4(Ename)=t1(Ename)=t2(Ename)=Smith

- t3(Pname)=t1(Pname)=Xandt4(Pname)=t2(Pname)=Y
- t3(Dname)=t2(Dname)=Annaandt4(Dname)=t1(Dname)=John
- Whenever $X \rightarrow\rightarrow Y$holds,we saythat *X***multidetermines** *Y*. Becauseof the symmetry in the definition, whenever $X \rightarrow\rightarrow Y$ holds in*R*, so does $X \rightarrow\rightarrow Z$. Hence, $X \rightarrow\rightarrow Y$ implies $X \rightarrow\rightarrow Z$, and therefore it is sometimes written as $X \rightarrow\rightarrow Y|Z$.
- AnMVDX→→YinRis called**trivialMVD** if

  (a) Yis asubsetofX,or

  (b) X ∪ Y=R

EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

- Forexample,therelationEMP_PROJECTShasthetrivialMVD

  Ename→→Pname
- AnMVDthatsatisfiesneither(a)nor(b)iscalleda **nontrivialMVD**
- If we have a nontrivial MVD in a relation, we may have to repeat values redundantly in the tuples
- In the EMP relation the values 'X' and 'Y' of Pname are repeated with each value of Dname (or, by symmetry, the values 'John' and 'Anna' of Dname are repeated with each value of Pname)
- Thisredundancyisclearlyundesirable.
- We now present the definition of **fourth normal form (4NF)**, which is violated when a relation has undesirable multivalued dependencies, and hence can be used to identify and decompose such relations
- **Definition: A relation schema *R* is in 4NF with respect to a set of dependencies *F* (that includes functional dependencies and multivalued dependencies) if, for every *nontrivial* multivalued dependency $X \rightarrow\rightarrow Y$ in F^+ *X* is a superkey for *R***
- The process of normalizing a relation involving the nontrivial MVDs that is not in 4NF consists of decomposing it so that each MVD is represented by a separate relationwhere it becomes a trivial MVD

EMP_PROJECTS

Ename	Pname
Smith	X
Smith	Y

EMP_DEPENDENTS

Ename	Dname
Smith	John
Smith	Anna

- WedecomposeEMPintoEMP_PROJECTSandEMP_DEPENDENTS
- Both EMP_PROJECTS and EMP_DEPENDENTS are in 4NF, because the MVDs Ename →→ Pname in EMP_PROJECTS and Ename →→ Dname in EMP_DEPENDENTS are trivial MVDs
- No other nontrivial MVDs hold in either EMP_PROJECTS or EMP_DEPENDENTS. No FDs hold in these relation schemas either

- Wecanstatethefollowingpoints:
  - Anall-keyrelationisalwaysinBCNFsinceithasnoFDs
  - An all-key relation suchas theEMP, which has no FDs but hastheMVDEname→→ Pname | Dname, is not in 4NF
  - ArelationthatisnotinNFduetoanontrivialMVDmust bedecomposedtoconvertit into a set of relations in 4NF
  - ThedecompositionremovestheredundancycausedbytheMVD

JoinDependenciesandFifthNormalForm

- A **join dependency**(**JD**), denoted by JD(R1, R2,..., Rn), specified on relation schema R, specifies a constraint on the states r of R. The constraint states that every legalstate r of R should have a nonadditive join decomposition into R1, R2, ..., Rn. Hence, for every such r we have

$$* (\pi_{R_1}(r), \pi_{R_2}(r), ..., \pi_{R_n}(r)) = r$$

- A join dependency JD(R1, R2,..., Rn), specifiedon relationschemaR, isa **trivial** JD if one of the relation schemas Ri in JD(R1, R2, ..., Rn) is equal to R.

Fifthnormalform(project-joinnormalform)

- A relation schema *R* is in **fifth normal form (5NF)** (or **project-join normal form(PJNF)**) with respect to a set *F* of functional, multivalued, and join dependencies if, for every nontrivial join dependency JD(*R*1, *R*2, ..., *Rn*) in F^+ every R_i is a superkey of*R*.
- Adatabaseissaidtobein5NF,ifandonlyif,
  - It'sin4NF

- If we can decompose table further to eliminate redundancy and anomaly, and when we re-join the decomposed tables by means of candidate keys, we should not be losing the original data or any new record set should not arise. In simple words, joining two or more decomposed table should not lose records nor create new records.

SUPPLY

Sname	Part_name	Proj_name
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

Fig:TherelationSUPPLYwithnoMVDsisin4NFbutnotin5NFifithastheJD(*R*1,*R*2, *R*3)

R_1

Sname	Part_name
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R_2

Sname	Proj_name
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R_3

Part_name	Proj_name
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

Fig:DecomposingtherelationSUPPLYintothe5NFrelations*R*1,*R*2,*R*3.

Chapter2:NormalizationAlgorithms

InferenceRulesforFunctionalDependencies

- LetFbethesetoffunctionaldependenciesthatarespecifiedonrelationschemaR
- Theschemadesignerspecifiesthefunctionaldependenciesthataresemantically obvious
- Numerousotherfunctionaldependenciesholdinallegalrelationinstancesamongsets of attributes that can be derived from and satisfy the dependencies inF
- ThoseotherdependenciescanbeinferredordeducedfromtheFDsinF.
- **Forexample:**
    - If each department has one manager, so that Dept_no uniquely determines Mgr_ssn (Dept_no → Mgr_ssn), and a manager has a unique phone number called Mgr_phone (Mgr_ssn→Mgr_phone),
    - Thenthesetwodependencies together implythat
      Dept_no → Mgr_phone
    - Thisisan**inferredFD**andneed*not* beexplicitlystated inadditionto the two given FDs.

- **Definition.** Formally, the set of all dependencies that include *F* as well as all dependenciesthatcanbeinferredfrom *F*iscalledthe**closure**of *F*; itisdenotedbyF^+.
- Forexample,supposethatwespecifythefollowingset*F*ofobviousfunctional dependencies on the relation schema EMP_DEPT

EMP_DEPT

ENAME	SSN	BDATE	ADDRESS	DNUMBER	DNAME	DMGRSSN

- F ={
  Ssn→{Ename,Bdate,Address,Dnumber}, Dnumber
  → {Dname, Dmgr_ssn}
  }
- Someoftheadditionalfunctionaldependenciesthatwecan *infer*from*F*arethe following:
    - Ssn→{Dname,Dmgr_ssn}
    - Ssn→Ssn

- Dnumber→Dname

- AnFDX→Yis**inferredfrom** aset of dependenciesFspecifiedonRifX→Yholdsin every legal relation state r of R
- TheclosureF+ofFisthesetofallfunctionaldependenciesthatcanbeinferredfromF
- Set of**inferencerules**canbeusedtoinfer newdependenciesfromagivensetof dependencies
- We usethenotation *F*|=*X*→ *Y*todenotethatthefunctionaldependency*X*→*Y*is inferred from the set of functional dependencies *F*
- weuseanabbreviatednotationwhendiscussingfunctionaldependencies. We concatenate attribute variables and drop the commas forconvenience
- TheFD{*X*,*Y*}→*Z*isabbreviatedto*XY*→*Z*, andtheFD{*X*,*Y*,*Z*}→{*U*, *V*}isabbreviated to *XYZ* → *UV*.
- ThreerulesIR1throughIR3arewell-knowninferencerulesforfunctionaldependencies.
- TheyareproposedbyArmstrongandhenceknownas**Armstrong'saxioms**
  - IR1(reflexiverule):If*X*⊇*Y*,then*X*→*Y*.
  - IR2(augmentationrule):{*X*→*Y*} |=*XZ*→*YZ*.
  - IR3(transitiverule):{*X*→*Y*,*Y*→*Z*}|=*X*→*Z*.
- Thereflexiverule(IR1)statesthatasetof attributesalwaysdeterminesitselforanyof its subsets, which is obvious.
- BecauseIR1generatesdependenciesthatarealwaystrue,suchdependenciesare called *trivial*.
- Formally,afunctionaldependency*X*→*Y*is**trivial**if*X*⊇*Y*;otherwise,itis**nontrivial**.
- Theaugmentationrule(IR2) saysthat addingthesamesetofattributestobothte left- and right-hand sides of a dependency results in another valid dependency
- AccordingtoIR3,functionaldependenciesaretransitive
- TherearethreeotherinferencerulesthatfollowfromIR1,IR2andIR3.Theyare:
  - IR4(decomposition,orprojective,rule):{*X*→*YZ*}|=*X*→*Y*
  - IR5(union,oradditive,rule):{*X*→*Y*,*X*→*Z*}|=*X*→*YZ*
  - IR6(pseudotransitiverule):{*X*→*Y*,W*Y*→*Z*} |=W*X*→*Z*
- The decomposition rule (IR4) says that we can remove attributes from the right-hand side of a dependency; applying this rule repeatedly can decompose the FD *X*→{*A*1, *A*2, ...,*An*}intothesetofdependencies{*X*→*A*1,*X*→*A*2, ...,*X*→*An*}.
- The union rule (IR5) allows us to do the opposite; we can combine a set of dependencies{*X*→*A*1,*X*→*A*2,...,*X*→*An*}intothesingleFD*X*→{*A*1,*A*2,...,*An*}.
- Thepseudotransitiverule(IR6)allowsustoreplaceasetofattributes *Y*onthelefthand side of a dependency with another set *X* that functionally determines *Y*, and canbe

derivedfromIR2andIR3ifweaugmentthefirstfunctionaldependency$X \rightarrow Y$with$W$ (theaugmentationrule)andthenapplythetransitiverule.

- In other words, the set of dependencies F+, which we called the **closure** of F, can be determined from F by using only inference rules IR1 through IR3.
- A systematic way to determine these additional functional dependencies is first to determine each set of attributes X that appears as a left-hand side of some functional dependency in F and then to determine the set of *all attributes* that are dependent on$X$.
- **Definition.** For each such set of attributes X, we determine the set X^+ of attributes that are functionally determined by X based on F; X^+ is called the **closure of X underF**.
- Algorithm16.1canbeusedtocalculate X^+.

Algorithm 16.1. Determining X^+, the Closure of X under F

Input: A set F of FDs on a relation schema R, and a set of attributes X, which is a subset of R.

$X^+ := X$;
repeat
    old$X^+ := X^+$;
    for each functional dependency $Y \rightarrow Z$ in F do
        if $X^+ \supseteq Y$ then $X^+ := X^+ \cup Z$;
until (X^+ = oldX^+);

- Algorithm16.1startsbysettingX^+ toalltheattributesinX.
- ByIR1,weknowthatalltheseattributesarefunctionallydependentonX.
- Using inference rules IR3 and IR4, we add attributes to X^+, using each functional dependency in F.
- We keep going through all the dependencies in F (the repeat loop) until no more attributes are added to X^+ during a complete cycle (of the for loop) through the dependencies in F.
- For example, consider the relation schema EMP_PRO. From the semantics of the attributes, we specify the following set F of functional dependencies that should hold on EMP_PROJ:

  F= { Ssn→ Ename,
  Pnumber→{Pname, Plocation},
  {Ssn,Pnumber}→ Hours}
- UsingAlgorithm16.1,wecalculatethefollowingclosuresetswithrespectto$F$:
  - {Ssn}$^+$={Ssn, Ename}

- {Pnumber}$^+$={Pnumber,Pname,Plocation}
- {Ssn,Pnumber}$^+$={Ssn,Pnumber,Ename,Pname,Plocation,Hours}

EquivalenceofSetsofFunctionalDependencies

Definition: A set of functional dependencies F is said to **cover** another set of functional dependencies E if every FD in E is also in F^+; that is, if every dependency in E can be inferred from F; alternatively, we can say that E is **covered by** F.

Definition: Two sets of functional dependencies E and F are **equivalent** if $E^+ = F^+$. Therefore, equivalence means that every FD in E can be inferred from F, and every FDin F can be inferred from E; thatis, E is equivalent to F if both the conditions—E coversF and F covers E—hold.

SetsofFunctionalDependencies

Asetoffunctionaldependencies*F*tobe**minimal**if itsatisfiesthefollowing conditions:

1. .Everydependencyin*F*hasasingleattributeforitsright-handside.
2. Wecannotreplaceanydependency$X \rightarrow A$in*F*withadependency $Y \rightarrow A$, where$Y$isapropersubset of$X$, andstillhaveasetofdependenciesthat is equivalent to *F*.

5We cannot remove any dependencyfrom *F* and still have a set ofdependencies that is equivalent to *F*.

Algorithm 16.2. Finding a Minimal Cover F for a Set of Functional Dependencies E

Input: A set of functional dependencies E.

1. Set $F := E$.
2. Replace each functional dependency $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ in F by the n functional dependencies $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n$.
3. For each functional dependency $X \rightarrow A$ in F
   for each attribute B that is an element of X
   if $\{ \{F - \{X \rightarrow A\}\} \cup \{ (X - \{B\}) \rightarrow A\} \}$ is equivalent to F
   then replace $X \rightarrow A$ with $(X - \{B\}) \rightarrow A$ in F.
4. For each remaining functional dependency $X \rightarrow A$ in F
   if $\{F - \{X \rightarrow A\}\}$ is equivalent to F,
   then remove $X \rightarrow A$ from F.

- Step2placesFDsinacanonicalformforsubsequenttesting
- Step3constitutesremovalofanextraneousattributeBcontained intheleft-handsideX of a functional dependency X->A from F when possible
- Step4constitutesremovalof aredundantfunctionaldependency x->A from Fwhen possible
- Example1:LetthegivensetofFDs be*E*:{*B*→*A*,*D*→*A*,*AB*→*D*}.We havetofindthe minimal cover of *E*.
  - Allabovedependenciesareincanonicalform(thatis,theyhaveonlyoneattributeon the right-hand side), so we have completed step 1 of Algorithm and can proceed to step 2
  - In step 2 we need to determine if *AB*→*D* has any redundant attribute on the left-hand side; that is, can it be replaced by *B*→*D* or *A*→*D*?
  - SinceB→A,byaugmentingwith *B*onbothsides(IR2), wehave *BB*→*AB*,or*B*→*AB* (i).However,*AB*→*D*asgiven (ii).
  - Hence by the transitive rule (IR3), we get from (i)and (ii),*B* → *D*. Thus *AB*→*D* maybe replaced by *B*→*D*.
  - We now have a set equivalent to original *E*, say *E*: {*B*→*A*, *D*→*A*, *B*→*D*}. No further reduction is possible in step 2 since all FDs have a single attribute on the left-handside.
  - Instep3welookforaredundant FDin*E*.Byusingthetransitiveruleon*B*→*D*and*D* →*A*,wederive*B*→*A*.Hence*B*→*A*isredundantin*E*andcanbeeliminated.

Algorithm 16.2(a). Finding a Key K for R Given a set F of Functional Dependencies

Input: A relation R and a set of functional dependencies F on the attributes of R.

1. Set $K := R$.
2. For each attribute A in K
   {compute $(K - A)^+$ with respect to F;
   if $(K - A)^+$ contains all the attributes in R, then set $K := K - \{A\}$ };

  - Therefore,theminimalcoverof*E*is{*B*→*D*,*D*→*A*}.

Westart bysetting *K*toalltheattributesof *R*; wethenremoveoneattributeatatimeandcheck whether the remaining attributes still form a superkey.

- Algorithm16.2(a)determinesonly *onekey*outofthepossiblecandidatekeysfor *R*;thekey returned depends on the order in which attributes are removed from *R* in step2.

PropertiesofRelationalDecompositions

Universal relation schema

- **Universal relation schema***R* = {*A*1, *A*2, ..., *An*}includes *all* the attributes ofthe database
- **universalrelationassumption**:everyattributenameisunique
- Theset*F*offunctionaldependenciesthat shouldholdontheattributesof *R*isspecified by the database designers
- Usingthefunctionaldependencies,thealgorithmsdecomposetheuniversalrelation schema *R* into a set of relation schemas *D*={*R*1,*R*2,...,*Rm*}thatwillbecometherelationaldatabaseschema;*D*is calleda **decomposition**of*R*.

AttributePreservationconditionofaDecomposition

- Eachattributein*R*willappearinat least onerelationschema $R_i$inthedecompositionso that no attributes are *lost*; formally, we have

$$\bigcup_{i=1}^{m} R_i = R$$

- Anothergoalofdecompositionistohaveeachindividualrelation R_i inthedecomposition D be in BCNF or 3NF
- Additional properties of decomposition are needed to prevent from generating spurious tuples

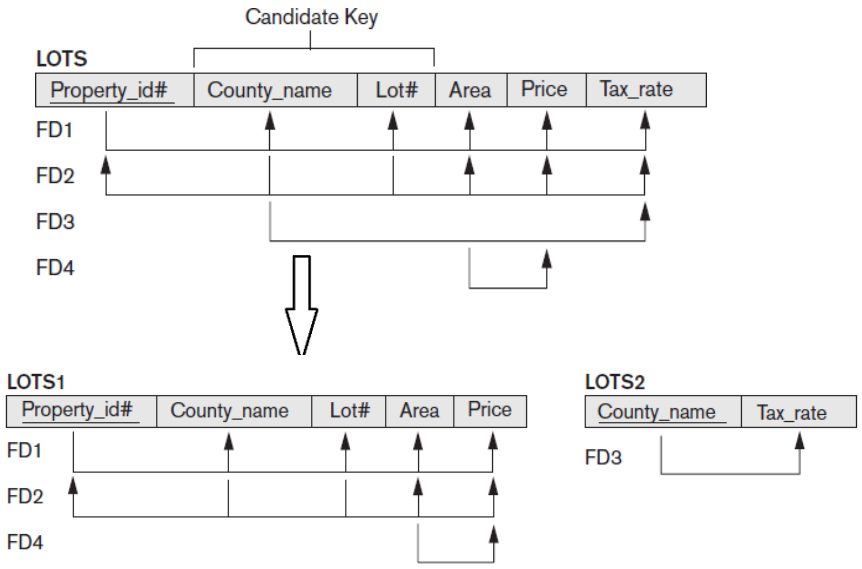
DesirablePropertiesofDecompositions

- Notalldecompositionofaschemaareuseful
- Werequiretwopropertiestobesatisfied:
  i) DependencyPreservationProperty
  ii) Nonadditive(Lossless)JoinProperty

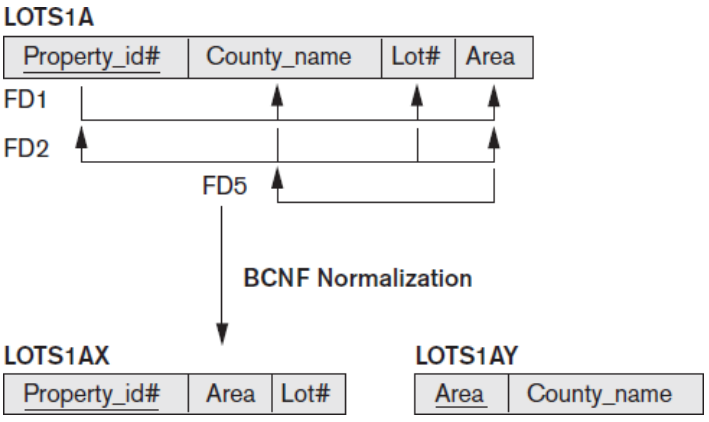
DependencyPreservationProperty

- Each functional dependency $X \rightarrow Y$ specified in *F* either appeared directly in one of the relation schemas R_i in the decomposition *D* or could be inferred from the dependencies that appear in some R_i

- We wanttopreservethedependenciesbecauseeachdependencyin*F*representsa constraint on the database
- Ifone ofthe dependencies is not represented in some individual relation *Ri* of the decomposition,wecannotenforcethisconstraintbydealingwithanindividualrelation
- Itisnotnecessarythattheexactdependenciesspecifiedin*F*appearthemselvesin individual relations of the decomposition *D*.
- Itissufficientthattheunionofthedependenciesthatholdontheindividualrelationsin *D*beequivalentto*F*
- **Example:**DependencyPreservingDecomposition



- **Example:**DecompositionthatdoesnotPreserveDependency



Nonadditive(Lossless)JoinProperty

- The nonadditive join property ensures that no spurious tuples result after the application of PROJECT and JOIN operations
- Theterm**lossydesign**refertoadesignthatrepresentsalossofinformation
- If a decomposition does not have the lossless join property, we may get additional spurious tuples after the PROJECT (π) and NATURAL JOIN (*) operations are applied; these additional tuples represent erroneous or invalid information

Algorithm 16.3. Testing for Nonadditive Join Property

Input: A universal relation R, a decomposition $D = \{R_1, R_2, ..., R_m\}$ of R, and a set F of functional dependencies.

