BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(Affiliated to the Visvesvaraya Technological University, Belagavi)

Department of Master of Computer Applications

Subject: Database Management System

Prepared by: Drakshaveni G
Assistant Professor
Dept.of MCA
BMSIT&M

Module -1

Introduction to Database

1.0 Introduction

Database is a collection of related data. Database management system is software designed to assist the maintenance and utilization of large scale collection of data. DBMS came into existence in 1960 by Charles. Integrated data store whichPPp is also called as the first general purpose DBMS. Again in 1960 IBM brought IMS-Information management system. In 1970 EdgorCodd at IBM came with new database called RDBMS. In 1980 then came SQL Architecture- Structure Query Language. In 1980 to 1990 there were

advances in DBMS e.g. DB2, ORACLE.

Data

.

- Data is raw fact or figures or entity.
- When activities in the organization takes place, the effect of these activities need

  to be recorded which is known as Data.

Information

- Processed data is called information
  The purpose of data . processing is to generate the information required for
- carrying out the business activities.

In general data management consists of following tasks

- Data capture: Which is the task associated with gathering the data as and when they originate.

- Data classification: Captured data has to be classified based on the nature and intended usage.

- Data storage: The segregated data has to be stored properly.

- Data arranging: It is very important to arrange the data properly

- Data retrieval: Data will be required frequently for further processing,

  Hence it is very important to create some indexes so that data can be retrieved easily.

- Data maintenance: Maintenance is the task concerned with keeping the data up-to-date.

- Data Verification: Before storing the data it must be verified for any error.

- Data Coding: Data will be coded for easy reference.

- Data Editing: Editing means re-arranging the data or modifying the data for presentation.

- Data transcription: This is the activity where the data is converted from one form into another.
- Data transmission: This is a function where data is forwarded to the place where it would be used further. sort**Metadata**.

metadata is definitional data thatprovidesinformationabout or documentation of other data managed within an application or environment. The term should be used with in any media. An item of metadata may describe   collection of data including multiple

content items and hierarchical levels, for example   database schema. In data processing, caution as all data is about something, and is therefore metadata.

- Database may be defined in simple terms as a collection of data

- A database is a collection of related data.

- The database can be of any size and of varying plexity.

- A database may be generated and maintained manually or it may be puterized.**Database** .

Database Management System

- A Database Management System (DBMS) is a collection of program that enables user to create and maintain a database.

- The DBMS is hence a general purpose software system that facilitates the process of defining constructing and manipulating database for various applications.

1.1 Characteristics of DBMS

- To incorporate the requirements of the organization, system should be designed for easy maintenance.
- Information systems should allow interactive access to data to obtain new information without writing fresh programs.
- System should be designed to co-relate different data to meet new requirements.
- An independent central repository, which gives information and meaning of available data is required.
- Integrated database will help in understanding the inter-relationships between data stored in different applications.
- The stored data should be made available for a ess by different users

  .

  simultaneously.
- Automatic recovery feature has to be provided to overe the problems with processing system failure.

DBMS Utilities

- A data loading utility:

  Which allows easy loading.of data from the external format without writing programs.
- A backup utility:

  Which allo s to make copies of the database periodically to help in cases of crashes and disasters.

- Recovery utility:

  Which allows to reconstruct the correct state of database from the backup and history of transactions.

- Monitoring tools:

  Which monitors the performance so that internal schema can be changed and database access can be optimized.

- File organization:

  Which allows restructuring the data from one type to another?

1.2 Difference between File system & DBMS

File System

1. File system is a collection of data. Any management with the file system, user has to write the procedures

2. File system gives the details of the data representation and Storage of data.
3. In File system storing and retrieving of data cannot be done efficiently. information5.Filesystemdoesn’tprovidecrashrecoverymechanim. .

Eg. While we are entering some data into the file if System crashes then content of the file is lost.

6. Protecting a file under file system is very difficult.

DBMS

1. DBMS is a collection of data nd user is not required to write the procedures for

.

managing the database.

2. DBMS provides an abstract view of data that hides the details.

3. DBMS is efficient to use since there are wide varieties of sophisticated techniques to store and retrieve the data.

4. DBMS takes care of Concurrent access using some form of locking.

5. DBMS has crash recovery mechanism, DBMS protects user from the effects of system failures.

6. DBMS has a good protection mechanism.

DBMS = Database Management System

RDBMS = Relational Database Management System

A database management system is, well, a system used to manage databases.
A relational database management system is a database management system used to

manage relational databases. A relational database is one where tables of data can have relationships based on primary and foreign keys.

1.3 Advantages of DBMS.

Due to its centralized nature, the database system can overe the disadvantages of the file system-based syste

DBMS provides the abstract view that hides these details.

1. **Data independency**:

Application program should not be exposed to details of data representation and storage

2. **Efficient data access**.:techniquestotore and retrieve data efficiently.

3. **Data integrity and security**:

Data is accessed through DBMS, it can enforce integrity constraints. E.g.: Inserting salary inform.tion for an employee.

4. **Data Administration** :

When users share data, centralizing the data is an important task, Experience professionals can minimize data redundancy and perform fine tuning which reduces retrieval time.

5. **Concurrent access and Crash recovery:** . DBMS utilizes a variety of sophisticated

DBMS schedules concurrent access to the data. DBMS protects user from the effects of system failure.

6. **Reduced application development time**.

DBMS supports important functions that are mon to many applications.

1.4 Functions of DBMS

 Data Definition: The DBMS provides functions to define the structure of the data in the application. These include defining and modifying the record structure, the type and size of fields and the various constraints to be satisfied by the data in

each field.

 Data Manipulation: Once the data structure is defined, data needs to be inserted, modified or deleted. These functions which perform these operations are part of DBMS. These functions can handle plashud and unplashuddata manipulation needs. Plashud queries are those which form part of the application. Unplashud

queries are ad-hoc queries which performed on a need basis.

 Data Security & Integrity: The DBMS contains modules which handle the

security and integrity of data in the application. .

 Data Recovery and Concurrency:Recoveryofthe data after system failure and concurrent access of records by multiple users is lso handled by DBMS.

 Data Dictionary Maintenance: Maintaining the data dictionary which contains the data definition of the application is also one of the functions of DBMS.

 Performance: Optimizing the performance of the queries is one of the important functions of DBMS. .

1.5 Role of Database Administrator.

Typically there are three types of users for a DBMS:

1. The END User who uses the application. Ultimately he is the one who actually puts the data into the system into use in business. This user need not know anything about the organization of data in the physical level.

2. The Application Programmer who develops the application programs. He/She has more knowledge about the data and its structure. He/she can manipulate the data using his/her programs. He/she also need not have access and knowledge of the plete data in the system.

3. The Data base Administrator (DBA) who is like the super-user of the system.

The role of DBA is very important and is defined by the following functions.

 Defining the schema: The DBA defines the schema which contains the structure of the data in the application. The DBA determines what data needs to be present in the system and how this data has to be presented and organized.

 Liaising with users: The DBA needs to interact continuously with the users to understand the data in the system and its use.

 Defining Security & Integrity checks: The DBA finds about the access restrictions to be defined and defines security checks accordingly. Data for backup and recovery. Defining backup proedureincludes specifying what data is to be backed up, the periodicity.of taking backups and also the medium and storage place to back p data.
Integrity checks are defined by the DBA.

 Defining Backup/Recovery Procedures: The DBA also defines procedures

 Monitoring performance: The DBA has to continuously monitor the performance of the queries and t ke the measures to optimize all the queries in the application.

1.6 Simplified Database System

Environment

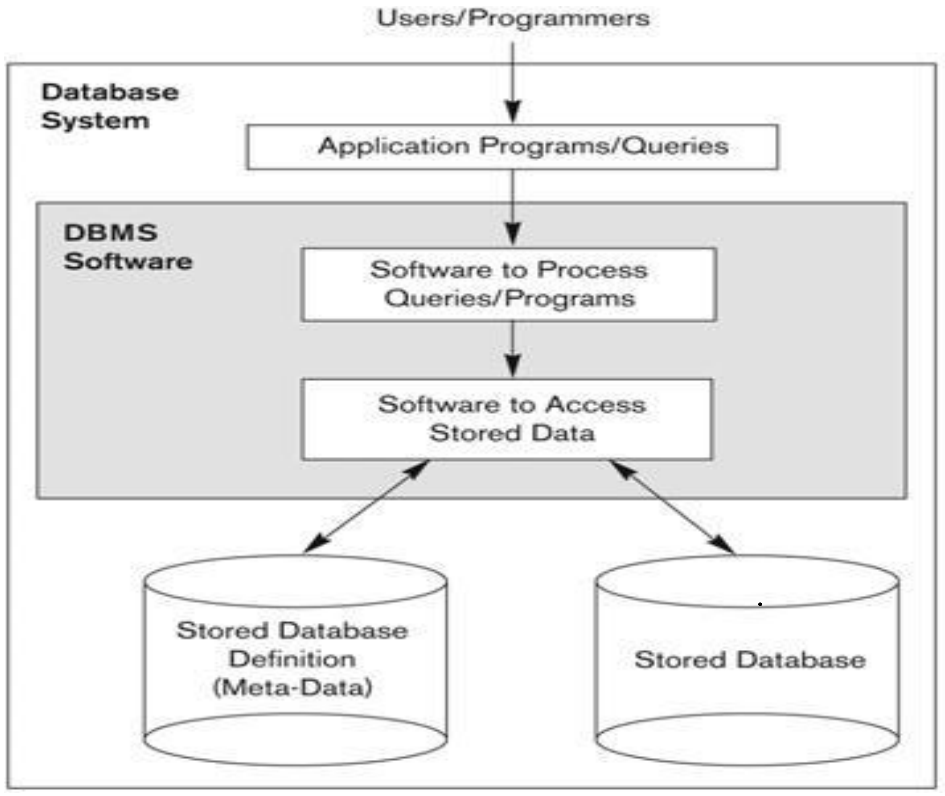

Users/Programmers
Database System
Application Programs/Queries
DBMS Software
Software to Process Queries/Programs
Software to Access Stored Data
Stored Database Definition (Meta-Data)
Stored Database

Figure 1.1 A simplified database system environment.