Note: Explanatory comments are given at the end of some of the steps. They follow the format: (* *comment* *).

1. Create an initial matrix S with one row i for each relation R_i in D, and one column j for each attribute A_j in R.
2. Set $S(i, j) := b_{ij}$ for all matrix entries. (* each b_{ij} is a distinct symbol associated with indices (i, j) *).
3. For each row i representing relation schema R_i
   {for each column j representing attribute A_j
   {if (relation R_i includes attribute A_j) then set $S(i, j) := a_j$;};}; (* each a_j is a distinct symbol associated with index (j) *).

4. Repeat the following loop until a *complete loop execution* results in no changes to S
{for each functional dependency $X \rightarrow Y$ in F
{for all rows in S *that have the same symbols* in the columns corresponding to attributes in X
{make the symbols in each column that correspond to an attribute in Y be the same in all these rows as follows: If any of the rows has an a symbol for the column, set the other rows to that *same* a symbol in the column. If no a symbol exists for the attribute in any of the rows, choose one of the b symbols that appears in one of the rows for the attribute and set the other rows to that same b symbol in the column ;} ; } ;};
5. If a row is made up entirely of a symbols, then the decomposition has the nonadditive join property; otherwise, it does not.

Example

(a) R = {Ssn, Ename, Pnumber, Pname, Plocation, Hours} $D = \{R_1, R_2\}$
R_1 = EMP_LOCS = {Ename, Plocation}
R_2 = EMP_PROJ1 = {Ssn, Pnumber, Hours, Pname, Plocation}

F = {Ssn → Ename; Pnumber → {Pname, Plocation}; {Ssn, Pnumber} → Hours}

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	b_{11}	a_2	b_{13}	b_{14}	a_5	b_{16}
R_2	a_1	b_{22}	a_3	a_4	a_5	a_6

(No changes to matrix after applying functional dependencies)

(b)

EMP

Ssn	Ename

PROJECT

Pnumber	Pname	Plocation

WORKS_ON

Ssn	Pnumber	Hours

(c) R = {Ssn, Ename, Pnumber, Pname, Plocation, Hours} $D = \{R_1, R_2, R_3\}$
R_1 = EMP = {Ssn, Ename}
R_2 = PROJ = {Pnumber, Pname, Plocation}
R_3 = WORKS_ON = {Ssn, Pnumber, Hours}

F = {Ssn → Ename; Pnumber → {Pname, Plocation}; {Ssn, Pnumber} → Hours}

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	b_{32}	a_3	b_{34}	b_{35}	a_6

(Original matrix S at start of algorithm)

	Ssn	Ename	Pnumber	Pname	Plocation	Hours
R_1	a_1	a_2	b_{13}	b_{14}	b_{15}	b_{16}
R_2	b_{21}	b_{22}	a_3	a_4	a_5	b_{26}
R_3	a_1	~~b_{32}~~ a_2	a_3	~~b_{34}~~ a_4	~~b_{35}~~ a_5	a_6

(Matrix S after applying the first two functional dependencies; last row is all "a" symbols so we stop)

TestingBinaryDecompositionsfortheNonadditiveJoinProperty

Property NJB (Nonadditive Join Test for Binary Decompositions). A decomposition $D = \{R_1, R_2\}$ of R has the lossless (nonadditive) join property with respect to a set of functional dependencies F on R *if and only if* either

- The FD $((R_1 \cap R_2) \rightarrow (R_1 - R_2))$ is in F^+, or
- The FD $((R_1 \cap R_2) \rightarrow (R_2 - R_1))$ is in F^+

AlgorithmsforRelationalDatabaseSchemaDesign

Twoalgorithmsforcreatingarelationaldecompositionfromuniversalrelation

1. Thefirstalgorithmdecomposesauniversalrelationintodependencypreserving3NF relations that also possess the nonadditive join property
2. ThesecondalgorithmdecomposesauniversalrelationschemaintoBCNFschemasthat possess the nonadditive join property

Dependency-PreservingandNonadditive(Lossless)Join Decomposition into 3NF Schemas

Algorithm16.4. RelationalSynthesisinto3NFwithDependencyPreservationandNonadditive Join Property

- **Input:**AuniversalrelationR andasetoffunctionaldependenciesFontheattributes of R.

1. FindaminimalcoverGforF(useAlgorithm16.2).
2. For each left-hand-side X of a functional dependency that appears in G, create a relation schema in D with attributes $\{X \cup \{A1\} \cup \{A2\} \ldots \cup \{Ak\}\}$, where $X \rightarrow A1$, $X \rightarrow A2$, ..., $X \rightarrow Ak$ are the only dependencies in G with X as left-hand-side (X is the key of thisrelation)

5 If none of the relation schemas in D contains a key of R, then create one more relation schema in D that contains attributes that form a key of R

6 Eliminate redundant relations from the resulting set of relations in the relational database schema. A relation R is considered redundant if R is a projection of another relation S inthe schema; alternately, R is subsumed by S

- **Example:**Considerthefollowinguniversalrelation:

  U(Emp_ssn,Pno,Esal,Ephone,Dno,Pname,Plocation)

- Emp_ssn, Esal, Ephone refer to the Social Security number, salary, and phone number of theemployee. Pno, Pname,andPlocationrefertothenumber, name, andlocationof the project. Dno is department number.
- Thefollowingdependenciesarepresent:
  -FD1:Emp_ssn→{Esal,Ephone,Dno}
  -FD2:Pno→{Pname, Plocation}
  -FD3:Emp_ssn,Pno→{Esal,Ephone,Dno,Pname,Plocation}
- By virtue of FD3, the attribute set {Emp_ssn, Pno} represents a key of the universal relation.
- Hence *F*, the set of given FDs includes {Emp_ssn → Esal, Ephone, Dno; Pno→Pname, Plocation; Emp_ssn, Pno→Esal, Ephone, Dno, Pname, Plocation}.
- By applying the minimal cover , in step 3 we see that Pno is a redundant attribute in Emp_ssn, Pno → Esal, Ephone, Dno. Moreover, Emp_ssn is redundant in Emp_ssn, Pno→Pname, Plocation.
- HencetheminimalcoverconsistsofFD1andFD2only
- Minimalcover*G*:{Emp_ssn→Esal,Ephone,Dno;Pno→Pname,Plocation}
- By applying Algorithm 16.4 to the above Minimal cover *G*, we get a 3NF design consistingof two relations with keys Emp_ssn and Pno as follows:

  *R*1(Emp_ssn,Esal,Ephone, Dno)

  *R*2(Pno,Pname,Plocation)
- In step 3, we generate a relation corresponding to the key(Emp_ssn,Pno) of U. Hence, the resulting design contains:

  *R*1(Emp_ssn,Esal,Ephone,Dno) *R*2

  (Pno, Pname, Plocation)

  R3(Emp_ssn, Pno)

Thisdesignachievesboththedesirablleproperties ofdependencypreservationandnon additive join

NonadditiveJoinDecompositionintoBCNFSchemas

Algorithm16.5. RelationalDecompositionintoBCNFwithNonadditive Join Property

- **Input:**AuniversalrelationRandasetoffunctionaldependenciesFontheattributesof*R*.

  1. Set *D*:={*R*};

  2. WhilethereisarelationschemaQinDthatisnotinBCNFdo

  {

  choosearelationschemaQinDthatisnotinBCNF;

find a functional dependency $X \rightarrow Y$ in Q that violates BCNF;
replace Q in D by two relation schemas $(Q - Y)$ and $(X \cup Y)$;
};

- Each time through the loop in Algorithm 16.5, we decompose one relation schema Q that is not in BCNF into two relation schemas.
- According to Property NJB for binary decompositions and Claim 2, the decomposition D has the nonadditive join property
- At the end of the algorithm, all relation schemas in D will be in BCNF
- Example: TEACH relation schema decomposed into TEACH1(Instructor, Student) and TEACH2(Instructor, Course) because the dependency FD2 Instructor→Course violates BCNF.
- In step 2 of Algorithm 16.5, it is necessary to determine whether a relation schema Q is in BCNF or not.
- whenever a relation schema Q has a BCNF violation, there exists a pair of attributes A and B in Q such that $\{Q - \{A, B\}\} \rightarrow A$; by computing the closure $\{Q - \{A, B\}\}^+$ for each pair of attributes $\{A, B\}$ of Q, and checking whether the closure includes A (or B), we can determine whether Q is in BCNF.

Dependency-Preserving and Nonadditive (Lossless) Join Decomposition into 3NF Schemas

- It is *not possible to have all three of the following:*
  (1) guaranteed nonlossy design,
  (2) guaranteed dependency preservation, and
  (3) all relations in BCNF
- The first condition is a must and cannot be compromised.
- The second condition is desirable, but not a must, and may have to be relaxed if we insist on achieving BCNF.
- Now we give an alternative algorithm where we achieve conditions 1 and 2 and only guarantee 3NF.
- A simple modification to Algorithm 16.4, shown as Algorithm 16.6, yields a decomposition D of R that does the following:
  - Preserves dependencies
  - Has the nonadditive join property

- Issuchthateachresultingrelationschemainthedecompositionisin3NF

Algorithm 16.6. Relational Synthesis into 3NF with Dependency Preservation and Nonadditive Join Property

Input: A universal relation R and a set of functional dependencies F on the attributes of R.

1. Find a minimal cover G for F (use Algorithm 16.2).
2. For each left-hand-side X of a functional dependency that appears in G, create a relation schema in D with attributes $\{X \cup \{A_1\} \cup \{A_2\} \ldots \cup \{A_k\}\}$, where $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_k$ are the only dependencies in G with X as left-hand-side (X is the key of this relation).
3. If none of the relation schemas in D contains a key of R, then create one more relation schema in D that contains attributes that form a key of R.[7] (Algorithm 16.2(a) may be used to find a key.)
4. Eliminate redundant relations from the resulting set of relations in the relational database schema. A relation R is considered redundant if R is a projection of another relation S in the schema; alternately, R is subsumed by S.

- Step3involvesidentifyingakey*K*of*R*.Algorithm16.2(a)canbeusedtoidentifyakey*K*of *R*basedonthesetofgivenfunctionaldependencies*F*.

Example 1 of Algorithm 16.6. Let us revisit the example given earlier at the end of Algorithm 16.4. The minimal cover G holds as before. The second step produces relations R_1 and R_2 as before. However, now in step 3, we will generate a relation corresponding to the key {Emp_ssn, Pno}. Hence, the resulting design contains:

R_1 (Emp_ssn , Esal, Ephone, Dno)
R_2 (Pno, Pname, Plocation)
R_3 (Emp_ssn, Pno)

This design achieves both the desirable properties of dependency preservation and nonadditive join.

AboutNulls,DanglingTuples,andAlternativeRelationalDesigns

ProblemswithNULLValuesandDanglingTuples

- Whenever a relational database schema is designed in which two or more relations are interrelated via foreign keys, particular care must be devoted to watching for potential NULL values in foreign keys.
- This can cause unexpected loss of information in queries that involve joins on thatforeign key.

- IfNULLsoccurinotherattributes,suchasSalary,theireffect onbuilt-infunctionssuch as SUM and AVERAGE must be carefully evaluated.

Dangling tuples mayoccurifwecarryadecompositiontoofar.Supposethatwedecompose the EMPLOYEE relation in Figure16.2(a)further into EMPLOYEE_1 and EMPLOYEE_2,shown in Figure 16.3(a) and 16.3(b)

(a)

EMPLOYEE

Ename	Ssn	Bdate	Address	Dnum
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4

DEPARTMENT

Dname	Dnum	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

(b)

Ename	Ssn	Bdate	Address	Dnum	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

(c)

Ename	Ssn	Bdate	Address	Dnum	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	NULL	NULL	NULL
Benitez, Carlos M.	888665555	1963-01-09	7654 Beech, Houston, TX	NULL	NULL	NULL

Figure16.2: IssueswithNULL-valuejoins.(a)SomeEMPLOYEEtupleshaveNULLforthejoinattribute Dnum (b)ResultofapplyingNATURALJOINtotheEMPLOYEEandDEPARTMENTrelations.(c)Resultofapplying LEFT OUTER JOIN to EMPLOYEE and DEPARTMENT.

(a) EMPLOYEE_1

Ename	Ssn	Bdate	Address
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX
Benitez, Carlos M.	888665555	1963-01-09	7654 Beech, Houston, TX

(b) EMPLOYEE_2

Ssn	Dnum
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1
999775555	NULL
888664444	NULL

(c) EMPLOYEE_3

Ssn	Dnum
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1

Figure16.3: Thedanglingtupleproblem.(a)TherelationEMPLOYEE_1 (includesallattributesofEMPLOYEEfrom Figure 16.2(a) except Dnum). (b) The relation EMPLOYEE_2 (includes Dnum attribute with NULL values). (c) The relation EMPLOYEE_3 (includes Dnum attribute but does not include tuples for which Dnum has NULL values).

- If we apply the NATURAL JOIN operation to EMPLOYEE_1 and EMPLOYEE_2, we get the original EMPLOYEE relation.
- we mayusethe alternative representation, shown in Figure 16.3(c), where we *do not include a tuple*in EMPLOYEE_3 if the employee has not been assigned a department (instead of including a tuple with NULL for Dnum as in EMPLOYEE_2).
- If we use EMPLOYEE_3 instead of EMPLOYEE_2 and apply a NATURAL JOIN on EMPLOYEE_1 and EMPLOYEE_3, the tuples for Berger and Benitez will not appear in the result; these are called **dangling tuples** in EMPLOYEE_1 because they are represented in only one of the two relations that represent employees, and hence are lost if we apply an (INNER) JOIN operation.

OtherDependenciesandNormalForms

InclusionDependencies

Inclusiondependencieswereredefinedinordertoformalizetwotypesofinterrelational constraints:

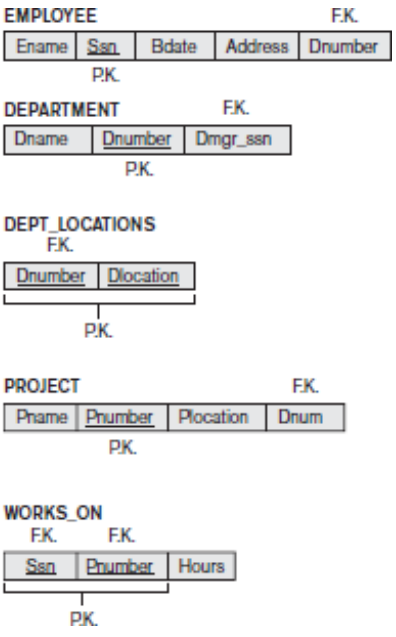
- Theforeignkey(orreferentialintegrity)constraintcannotbespecifiedasafunctionalor multivalued dependency because it relates attributes across relations.
- Theconstraintbetweentworelationsthatrepresentaclass/subclassrelationshipalsohas no formal definition in terms of the functional,multivalued, and joindependencies.

Definition. An **inclusion dependency** $R.X < S.Y$ between two sets of attributes—X of relation schema R, and Y of relation schema S—specifies the constraint that, at any specific time when r is a relation state of R and s a relation state of S, we must have

$$\pi_X(r(R)) \subseteq \pi_Y(s(S))$$

- The subset relationship does not necessarily have to be a proper subset. Obviously, the sets of attributes on which the inclusion dependency is specified—*X* of *R* and *Y* of *S*—must have the same number of attributes.
- Inaddition,thedomainsforeachpairofcorrespondingattributesshouldbecompatible.
- For example, we can specify the following inclusion dependencies on the relationalvschema in Figure 15.1:

Figure 15.1
A simplified COMPANY relational database schema.



- DEPARTMENT.Dmgr_ssn<EMPLOYEE.Ssn
- WORKS_ON.Ssn<EMPLOYEE.Ssn
- EMPLOYEE.Dnumber<DEPARTMENT.Dnumber
- PROJECT.Dnum<DEPARTMENT.Dnumber
- WORKS_ON.Pnumber<PROJECT.Pnumber
- DEPT_LOCATIONS.Dnumber<DEPARTMENT.Dnumber

- Alltheprecedinginclusiondependenciesrepresent**referentialintegrityconstraints**.

- Wecanalsouseinclusiondependenciestorepresent **class/subclass.**Forexample,inthe relational schema of Figure 9.6, we can specify the following inclusion dependencies:
  - EMPLOYEE.Ssn<PERSON.Ssn
  - ALUMNUS.Ssn<PERSON.Ssn
  - STUDENT.Ssn<PERSON.Ssn

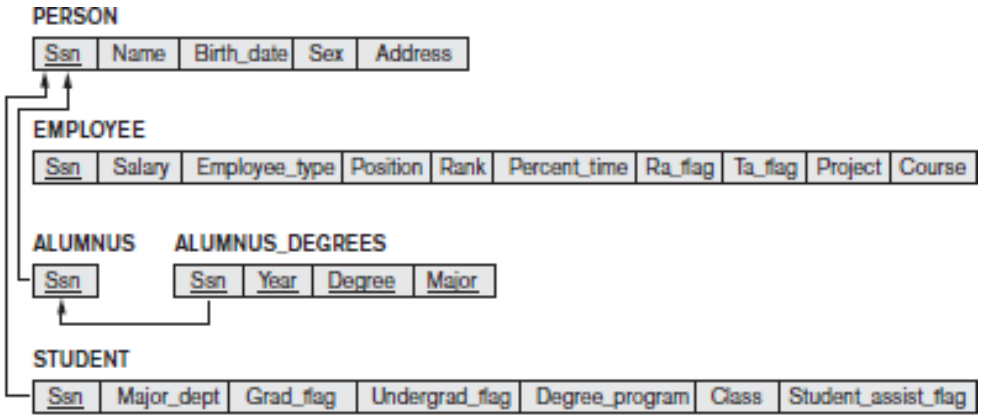


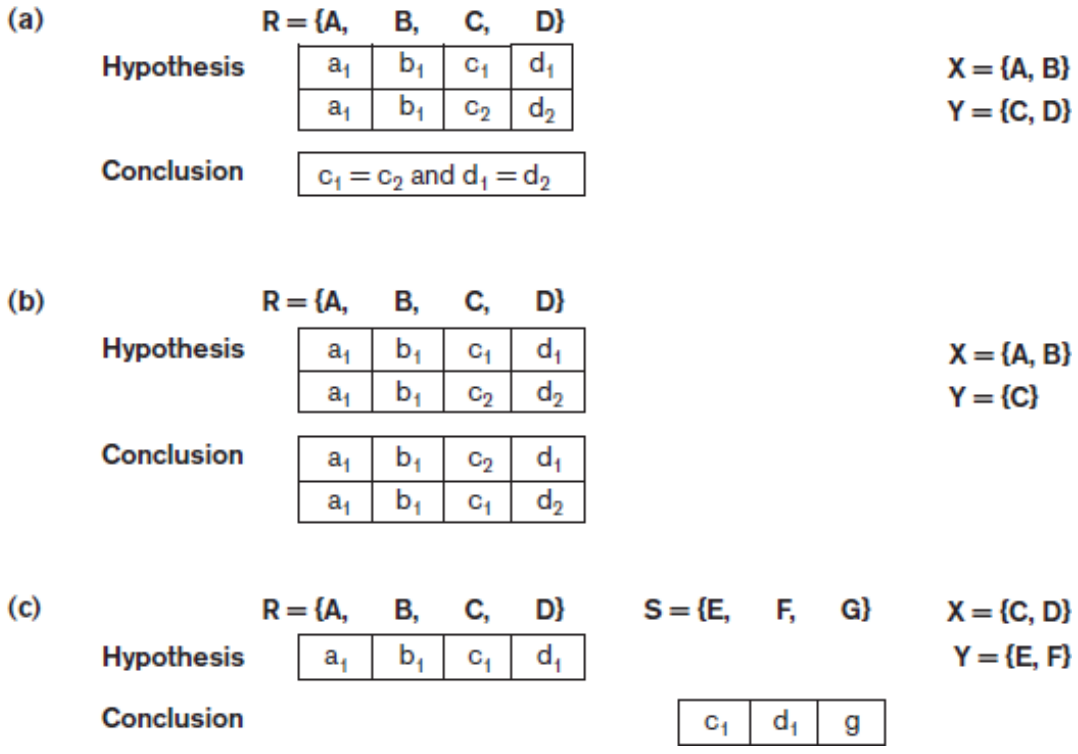
Figure 9.6
Mapping the EER specialization lattice in Figure 8.8 using multiple options.