A database management system(DBMS)iscollectionofprograms that enables users to create and maintain database. The DBMS is general purpose software system that .facilitates the processofdefining, constructing, manipulating and sharing databases

among various users and applications. Defining a database specifying the database involves specifying the data types, constraints and structures of the data to be stored in the database. The descriptive information is also stored in the database in the form database catalog or dictionary; it is called meta-data.

Manipulating the data includes the querrying the database to retrieve the specific data. An application program accesses the database by sending the qurries or requests for data to DBMS.

The important function provided by the DBMS includes protecting the database and maintain the database.

1.7 Example of a Database (with a Conceptual Data Model)

- **Mini-world for the example:**

  Part of a UNIVERSITY environment.

- **Some mini-world *entities:***

  STUDENTs COURSEs

  SECTIONs (of COURSEs)

  (academic) DEPARTMENTs

  INSTRUCTORs

Example of a Database (with a Conceptual Data Model)

.

- **Some mini-world*relationships:***

  SECTIONs *are of specific COURSEs*

  STUDENTs *take SECTIONs*

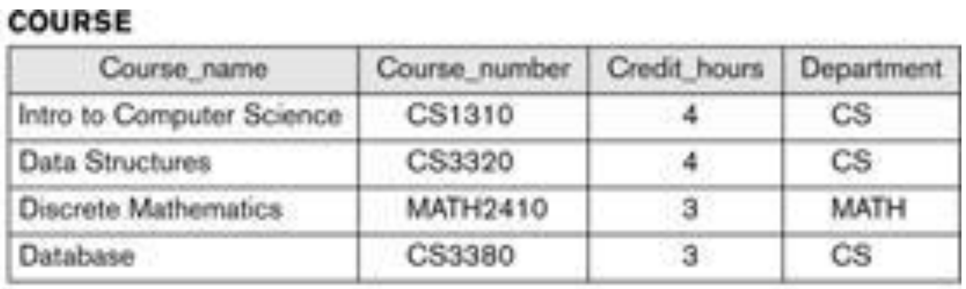
  COURSEs *have prerequisite COURSEs*

  INSTRUCTORs *teach SECTIONs*

  COURSEs *are offered by DEPARTMENTs*

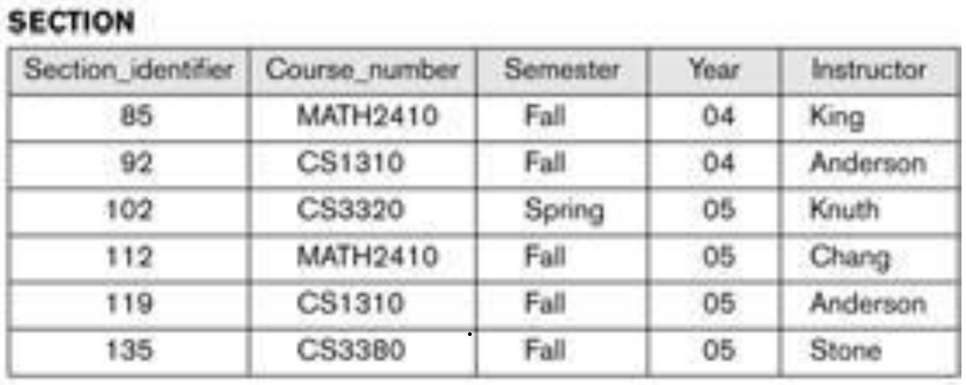
  STUDENTs *major in DEPARTMENTs*

Example of a simple Database



COURSE

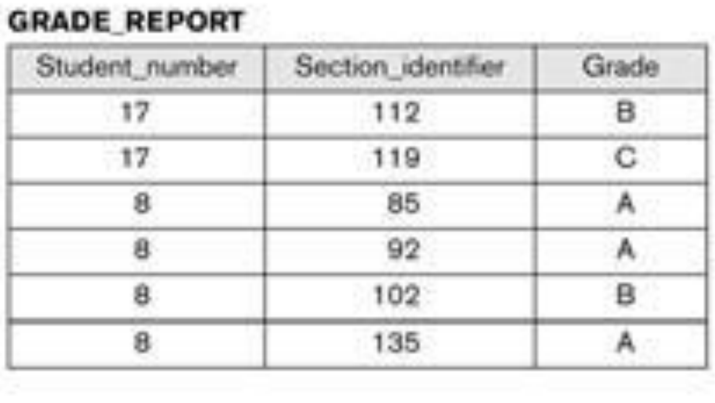
Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database	CS3380	3	CS



SECTION

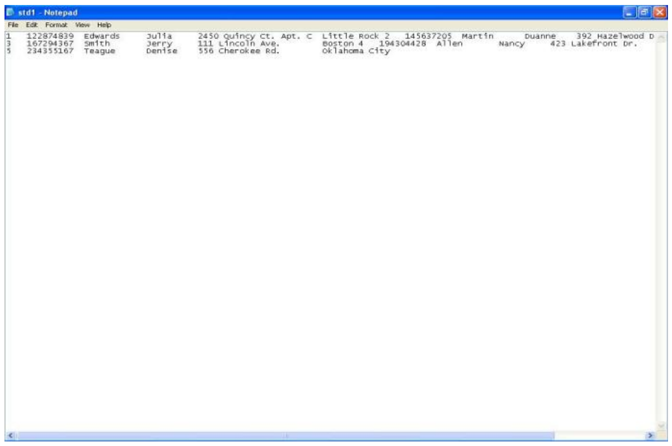
Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	Stone

Example of a simple Database



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

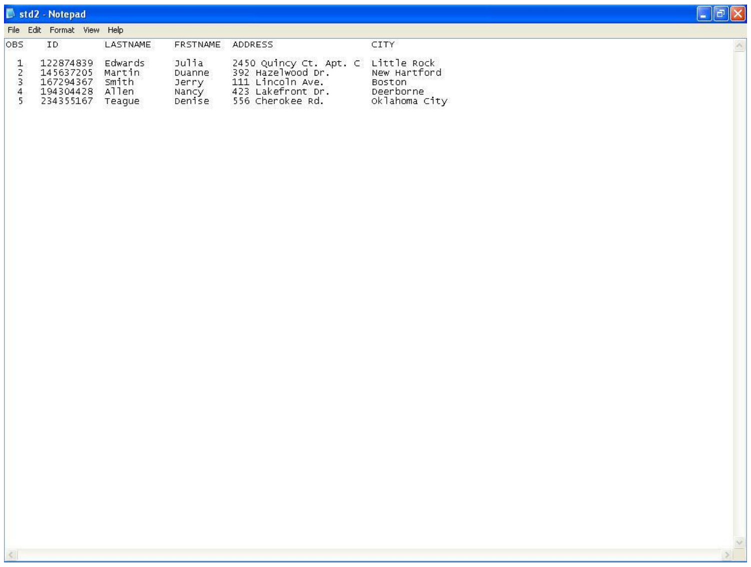


```
1    122874839  Edwards     Julia     2450 Quincy Ct. Apt. C  Little Rock 2   145637205  Martin      Duanne    392 Hazelwood D
3    167294367  Smith       Jerry     111 Lincoln Ave.        Boston 4   194304428  Allen       Nancy     423 Lakefront Dr.
5    234355167  Teague      Denise    556 Cherokee Rd.        Oklahoma City
```

Example of
a Student File

.

Example of a Student File

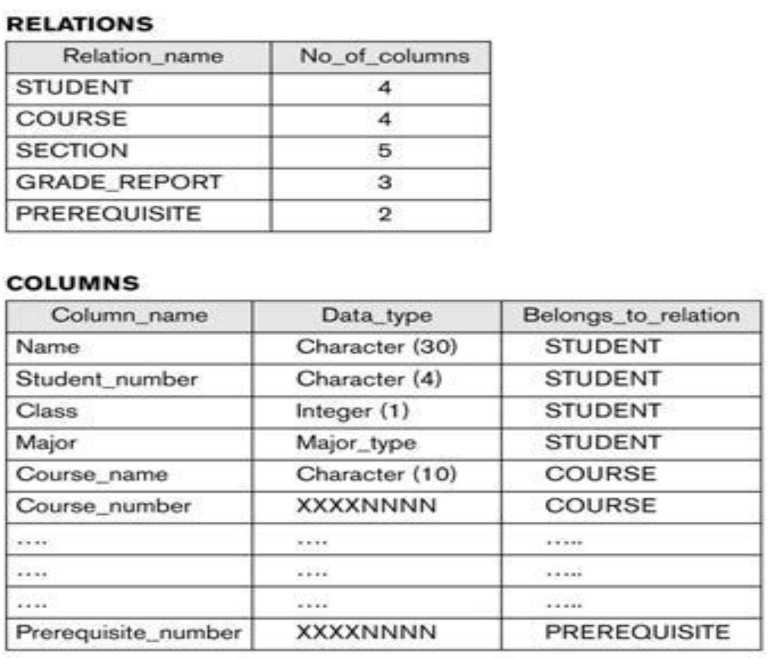


```
OBS    ID         LASTNAME   FRSTNAME  ADDRESS                CITY

  1   122874839   Edwards    Julia     2450 Quincy Ct. Apt. C  Little Rock
  2   145637205   Martin     Duanne    392 Hazelwood Dr.       New Hartford
  3   167294367   Smith      Jerry     111 Lincoln Ave.        Boston
  4   194304428   Allen      Nancy     423 Lakefront Dr.       Deerborne
  5   234355167   Teague     Denise    556 Cherokee Rd.        Oklahoma City
```

.

Example of a simplified database catalog



RELATIONS

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

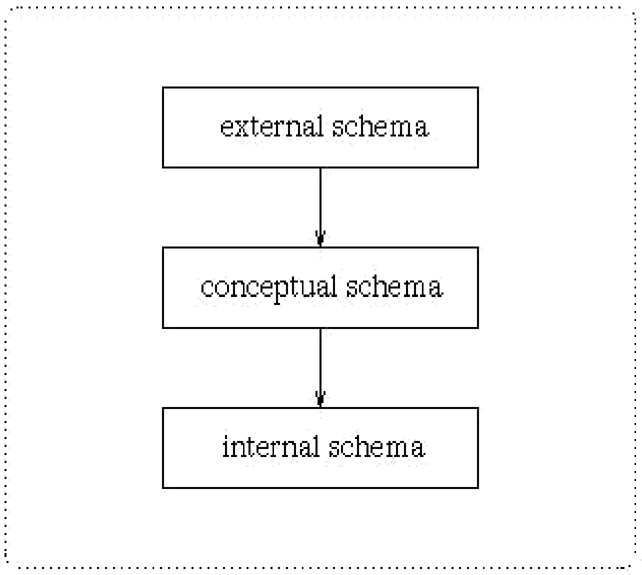
COLUMNS

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 1.3
An example of a database catalog for the database in Figure 1.2.

Note: Major_type is defined as an enumerared type with all known majors. XXXXNNNN is used to define a type with four alpha characters followed by four digits

1.8 Architecture of DBMS .


external schema
conceptual schema
internal schema

A monly used views of data approach is the three-level architecture suggested by ANSI/SPARC (American National Standards Institute/Standards Planning and Requirements mittee). ANSI/SPARC produced an interim report in 1972 followed by a final report in 1977. The reports proposed an architectural framework for databases. Under this approach, a database is considered as containing data about an *enterprise*. The three levels of the architecture are three different views of the data:

External - individual user view
Conceptual - munity user view
Internal - physical or storage view

The three level database architecture allows a clear separationoftheinforation meaning (conceptual view) from the external data representation and fr the physical data

structure layout. A database system that is able to separate.the three different views of data is likely to be flexible andadaptable.Thisflexibility and adaptability is data independence that we have discussed earlier.

We now briefly discuss the three different views.

The external level is the view that the individual user of the database has. This view is often a restricted view of the.dt se and the same database may provide a number of different views fordifferent classes of users. In general, the end users and even the application programmers are only interested in a subset of the database. For example, a department head may only be interested in the departmental finances and student enrolments but not the library information. The librarian would not be expected to have any interest in the information about academic staff. The payroll office would have no interest in student enrolments.

The conceptual view is the information model of the enterprise and contains the view of the whole enterprise without any concern for the physical implementation. This view is normally more stable than the other two views. In a database, it may be desirable to change the internal view to improve performance while there has been no change in the conceptual view of the database. The conceptual view is the overall munity view of the database and it includes all the information that is going to be represented in the database. The conceptual view is defined by the conceptual schema which includes definitions of each of the various types of data.

The internal view is the view about the actual physical storage of data. It tells us what data is stored in the database and how. At least the following aspects are considered at this level:

Storage allocation e.g. B-trees, hashing etc.

Access paths e.g. specification of primary and secondary keys, indexes and pointers and sequencing.

Miscellaneous e.g. data pression and encryption .techniques, optimization of

the internal structures.

Efficiency considerations are the most important at this level and the data structures are chosen to provide an efficient database. The internal view does not deal

with the physical devices directly. In tead it view physical device as a collection of physical s and allocates spce in terms of logical s.

.

The separation of the conceptual view from the internal view enables us to provide a logicaldescription of the database without the need to specify physical structures. This is often called *physical data independence*. Separating the external views from the conceptual view enables us to change the conceptual view without affecting the external views. This separation is sometimes called *logical data independence*.

Assuming the three level view of the database, a number of mappings are needed to enable the users working with one of the external views. For example, the payroll office may have an external view of the database that consists of the following information only:

Staff number, name and address.

Staff tax information e.g. number of dependents.

Staff bank information where salary is deposited.

Staff employment status, salary level, leave information etc.

The conceptual view of the database may contain academic staff, general staff, casual staff etc. A mapping will need to be created where all the staff in the different categories arebined into one category for the payroll office. The conceptual view would include information about each staff's position, the date employment started, full-time or part-time etc. This will need to be mapped to the salary level for the salary office. Also, if there is some change in the conceptual view, the external view can stay the same if the

mapping is changed.

1.9 Data Independence

.

Data independence can be defined as the capacity to change the schema at one level without changing the schema at next higher level. There are two types of data

Independence. They are

1. Logical data independence.
2. Physical data independence.

.

1. Logical data independence is the capacity to change the conceptual schema without having to change the external schema.
2. Physical data independence is the capacity to change the internal schema without

   changing the conceptual schema.

When not to use a DBMS

- Main inhibitors (costs) of using a DBMS:
- High initial investment and possible need for additional hardware.
- Overhead for providing generality, security, concurrency control, recovery, and

  integrity functions When a DBMS may be unnecessary:
- If the database and applications are simple, well defined and not expected to change.
- If there are stringent real-time requirements that may not be met because of DBMS overhead.
- If access to data by multiple users is not required.
- When no DBMS may suffice:
- If the database system is not able to handle the plexity of data because of modeling limitations
- If the database users need special operations not supported by the DBMS.

1.10 Types of Databases and Database Applications

- Traditional Applications:

  .

  Numeric and Textual Databases
- More Recent Applications:

  Multimedia Databases

  Geographic Information Systems (GIS)

  Data Warehouses

  Real-time and Active Databases

Many other applications

.

1.11 Data Model

A model is an abstraction process that hides superfluous details. Data modeling is used for representing entities of interest and their relationship in the database.

Data model and different types of Data Model

Data model is a collection of concepts that can be used to describe the structure of a database which provides the necessary means to achieve the abstraction. The structure of a database means that holds the data.

data types
relationships

constraints

Types of Data Models

1. High Level- Conceptual data model.
2. Low Level – Physical data model.
3. Relational or Representational
4. Object-oriented Data Models:
5. Object-Relational Models:

1. High Level-conceptual data model: User level data del is the high level or conceptual model. This provides concepts that are lose to the way that many

users perceive data. .

2 .Low level-Physical data model:providesconceptthat describe the details of how data is stored in the puter model. Low level dat model is only for puter specialists not for end-user.

3. Representation data model: It is between High level & Low level data model . Which providesconcepts that may be understood by end-user but that are not too far removed from the ay data is organized by within the puter.

The most mon data models are

1. Relational Model

The Relational Model uses a collection of tables both data and the relationship among those data. Each table have multiple column and each column has a unique name .

Relational database prising of two tables

Customer –Table.