TemplateDependencies

- Templatedependenciesprovideatechniqueforrepresentingconstraintsinrelationsthat typically have no easy and formaldefinitions.
  - Therearetwotypesoftemplates:
    - tuple-generatingtemplatesand
    - constraintgeneratingtemplates.
  - Atemplateconsistsofanumberof**hypothesistuples**thataremeanttoshowanexample of the tuples that may appear in one or morerelations.
  - Theotherpartofthetemplateisthe**templateconclusion**.
  - Fortuple-generatingtemplates,theconclusionisa*setoftuples*thatmustalsoexistinthe relations if the hypothesis tuples are there.

- For constraint-generating templates, the template conclusion is a *condition* that must hold on the hypothesis tuples.
- Using constraint generating templates, we are able to define **semantic constraints**—those that are beyond the scope of the relational model in terms of its data definition language and notation.
- Figure 16.5 shows how we may define functional, multivalued, and inclusion dependencies by templates.

Figure 16.5
Templates for some common type of dependencies.
(a) Template for functional dependency X → Y.
(b) Template for the multivalued dependency X →→ Y.
(c) Template for the inclusion dependency R.X < S.Y.



- Figure 16.6 shows how we may specify the constraint that an employee's salary cannot be higher than the salary of his or her direct supervisor on the relation schema EMPLOYEE

Figure 16.6
Templates for the constraint that an employee's salary must be less than the supervisor's salary.

EMPLOYEE = {Name, Ssn, . . . , Salary, Supervisor_ssn}

	a	b	c	d
Hypothesis	e	d	f	g
Conclusion			c < f	

FunctionalDependenciesBasedonArithmeticFunctionsandProcedures

- Sometimessomeattributes in arelationmayberelated viasomearithmetic functionora more complicated functional relationship.
- As long as a unique value of *Y* is associated with every *X*, we can still consider that the FD *X*→*Y* exists.
- Forexample,intherelation
  ORDER_LINE (Order#, Item#, Quantity, Unit_price, Extended_price, Discounted_price)
- each tuple represents an item from an order with a particular quantity, and the price per unit for that item.
- Inthisrelation, (Quantity, Unit_price)→Extended_price bytheformula
  Extended_price = Unit_price * Quantity.
- Hence, there is a unique value for Extended_price for every pair (Quantity, Unit_price), and thus it conforms to the definition of functional dependency.
- Moreover, there may be a procedure that takes into account the quantity discounts, the type of item, and so on and computes a discounted price for the total quantity orderedfor that item.
- Therefore,wecansay
  (Item#, Quantity, Unit_price) → Discounted_price, or
  (Item#,Quantity,Extended_price)→Discounted_price.

Domain-KeyNormalForm

- The idea behind **domain-key normal form** (**DKNF**) is to specify the *ultimate normalform* that takes into account all possible types of dependencies andconstraints.

- A relation schema is said to be in **DKNF** if all constraints and dependencies that should hold on the valid relation states can be enforced simply by enforcing the domain constraints and key constraints on the relation
- For a relation in DKNF, it becomes very straightforward to enforce all database constraints by simply checking that each attribute value in a tuple is of the appropriate domain and that every key constraint is enforced.
- For example, consider a relation CAR(Make, Vin#) (where Vin# is the vehicle identification number) and another relation MANUFACTURE(Vin#,Country) (where Country is the country of manufacture).
- A general constraint may be of the following form: *If the Make is either 'Toyota' or 'Lexus,' then the first character of the Vin# is a 'J' if the country of manufacture is'Japan'; if the Make is 'Honda' or 'Acura,' the second character of the Vin# is a 'J' if the country of manufacture is 'Japan.'*
- There is nosimplified way torepresentsuchconstraintsshortofwriting aprocedure (or general assertions) to test them.
- TheprocedureCOMPUTE_TOTAL_PRICEaboveis anexampleofsuchprocedures needed to enforce an appropriate integrity constraint.

Problem1

Considerthefollowingrelationforpublishedbooks:

BOOK(BookTitle,AuthorName,BookType,ListPrice,AuthorAffiliation, Publisher)

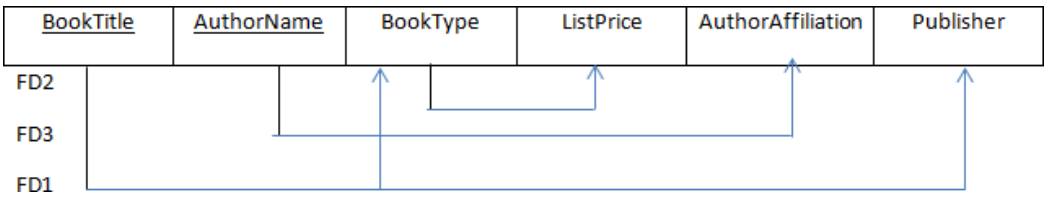
Supposethefollowingdependenciesexist:

- **BookTitle→BookType,Publisher**
- **BookType→ListPrice**
- **AuthorName→AuthorAffiliation**

What normal form is the relation in? explain your answer. Apply normalization until you cannot decomposetherelationsfurther.Statethereasonsbehindeachdecomposition.

Solution:

Therelationisin1NFandnotin2NFasnoattributesarefullyfunctionallydependent on the key (BookTitle and AuthorName). It is also not in 3NF.

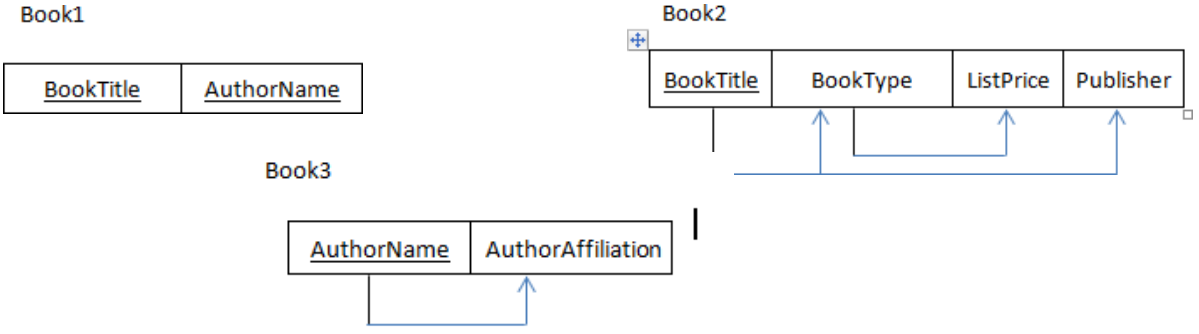


- The relationis notin2NFbecausethepartialDependenciesexist

{BookTitle,AuthorName}→{Publisher,BookType}

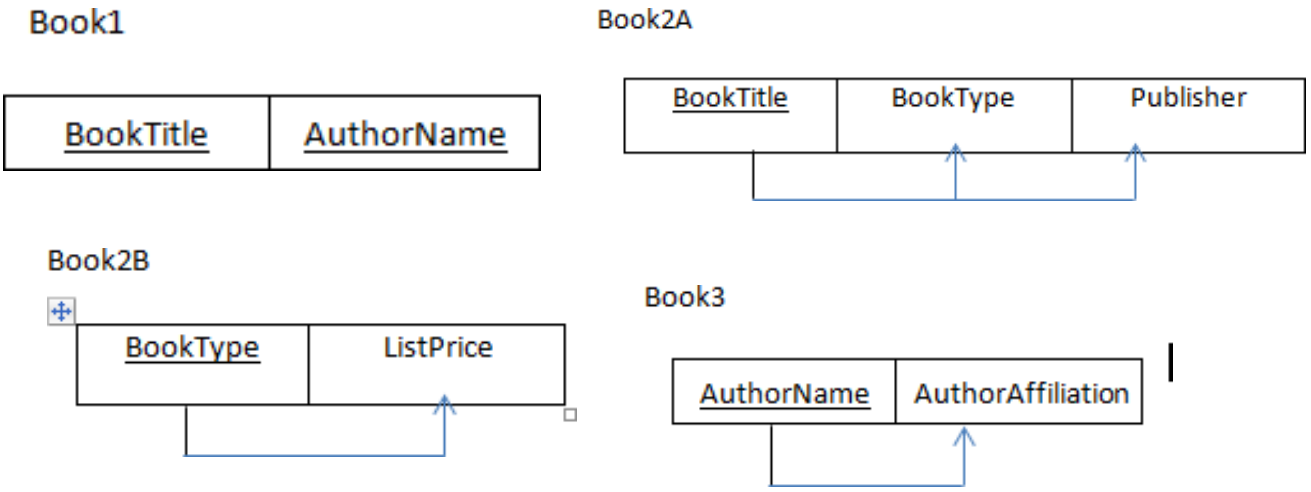
{BookdTitle,AuthorName}→AuthorAffiliation

- Thus,theseattributesarenotfullyfunctionally dependent ontheprimarykeyThe2NF decomposition will eliminate the partial dependencies.
- 2NF decomposition:
  - Book1(BookTitle,AuthorName)
  - Book2(BookTitle,BookType,ListPrice,Publisher)
  - Book3(AuthorName,AuthorAffiliation)



- Therelationsarenotin3NFbecause:
- BookTitle→BookType→ListPrice

  BookTypeisneither akey itselfnorasubsetofakeyandListPriceisnotaprime attribute
- The3NFdecompositionwilleliminatethetransitivedependencyofListprice. 3NF decomposition:
  - Book1(BookTitle,AuthorName)
  - Book2A(BookTitle,BookType,Publisher)
  - Book2B(BookType,ListPrice)
  - Book3(AuthorName,AuthorAffiliation)



Problem2

Considerthefollowingrelation:

CAR_SALE(Car#,DateSold,Salesman#,Commission%,DiscountAmount)
Assume that a car may be sold by multiple salesmen, and hence
{Car#,Salesman#}istheprimarykey.
Additional dependencies are:
Car#→DateSold
Car# → DiscountAmount
DateSold → DiscountAmount
Salesman#→Commission%
Basedonthegivenprimarykey,istherelationin1NF,2NF,3NF? Why or why not?
Howwouldyousuccessivelynormalizeitcompletely?

Solution:

- Therelationisin1NFbecauseallattributevaluesaresingleatomicvalues.
- Therelationisnotin2NFbecause:
  - Car#→DateSold
  - Car#→DiscountAmount
  - Salesman#→Commission%

Thus,theseattributesarenotfullyfunctionallydependentontheprimarykey.

- 2NF decomposition:
  - CAR_SALE1(Car#,DateSold,DiscountAmount)
  - CAR_SALE2(Car#,Salesman#)
  - CAR_SALE3(Salesman#,Commission%)
- Therelationsarenotin3NFbecause:
  - Car#→DateSold→DiscountAmount

DateSoldisneitherakeyitselfnorasubsetofakeyandDiscountAmount is not aprime attribute.

- 3NF decomposition:
  - CAR_SALES1A(Car#,DateSold)
  - CAR_SALES1B(DateSold,DiscountAmount)
  - CAR_SALE2(Car#,Salesman#)
  - CAR_SALE3(Salesman#,Commission%)

AssignmentQuestions

1. Considerthefollowingrelationforpublishedbooks:

   BOOK(BookTitle,AuthorName,BookType,ListPrice,AuthorAffiliation,Publisher)

   Suppose the following dependencies exist:

   BookTitle→BookType,Publisher

   BookType → ListPrice

   AuthorName → AuthorAffiliation

   Whatnormalformistherelationin?Explainyouranswer.

2. Considerthefollowingrelation:

   CAR_SALE(Car#,DateSold,Salesman#,Commission%,DiscountAmount)

   Assume that a car may be sold by multiple salesmen, and hence

   {Car#,Salesman#}istheprimarykey.

   Additional dependencies are:

   Car#→DateSold

Car# → DiscountAmount

DateSold → DiscountAmount

Salesman#→Commission%

Basedonthegivenprimarykey,istherelationin1NF,2NF,3NF? Why or why not?

Howwouldyousuccessivelynormalizeitcompletely?

3. LetR={Ssn,Ename, Pnumber, Pname,Plocation,Hours}and0={RI, R2,R3}where RI = EMP = {Ssn, Ename}
   R2 = PRO] = {Pnumber, Pname, Plocation}
   R3=WORKS-ON={Ssn,Pnumber,Hours}
   ThefollowingfunctionaldependenciesholdonrelationR. F = {Ssn ->Ename; Pnumber -> {Pname, Plocation};
   {Ssn,Pnumber}->Hours}
   ProvethattheabovedecompositionofrelationRhasthelosslessjoinproperty.
4. ConsiderR={ABCDEF} FDS{AB->B->EA->DF}
   Checkwhetherdecompositionislossless.
5. WhatisasetoffunctionaldependenciesFsaidtobeminimal?Giveanalgorithmfor finding a minimal cover G for F.

ExpectedOutcome

- Todesign adatabasewhich willhaveminimum redundancy
- Toapplynormalization tothedesigneddatabase.
- Todecomposethetablesandnormalizethedesignupto4NFand5NFthetablesupto 4NF and 5NF
- Toapply losslessand lossyjoinoperations
- Toapplyinference rulesanddeduceotherrules fromthegivenset.

FurtherReading

1. https://www.smartdraw.com/entity-relationship-diagram/
2. https://en.wikipedia.org/wiki/Database_normalization
3. www.databasteknik.se/webbkursen/relalg-lecture
4. https://technet.microsoft.com/en-us/library/bb264565(v=sql.90).aspx
5. pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/.../Ch16_Overview_Xacts.pdf

Module5

Chapter1:TransactionProcessing

Introduction

The concept of transaction provides a mechanism for describing logical units of database processing. Transactionprocessing systems are systems withlarge databases and hundreds of concurrent users executing database transactions. Examples:

- airlinereservations
- banking
- creditcardprocessing,
- onlineretailpurchasing,
- Stockmarkets,supermarketcheckouts,andmanyotherapplications

These systems require high availability and fast response time for hundreds of concurrentusers. A transaction is typically implemented by a computer program, which includes database commands such as retrievals, insertions, deletions, and updates.

Objectives

- Tostudytransactionproperties
- Tostudy creation ofschedule and maintaining scheduleequivalence.
- Tocheckwhetherthegivenscheduleisserailizableornot.
- Tostudyprotocols usedfor lockingobjects
- Differentiatingbetween 2PLandStrict2PL

IntroductiontoTransactionProcessing

Single-UserversusMultiuserSystems

- Onecriterionforclassifyingadatabasesystemisaccordingtothenumberofuserswho can use the system **concurrently**

Single-UserversusMultiuserSystems

- ADBMSis
- **single-user**
  - atmostoneuseratatimecanusethesystem
  - Eg:PersonalComputerSystem
- **multiuser**
  - manyuserscanusethesystemandhenceaccessthedatabaseconcurrently
  - Eg:Airlinereservationdatabase

- Concurrentaccessis possible becauseof **Multiprogramming.** Multiprogrammingcan be achieved by:
  - interleavedexecution
  - ParallelProcessing
- **Multiprogramming operating systems** execute some commands from one process, then suspend that process and execute some commands from the next process, and so on
- A process is resumed at the point where it was suspended whenever it gets its turn to use the CPU again
- Hence, concurrent execution of processes is actually **interleaved**, as illustratedin Figure 21.1

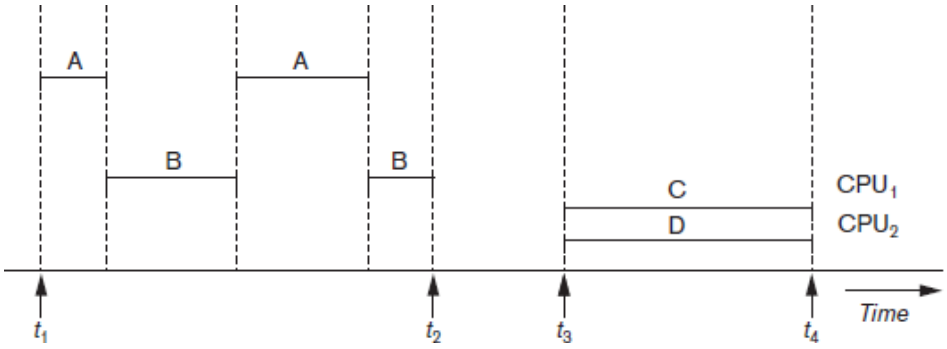


Figure 21.1
Interleaved processing versus parallel processing of concurrent transactions.

- Figure21.1,showstwoprocesses,AandB,executingconcurrentlyinaninterleaved fashion
- InterleavingkeepstheCPUbusywhenaprocessrequiresaninputoroutput(I/O) operation, such as reading a block from disk
- TheCPUisswitchedtoexecuteanotherprocessratherthanremainingidleduringI/O time
- Interleavingalsopreventsalongprocessfromdelayingotherprocesses.
- Ifthecomputersystemhasmultiplehardwareprocessors(CPUs),**parallelprocessing** ofmultipleprocessesispossible,asillustratedbyprocessesCandDinFigure21.1
- Mostofthetheoryconcerningconcurrencycontrolindatabasesisdevelopedintermsof **interleavedconcurrency**
- In a multiuser DBMS, the stored data items are the primary resources that may be accessedconcurrentlybyinteractiveusersorapplicationprograms,whichareconstantly retrieving information from and modifying the database.

Transactions,DatabaseItems,ReadandWriteOperations,andDBMS Buffers

- ATransactionanexecutingprogramthatformsalogicalunitofdatabaseprocessing
- It includes oneor moreDB access operations suchas insertion, deletion,modification or retrieval operation.
- It can be either embedded within an application program using **begin transaction** and **end transaction** statements Or specified interactively via a high level query language such as SQL
- Transactionwhichdonotupdatedatabaseareknownas**readonlytransactions.**
- Transactionwhichdoupdatedatabaseareknownas**readwritetransactions.**
- A **database** is basically represented as a collection of named data items The size of a data item is called its **granularity**.
- A **data item** can be a database record, but it can also be a larger unit such as a whole disk block, or even a smaller unit such as an individual field (attribute) value of some record in the database
- Eachdataitemhasauniquename
- **BasicDBaccessoperationsthatatransactioncanincludeare:**
  - **read_item(X)**:ReadsaDBitemnamedXintoaprogramvariable.
  - **write_item(X)**:WritesthevalueofaprogramvariableintotheDBitemnamedX
- **Executingread_item(X)includethefollowing steps:**
  1. FindtheaddressofthediskblockthatcontainsitemX
  2. Copytheblockintoabufferinmainmemory
  3. CopytheitemXfromthebuffertoprogramvariablenamedX.
- **Executingwrite_item(X)includethefollowing steps:**
  1. FindtheaddressofthediskblockthatcontainsitemX
  2. Copythediskblockintoabufferinmainmemory
  3. CopyitemXfromprogramvariablenamedXintoitscorrectlocationinbuffer.
  4. Storetheupdateddiskblockfrombufferbacktodisk(eitherimmediatelyorlater).
- Decisionofwhentostoreamodifieddiskblockishandledby**recoverymanager**ofthe DBMS in cooperation with operating system.
- ADBcacheincludesanumberofdatabuffers.
- Whenthebuffersarealloccupiedabufferreplacementpolicyisusedtochooseoneof the buffers to be replaced. EG: LRU

- Atransactionincludesread_itemandwrite_itemoperationstoaccessandupdateDB.

(a)

T_1

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

(b)

T_2

read_item(X);
$X := X + M$;
write_item(X);

Figure 21.2
Two sample transactions. (a) Transaction T_1. (b) Transaction T_2.

- The**read-set**ofatransactionisthesetofallitemsthatthetransactionreads
- The**write-set**isthesetofallitemsthatthetransactionwrites
- Forexample,theread-setof$T1$inFigure21.2is{X,Y}anditswrite-setisalso{X,Y}.