Customer-Name	Security Number	Address	City	Account-Number
Preethi	111-222-3456	Yelhanka	Bangalore	A-101
Sharan	111-222-3457	Hebbal	Bangalore	A-125
Preethi	112-123-9878	Jaynagar	Bangalore	A-456
Arun	123-987-9909	MG road	Bangalore	A-987
Preethi	111-222-3456	Yelhanka	Bangalore	A-111
Rocky	222-232-0987	Sanjay Nagar	Bangalore	A-111

Account –Table

Account-Number	Balance
A-101	1000.00
A-125	1200.00

A-456	5000.00
A-987	1234.00
A-111	3000.00

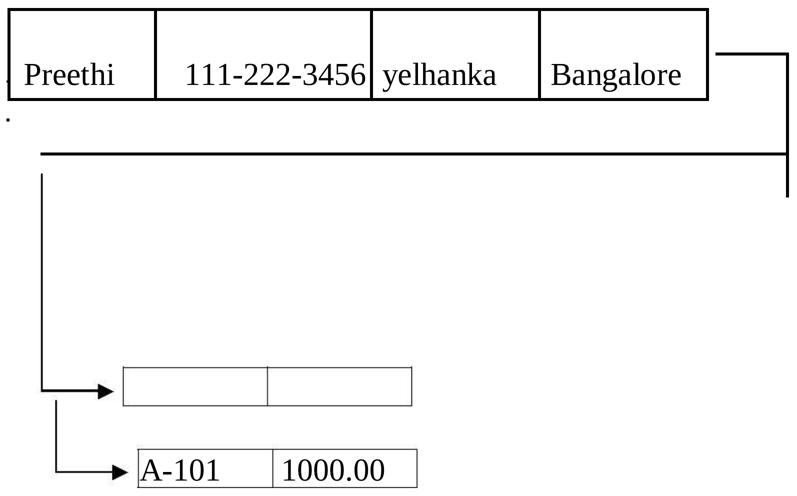
Customer Preethi and Rocky share the same account number A-111

Advantages

1. The main advantage of this model is its bility to represent data in a simplified format.
2. The process of manipulating record is simplified with the use of certain key attributes used to retrieve data.
3. Representation of different types of relationship i possible with this model.

2. Network Model

The data in the net ork model are represented by collection of records and relationships among data are represented by links, which can be viewed as pointers.



A-111 3000.00

The records in the database are organized as collection of arbitrary groups.

Advantages:

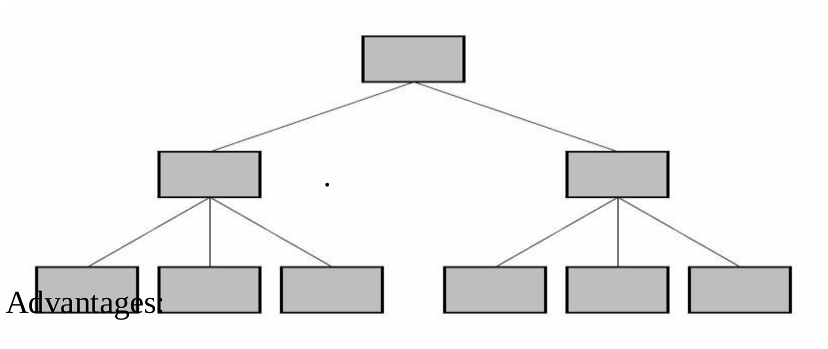
1. Representation of relationship between entities is implemented using pointers which allows the representation of arbitrary relationship

2. Unlike the hierarchical model it is easy.

3. data manipulation can be done easily with this model.

3. Hierarchical Model likerelationships: .

each parent can have many children but each child only has one parent. All attributes of a specific record are listed under an entity type.



Advantages:

1. The representation of records is done using an ordered tree, which is natural method of implementation of one–to-many relationships.

2. Proper ordering of the tree results in easier and faster retrieval of records.

3. Allows the use of virtual records. This result in a stable database especially when modification of the data base is made.

4.0 Object-oriented Data Models

- Several models have been proposed for implementing in a database system.

- One set prises models of persistent O-O Programming Languages such as C++ (e.g., in OBJECTSTORE or VERSANT), and Smalltalk (e.g., in GEMSTONE).

- Additionally, systems like O2, ORION (at MCC – then ITASCA), IRIS (at H.P.- used in Open OODB).

5.0 Object-Relational Models

- Most Recent Trend. Started with Informix .
- Universal Server.
- Relational systems incorporate concepts from object databases leading to object-relational.

- Object Database Standard: ODMG-93, ODMG-version 2.0,ODMG-version 3.0.

- Exemplified in the latest versions of Oracle-10i,DB2, and SQL Server and other DBMSs.

- Standards included in.SQL-99 and expected to be enhanced in future SQL standards.

The *description of a database.*

Includes descriptions of the database structure, data types, and the constraints on the database.

- Schema Diagram:

An ***illustrative display of (most aspects of) a*** database schema.

- Schema Construct:

A ***ponent of the schema or an object within*** the schema, e.g., STUDENT, COURSE.

- Database State:

The actual data stored in a database at a

particular moment in time. This includes the collection of all the data in the database. Also called database instance (or occurrence or snapshot).

- The term *instance is also applied to individual* database ponents, e.g. *recordinstance, table instance, entity instance*

Database Schema vs. Database State

- Database State:

Refers to the ***content of a database at a moment*** in time.

- Initial Database State:

Refers to the database state when it is initially loaded into .the system.

- Valid State:

A state that satisfies the structure and constraints of the database.

- Distinction

The ***database schema changesvery infrequently.***

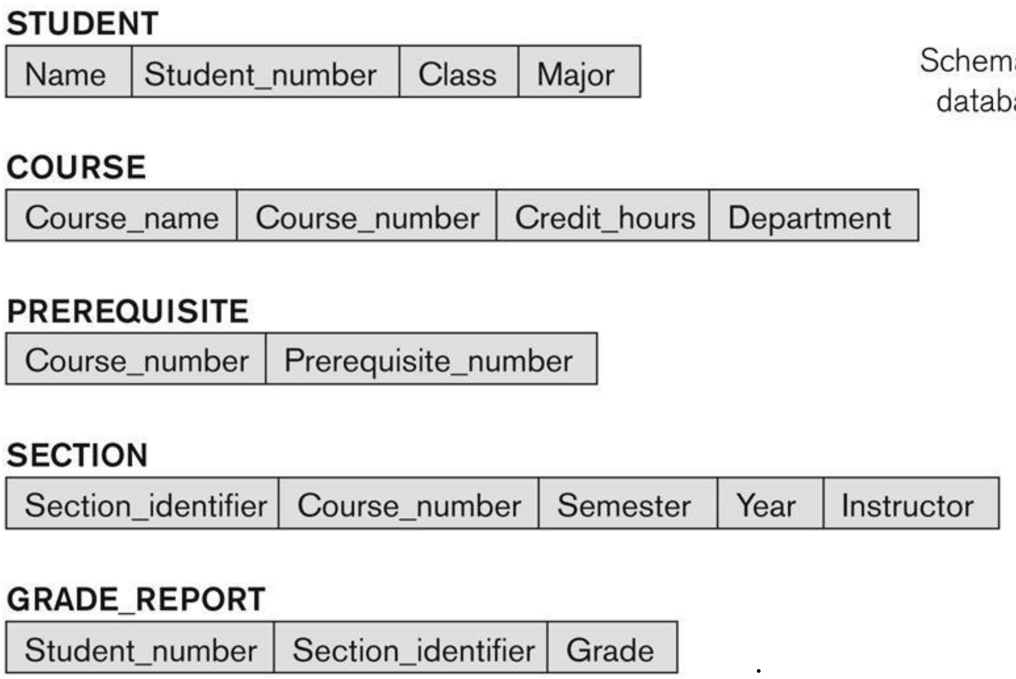
The ***database state changes every time the*** database is updated

. • Schema

isalsocalled intension

- State is also called extension

Example of a Database Schema



STUDENT

Name	Student_number	Class	Major

COURSE

Course_name	Course_number	Credit_hours	Department

PREREQUISITE

Course_number	Prerequisite_number

SECTION

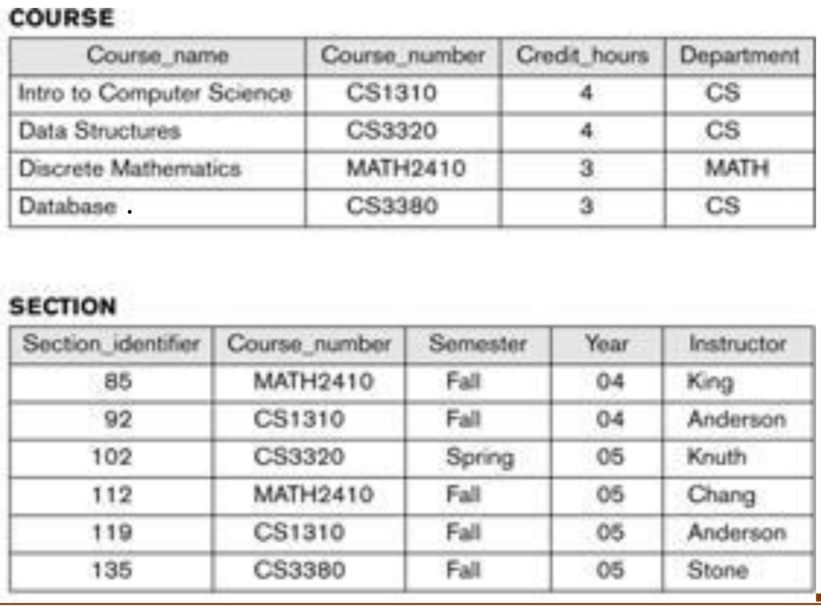
Section_identifier	Course_number	Semester	Year	Instructor

GRADE_REPORT

Student_number	Section_identifier	Grade

Figure 2.1
Schema diagram for the database in Figure 1.2.

Example of a database state

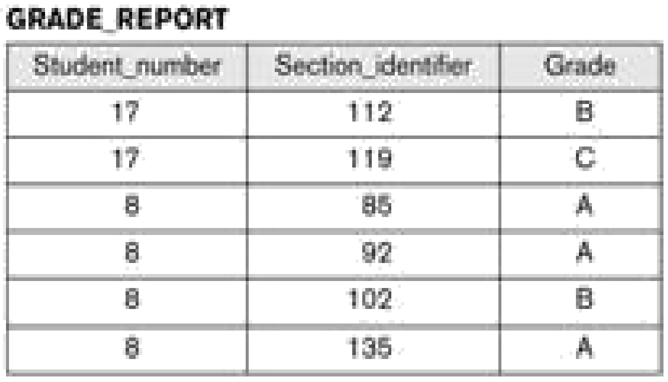


COURSE

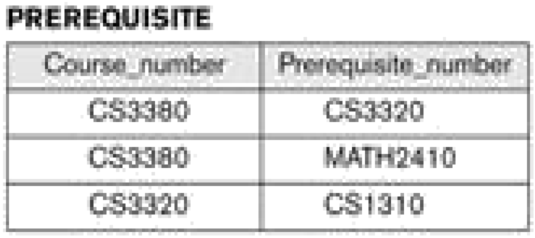
Course_name	Course_number	Credit_hours	Department
Intro to Computer Science	CS1310	4	CS
Data Structures	CS3320	4	CS
Discrete Mathematics	MATH2410	3	MATH
Database .	CS3380	3	CS

SECTION

Section_identifier	Course_number	Semester	Year	Instructor
85	MATH2410	Fall	04	King
92	CS1310	Fall	04	Anderson
102	CS3320	Spring	05	Knuth
112	MATH2410	Fall	05	Chang
119	CS1310	Fall	05	Anderson
135	CS3380	Fall	05	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

Figure 1.2
A database that stores student and course information.

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- High-Level or Non-procedural Languages: These include the relational language SQL .
- May be usedinastandalone way or may be embedded in a programming language
- Low Level or Procedural Languages: .**DBMS Languages**

These must be embedded in a programming language

Data Definition Language (DDL)

Used by the DBA and database designers to specify the conceptual schema of a database.

- In many DBMSs, the DDL is also used to define internal and external schemas (views).

- In some DBMSs, separate **storage definition language (SDL) and viewdefinition language (VDL) are used to define internal and external** schemas.

- SDL is typically realized via DBMS mands provided to the DBA and database designers

Data Manipulation Language (DML)

Used to specify database retrievals and updates DML mands (data sublanguage) can be *embedded in a general-purpose programming* language (host language), such as COBOL, C, C++, or Java.

Types of DML

. •

Alternatively, stand-aloneDMLmandscanbeapplied directly (called a *query*

- A library of functions can also be provided to access the DBMS from a

  programming language

  language).

- **High Level or Non-procedural Language:**

  For example, the SQL reltional language are “set”-oriented and specify what .datatoretrieverather than how to retrieve it.
  Also called **declarative languages.**

- **Low Level or Procedural Language:**

  Retrieve data one record-at-a-time;

  Constructs such as looping are needed to retrieve multiple records, along with positioning pointers.

DBMS Interfaces

- Stand-alone query language interfaces

  Example: Entering SQL queries at the DBMS interactive SQL interface (e.g. SQL*Plus in ORACLE)

Programmer interfaces for embedding DML in programming languages

- User-friendly interfaces
- Menu-based, forms-based, graphics-based, etc.

DBMS Programming Language Interfaces

languages

Database Programming Language Approach:

- Programmer interfaces for embedding DML in a programming languages:

- **Embedded Approach: e.g embedded SQL (for C,**C++, etc.), SQLJ (for Java)
- **Procedure Call Approach: e.g. JDBC for Java,** ODBC for other programming
- e.g. ORACLE has PL/SQL, a programming language basedponents/nSQL;

language incorporates SQL and its data types as integral

.

User-Friendly DBMS Interfaces

- Menu-based, popular for browsing on the web
- Forms-based, designed for naïve users
- Graphics-based (Point and C ick, Drag and Drop, etc.)
- Natural language: requests in written English

.

- binations of the above:For example, both menus and forms usedextensively in Web database interfaces

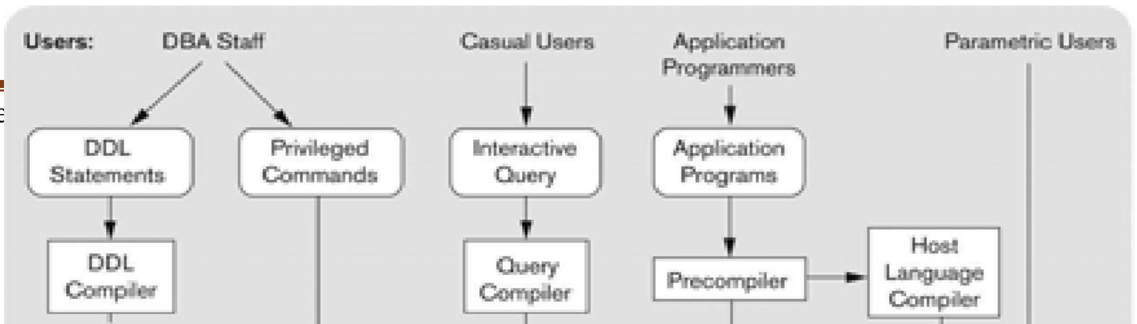
Other DBMS Interfaces

- Speech as Input and Output
- Web Browser as an interface
- Parametric interfaces, e.g., bank tellers using function keys.
- Interfaces for the DBA:
- Creating user accounts, granting authorizations
- Setting system parameters
- Changing schemas or access paths

2.0 The database system environment

The DBMS is a plex software system.

Typical DBMS ponent Modules



.

.

The figure is dividedintotwo halves. The top half of the figure refers to the various users of the database environment and their interfaces. The lower half shows the internals of the DBMS responsible for storage of data and processing of transaction.

The database and the DBMS catalog are usually stored on disk.Access to the disk is primarily controlled by operating system(OS).which inclues disk input/Output.A higher level stored data manager module of DBMS controls access to DBMS information that is stored on the disk.

If we consider the top half of the figure, It shows interfaces to DBA staff, casual users, application programmers and parametric users

The DDL piler processes schema definitions, specified in the DDL,and stores the description of the schema in the DBMS Catalog..The catalog includes information such as names and sizes of the sizes of the files, data types of data of data items. Storage details of each file, mapping information among schemas and constraints.

Casual users and persons with occasional need of information from database interact using some for of interface which is interactive query interface. The queries are parsed, analysed for correctness of the operations for

the model. the names of the data elements and so on by a query mpiler that piles . them into internal form. The internalqueryissubjectedtoqueryoptimization..The query optimizer is concerned with rearrangement and possible recording of operations,
eliminations of redundancies.