WhyConcurrencyControlIsNeeded

- Severalproblemscanoccurwhenconcurrenttransactionsexecuteinanuncontrolled manner
- Example:
  - WeconsideranAirlinereservationDB
  - EachrecordsisstoredforanairlineflightwhichincludesNumberofreservedseats among other information.
  - Typesofproblemswemayencounter:
    1. TheLostUpdateProblem
    2. TheTemporaryUpdate(orDirtyRead)Problem
    3. TheIncorrectSummaryProblem
    4. TheUnrepeatableReadProblem

T_2

read_item(X);
$X := X + M$;
write_item(X);

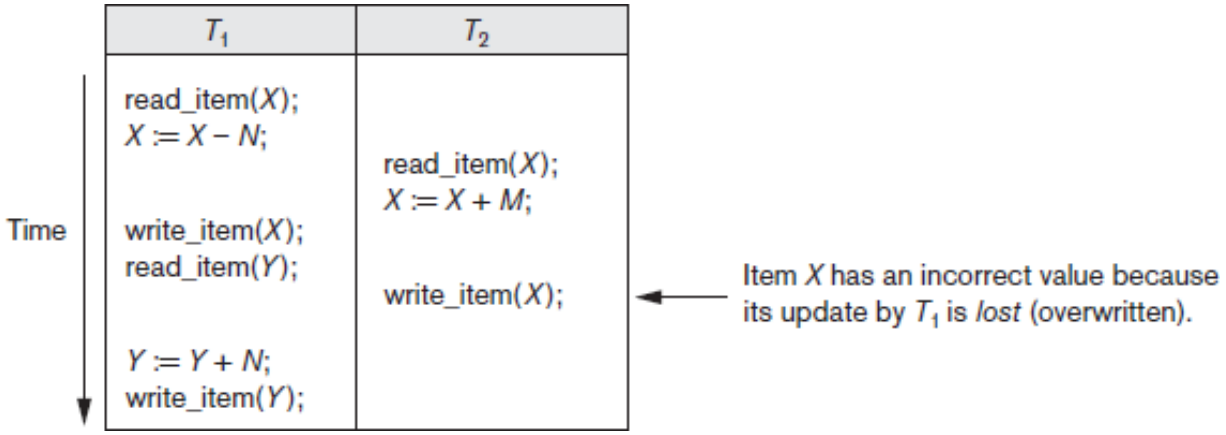
T_1

read_item(X);
$X := X - N$;
write_item(X);
read_item(Y);
$Y := Y + N$;
write_item(Y);

- TransactionT1
  - transfersNreservationsfromoneflight whosenumberof reservedseatsis storedin the database item named X to another flight whose number of reserved seats isstored in the database item named Y.
- TransactionT2
  - reservesMseatsonthefirstflight(X)

1. TheLostUpdateProblem

- occurswhentwotransactionsthataccessthesameDBitemshavetheiroperations interleaved in a way that makes the value of some DB itemincorrect
- SupposethattransactionsT1andT2aresubmittedatapproximatelythesametime,and suppose that their operations are interleaved as shown in Figure below



- FinalvalueofitemXisincorrect becauseT2readsthevalueof XbeforeT1changesit in the database, and hence the updated value resulting from T1 is lost.
- Forexample:

  X=80atthestart(therewere80reservationsontheflight)

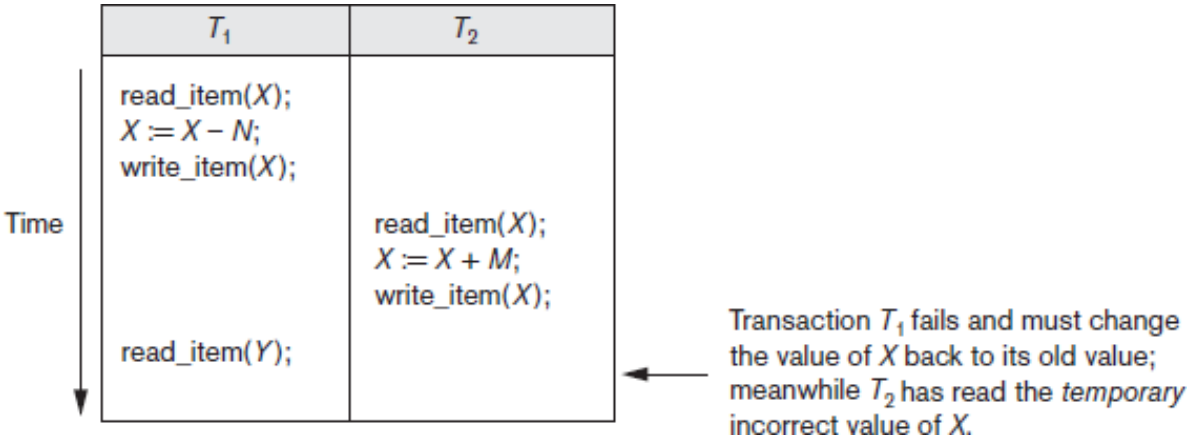
  N=5(T1transfers5seatreservationsfromtheflightcorresponding to X to the flight corresponding to Y)

  M=4(T2reserves4seatsonX)

  ThefinalresultshouldbeX=79.
- TheinterleavingofoperationsshowninFigureisX=84becausetheupdateinT1that removed the five seats from X was lost.

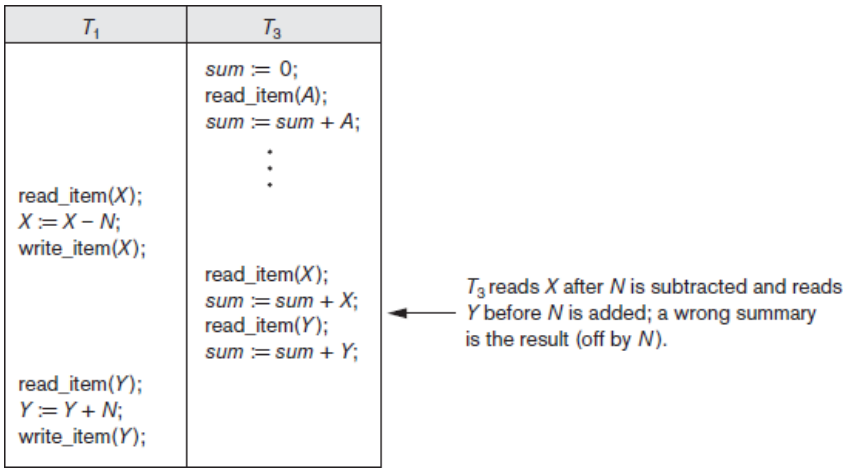
2. TheTemporaryUpdate(orDirtyRead)Problem

- occurswhenonetransactionupdatesadatabaseitem andthenthetransactionfailsfor some reason
- Meanwhiletheupdateditemisaccessedbyanothertransactionbeforeitischangedback to its original value



3. TheIncorrectSummaryProblem

- If one transaction is calculating an aggregate summary function on a number of db items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they areupdated.



4. TheUnrepeatableReadProblem

- Transaction T reads the same item twice and gets different values on each read, since the item was modified by another transaction T` between the two reads.
- for example, if during an airline reservation transaction, a customer inquires about seat availability on several flights
- When the customer decides on a particular flight, the transaction then reads the number of seats on that flight a second time before completing the reservation, and it may endup reading a different value for the item.

WhyRecoveryIsNeeded

- Whenever atransactionis submittedto a DBMSfor execution,thesystemis responsible for making sure that either
    1. All the operations in the transaction are completed successfully and their effect is recorded permanently in the database or
    2. Thetransactiondoesnothaveanyeffectonthedatabaseoranyother transactions
- In the first case, the transaction is said to be committed, whereasin the second case,the transaction is aborted
- Ifatransactionfailsafterexecutingsomeofitsoperationsbutbeforeexecutingallof them, the operations already executed must be undone and have no lastingeffect.

Typesoffailures

1. **Acomputerfailure(systemcrash):**
    - Ahardware,software,ornetworkerroroccursinthecomputer systemduring transaction execution
    - Hardwarecrashesareusuallymediafailures—forexample,mainmemoryfailure.
2. **Atransactionorsystemerror:**
    - Someoperationinthetransactionmaycauseittofail,suchasintegeroverflowor division by zero
    - Alsooccurbecauseoferroneousparametervalues
3. **Localerrorsorexceptionconditionsdetectedbythetransaction:**
    - Duringtransactionexecution,certainconditionsmayoccurthatnecessitatecancellation of the transaction

- Forexample,dataforthetransactionmaynotbefound

4. Concurrencycontrolenforcement:

- Theconcurrencycontrol maydecidetoabortatransactionbecauseitviolates serializability or several transactions are in a state ofdeadlock

5. Diskfailure:

- Somediskblocksmaylosetheir databecauseofaread orwrite malfunctionor because of a disk read/write head crash.

6. Physicalproblemsandcatastrophes:

- referstoanendlesslistofproblemsthatincludespowerorair-conditioningfailure,fire, theft, overwriting disks or tapes by mistake

- Failuresoftypes1, 2,3,and4aremorecommonthanthoseoftypes5or6.
- Wheneverafailureoftype1through4occurs,thesystemmustkeepsufficient informationto quickly recover from the failure.
- Diskfailureorothercatastrophicfailuresoftype5or6donothappenfrequently; iftheydo occur, recovery is a major task.

TransactionandSystemConcepts

TransactionStatesandAdditionalOperations

- A transaction is an atomic unit of work that should either be completed in its entirety or not done at all. For recovery purposes, the system keeps track of start of a transaction, termination, commit or aborts.
  - **BEGIN_TRANSACTION**:marksthebeginningoftransactionexecution
  - **READ or WRITE**: specify read or write operations on the database items that are executed as part of a transaction
  - **END_TRANSACTION:** specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution
  - **COMMIT_TRANSACTION**: signals a *successful end* of the transaction so that any changes (updates) executed by the transaction can be safely **committed** to the database and will not be undone
  - **ROLLBACK**: signals that the transaction has *ended unsuccessfully,* so that any changes or effects that the transaction may have applied to the database must be **undone**

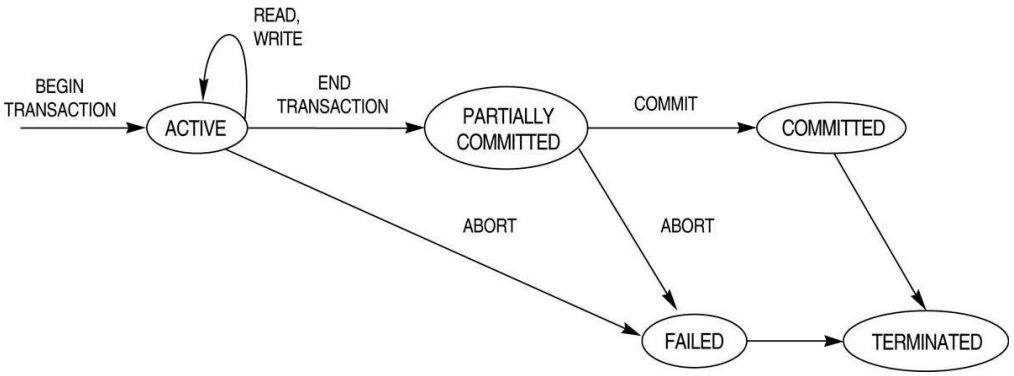


Figure:Statetransitiondiagramillustratingthestatesfortransactionexecution

- A transaction goes into **active state** immediately after it starts execution and canexecute read and write operations.
- Whenthetransactionendsitmovesto**partiallycommitted**state.
- At this end additional checks are done to see if thetransaction can be committed or not. If these checks are successful the transaction is said to have reached commit point and enters **committed state.** All the changes are recorded permanently in thedb.
- A transaction can go to the **failed state** if one of the checks fails or if the transaction is aborted during its active state. The transaction may then have to be rolled back to undo the effect of its write operation.
- Terminated state corresponds to the transaction leaving the system. All the information about the transaction is removed from system tables.

TheSystemLog

- **LogorJournal**keepstrackofalltransactionoperationsthataffectthevaluesof database items
- Thisinformationmaybeneededtopermitrecoveryfromtransactionfailures.
- Thelogiskept ondisk,soitisnot affectedbyanytypeoffailureexceptfordiskor catastrophic failure
- one(ormore)mainmemorybuffersholdthelastpartofthelogfile, sothat logentries are first added to the main memory buffer
- Whenthe**logbuffer**isfilled,orwhencertainotherconditionsoccur,thelogbufferis *appendedtotheendofthelogfileondisk*.

- Inaddition,thelogisperiodicallybackeduptoarchivalstorage(tape)to guardagainst such catastrophic failures
- Thefollowingarethetypesofentries—called**logrecords**—thatarewrittentothelogfile and the corresponding action for each log record.
- Intheseentries, Treferstoaunique **transaction-id**thatisgeneratedautomaticallyby the system for each transaction and that is used to identify eachtransaction:
  1. **[start_transaction,T].**IndicatesthattransactionThasstartedexecution.
  2. **[write_item,T,X,old_value,new_value].**IndicatesthattransactionThaschanged the value of database item X from old_value to new_value.
  3. **[read_item,T,X].**IndicatesthattransactionThasreadthevalueofdatabaseitemX.
  4. **[commit,T].**IndicatesthattransactionThascompletedsuccessfully,andaffirmsthat its effect can be committed (recorded permanently) to the database.
  5. **[abort,T].**IndicatesthattransactionThasbeenaborted.

CommitPointofaTransaction:

- **DefinitionaCommitPoint:**
  - A transaction T reaches its **commit point** when all its operations that access the database have been executed successfully *and* the effect of all the transaction operations on the database has been recorded in thelog.
  - Beyondthecommitpoint, thetransaction is saidto becommitted, anditseffect is assumed to be permanently recorded in the database.
  - Thetransactionthenwritesanentry[commit,T]intothelog.
- **RollBackoftransactions:**
  - Needed for transactions that have a [start_transaction,T] entry into the log but no commit entry [commit,T] into the log.

DBMSspecificbufferReplacementpolicies

DomainSeparation(DS)method

- DBMScacheisdividedintoseparatedomains,eachhandlesonetypeofdiskpagesand replacements within each domain are handled via basic LRU pagereplacement.
- LRUisa**static**algorithmanddoesnotadoptstodynamicallychangingloadsbecause the number of available buffers for each domain ispredetermined.
- **GroupLRU**addsdynamicallyloadbalancingfeaturesinceitgiveseachdomaina priority and selects pages from lower priority level domain first forreplacement.

HotSet Method:

- Thisisusefulinqueriesthathavetoscanasetofpagesrepeatedly.
- The hot set method determines for each db processing algorithm the set of disk pages that will be accessed repeatedly and it does not replace them until their processing is completed.

TheDBMINmethod:

- uses a model known as QLSM (Query Locality set model), which predetermines the pattern of page references for each algorithm for a particular dboperation
- Depending on the type of access method, the file characteristics, and the algorithm used the QLSM will estimate the number of main memory buffers needed for each file involved in the operation.

DesirablePropertiesofTransactions

- Transactionsshouldpossessseveralproperties,oftencalledthe**ACID**properties
  A**Atomicity:**atransactionisanatomicunitofprocessinganditiseitherperformed entirely or not at all.
  C **ConsistencyPreservation:**atransactionshouldbeconsistencypreservingthatisit must take the database from one consistent state to another.
  I**Isolation/Indepen dence:** Atransactionshouldappearasthoughitisbeingexecuted in isolation from other transactions, even though many transactions are executed concurrently.
  D**Durability(orPermanency):**ifatransactionchangesthedatabaseandiscommitted, the changes must never be lost because of any failure.
- The **atomicity** property requires that we execute a transaction to completion. It is the responsibility of the transaction recovery subsystem of a DBMS to ensureatomicity.
- The preservation of ***consistency*** is generally considered to be the responsibility of the programmers who write the database programs or of the DBMS module that enforces integrity constraints.
- The **isolation** property is enforced by the concurrency control subsystem of the DBMS.If every transaction does not make its updates (write operations) visible to other transactions until it is committed, one form of isolation is enforced that solves the temporary update problem and eliminates cascading rollbacks
- **Durability**istheresponsibilityofrecovery subsystem.

CharacterizingSchedulesBasedonRecoverability

- **schedule**(or**history):**theorderofexecutionofoperationsfromallthevarious transactions
- **Schedules(Histories)ofTransactions:** AscheduleSofntransactions$T_1, T_2, \ldots\ldots T_n$is a sequential ordering of the operations of the n transactions.
  - Thetransactionsareinterleaved
- Twooperationsinascheduleare saidto**conflict**iftheysatisfyallthreeofthefollowing conditions:
  (1) theybelongto*differenttransactions*;
  (2) theyaccessthe*sameitemX*;and
  (3) *atleastone*oftheoperationsisawrite_item(*X*)
- **Conflictingoperations:**
  - $r_1(X)$conflictswith$w_2(X)$ Readwriteconflict
  - $r_2(X)$conflictswith$w_1(X)$
  - $w_1(X)$conflictswith$w_2(X)$ Writeconflict
  - $r_1(X)$donotconflictswith$r_2(X)$

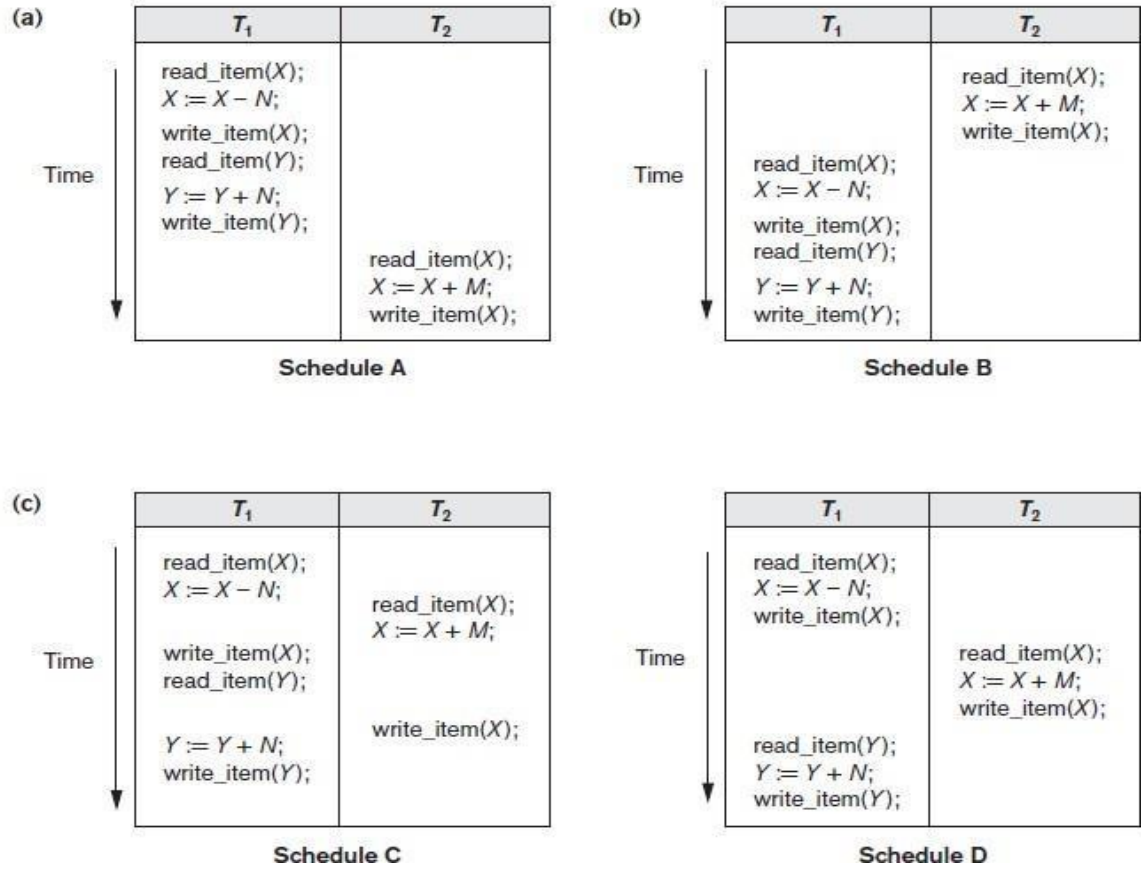
Schedulesclassifiedonrecoverability:

- **Recoverableschedule**:
  - Onewherenotransactionneedstoberolledback.
  - AscheduleSisrecoverableifnotransactionTinScommitsuntilalltransactions T' that have written an item that T reads have committed.
  - Example:
    - S_c:$r_1(X)$;$w_1(X)$;$r_2(X)$;$r_1(Y)$;$w_2(X)$;c_2;a_1;
    - S_d:$r_1(X)$;$w_1(X)$;$r_2(X)$;$r_1(Y)$;$w_2(X)$;$w_1(Y)$;c_1;c_2;
- **Cascadelessschedule**:
  - Onewhereeverytransactionreadsonlytheitemsthatarewrittenbycommitted transactions.
- **Schedulesrequiringcascadedrollback**:
  - Ascheduleinwhichuncommittedtransactionsthatreadanitemfromafailed transaction must be rolled back.
- **StrictSchedules**:
  - AscheduleinwhichatransactioncanneitherreadorwriteanitemXuntilthe last transaction that wrote X has committed.