Application programmer writes programs in host languages. The prepiler extracts DML mands from an application program

2.1**Centralized and Client-Server DBMS Architectures**

- bineseverythinginto single system including- DBMS software, hardware, application programs, and user interface processing software.
- User can still connect through a remote terminal – however, all processing is done at centralized site. .**CentralizedDBMS:**

A Physical Centralized Architecture

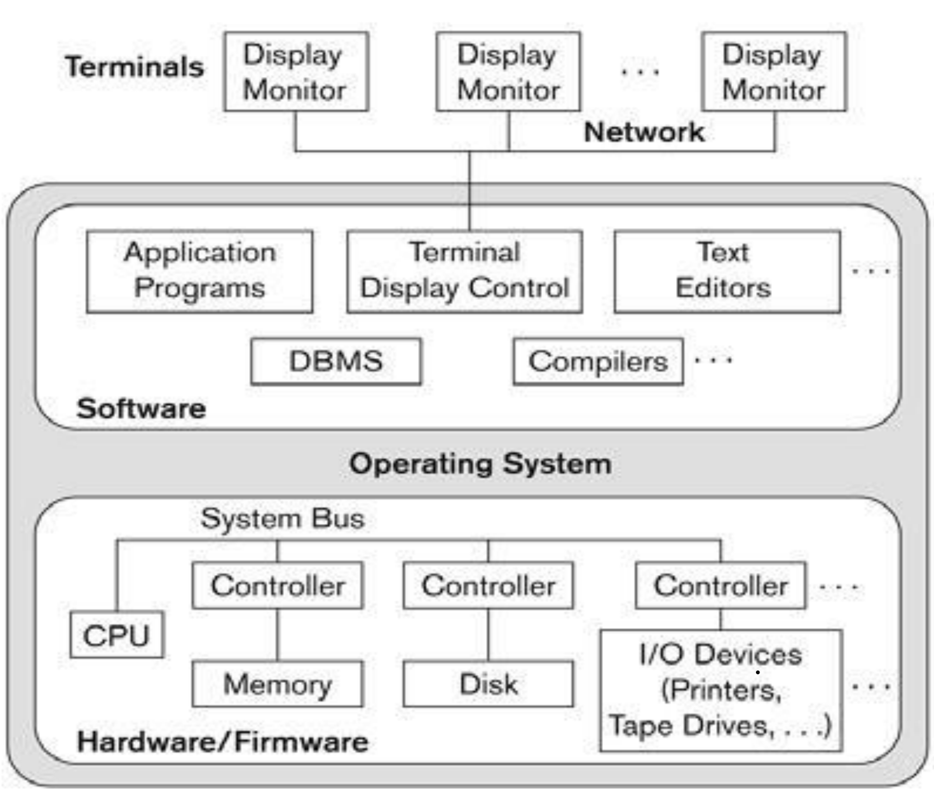


Figure 2.4
A physical centralized architecture.

main processing for all system.functions,includinguserapplication programs and user interface programs as ell all DBMS functionality. The reason was that most users Architectures for DBMS have followed trends similar to those generating puter

system architectures. Earlier architectures used mainframes puters to provide the

accessed such systemsviaputer terminals that did not have processing power and only provided display capabilities. Therefore all processing was performed remotely on the puter system, and only display information and controls were sent from the puter to the display terminals, which were connected to central puter via various types of munication networks.

As prices of hardware declined, most users replaced their terminals with PCs and workstations. At first database systems used these puters similarly to how they have used is play terminals, so that DBMS itself was still a Centralized DBMS in which all the DBMS functionality, application program execution and user interface processing were carried out on one Machine.

Basic 2-tier Client-Server Architectures

- Specialized Servers with Specialized functions
- Print server
- File server
- DBMS server
- Web server
- Email server
- Clients can access the specialized servers as needed

Logical two-tier client server architecture .

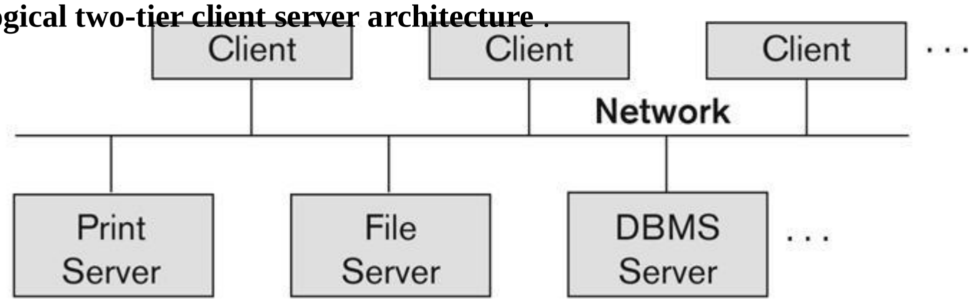


Figure 2.5
Logical two-tier client/server architecture.

Clients

- Provide appropriate interfacesthroughclientsoftware module to access and
- Clientsmaybediskless machines or PCs or Workstations with disks with only the client software installed.

- Connected to the servers via some form of a network.
- (LAN: local area network, wireless network, etc.)utilizethevariousserver .resources.

DBMS Server

- Provides database query and transaction services to the clients
- Relational DBMS servers are often called SQL servers, query servers, or transaction servers
- Applications running on clients utilize an Application Program Interface (**API) toaccess server databases via** standard interface such as:

- ODBC: Open Database Connectivity standard
- JDBC: for Java programming access
- Client and server must install appropriate client module and server module software for ODBC or JDBC

Two Tier Client-Server Architecture

- A client program may connect to several DBMSs, sometimes called the data sources.
- In general, data sources can be files or other non-DBMS software that manages

  data. Other variations of clients are possible: e.g., insomeobject DBMSs, more functionality is transferred to clients including data dictionary functions, optimization and recovery across multiple servers, etc.

Three Tier Client-Server Architecture

- mon for Web applications
- Intermediate Layer called Application Server or Web Server:
- Stores the web connectivity software and the business logic part of the application used to access the corresponding data from the database server

- Acts like a conduit for sending partially processed data between the database server and the client.

- Three-tier Architecture Can Enhance Security:

- Database server only accessible via middle tier
- Clients cannot directly access database server .

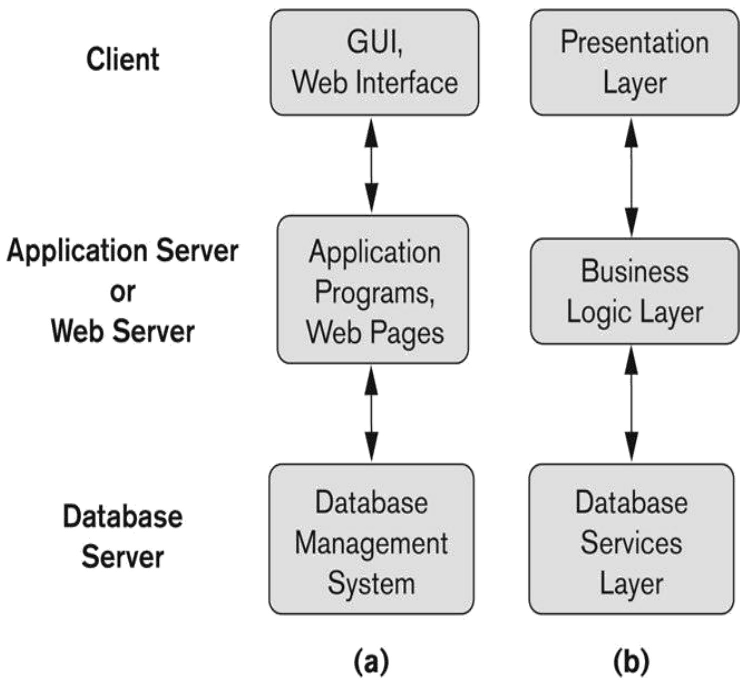


Figure 2.7
Logical three-tier client/server architecture, with a couple of commonly used nomenclatures.

- Based on the data model used
- Traditional: Relational, Network, Hierarchical.

- Emerging: Object-oriented, Object-relational.
- Other classifications .

- Single-user(typically used with personal puters) vs. multi-user (most DBMSs).

- Centralized (uses a single puter with one database) vs. distributed (uses multiple puters, multiple databases) .**ClassificationofDBMSs**

Variations of Distributed DBMSs (DDBMSs)

- Homogeneous DDBMS

- Heterogeneous DDBMS

- Federated or Multidatabase Systems

- Distributed Database Systems have now e to be known as client-server based database systems because:
- They do not support a totally distributed environment, but rather a set of database servers supporting a set of clients.

Cost considerations for DBMSs

- Cost Range: from free open-source systems to configurations costing millions of dollars

- Examples of free relational DBMSs: MySQL, PostgreSQL, others

Entity-Relationship Model

Introduction to ER Model

ER model is represents real world situations using concepts, which are monly used by people. It allows defining a representation of the real world at logical level.ER model has no facilities to describe machine-related aspects.

In ER model the logical structure of data is captured by indicating the grouping of data into entities. The ER model also supports a top-down approach by which details can be given in successive stages.

Entity: An entity is something whichisdescribedinthedatabase by storing its data, itmay be a concrete entity a conceptual entity.

Entity set: An entity set is a collection of similar entities.

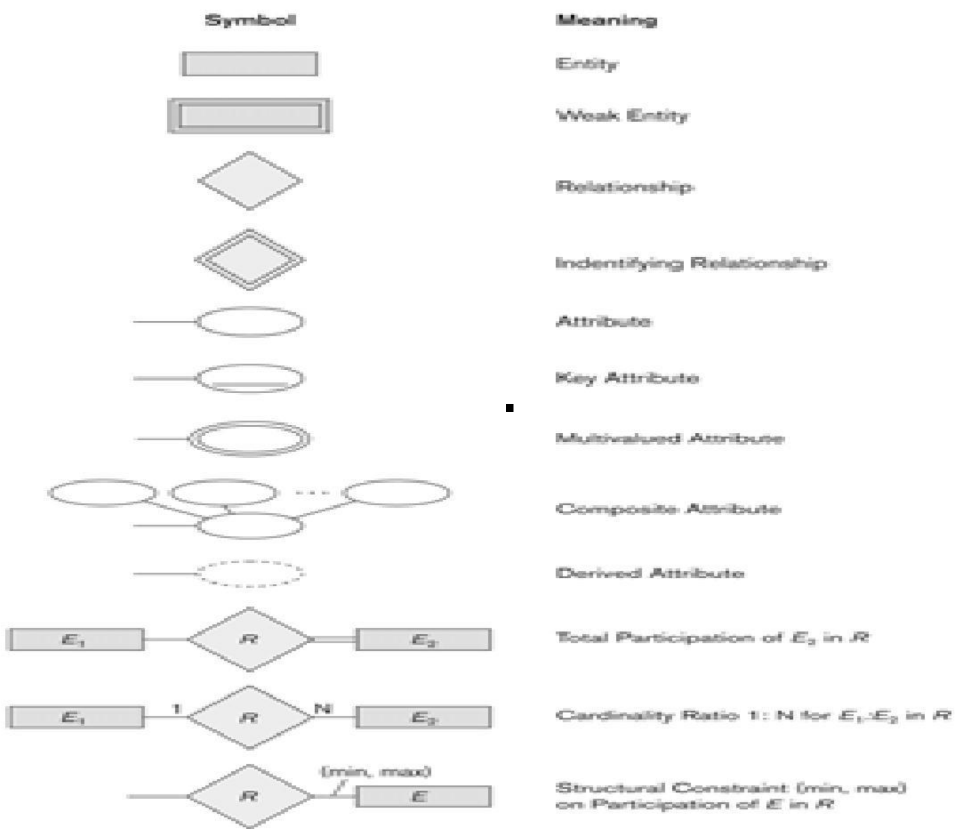
Attribute: An attribute describes property associated with entities. Attribute will have aname and a value for each entity.

Domain: A domaindefines a set of permitted values for a attribute

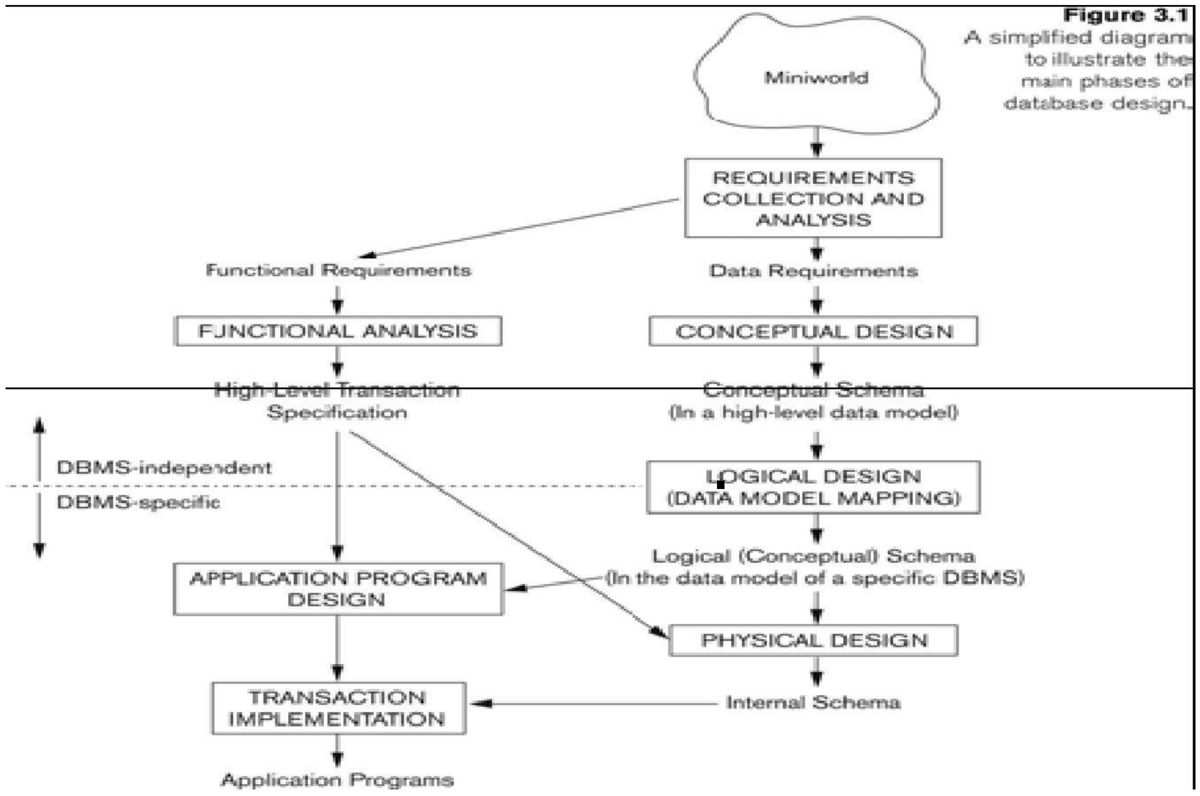
SYMBOLS IN E-R DIAGRAM

The ER model is represented using different symbols as shown in Fig .a

Figure 3.14 Summary of the notation for ER diagrams.


Symbol
Meaning
Entity
Weak Entity
Relationship
Indentifying Relationship
Attribute
Key Attribute
Multivalued Attribute
Composite Attribute
Derived Attribute
E1
R
E2
Total Participation of E2 in R
1
N
Cardinality Ratio 1: N for E1:E2 in R
(min, max)
E
Structural Constraint (min, max) on Participation of E in R

Overview of Database Design Process


Miniworld
REQUIREMENTS COLLECTION AND ANALYSIS
Functional Requirements
Data Requirements
FJNCTIONAL ANALYSIS
CONCEPTUAL DESIGN
High-Level Transaction Specification
Conceptual Schema (In a high-level data model)
DBMS-independent
DBMS-specific
LOGICAL DESIGN (DATA MODEL MAPPING)
Logical (Conceptual) Schema (In the data model of a specific DBMS)
APPLICATION PROGRAM DESIGN
PHYSICAL DESIGN
TRANSACTION IMPLEMENTATION
Internal Schema
Application Programs
Figure 3.1 A simplified diagram to illustrate the main phases of database design.

Example PANY D

We need to createa database schema design based on the following (simplified) **requirements**of the PANY Database:

The pany is organized into DEPARTMENTs.

Each department has a name, number and an employee who *manages the department.*

We keep track of the start date of the department manager. A department may have several locations.

Each department *controls a number of*

PROJECTs. Each project has a unique name, unique number and is located at a single location.

We store each EMPLOYEE's social security number, address, salary, sex, and birth date. Each employee *works for one department but may work on several projects.*

We keep track of the number of hours per week that an employee currently works on each project.

We also keep track of the *direct supervisor of each* employee.
Each employee may *have a number of* DEPENDENTs.

For each dependent, we keep track of their name, sex, birth date, and relationship to the employee.

ER Model Concepts

Entities and Attributes

For example the EMPLOYEE John Smith, the Research DEPARTMENT, ▪ theProductX PROJECT.
Entities are specific objects or things in the mini-world that are represented in the database.
Attributes are properties used todescribeanentit.

For example an EMPLOYEE entity may have the attributes Name, SSN, Address, Sex, BirthDate .

A specific entity will have a ▪ value for each of its attributes.
For example a specificemployee entity may have Name='John Smith', SSN='123456789', Address ='731, Fondren, Houston, TX', Sex='M', BirthDate='09-JAN-55‘

Each attribute has a *value set (or data type) associated with* it – e.g. integer, string, subrange, enumerated type,

Types of Attributes

There are two types of Attributes

Simple

Each entity has a single atomic value for the attribute.
For example, SSN or Sex.

posite

The attribute may be posed of several ponents. For example:

Address(Apt#, House#, Street, City, State, ZipCode, Country), or Name(FirstName, MiddleName, LastName).

position may form a hierarchy where some ponents are themselves posite.

Multi-valued

An entity may have multiple values for that attribute. For example,Color of a CAR or Previous Degrees of a STUDENT.
Denoted as {Color} or {Previous Degrees}.

In general, posite and multiallsyllabus-valuedattributesmaybene ted arbitrarily to any number of levels, although this is rare.