CharacterizingSchedulesBasedonSerializability

- schedules that are always considered to be correct when concurrent transactions are executing are known as **serializable** schedules
- Suppose that two users—for example, two airline reservations agents—submit to the DBMS transactions T1 and T2 at approximately the same time. If no interleaving of operations is permitted, there are only two possible outcomes:
  1. Execute alltheoperations oftransaction T1(in sequence)followedby allthe operations of transaction T2 (in sequence).
  2. Executealltheoperationsoftransaction*T*2(insequence)followedbyallthe operations of transaction *T*1 (in sequence).

Figure 21.5
Examples of serial and nonserial schedules involving transactions T_1 and T_2. (a) Serial schedule A: T_1 followed by T_2. (b) Serial schedule B: T_2 followed by T_1. (c) Two nonserial schedules C and D with interleaving of operations.



- **Serialschedule:**
  - AscheduleSisserialif,foreverytransactionTparticipatingintheschedule,all the operations of T are executed consecutively in theschedule.
    - Otherwise,thescheduleiscallednonserialschedule.
- **Serializableschedule:**
  - AscheduleSisserializableifitisequivalenttosomeserialscheduleofthesame n transactions.
- **Resultequivalent:**
  - Twoschedulesarecalledresultequivalent iftheyproducethesamefinalstateof the database.
- **Conflictequivalent:**
  - Twoschedulesaresaidtobeconflictequivalentiftheorderofanytwoconflicting operations is the same in both schedules.
- **Conflictserializable:**
  - AscheduleSissaidtobeconflictserializable ifitisconflictequivalenttosome serial schedule S'.
- Beingserializableis<u>not</u>thesameasbeingserial
- Beingserializableimpliesthatthescheduleisa<u>correct</u>schedule.
  - Itwillleavethedatabaseinaconsistentstate.
  - The interleaving is appropriate and will result in a state as if the transactions wereseriallyexecuted,yetwillachieveefficiencyduetoconcurrentexecution.

TestingconflictserializabilityofaScheduleS

ForeachtransactionTiparticipatinginscheduleS,createanodelabeledTiinthe precedence graph.

ForeachcaseinSwhereTjexecutesaread_item(X)afterTiexecutesawrite_item(X), create an edge (Ti→Tj) in the precedence graph.

ForeachcaseinSwhereTjexecutesawrite_item(X)afterTiexecutesaread_item(X) ,createanedge(Ti→Tj)intheprecedencegraph.

ForeachcaseinSwhereTjexecutesawrite_item(X)afterTiexecutesawrite_item(X), create an edge (Ti→Tj) in the precedence graph.

ThescheduleSisserializableifandonlyiftheprecedencegraphhasnocycles.

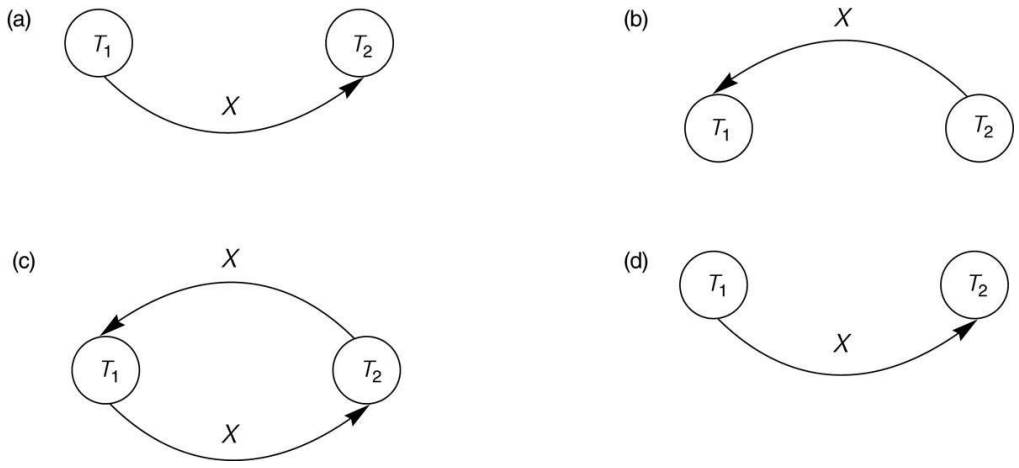


Fig:Constructingtheprecedencegraphsforschedules *A*and*D*fromfig21.5totestforconflict serializability.

(a) Precedencegraphforserialschedule*A*.

(b) Precedencegraphforserialschedule*B*.

(c) Precedencegraphforschedule*C*(notserializable).

(d) Precedencegraphforschedule*D*(serializable,equivalenttoschedule*A*).

- Anotherexampleofserializabilitytesting.(a)TheREADandWRITEoperationsofthree transactions T_1, T_2, and T_3.

(a)

transaction T_1
read_item (X); write_item (X); read_item (Y); write_item (Y);

transaction T_2
read_item (Z); read_item (Y); write_item (Y); read_item (X); write_item (X);

transaction T_3
read_item (Y); read_item (Z); write_item (Y); write_item (Z);

(b)

transaction T_1	transaction T_2	transaction T_3
	read_item (Z);	
	read_item (Y);	
	write_item (Y);	
		read_item (Y);
		read_item (Z);
read_item (X);		
write_item (X);		
		write_item (Y);
		write_item (Z);
	read_item (X);	
read_item (Y);		
write_item (Y);	write_item (X);	

Time ↓

Schedule E

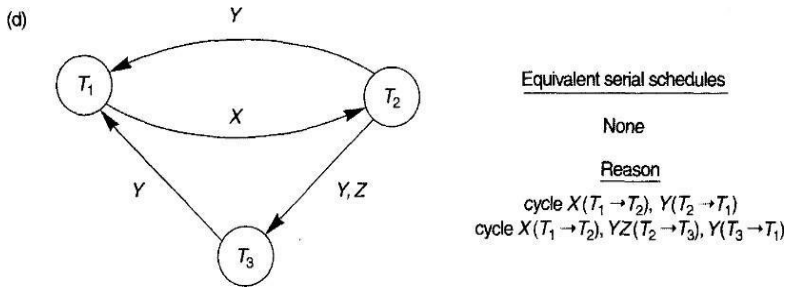
(c)

transaction T_1	transaction T_2	transaction T_3
		read_item (Y);
		read_item (Z);
read_item (X);		
write_item (X);		
		write_item (Y);
		write_item (Z);
	read_item (Z);	
read_item (Y);		
write_item (Y);	read_item (Y);	
	write_item (Y);	
	read_item (X);	
	write_item (X);	

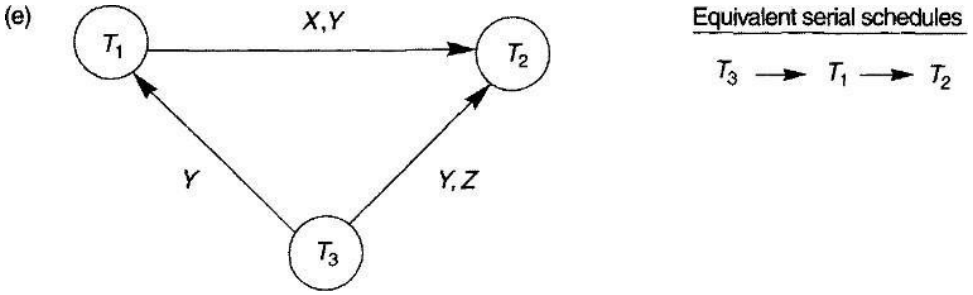
Time ↓

Schedule F

- PrecedencegraphforscheduleE



- PrecedencegraphforscheduleF



TransactionSupportinSQL

- ThebasicdefinitionofanSQLtransactionis,it isalogicalunit ofworkandisguaranteed to be atomic
- AsingleSQLstatementisalwaysconsideredtobeatomic—eitheritcompletes execution without an error or it fails and leaves the databaseunchanged
- WithSQL,thereisnoexplicitBegin_Transactionstatement.Transactioninitiationis done implicitly when particular SQL statements are encountered
- Everytransactionmusthaveanexplicitendstatement,whichiseitheraCOMMITora ROLLBACK
- EverytransactionhascertaincharacteristicsattributedtoitandarespecifiedbyaSET TRANSACTION statement in SQL

- Thecharacteristicsare:
  - **Theaccessmode**
    - canbespecifiedasREADONLYorREADWRITE
    - ThedefaultisREADWRITE
    - AmodeofREADWRITEallowsselect,update,insert,delete,andcreate commands to be executed
    - AmodeofREADONLY,asthenameimplies,issimplyfordataretrieval.
  - **Thediagnosticareasize**
    - DIAGNOSTICSIZEn,specifiesanintegervaluen, whichindicatesthe number of conditions that can be held simultaneously in the diagnosticarea
    - Theseconditionssupplyfeedbackinformation(errorsorexceptions)tothe user or program on the n most recently executed SQLstatement
- **Theisolationlevel**
  - specifiedusingthestatementISOLATIONLEVEL<isolation>,wherethevaluefor <isolation>canbeREADUNCOMMITTED,READCOMMITTED,REPEATABLE READ, or SERIALIZABLE
    - ThedefaultisolationlevelisSERIALIZABLE
    - TheuseofthetermSERIALIZABLEhereis basedonnotallowingviolationsthat cause dirty read, unrepeatable read, and phantoms
    - Ifatransactionexecutesatalowerisolationlevelthan SERIALIZABLE,thenone or more of the following three violations may occur:
      1. **Dirtyread.** Atransaction*T*1mayreadtheupdateofatransaction *T*2,which has not yet committed. If*T*2 fails andis aborted, then *T*1 would havereada value that does not exist and is incorrect.
      2. **Nonrepeatableread.** Atransaction*T*1mayreada givenvaluefromatable.If anothertransaction*T*2laterupdatesthatvalueand*T*1readsthatvalueagain, *T*1 will see a different value.
      3. **Phantoms.** A transaction *T*1 may read a set of rows from a table, perhaps basedonsomeconditionspecifiedintheSQLWHERE-clause.Nowsuppose that a transaction *T*2inserts a newrowthat also satisfies the WHERE-clause condition used in *T*1, into the table used by *T*1. If*T*1 is repeated, then *T*1will see a phantom, a row that previously did notexist.

Table 21.1 Possible Violations Based on Isolation Levels as Defined in SQL

	Type of Violation		
Isolation Level	**Dirty Read**	**Nonrepeatable Read**	**Phantom**
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION
    READ WRITE
    DIAGNOSTIC SIZE 5
    ISOLATION LEVEL SERIALIZABLE;
EXEC SQL INSERT INTO EMPLOYEE (Fname, Lname, Ssn, Dno, Salary)
    VALUES ('Robert', 'Smith', '991004321', 2, 35000);
EXEC SQL UPDATE EMPLOYEE
    SET Salary = Salary * 1.1 WHERE Dno = 2;
EXEC SQL COMMIT;
GOTO THE_END;
UNDO: EXEC SQL ROLLBACK;
THE_END: ... ;
```

- Thetransactionconsistsoffirstinsertinganewrowin theEMPLOYEEtableandthen updating the salary of all employees who work in department 2
- Ifanerroroccursonanyof theSQLstatements,theentiretransactionisrolledback
- Thisimpliesthatanyupdatedsalary(bythistransaction)wouldberestoredto its previous value and that the newly inserted row would be removed.

Chapter2:ConcurrencyControlinDatabases

IntroductiontoConcurrencyControl

- PurposeofConcurrencyControl
  - ToenforceIsolation(throughmutualexclusion)amongconflictingtransactions.
  - Topreservedatabaseconsistencythroughconsistencypreserving executionof transactions.
  - Toresolveread-writeandwrite-writeconflicts.
- Example:
  - In concurrent execution environmentif T1 conflicts with T2 over a data item A, thenthe existing concurrency control decides if T1 or T2 should get the A and if the other transaction is rolled-back or waits.

Two-PhaseLockingTechniquesforConcurrencyControl

- Theconceptoflockingdataitemsisoneofthemaintechniquesusedforcontrollingthe concurrent execution of transactions.
- Alockisavariableassociatedwithadataitem inthedatabase.Generallythereisalock for each data item in the database.
- Alockdescribesthestatusofthedataitemwithrespecttopossibleoperationsthatcanbe applied to that item.
- Itisusedforsynchronizingtheaccessbyconcurrenttransactionstothedatabaseitems.
- Atransactionlocksanobjectbeforeusing it
- Whenanobjectislockedbyanothertransaction,therequestingtransactionmustwait

TypesofLocksandSystemLockTables

1. **BinaryLocks**

- A**binarylock** canhavetwo**states**or**values:**lockedandunlocked(or1 and 0).
- IfthevalueofthelockonXis1,itemXcannotbeaccessedbyadatabase operation that requests the item

- IfthevalueofthelockonXis0,theitem canbeaccessedwhen requested, and the lock value is changed to 1
- We refertothecurrentvalue(orstate)ofthelockassociatedwithitemX as **lock**(***X***).
- Twooperations**,lock_item**and**unlock_item**,areusedwithbinary locking.
- Atransactionrequestsaccesstoanitem *X*byfirstissuinga**lock_item(*X*)** operation
- IfLOCK(*X*)=1,thetransactionisforcedtowait.
- IfLOCK(*X*)=0,itissetto1(thetransaction**locks**theitem)andthe transaction is allowed to access item *X*
- When the transaction is through using the item, it issues an **unlock_item(*X*)**operation,whichsetsLOCK(*X*)backto0(**unlocks**the item) so that *X* may be accessed by other transactions
- Hence,abinarylockenforces**mutualexclusion**onthedataitem.

```
lock_item(X):
B:ifLOCK(X)=0(*itemisunlocked*)  then
  LOCK(X) ←1 (* lock the item *) else
  begin
        wait(untilLOCK(X)=0
        andthelockmanagerwakesupthetransaction); go to
         B
   end;
unlock_item(X):
LOCK(X)←0;(*unlocktheitem*) if
any transactions are waiting
thenwakeuponeofthewaitingtransactions;
```

Fig:2.1.1Lockandunlockoperationsforbinarylicks.

- The lock_item and unlock_item operations must be implemented as indivisible units that is, no interleaving should be allowed once a lock or unlock operation is started until the operation terminates or the transaction waits
- The wait command within the lock_item(*X*) operation is usually implemented by putting the transaction in a waiting queue for item *X* until *X* is unlocked and the transaction can be granted access to it
- Other transactions that also want to access *X* are placed in the same queue.Hence, the wait command is considered to be outside the lock_item operation.
- It is quite simple to implement a binary lock; all that is needed is a binary-valuedvariable, LOCK, associated with each data item X in thedatabase
- In its simplest form, each lock can be a record with three fields: <Data_item_name, LOCK, Locking_transaction> plus aqueuefortransactionsthatarewaitingto accessthe item
- If thesimple binary locking scheme described here is used, every transaction must obey the following rules:
  1. A transaction *T* must issue the operation lock_item(*X*) before any read_item(*X*) or write_item(*X*) operations are performed in *T.*
  2. A transaction *T* must issue the operation unlock_item(*X*) after allread_item(*X*) and write_item(*X*) operations are completed in *T.*
  3. Atransaction*T*willnotissuealock_item(*X*)operationifitalreadyholdsthelock on item *X*.
  4. Atransaction*T*willnotissueanunlock_item(*X*)operationunlessitalreadyholds the lock on item *X*.

2. Shared/Exclusive(orRead/Write)Locks

- binarylockingschemeistoorestrictivefordatabaseitemsbecauseatmost,one transaction can hold a lock on a given item
- shouldallowseveraltransactionstoaccessthesameitemX iftheyallaccessXfor reading purposes only
- ifatransactionistowriteanitem*X*,itmusthaveexclusiveaccessto*X*
- Forthispurpose,adifferenttypeoflockcalleda**multiple-modelock**isused
- Inthisscheme—called**shared/exclusive**or**read/write**locks—therearethreelocking operations: **read_lock(*X*), write_lock(*X*), and unlock(*X*)**.

- A **read-locked item** is also called **share-locked** because other transactions are allowed to read the item, whereas a **write-locked item** is called **exclusive-locked** because a single transaction exclusively holds the lock on the item
- Methodtoimplementread/writelockisto
  - keeptrackofthenumberoftransactions that hold a shared(read)lock on an item in the lock table
  - Eachrecordinthelocktablewillhavefourfields: <Data_item_name,LOCK,No_of_reads,Locking_transaction(s)>.
- IfLOCK(*X*)=write-locked,thevalueoflocking_transaction(s)isasingletransactionthat holds the exclusive (write) lock on *X*
- IfLOCK(*X*)=read-locked,thevalueof lockingtransaction(s)isalistofoneormore transactions that hold the shared (read) lock on *X*.

```
read_lock(X):
B:  if LOCK(X) = "unlocked"
        then begin LOCK(X) ← "read-locked";
                no_of_reads(X) ← 1
                end
    else if LOCK(X) = "read-locked"
        then no_of_reads(X) ← no_of_reads(X) + 1
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
write_lock(X):
B:  if LOCK(X) = "unlocked"
        then LOCK(X) ← "write-locked"
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
```

```
unlock (X):
    if LOCK(X) = "write-locked"
        then begin LOCK(X) ← "unlocked";
                wakeup one of the waiting transactions, if any
                end
    else it LOCK(X) = "read-locked"
        then begin
                no_of_reads(X) ← no_of_reads(X) –1;
                if no_of_reads(X) = 0
                    then begin LOCK(X) = "unlocked";
                            wakeup one of the waiting transactions, if any
                            end
                end;
```

- Whenweusetheshared/exclusivelockingscheme,thesystemmustenforcethefollowing rules:

1. AtransactionTmustissuetheoperationread_lock(X)orwrite_lock(X)beforeany read_item(X) operation is performed in T.
2. AtransactionTmustissuetheoperationwrite_lock(X)beforeanywrite_item(X) operation is performed in T.

3AtransactionTmustissuetheoperationunlock(X)afterallread_item(X)and write_item(X) operations are completed in T.3

4. AtransactionTwillnot issuearead_lock(X)operationifitalreadyholdsaread(shared) lock or a write (exclusive) lock on item X.

ConversionofLocks

- Atransactionthatalreadyholdsalockonitem*X*isallowedundercertainconditionsto **convert**thelockfromonelockedstatetoanother
- Forexample,itispossibleforatransaction *T*toissuearead_lock(*X*)andthenlaterto **upgrade**thelockbyissuingawrite_lock(*X*)operation

  -If*T*istheonlytransactionholdingareadlockon *X*atthetimeit issues the write_lock(*X*) operation, the lock can be upgraded;otherwise, the transaction must wait

GuaranteeingSerializabilitybyTwo-PhaseLocking

- Atransactionissaidtofollowthe**two-phaselockingprotocol**if*all*lockingoperations (read_lock, write_lock) precede the *first* unlock operation in thetransaction
- Suchatransactioncanbedividedintotwophases:
  - **Expanding**or**growing(first)phase**,duringwhichnewlocksonitemscanbe acquired but none can be released
  - **Shrinking(second)phase**,duringwhichexistinglockscanbereleasedbutno new locks can be acquired
- If lock conversion is allowed, then upgrading of locks (from read-locked to write-locked) mustbedoneduringtheexpanding phase, anddowngrading oflocks(from write-locked to read-locked) must be done in the shrinking phase.
- Transactions *T*1 and *T*2 in Figure 22.3(a) do not follow the two-phase locking protocol because the write_lock(*X*) operation follows the unlock(*Y*) operation in *T*1, and similarly the write_lock(*Y*) operation follows the unlock(*X*) operation in*T*2.

(a)

T_1	T_2

read_lock(*Y*);
read_item(*Y*);
unlock(*Y*);
write_lock(*X*);
read_item(*X*);
X := *X* + *Y*;
write_item(*X*);
unlock(*X*);
	read_lock(*X*);
	read_item(*X*);
	unlock(*X*);
	write_lock(*Y*);
	read_item(*Y*);
	Y := *X* + *Y*;
	write_item(*Y*);
	unlock(*Y*);

(b) Initial values: *X*=20, *Y*=30

Result serial schedule T_1 followed by T_2: *X*=50, *Y*=80

Result of serial schedule T_2 followed by T_1: *X*=70, *Y*=50

(c)

Time

T_1	T_2

read_lock(*Y*);
read_item(*Y*);
unlock(*Y*);	

	read_lock(*X*);
	read_item(*X*);
	unlock(*X*);
	write_lock(*Y*);
	read_item(*Y*);
	Y := *X* + *Y*;
	write_item(*Y*);
	unlock(*Y*);

write_lock(*X*);
read_item(*X*);
X := *X* + *Y*;
write_item(*X*);
unlock(*X*);	

Figure 21.3 Transactions that do not obey two-phase locking (a) Two transactions *T*1 and *T*2 (b) Results of possible serial schedules of *T*1 and *T*2 (c) A nonserializable schedule *S* that uses locks

- If we enforce two-phase locking, the transactions can be rewritten as *T*1' and *T*2' as shown in Figure 22.4.
- Now, the schedule shown in Figure 22.3(c) is not permitted for *T*1_ and *T*2_ (with their modified order of locking and unlocking operations) under the rules of locking because *T*1_ will issue its write_lock(*X*) *before* it unlocks item *Y*; consequently, when *T*2_ issues its read_lock(*X*), it is forced to wait until *T*1_ releases the lock by issuing an unlock (*X*) in the schedule.

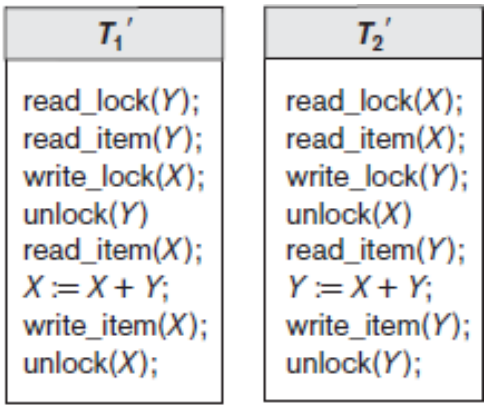


Figure 22.4
Transactions T_1' and T_2', which are the same as T_1 and T_2 in Figure 22.3, but follow the two-phase locking protocol. Note that they can produce a deadlock.

- Ifeverytransactioninaschedulefollowsthetwo-phaselockingprotocol,schedule guaranteed to be serializable
- Two-phaselockingmaylimittheamountofconcurrencythatcanoccurinaschedule
- Someserializableschedules willbeprohibitedbytwo-phaselockingprotocol

VariationsofTwo-PhaseLocking

- **Basic2PL**
  - Techniquedescribedpreviously
- **Conservative(static)2PL**
  - Requiresatransactiontolockalltheitemsitaccessesbeforethetransaction begins execution by predeclaring read-set and write-set
  - ItsDeadlock-freeprotocol

- **Strict2PL**
  - guaranteesstrictschedules
  - Transactiondoesnotreleaseexclusivelocksuntilafteritcommitsoraborts
  - noothertransactioncanreadorwriteanitemthatiswrittenby*T*unless*T*has committed, leading to a strict schedule forrecoverability
  - Strict2PLisnotdeadlock-free

- **Rigorous2PL**
  - guaranteesstrictschedules
  - Transactiondoesnotreleaseanylocksuntilafteritcommitsoraborts
  - easiertoimplementthanstrict2PL

DealingwithDeadlockandStarvation

- **Deadlock** occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T' in theset.
- Hence, each transaction in the set is in a waiting queue, waiting for one of the other transactions in the set to release the lock on an item.
- Butbecausetheothertransactionisalsowaiting,itwillneverreleasethelock.
- A simple example is shown in Figure 22.5(a), where the two transactions *T*1' and *T*2_'are deadlocked in a partial schedule; *T*1' is in the waiting queue for *X*, which is locked by *T*2', while *T*2' is inthe waiting queue for *Y*, which is locked by *T*1'. Meanwhile, neither *T*1' nor *T*2' nor any other transaction can access items *X* and*Y*

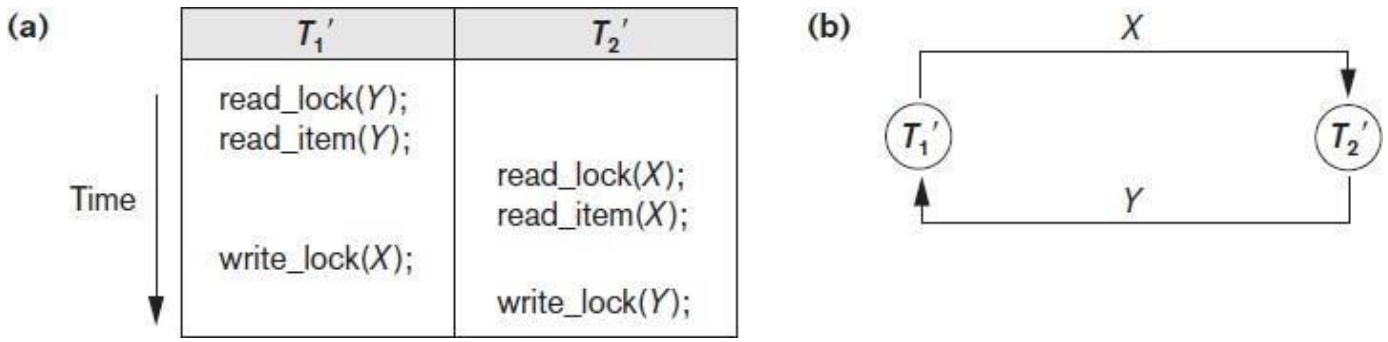


Figure22.5Illustratingthedeadlockproblem(a)Apartialscheduleof*T*1′and*T*2′thatisina state of deadlock (b) A wait-for graph for the partial schedule in (a)

Deadlockpreventionprotocols

- Onewaytopreventdeadlockistousea**deadlockpreventionprotocol**
- One deadlock prevention protocol, which is used in conservative two-phase locking, requires that every transaction lock *all the items it needs in advance*. If any of the items cannot be obtained, none of the items are locked. Rather, the transaction waits and then tries again to lock all the items it needs.
- A second protocol, which also limits concurrency, involves *ordering all the items* in the database and making sure that a transaction that needs several items will lock them according to that order. This requires that the programmer (or the system) is aware ofthe chosen order of the items
- Bothapproachesimpractical
- Someofthesetechniquesusetheconceptof**transactiontimestamp**TS(*T*),whichisa unique identifier assigned to each transaction
- Thetimestampsaretypicallybasedontheorderinwhichtransactionsarestarted;hence,if transaction T_1 starts before transaction T_2, then TS(T_1) <TS(T_2).
- The*older*transaction(whichstartsfirst)hasthe*smaller*timestampvalue.
- Protocolsbasedonatimestamp
  - Wait-die
  - Wound-wait
- Suppose that transaction *Ti* tries to lock an item *X* but is not able to because *X* is locked by some other transaction *Tj* with a conflicting lock. The rules followed by theseschemes are:
  - **Wait-die.** If TS(*Ti*) < TS(*Tj*), then (*Ti* older than *Tj*) *Ti* is allowed to wait; otherwise (*Ti* younger than *Tj*) abort *Ti* (*Ti dies*) and restart it later *with the same timestamp.*
  - **Wound-wait.** If TS(*Ti*) < TS(*Tj*), then (*Ti* older than *Tj*) abort *Tj* (*Ti wounds Tj*) and restart it later *with the same timestamp;* otherwise (*Ti* younger than *Tj*) *Ti* is allowed to wait.
- In wait-die, an older transaction is allowed to *wait for a younger transaction*, whereas a younger transaction requesting an item held by an older transaction is aborted and restarted.
- The wound-wait approach does the opposite: A younger transaction is allowed to *wait for an older one*, whereas an older transaction requesting an item held by a younger transaction *preempts* the younger transaction by aborting it.

- Bothschemesendupabortingthe*younger*ofthetwotransactions(thetransactionthat started later) that *may be involved* in a deadlock, assuming that this will waste less processing.
- Itcanbeshownthatthesetwotechniquesare*deadlock-free*,sinceinwait-die, transactions only wait for younger transactions so no cycle iscreated.
- Similarly,inwound-wait,transactionsonlywaitforoldertransactionssonocycleis created.
- Anothergroupofprotocolsthatpreventdeadlockdonotrequiretimestamps.These include the
  - nowaiting(NW)and
  - cautiouswaiting(CW)algorithms
  - **Nowaitingalgorithm**,
    - if a transaction is unable to obtain a lock, it is immediately aborted and then restarted after a certain time delay without checking whether a deadlock will actually occur or not.
    - notransactioneverwaits,sonodeadlockwill occur
    - thisschemecancausetransactionstoabortandrestartneedlessly
  - **cautiouswaiting**
    - trytoreducethenumberofneedlessaborts/restarts
    - Supposethattransaction*Ti*triestolockanitem*X*butisnot abletodosobecause *X*islockedbysomeothertransaction*Tj*withaconflictinglock.
    - Thecautiouswaitingrulesareasfollows:
      - If*Tj*isnotblocked(notwaitingforsomeotherlockeditem),then *Ti*is blocked and allowed to wait; otherwise abort *Ti.*
    - Itcanbeshownthatcautiouswaitingisdeadlock-free,becausenotransactionwill ever wait for another blocked transaction.

DeadlockDetection.

- A second, more practical approach to dealing with deadlock is **deadlock detection**, where the system checks if a state of deadlock actually exists.
- This solution is attractive if we know there will be little interference among the transactions—that is, if different transactions will rarely access the same items at the same time.

- This can happen if the transactions are short and each transaction locks only a few items, or if the transaction load is light.
- On the other hand, if transactions are long and each transaction uses many items, or if the transaction load is quite heavy, it may be advantageous to use a deadlockprevention scheme.
- Asimplewaytodetectastateofdeadlockis forthesystemtoconstructandmaintaina **wait-forgraph**.
- Onenodeiscreatedinthewait-forgraphforeachtransactionthatiscurrentlyexecuting.
- Whenever a transaction *Ti* is waiting to lock an item *X* that is currently locked by a transaction *Tj*, a directed edge (*Ti* → *Tj*) is created in the wait-forgraph.
- When *Tj* releases the lock(s) on the items that *Ti* was waiting for, the directed edge is dropped from the wait-for graph.We have a state of deadlock if and only if the wait-for graph has a cycle.
- One problem with this approach is the matter of determining *when* the system should check for a deadlock.
- One possibility is to check for a cycle every time an edge is added to the wait-for graph, but this may cause excessive overhead.
- Criteria such as the number of currently executing transactions or the period of time several transactions have been waiting to lock itemsmay be used instead to check for a cycle. Figure 22.5(b) shows the wait-for graph for the (partial) schedule shown in Figure 22.5(a).

– If the system is in a state of deadlock, some of the transactions causing the deadlock must be aborted.
– Choosingwhichtransactionstoabortisknownas**victimselection**.
– The algorithmfor victimselection should generally avoid selecting transactions thathave been running for a long time and that have performed many updates, and it should try instead to select transactions that have not made many changes (youngertransactions).

- **Timeouts**
  - Anothersimpleschemetodealwithdeadlockistheuseof**timeouts**.
  - Thismethodispracticalbecauseofitslowoverheadandsimplicity.
  - In this method, if a transaction waits for a period longer than a system-defined timeout period, the system assumes that the transaction may be deadlocked and aborts it—regardless of whether a deadlock actually exists or not.

- **Starvation.**
  - Another problem that may occur when we use locking is **starvation**, which occurs when a transaction cannot proceed for an indefinite period of time while other transactions in the system continue normally.
  - This may occur if the waiting scheme for locked items is unfair, giving priority to some transactions over others
  - One solution for starvation is to have a fair waiting scheme, suchas using a **first-come-first-served**queue;transactionsareenabledtolockanitemintheorder in which they originally requested the lock.
  - Another scheme allows some transactions to have priority over others but increases the priority of a transaction the longer it waits, until it eventually gets the highest priority and proceeds.
  - Starvation can also occur because of victim selection if the algorithm selects the same transaction as victim repeatedly, thus causing it to abort and never finish execution.
  - The algorithm can use higher priorities for transactions that have been aborted multiple times to avoid this problem.

ConcurrencyControlBasedonTimestampOrdering

guaranteesserializabilityusingtransactiontimestampstoordertransactionexecution for an equivalent serial schedule

Timestamps

- **timestamp**isauniqueidentifiercreatedbytheDBMStoidentifyatransaction.
- Typically, timestamp values are assigned in the order in which the transactions are submitted to the system, so a timestamp can be thought of as the *transaction start time*.
- Wewillrefertothetimestampoftransaction*T*as**TS(*T*)**.
- Concurrency control techniques based on timestamp ordering do not use locks;hence, *deadlocks cannot occur*.
- Timestampscanbegeneratedinseveralways.
  - One possibility is to use a counter that is incremented each time its value is assigned to a transaction. The transaction timestamps are numbered 1, 2, 3,

... in this scheme. A computer counter has a finite maximum value, so the system must periodically reset the counter to zero when no transactions are executing for some short period of time.

– Another wayto implement timestamps isto use thecurrent date/timevalue of the system clock and ensure that no two timestamp values are generated during the same tick of the clock.

TheTimestampOrdering Algorithm

- The idea for this scheme is to order the transactions based on their timestamps.
- A schedule in which the transactions participate is then serializable, and the onlyequivalentserialschedulepermittedhasthetransactionsinorderoftheir timestamp values. This is called **timestamp ordering (TO**).
- The algorithm must ensure that, for each item accessed by *conflicting Operations* in the schedule, the order in which the item is accessed does not violate the timestamp order.
- To do this, the algorithm associates with each database item *X* two timestamp (**TS**) values:
    1. **read_TS(*X*).** The **read timestamp** of item *X* is the largest timestamp among all the timestamps of transactions that have successfullyread item *X*—that is, read_TS(*X*) = TS(*T*), where *T* is the *youngest* transaction that has read *X* successfully.
    2. **write_TS(*X*).** The **write timestamp** of item *X* is the largest of all the timestamps of transactions that have successfully written item *X*—that is, write_TS(*X*) = TS(*T*), where *T* is the *youngest* transactionthat has written *X* successfully.

BasicTimestampOrdering(TO).

- Whenever some transaction *T* tries to issue a read_item(*X*) or a write_item(*X*) operation, the **basic TO** algorithm compares the timestamp of *T* with read_TS(*X*) and write_TS(*X*) to ensure that the timestamp order of transaction execution is not violated.
- If this order is violated, then transaction *T* is aborted and resubmitted to the system as a new transaction with a *new timestamp*.
- If*T*isabortedandrolledback,any transaction*T*1 thatmay haveusedavaluewrittenby*T* mustalsoberolledback.

- Similarly, any transaction *T*2 that may have used a value written by *T*1 must also be rolled back, and so on. This effect is known as **cascading rollback** and is one of the problems associated with basic TO, since the schedules produced are not guaranteed to be recoverable.
- An *additional protocol* must be enforced to ensure that the schedules are recoverable, cascadeless, or strict.
- **ThebasicTOalgorithm:**
  - The concurrencycontrolalgorithmmust checkwhether conflicting operations violate the timestamp ordering in the following two cases:

1. Whenever a transaction *T* issues a write_item(*X*) operation, the following ischecked:
   a. If read_TS(*X*) > TS(*T*) or if write_TS(*X*) > TS(*T*), then abort and roll back *T* and reject the operation. This should be done because some *younger* transactionwith a timestamp greater than TS(*T*)—and hence *after T* in the timestamp ordering—has already read or written the value of item *X* before *T* had a chance to write *X*, thus violating the timestamp ordering.
   b. If the condition in part (a) does not occur, then execute the write_item(*X*)operation of *T* and set write_TS(*X*) to TS(*T*).
2. Wheneveratransaction*T*issuesaread_item(*X*)operation,thefollowingischecked:
   a. If write_TS(*X*) > TS(*T*), then abort and roll back *T* and reject the operation. This should be done because some younger transaction with timestamp greater than TS(*T*)—and hence *after T* in the timestamp ordering—has already written the value of item *X* before *T* had a chance to read *X*.
   b. If write_TS(*X*) ≤ TS(*T*), then execute the read_item(*X*) operation of *T* and set read_TS(*X*) to the *larger* of TS(*T*) and the current read_TS(*X*).

- Whenever the basic TO algorithm detects two *conflicting operations* that occur in the incorrect order, it rejects the later of the two operations by aborting the transaction that issued it. The schedules produced by basic TO are hence guaranteed to be *conflict serializable*

StrictTimestampOrdering(TO)

- AvariationofbasicTOcalled**strictTO**ensuresthattheschedulesareboth**strict** (foreasyrecoverability)and(conflict)serializable.

- In this variation, a transaction *T* that issues a read_item(*X*) or write_item(*X*) such that TS(*T*) > write_TS(*X*) has itsread or write operation *delayed* untilthe transaction *T*' that *wrote* the value of *X* (hence TS(*T*') = write_TS(*X*)) has committed oraborted.
- Toimplementthisalgorithm,it isnecessarytosimulatethelockingofanitem *X*that has been written by transaction *T*' until *T*' is either committed or aborted. This algorithm *does not cause deadlock*, since *T* waits for *T*' only if TS(*T*) >TS(*T*_).

Thomas'sWriteRule

- A modification of the basic TO algorithm, known as **Thomas's write rule**, does not enforce conflict serializability, but it rejects fewer write operations by modifying the checks for the write_item(*X*) operation asfollows:
  1. Ifread_TS(*X*)>TS(*T*),thenabortandrollback*T*andrejecttheoperation.
  2. Ifwrite_TS(*X*) > TS(*T*), then do not execute the write operation but continue processing. This is because some transaction with timestamp greater than TS(*T*)—and hence after *T* in the timestamp ordering—has already written the value of *X*. Thus,wemustignorethewrite_item(*X*)operationof*T*becauseitisalreadyoutdated andobsolete.Noticethatanyconflictarisingfromthissituationwouldbedetectedby case (1).

Ifneitherthecondition inpart(1)northecondition inpart(2)occurs,thenexecute the write_item(*X*) operation of *T* and set write_TS(*X*) to TS(*T*).

MultiversionConcurrencyControlTechniques

- Other protocols for concurrency control keep the old values of a data item when theitem is updated. These are known as **multiversion concurrency control**, because several versions (values) of an item aremaintained
- When a transaction requires access to an item, an *appropriate* version is chosen to maintain the serializability of the currently executing schedule, if possible.
- The idea is that some read operations that would be rejected in other techniques can still be accepted by reading an *older version* of the item to maintain serializability.When a transaction writes an item, it writes a *new version* and the old version(s) of the item are retained
- An obvious drawback ofmultiversion techniques is that more storage is needed to maintain multiple versions of the database items

MultiversionTechniqueBasedonTimestampOrdering

- Inthismethod,severalversions*X*1,*X*2,...,*Xk*ofeachdataitem *X*aremaintained.
- For*eachversion*,thevalueofversion*Xi*andthefollowingtwotimestampsarekept:
  1. **read_TS(*Xi*).**The**readtimestamp**of*Xi*isthelargestofallthetimestampsof transactions that have successfully read version *Xi*.
  2. **write_TS(*Xi*).**The**writetimestamp**of*Xi*isthetimestampofthetransaction that wrote the value of version *Xi*.
- Wheneveratransaction*T*isallowedtoexecuteawrite_item(*X*)operation,anew version *Xk*+1 of item *X* is created, with both the write_TS(*Xk*+1) and the read_TS(*Xk*+1) set to TS(*T*)
- Correspondingly,whenatransaction*T*isallowedtoreadthevalueofversion *Xi*,the value of read_TS(*Xi*) is set to the larger of the current read_TS(*Xi*) andTS(*T*).
- Toensureserializability,thefollowingrulesareused:
  1. If transaction *T* issues a write_item(*X*) operation, and version *i* of *X* has the highest write_TS(*Xi*) of all versions of *X* that is also *less than or equal to* TS(*T*), and read_TS(*Xi*) > TS(*T*), then abort and roll back transaction *T*; otherwise, create a new version *Xj* of *X* with read_TS(*Xj*) = write_TS(*Xj*) = TS(*T*).
  2. If transaction *T* issues a read_item(*X*) operation, find the version *i* of *X* that has the highest write_TS(*Xi*) of all versions of *X* that is also *less than or equal to* TS(*T*); thenreturnthe value of *Xi*totransaction *T,* and set thevalue ofread_TS(*Xi*) to the larger of TS(*T*) and the current read_TS(*Xi*).

MultiversionTwo-PhaseLockingUsingCertifyLocks

- Inthismultiple-modelockingscheme,thereare*threelockingmodes*foranitem: read, write, and *certify*
- Hence,thestateofLOCK(*X*)foranitem*X*canbeoneofread-locked,writelocked, certify-locked, or unlocked
- We candescribetherelationshipbetweenreadandwritelocksinthestandard scheme by means of the **lock compatibility table** shown in Figure 22.6(a)
- An entry of *Yes* means that if a transaction *T* holds the type of lock specified in the column header on item *X* and if transaction *T*_ requests the type of lock specified in

therowheaderonthesameitem*X*,then*T_canobtainthelock*becausethelocking modes are compatible

(a)

	Read	Write
Read	Yes	No
Write	No	No

(b)

	Read	Write	Certify
Read	Yes	Yes	No
Write	Yes	No	No
Certify	No	No	No

Figure22.6: Lockcompatibilitytables.(a)Acompatibilitytableforread/writelockingscheme. (b)Acompatibilitytableforread/write/certifylockingscheme.

- Ontheotherhand,anentryof*No*inthetableindicatesthatthelocksarenotcompatible, so *T' must wait* until *T releases* the lock
- Theideabehindmultiversion2PListoallowothertransactions*T'*toreadanitem*X* whileasingletransaction*T*holdsawritelockon*X*
- Thisisaccomplishedbyallowing*twoversions*foreachitem*X*;oneversionmustalways have been written by some committed transaction
- Thesecondversion*X'*iscreatedwhenatransaction*T*acquiresawritelockontheitem

Validation(Optimistic)ConcurrencyControlTechniques

- In**optimisticconcurrencycontroltechniques**,alsoknownas**validation**or **certificationtechniques**,*nochecking*isdonewhilethetransactionisexecuting
- Inthisscheme,updatesinthetransactionare*not*applieddirectlytothedatabaseitems until the transaction reaches its end

- Duringtransactionexecution,allupdatesareappliedto*localcopies*ofthedataitems that are kept for the transaction
- At theendoftransactionexecution,a **validationphase**checkswhetheranyofthe transaction's updates violate serializability.
- Therearethreephasesforthisconcurrencycontrolprotocol:
  1. **Read phase.** A transaction can read values of committed data items from the database. However, updates are applied only to local copies (versions) of the data items kept in the transaction workspace.
  2. **Validation phase.** Checking is performed to ensure that serializability will not be violated if the transaction updates are applied to the database.
  3. **Write phase.** If the validation phase is successful, the transaction updates are appliedto thedatabase;otherwise,theupdates are discarded andthetransaction is restarted.
- The idea behind optimistic concurrency control is to do all the checks at once; hence, transaction execution proceeds with a minimum of overhead until the validation phase is reached
- The techniques are called *optimistic* because they assume that little interference will occur and hence that there is no need to do checking during transactionexecution.
- ThevalidationphaseforTichecksthat, for*each*suchtransaction*Tj*that iseither committed or is in its validation phase, *one* of the following conditionsholds:
  1. Transaction*Tj*completesitswritephasebefore*Ti*startsitsreadphase.
  2. *Ti*startsitswritephaseafter *Tj*completesitswritephase,andtheread_set of *Ti* has no items in common with the write_set of *Tj*.
  3. Both the read_set and write_set of *Ti* have no items in common with the write_setof*Tj*,and*Tj*completesitsreadphasebefore*Ti*completesitsread phase.

GranularityofDataItemsandMultipleGranularityLocking

- Allconcurrencycontroltechniquesassumethatthedatabaseisformedofanumber of named data items. A database item could be chosen to be one of thefollowing:
  - Adatabaserecord
  - Afieldvalueof adatabaserecord
  - Adiskblock
  - Awholefile

- Thewholedatabase

- Thegranularitycanaffecttheperformanceofconcurrencycontrolandrecovery

GranularityLevelConsiderationsforLocking

- Thesizeofdataitemsisoftencalledthe**dataitemgranularity**.
- *Finegranularity*referstosmallitemsizes,whereas *coarsegranularity*referstolarge item sizes
- Thelargerthedataitemsizeis,thelowerthedegreeofconcurrencypermitted.
- For example, if the data item size is a disk block, a transaction *T* that needs to lock a record *B* must lock the whole disk block *X* that contains *B* because a lock is associated with the whole data item (block). Now, if another transaction *S* wants to lock a different record *C* that happens to reside in the same block *X* in a conflicting lock mode, it is forced to wait. If the data item size was a single record, transaction *S* would be able to proceed, because it would be locking a different data item (record).
- The smaller the data item size is, the more the number of items in the database. Because every item is associated with a lock, the system will have a larger number of active locks to be handled by the lock manager. More lock and unlock operations will be performed, causing a higher overhead
- Thebestitemsize*dependsonthetypesoftransactionsinvolved*.
- If a typical transaction accesses a small number of records, it is advantageous to have the data item granularity be one record
- On the other hand, if a transaction typically accesses many records in the same file, it may be better to have block or file granularity so that the transaction will consider all those records as one (or a few) data items

MultipleGranularityLevelLocking

- Since the best granularity size depends on the given transaction, it seems appropriate that a database system should support multiple levels of granularity, where the granularity level can be different for various mixes oftransactions
- Figure 22.7 shows a simple granularity hierarchy with a database containing two files, each file containing several disk pages, and each page containing severalrecords.
- This can be used to illustrate a **multiple granularity level** 2PL protocol, where a lock can be requested at any level

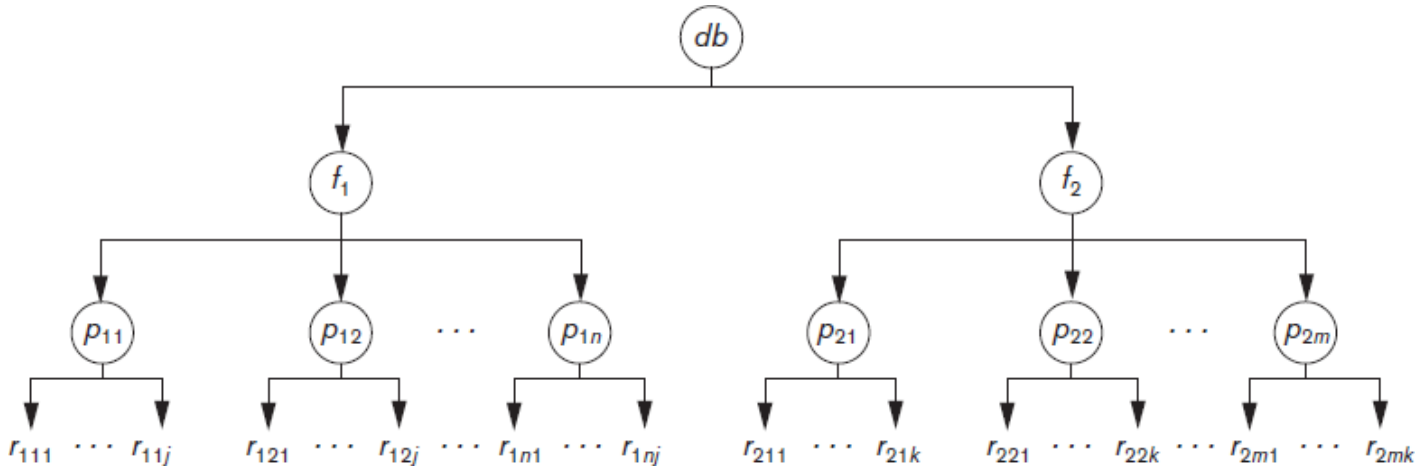


Figure22.7Agranularityhierarchyforillustratingmultiplegranularitylevellocking

- Tomakemultiplegranularitylevellockingpractical,additionaltypesoflocks,called **intentionlocks**,areneeded
- The idea behind intention locks is for a transaction to indicate, along the path from the rootto the desired node, what type of lock (shared or exclusive) it will require from one of the node's descendants.
- Therearethreetypesofintentionlocks:
  1. Intention-shared(IS)indicatesthat oneormoresharedlockswillberequestedonsome descendant node(s).
  2. Intention-exclusive(IX)indicatesthatoneormoreexclusivelockswillberequestedon some descendant node(s).
  3. Shared-intention-exclusive(SIX)indicatesthatthecurrentnodeislockedinshared mode but that one or more exclusive locks will be requested on some descendant node(s).
- Thecompatibilitytableofthethreeintentionlocks,andthesharedandexclusive locks, is shown in Figure 22.8.

IS	Yes	Yes	Yes	Yes	No
IX	Yes	Yes	No	No	No
S	Yes	No	Yes	No	No
SIX	Yes	No	No	No	No
X	No	No	No	No	No

Figure22.8: Lockcompatibilitymatrixformultiplegranularitylocking.

- The**multiplegranularitylocking(MGL)**protocolconsistsofthefollowingrules:
  1. Thelockcompatibility(basedonFigure22.8)mustbeadheredto.
  2. Therootofthetreemustbelockedfirst, inanymode.
  3. Anode*N*canbelockedbyatransaction*T*inSorISmodeonly iftheparent node *N* is already locked by transaction *T* in either IS or IXmode.
  4. A node *N* can be locked by a transaction *T* in X, IX,or SIX mode only ifthe parentofnode*N*isalreadylockedbytransaction*T*ineitherIXorSIXmode.
  5. Atransaction*T*canlockanodeonlyifithasnotunlockedanynode(to enforce the 2PL protocol).
  6. Atransaction*T*canunlockanode,*N*,onlyifnoneofthechildrenofnode*N* arecurrentlylockedby*T.*
- Themultiplegranularitylevelprotocolisespeciallysuitedwhenprocessingamixof transactions that include
  (1) shorttransactionsthataccessonlyafewitems(recordsorfields)and
  (2) longtransactionsthataccessentirefiles.

Chapter3:IntroductiontoDatabaseRecoveryProtocols

RecoveryConcepts

RecoveryOutlineandCategorizationofRecovery Algorithms

- Recoveryfromtransactionfailuresusuallymeansthatthedatabaseis *restored*to the most recent consistent state just before the time of failure
- To do this, the system must keep information about the changes that were appliedto data items by the various transactions. This information is typically kept in the **system log**.
- Conceptually, we can distinguish two main techniques for recovery from noncatastrophic transaction failures: **deferred update** and **immediate update**.
- The**deferredupdate**techniques
  - donotphysicallyupdatethedatabaseondiskuntil*after*atransactionreachesits commit point; then the updates are recorded in the database
  - Before reaching commit, all transaction updates are recorded in the local transactionworkspaceorinthemainmemorybuffersthattheDBMSmaintains
  - Beforecommit,theupdatesarerecordedpersistentlyinthelog,andthenafter commit, the updates are written to the database on disk
  - If a transaction fails before reaching its commit point, it will not have changed the database in any way, so UNDO is not needed
  - It may be necessary to REDO the effect of the operations of a committed transaction fromthe log,because theireffect maynot yet have beenrecorded in the database on disk
  - Hence,deferredupdateisalsoknownasthe**NO-UNDO/REDO algorithm**
- The**immediateupdate**techniques
  - the database *may be updated* by some operations of a transaction *before* the transaction reaches its commit point.
  - However, these operations must also be recorded in the log *on disk* by force-writing *before* they are applied to the database on disk, making recovery still possible

- If a transaction fails after recording some changes in the database on disk but before reaching its commit point, the effect of its operations on the database must be undone; that is, the transaction must be rolled back
- In the general case of immediate update, both *undo* and *redo* may be required during recovery.
- This technique, known as the **UNDO/REDO algorithm**, requires both operations during recovery, and is used most often in practice.

Caching(Buffering)ofDiskBlocks

- Itisconvenienttoconsiderrecoveryintermsofthedatabasediskpages(blocks).
- Typicallyacollectionofin-memorybuffers,calledthe**DBMScache**,iskeptunderthe control of the DBMS for the purpose of holding these buffers.
- A**directory**forthecacheisusedtokeeptrackofwhichdatabaseitemsareinthe buffers
- Thiscanbeatableof<Disk_page_address,Buffer_location,...>entries.
- WhentheDBMSrequestsactiononsomeitem,first itchecksthecachedirectoryto determine whether the disk page containing the item is in the DBMScache.
- Ifit isnot,theitemmustbelocatedondisk, andtheappropriatediskpagesarecopied into thecache.Itmay benecessaryto**replace**(or **flush**)someofthecachebuffersto make space available for the new item.
- TheentriesintheDBMScachedirectoryholdadditionalinformationrelevanttobuffer management.
- Associatedwitheachbufferinthecacheisa**dirtybit**,whichcanbeincludedinthe directory entry, to indicate whether or not the buffer has beenmodified.
- When a page is first readfrom the database disk into a cache buffer, a new entry is insertedinthecachedirectorywiththenewdiskpageaddress,andthedirtybit issetto 0 (zero).
- Assoonasthebufferismodified,thedirtybitforthecorrespondingdirectoryentryisset to 1 (one)
- Additionalinformation,suchasthetransactionid(s)ofthetransaction(s)thatmodified the buffer can also be kept in the directory
- Whenthebuffercontentsarereplaced(flushed)fromthecache,thecontentsmustfirst be written back to the corresponding disk page *only if its dirty bit is 1*

- Anotherbit,calledthe**pin-unpin**bit,isalsoneeded—apageinthecacheis **pinned**(bit value 1 (one)) if it cannot be written back to disk as yet.
- Twomainstrategiescanbeemployedwhenflushingamodifiedbufferbacktodisk.
  - The first strategy, known as **in-place updating**, writes the buffer to the *same original disk location,* thus overwriting the old value of any changed data items on disk. Hence, a single copy of each database disk block ismaintained.
  - The second strategy, known as **shadowing**, writes an updated buffer at adifferent disk location, so multiple versions of data items can be maintained, but this approach is not typically used in practice.

Write-AheadLogging,Steal/No-Steal,andForce/No-Force

- Whenin-placeupdatingisused,itisnecessarytousealogforrecovery
- In this case, the recovery mechanism must ensure that the BFIM of the data item is recorded in the appropriate log entry and that the log entry is flushed to disk before the BFIM is overwritten with the AFIM in the database ondisk.
- This process is generally known as **write-aheadlogging**, and is necessary to be ableto UNDO the operation if this is required during recovery
- A **REDO-type log entry** includes the **new value** (AFIM) of the item written by the operation since this is needed to *redo* the effect of the operation from the log (by setting the item value in the database on disk to its AFIM).
- The **UNDO-type log entries** include the **old value** (BFIM) of the item since this is needed to *undo* the effect of the operation from the log (by setting the item value in the database back to its BFIM)
- In an UNDO/REDO algorithm, both types of log entries are combined. Additionally, when cascading rollback is possible, read_item entries in the log are considered to be UNDO-type entries
- Standard DBMS recovery terminology includes the terms **steal/no-steal** and **force/no-force**, whichspecifytherulesthat govern *when* apagefromthedatabasecan bewritten to disk from the cache:
  1. If a cache buffer page updated by a transaction *cannot* be written to disk before the transaction commits, the recovery method is called a **no-steal approach**.The pin-unpin bit will be used to indicate if a page cannot be written back to disk. On the other hand, if the recovery protocol allows writing an updated buffer*before* the transaction commits, it is called **steal**. Steal is used when the DBMS

cache (buffer) manager needs a buffer frame for another transaction and the buffer manager replaces an existing page that had been updated but whose transaction has not committed. The *no-steal rule* means that UNDO will never be needed during recovery, since a committed transaction will not have any of its updates on disk before it commits.

2. If all pages updated by a transaction are immediately written to disk *before* the transaction commits, it is called a **force approach**. Otherwise, it is called **no-force**. The *force rule* means that REDO will never be needed during recovery, since any committed transaction will have all its updates on disk before it is committed.

- Thedeferredupdate(NO-UNDO)recoveryschemefollowsa*no-steal*approach.
- However,typicaldatabasesystemsemploya*steal/no-force*strategy.
- The *advantage of steal* is that it avoids the need for a very large buffer space to store all updated pages in memory.
- The *advantage of no-force* is that an updated page of a committed transaction may still be in the buffer when another transaction needs to update it, thus eliminatingthe I/O cost to write that page multiple times to disk, and possibly to have to read it again from disk.
- To permit recovery when in-place updating is used, the appropriate entries required for recovery must be permanently recorded inthe log on disk before changes are applied to the database.
- For example,considerthe following **write-aheadlogging(WAL**) protocolfor arecovery algorithm that requires both UNDO and REDO:
  1. The before image of an item cannot be overwritten by its after image in thedatabase on disk until all UNDO-type log records for the updating transaction— upto this point—have been force-written to disk.
  2. The commit operation of a transaction cannot be completed until all the REDO-type and UNDO-type log records for that transaction have been force written todisk.

CheckpointsintheSystemLogandFuzzyCheckpointing

- Anothertypeofentryinthelogiscalleda**checkpoint**.
- A[checkpoint,list of active transactions] record iswritteninto thelogperiodicallyat that point when the system writes out to the database on disk all DBMS buffers that have been modified

- As a consequence of this, all transactions that have their [commit, T] entries in the log before a [checkpoint] entry do not need to have their WRITE operations redone in case of a system crash, since all their updates will be recorded in the database on diskduring checkpointing
- As part of checkpointing, the list of transaction idsfor active transactions at the time of the checkpoint is included in the checkpoint record, so that these transactions can be easily identified during recovery.
- TherecoverymanagerofaDBMSmustdecideatwhatintervalstotakeacheckpoint.
- The interval may be measured intime—say, every *m* minutes—or in the number *t* of committedtransactionssincethelastcheckpoint,wherethevaluesof *m*or*t*aresystem parameters
- Takingacheckpointconsistsofthefollowingactions:
  1. Suspendexecutionoftransactionstemporarily.
  2. Force-writeallmainmemorybuffersthathavebeenmodifiedtodisk.
  3. Writea[checkpoint]recordtothelog,andforce-writethelogtodisk.
  4. Resumeexecutingtransactions.

- Thetimeneededtoforce-writeallmodifiedmemorybuffersmaydelaytransaction processing because of step 1
- Toreducethisdelay,it iscommontouseatechniquecalled**fuzzycheckpointing**.
- In this technique, the system can resume transaction processing after a [begin_checkpoint]record is writtentothelog without havingtowaitforstep 2tofinish.
- Whenstep2iscompleted,an[end_checkpoint,...]recordiswritten inthelogwith the relevant information collected during checkpointing.

TransactionRollbackandCascadingRollback

- Ifatransactionfailsforwhateverreasonafterupdatingthedatabase,butbeforethe transaction commits, it may be necessary to **roll back** the transaction
- Ifanydataitemvalueshavebeenchangedbythetransactionandwrittentothe database, they must be restored to their previous values (BFIMs)
- Theundo-typelogentriesareusedtorestoretheoldvalues ofdataitemsthatmustbe rolled back

- IfatransactionTisrolledback, anytransactionSthat has,intheinterim, readthevalue of some data item *X* written by *T* must also be rolled back
- Similarly, once *S* is rolled back, any transaction *R* that has read the value of some data item *Y* written by *S* must also be rolled back; and so on.
- This phenomenon is called **cascading rollback**, and can occur when the recovery protocol ensures *recoverable* schedules but does not ensure *strict* or *cascadeless* schedules
- Figure23.1showsanexamplewherecascadingrollbackisrequired.
- Thereadandwriteoperationsofthreeindividualtransactionsareshowninfigure 23.1(a).
- Figure23.1(b)showsthesystemlogatthepointof asystemcrashforaparticular execution schedule of these transactions.
- Thevaluesofdataitems*A*,*B*,*C*,and*D*,whichareusedbythetransactions,areshown to the right of the system log entries.
- We assumethattheoriginalitemvalues,shown inthefirstline, are *A*=30,*B*=15,*C*= 40, and *D* = 20.
- At thepointofsystemfailure,transaction*T*3hasnotreacheditsconclusionandmust be rolled back.
- TheWRITEoperationsof *T*3,markedbyasingle*inFigure23.1(b), arethe *T*3 operations that are undone during transaction rollback.
- Figure23.1(c)graphicallyshowstheoperationsofthedifferenttransactionsalongthe time axis

(a)

T_1
read_item(*A*)
read_item(*D*)
write_item(*D*)

T_2
read_item(*B*)
write_item(*B*)
read_item(*D*)
write_item(*D*)

T_3
read_item(*C*)
write_item(*B*)
read_item(*A*)
write_item(*A*)

(b)

		A	B	C	D
		30	15	40	20
	[start_transaction, T_3]				
	[read_item, T_3, C]				
*	[write_item, T_3, B, 15, 12]		12		
	[start_transaction, T_2]				
	[read_item, T_2, B]				
**	[write_item, T_2, B, 12, 18]		18		
	[start_transaction, T_1]				
	[read_item, T_1, A]				
	[read_item, T_1, D]				
	[write_item, T_1, D, 20, 25]				25
	[read_item, T_2, D]				
**	[write_item, T_2, D, 25, 26]				26
	[read_item, T_3, A]				

← System crash

* T_3 is rolled back because it did not reach its commit point.

** T_2 is rolled back because it reads the value of item B written by T_3.

(c)

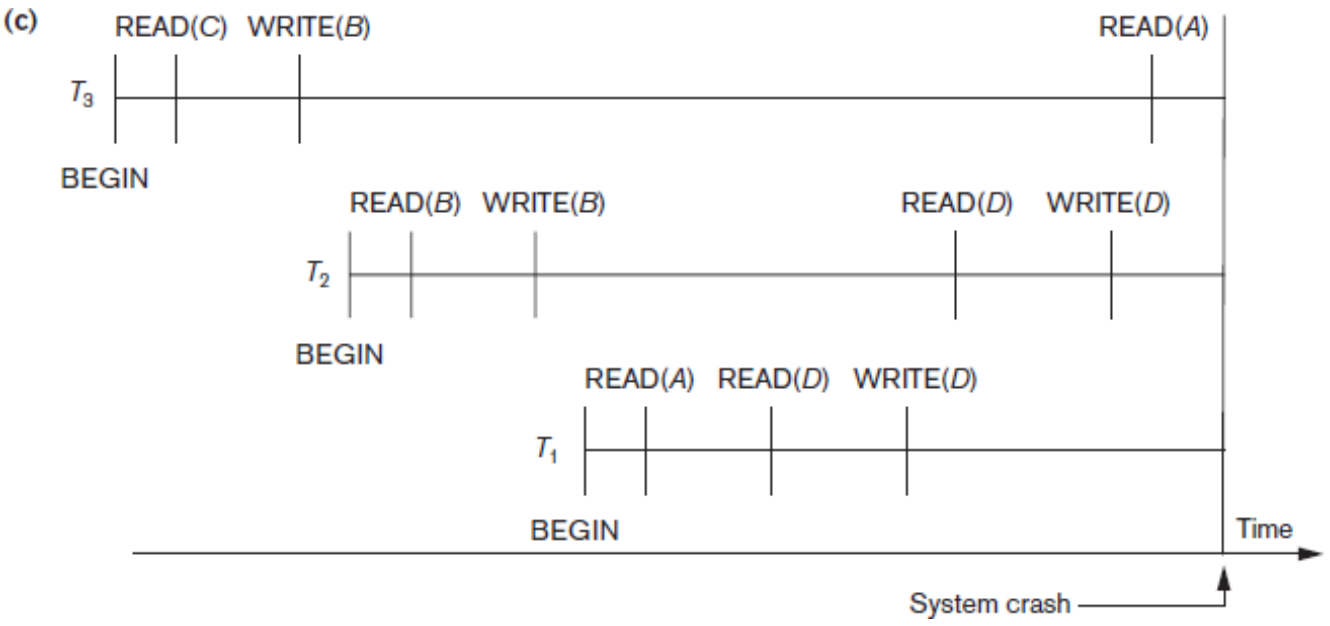


Figure23.1: Illustratingcascadingrollback(aprocessthatneveroccursinstrictorcascadeless schedules). (a) The read and write operations of three transactions. (b) System log at point of crash. (c) Operations before the crash.

TransactionActionsThatDoNotAffecttheDatabase

- In general, a transaction will have actions that do *not* affect the database, such as generating and printing messages or reports from information retrieved from the database
- If a transaction fails before completion, we may not want the user to get these reports, since the transaction has failed to complete.
- If such erroneous reports are produced, part of the recovery process would have to inform the user that these reports are wrong, since the user may take an action basedon these reports that affects the database.
- Hence, such reports should be generated only *after the transaction reaches its commit point.*
- A common method of dealing with such actions is to issue the commands that generate the reports but keep them as batch jobs, which are executed only after the transaction reaches its commit point. Ifthe transaction fails, the batch jobs are canceled.

NO-UNDO/REDORecoveryBasedonDeferredUpdate

- The idea behind deferred update is to defer or postpone any actual updates to the database on disk until the transaction completes its execution successfully and reaches its commit point.
- During transaction execution, the updates are recorded only in the log and in the cache buffers.
- After the transaction reaches its commit point and the log is forcewritten to disk, the updates are recorded in the database.
- If a transaction fails before reaching its commit point, there is no need to undo any operations because the transaction has not affected the database on disk in anyway.
- Therefore, only **REDO type log entries** are needed in the log, which include the **new value** (AFIM) of the item written by a write operation.
- The **UNDO-type log entries** are not needed since no undoing of operations will be required during recovery.
- Wecanstateatypicaldeferredupdateprotocolasfollows:
  1. Atransactioncannotchangethedatabaseondiskuntilitreachesitscommitpoint.
  2. Atransactiondoesnotreachitscommitpointuntilallits REDO-typelogentriesare recorded in the log *and* the log buffer is force-written todisk.

- Formultiusersystemswithconcurrencycontrol,theconcurrencycontrolandrecovery processes are interrelated.
- Assumingthat[checkpoint]entriesareincludedinthelog,apossiblerecoveryalgorithm for this case, which we call RDU_M (Recovery using Deferred Update in a Multiuser environment), is as follows:
    - **Procedure RDU_M (NO-UNDO/REDO with checkpoints).** Use two lists of transactions maintained by the system: the committed transactions *T* since the last checkpoint (**commit list**), and the active transactions *T*_ (**active list**). REDO all the WRITE operations of the committed transactions from the log, *in the order in which they were written into the log.* The transactions that are active and did not commit are effectively canceled and must be resubmitted.
- TheREDOprocedureisdefinedasfollows:
    - **Procedure REDO (WRITE_OP).** Redoing a write_item operation WRITE_OP consists of examining its log entry [write_item, *T*, *X*, new_value] and setting the value of item *X* in the database to new_value, which is the after image(AFIM).
- Figure23.2illustratesatimelineforapossiblescheduleofexecutingtransactions.

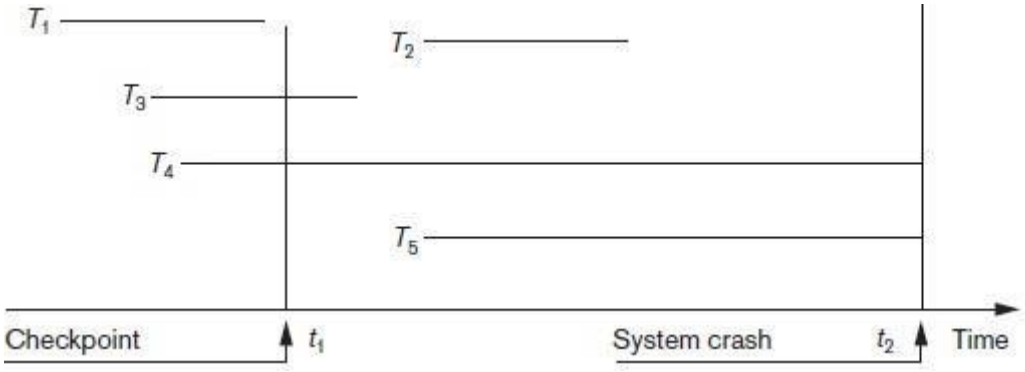


Figure 23.2
An example of a recovery timeline to illustrate the effect of checkpointing.

- Figure23.3showsanexampleofrecoveryforamultiusersystemthatutilizestherecovery and concurrency control method
-

(a)

T_1	T_2	T_3	T_4
read_item(A)	read_item(B)	read_item(A)	read_item(B)
read_item(D)	write_item(B)	write_item(A)	write_item(B)
write_item(D)	read_item(D)	read_item(C)	read_item(A)
	write_item(D)	write_item(C)	write_item(A)

(b)

[start_transaction,T_1]
[write_item, T_1, D, 20]
[commit, T_1]
[checkpoint]
[start_transaction, T_4]
[write_item, T_4, B, 15]
[write_item, T_4, A, 20]
[commit, T_4]
[start_transaction, T_2]
[write_item, T_2, B, 12]
[start_transaction, T_3]
[write_item, T_3, A, 30]
[write_item,T_2, D, 25] ← System crash

T_2 and T_3 are ignored because they did not reach their commit points.

T_4 is redone because its commit point is after the last system checkpoint.

Figure 23.3
An example of recovery using deferred update with concurrent transactions. (a) The READ and WRITE operations of four transactions. (b) System log at the point of crash.

- The method's main benefit is that transaction operations *never need to be undone,* for two reasons:
  1. A transaction does not record any changes in the database on disk until after it reaches its commit point—that is, until it completes its execution successfully. Hence, a transaction is never rolled back because of failure during transactionexecution.
  2. A transaction will never read the value of an item that is written by an uncommitted transaction, becauseitems remain locked until atransaction reaches its commit point. Hence, no cascading rollback will occur.

RecoveryTechniquesBasedonImmediateUpdate

- In these techniques, when a transaction issues an update command, the database on disk can be updated *immediately*, without any need to wait for the transaction to reachits commit point.
- Provisions must be made for *undoing* the effect of update operations that have been applied to the database by a *failed transaction.* This is accomplished by rolling back the transaction and undoing the effect of the transaction's write_item operations.
- Therefore, the **UNDO-type log entries**, which include the **old value** (BFIM) of the item, must be stored in the log. Because UNDO can be needed during recovery, these methodsfollowa **steal strategy**fordeciding when updatedmainmemorybufferscan be written back to disk
- Theoretically,wecandistinguishtwomaincategoriesofimmediateupdatealgorithms.
- If the recovery technique ensures that all updates of a transaction are recorded in the database on disk *before the transaction commits,* there is never a need to REDO any operations of committed transactions. This is called the **UNDO/NO-REDO recovery algorithm**.
- In this method, all updates by a transaction must be recorded on disk *before the transactioncommits*,sothat REDO is never needed. Hence,thismethod mustutilize the **force strategy** for deciding when updated main memory buffers are written back todisk
- If the transaction is allowed to commit before all its changes are written to the database, we havethemostgeneralcase, knownasthe **UNDO/REDO recoveryalgorithm**.Inthis case, the **steal/no-force strategy** is applied.
- When concurrent execution is permitted, the recovery process again depends on the protocols used for concurrency control.
- The procedureRIU_M(Recovery usingImmediate Updates for aMultiuser environment) outlines a recovery algorithm for concurrent transactions with immediate update (UNDO/REDO recovery).
- **ProcedureRIU_M(UNDO/REDOwithcheckpoints).**

  1. Usetwo lists of transactions maintained by the system: the committed transactions since the last checkpoint and the active transactions.
  2. Undo all the write_item operations of the *active* (uncommitted) transactions, using the UNDO procedure. The operations should be undone in the reverse of the order in which they were written into the log.
  3. Redoallthewrite_itemoperationsofthe*committed*transactionsfromthelog,in

theorderinwhichtheywerewrittenintothelog,usingtheREDOproceduredefined earlier.

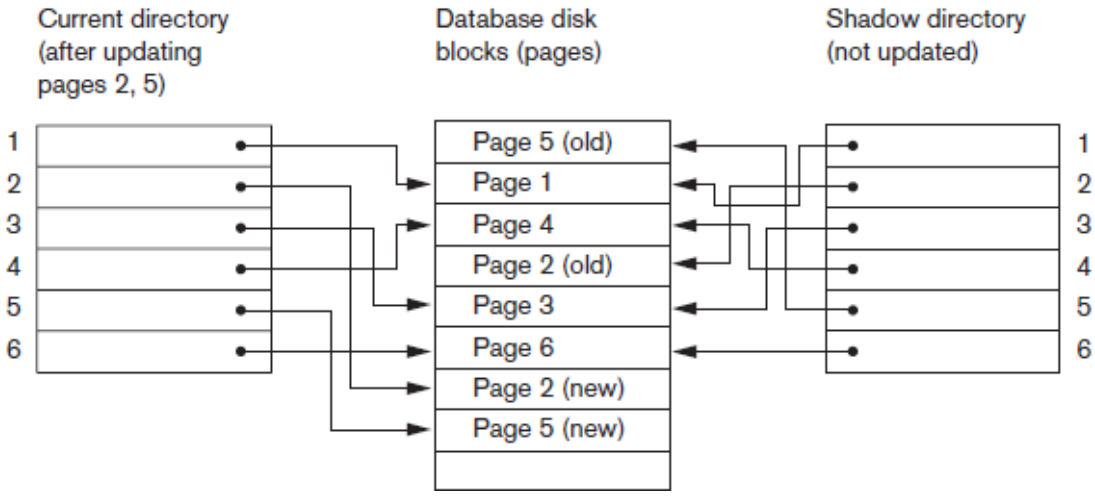
- TheUNDOprocedureisdefinedasfollows:

  Procedure UNDO (WRITE_OP). Undoing a write_item operation write_op consists of examining its log entry [write_item, *T*, *X*, old_value, new_value] and setting the value of item *X* in the database to old_value, which is the before image (BFIM). Undoing a number of write_item operations from one or more transactions from the log must proceed in the *reverse order* from the order in which the operations were written in the log.

ShadowPaging

- Thisrecoveryschemedoesnotrequiretheuseofaloginasingle-userenvironment.
- Inamultiuserenvironment,alogmaybeneededfortheconcurrencycontrolmethod.
- Shadowpagingconsidersthedatabasetobemadeupofanumberoffixedsizediskpages (or disk blocks)—say, *n*—for recovery purposes
- A**directory**with*n*entries5isconstructed, wherethe*i*thentrypointstothe*i*thdatabase page on disk.
- Thedirectoryiskept inmainmemory ifit isnottoolarge,andallreferences—readsor writes—to database pages on disk go through it.
- Whenatransactionbeginsexecuting,the**currentdirectory**—whoseentriespointtothe most recent or current database pages on disk—is copied into a **shadowdirectory**.
- Theshadowdirectoryisthensavedondiskwhilethecurrentdirectoryisusedbythe transaction.
- Duringtransactionexecution,theshadowdirectoryis*never*modified.
- When a write_item operation is performed, a new copy of the modified database page is created, but the old copy of that page is *not overwritten.* Instead, the new page is written elsewhere—on some previously unused disk block.
- The current directory entry is modified to point to the new disk block, whereas the shadow directory is not modified and continues to point to the old unmodified diskblock.
- Figure 23.4 illustrates the concepts of shadow and current directories. For pages updated by the transaction, two versions are kept.
- The old version is referenced by the shadow directory and the new version by the current directory.

Figure 23.4
An example of shadow paging.



- Torecoverfromafailureduringtransactionexecution,itissufficienttofreethemodified database pages and to discard the currentdirectory.
- Thestateofthedatabasebeforetransactionexecutionisavailablethroughtheshadow directory, and that state is recovered by reinstating the shadowdirectory.
- Sincerecoveryinvolvesneitherundoingnorredoingdataitems,thistechniquecanbe categorized as a NOUNDO/ NO-REDO technique for recovery.
- Disadvantageofshadowpaging:
  - theupdateddatabasepageschangelocationondisk
  - ifthedirectoryislarge,theoverheadofwritingshadowdirectoriestodiskas transactions commit is significant
  - Afurthercomplicationishowtohandle**garbagecollection**whenatransactioncommits
  - Anotherissueisthattheoperationtomigratebetweencurrentandshadowdirectories must be implemented as an atomicoperation.

TheARIESRecoveryAlgorithm

- Itisusedinmanyrelationaldatabase-relatedproductsofIBM.
- ARIESusesasteal/no-forceapproachforwriting,anditisbasedonthreeconcepts:
  1. write-aheadlogging
  2. repeatinghistoryduringredo,and
  3. loggingchangesduringundo.

- **repeating history**, means that ARIES will retrace all actions of the database system prior to the crash to reconstruct the database state *when the crash occurred.* Transactions that were uncommitted at the time of the crash (active transactions) are undone.
- **logging during undo**, will prevent ARIES from repeating the completed undooperations if a failure occurs during recovery, which causes a restart of the recovery process.
- TheARIESrecoveryprocedureconsistsofthreemainsteps:
  1. Analysis
  2. REDO
  3. UNDO.

- The**analysisstep**
  - identifiesthedirty(updated)pagesinthebufferandthesetoftransactionsactive at the time of the crash
  - TheappropriatepointinthelogwheretheREDOoperationshouldstartisalso determined
- The**REDO phase**
  - reappliesupdatesfromthelogtothedatabase.
  - CertaininformationintheARIESlogwillprovidethestartpointforREDO,from which REDO operations are applied until the end of the log isreached
- The**UNDOphase**
  - thelogisscannedbackwardandtheoperationsoftransactionsthatwereactiveat the time of the crash are undone in reverse order.
- The information needed for ARIES to accomplish its recovery procedure includes the log, theTransactionTable,andtheDirtyPageTable.Additionally, checkpointingisused.
- Thesetablesaremaintainedbythetransactionmanagerandwrittentothelogduring checkpointing.
- InARIES,everylogrecordhasanassociated**logsequencenumber(LSN)** thatis monotonically increasing and indicates the address of the log record ondisk.
- EachLSNcorrespondstoa*specificchange*(action)ofsometransaction.
- Besidesthelog,twotablesareneededforefficientrecovery:the**TransactionTable**and the **Dirty Page Table**, which are maintained by the transaction manager.
- Whenacrashoccurs,thesetablesarerebuiltintheanalysisphaseofrecovery.

- The Transaction Table contains an entry for *each active transaction,* with information such asthe transaction ID, transaction status, and the LSN of themostrecent log record for the transaction.
- The Dirty Page Table contains an entry for each dirty page in the buffer, which includes the page ID and the LSN corresponding to the earliest update to that page.
- **Checkpointing** in ARIES consists of the following: writing a begin_checkpoint record to the log, writing an end_checkpoint record to the log, and writing *the LSN of* the begin_checkpoint record to a special file.
- Thisspecialfileisaccessedduringrecoverytolocatethelastcheckpointinformation
- After a crash, the ARIES recovery manager takes over. Information from the last checkpoint is first accessed through the special file.
- The **analysis phase** starts at the begin_checkpoint record and proceeds to the end of the log
- The **REDO phase** follows next. To reduce the amount of unnecessary work, ARIES starts redoing at a point in the log where it knows (for sure) that previous changes todirty pages *have already been applied to the database on disk.*

DatabaseBackupandRecoveryfromCatastrophicFailures

- Akeyassumptionhasbeenthatthesystemlogismaintainedonthediskandisnotlost as a result of the failure.
- Similarly,theshadowdirectorymustbestoredondisktoallowrecoverywhenshadow paging is used.
- Therecoverytechniquesusetheentriesinthesystemlogortheshadowdirectoryto recover from failure by bringing the database back to a consistentstate.
- TherecoverymanagerofaDBMSmustalsobeequippedtohandlemore catastrophic
- failuressuchasdiskcrashes.
- The main technique used to handle such crashes is a **database backup**, in which the whole database and the log are periodically copied onto a cheap storage medium such as magnetic tapes or other large capacity offline storage devices.
- Incaseof acatastrophicsystemfailure, thelatestbackupcopycanbereloadedfromthe tape to the disk, and the system can be restarted.
- Data from critical applications such as banking, insurance, stock market, and other databases is periodically backed up in its entirety and moved to physically separate safe locations.

- Toavoid losing all the effects of transactions that have been executed since the last backup, it is customary to back up the system log at more frequent intervals than full databasebackup by periodically copying it to magnetictape.

AssignmentQuestions

1. Explainpropertiesofatransactionwithstatetransitiondiagram.
2. Discusstheproblemsthatcanoccurwhenconcurrenttransactionsareexecuted.
3. Discussthedifferenttypesoffailures.Whatismeantbycatastrophicfailure?
4. Discusstheactionstakenbytheread_itemandwrite_itemoperationsonadatabase.
5. Whatistwo-phaselockingprotocol?Howdoesitguaranteeserializaility?
6. Whatisaschedule?Explainwithexampleserial,nonserialandconflictserializable schedules.
7. Writeshortnoteson
   1. Writeaheadlogprotocol
   2. TimestampOrdering
   3. Twophaselockingprotocol
8. Discusstheproblemsofdeadlockandstarvation,andthedifferentapproachestodealing with these problems.
9. Describethewait-dieandwound-waitprotocolsfordeadlockprevention.
10. Discussthedeferredupdatetechniqueofrecovery.Whataretheadvantagesand disadvantages of this technique?
11. Describetheshadowpagingrecoverytechnique.
12. DescribethethreephasesoftheARIESrecoverymethod.

ExpectedOutcome

- Toexecutetransactionsbycreatingschedules
- Toobtainequivalentand serializableschedulestoavoidanomalies.
- Tocheckwhetherthegivenscheduleisserailizableornot.
- Tostudy locking protocols
- Toimproveresourceutilization byapplyingvarious formsoflocking protocol.

FurtherReading

- https://www.smartdraw.com/entity-relationship-diagram/
- https://en.wikipedia.org/wiki/Database_normalization
- www.databasteknik.se/webbkursen/relalg-lecture
- https://technet.microsoft.com/en-us/library/bb264565(v=sql.90).aspx
- pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/.../Ch16_Overview_Xacts.pdf