For example, Previous Degrees of STUDENT is posite multi-valued attribute denoted by

{Previous Degrees (College, Year, Degree, Field)}

Multiple Previous Degrees values can exi t. Each ha foursubponent attributes:
College, Year, Degree, Field

Example of a posite attribute

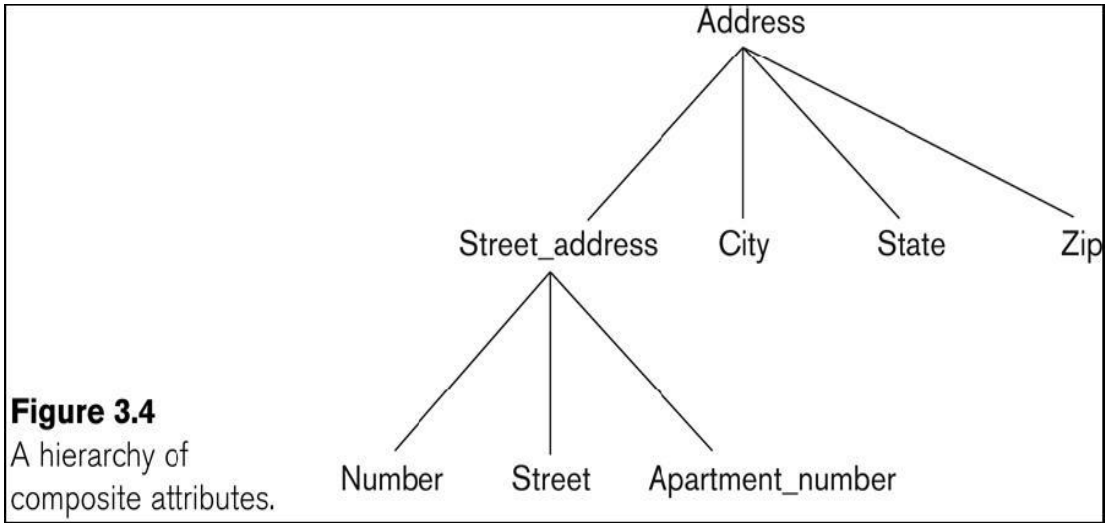


Figure 3.4
A hierarchy of composite attributes.

Entity Types and Key Attributes

Entities with the same basic attributes are grouped or typed into an entity type. For example, the entity type EMPLOYEE and PROJECT.

An attribute of an entity type for which each entity must have a unique value is called a key attribute of the entity type.

For example, SSN of EMPLOYEE.

A key attribute may be posite.

Vehicle Tag Number is a key of the CAR entity type with (Number, State).

▪ ponents

An entity type may have more than one key.

The CAR entity type may have two keys:

VehicleIdentificationNumber (popularly called VIN)

VehicleTagNumber (Number, State), icense plate number.

Each key is underlined

Displaying an Entity type

In ER diagrams, an entity type is displayed in a rectangular box
Attributes aredisplayed in ovals.

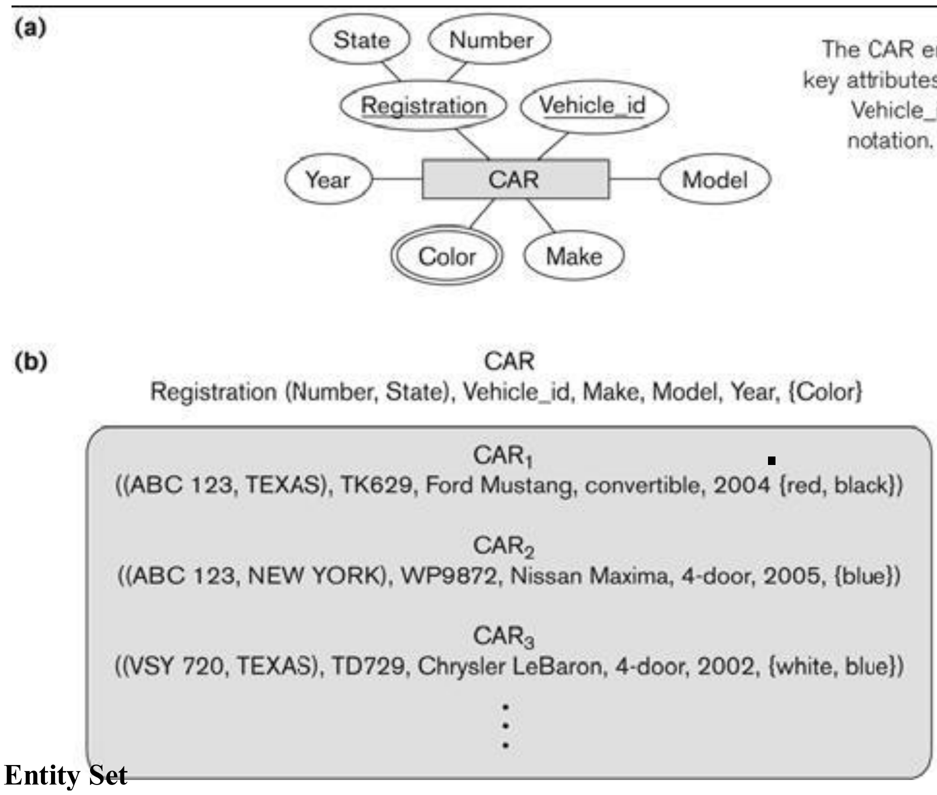
Each attribute is connected to its entity type

ponents of a posite attribute are connected to the oval representing the posite attribute.

Each key attribute is underlined.

Multivalued attributes displayed in double ovals.

Entity Type CAR with two keys and a corresponding Entity Set



(a)
State
Number
Registration
Vehicle_id
Year
CAR
Model
Color
Make
(b)
CAR
Registration (Number, State), Vehicle_id, Make, Model, Year, {Color}
CAR₁
((ABC 123, TEXAS), TK629, Ford Mustang, convertible, 2004 {red, black})
CAR₂
((ABC 123, NEW YORK), WP9872, Nissan Maxima, 4-door, 2005, {blue})
CAR₃
((VSY 720, TEXAS), TD729, Chrysler LeBaron, 4-door, 2002, {white, blue})

Figure 3.7 The CAR entity type with two key attributes, Registration and Vehicle_id. (a) ER diagram notation. (b) Entity set with three entities.

Entity Set

entity set.

Each entity type will have ▪ collection of entities stored in the database Called the
The above example sho s three CAR entity instances in the entity set for CAR

Same name (CAR)used to refer to both the entity type and the entity set.

Entity set is the current *state of the entities of that*type that are stored in the database.

Initial Design of Entity Types for the PANY Database Schema

Based on the requirements, we can identify four initial entity types in the PANY database:

DEPARTMENT

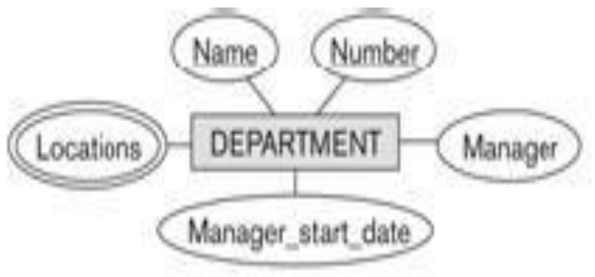
PROJECT

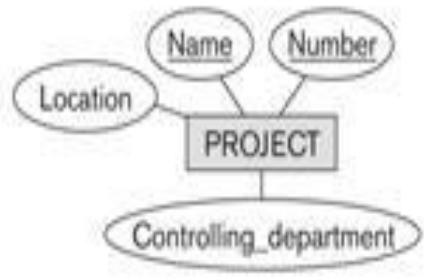
EMPLOYEE

DEPENDENT

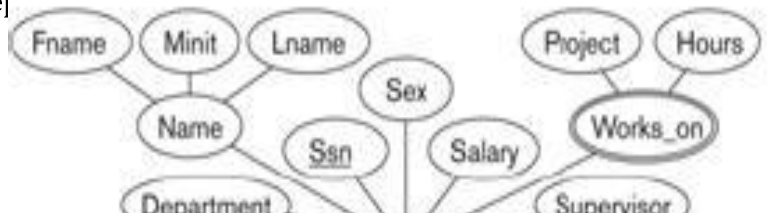
Their initial design is shown below.

The initial attributes shown are derived from the requirements description





Initial Design of Entity Typesfor the PANY Database Schema



8

Refining the initial design by introducing relationships

The initial design is typically not plete. Some aspects in the requirements will be represented as relationships.

ER model has three main concepts:

Entities (and their entity types and entity sets)

Attributes (simple, posite, multi valued)

Relationships and Relationship Types

Relationships (and their relationship types and relationship sets)

A **relationship relates two or more distinct entities with a** *specific meaning. For example,* EMPLOYEE John Smith *works on the* ProductX PROJECT, or EMPLOYEE Franklin Wong *manages the* Research DEPARTMENT.

PROJECTs participate, or theMANAGESrelationshiptype in which EMPLOYEEs and

DEPARTMENTs participate. ▪

Relationships of the same type are grouped or typed into a **relationship type.**

For example, the WORKS_ON re ationship type in which EMPLOYEEs and

The degree ofarelationship type is the number of participating entity type.

Both MANAGES and WORKS_ON are *binary relationships.*

Relationship instances of the WORKS_FOR N:1 relationship between EMPLOYEE and DEPARTMENT

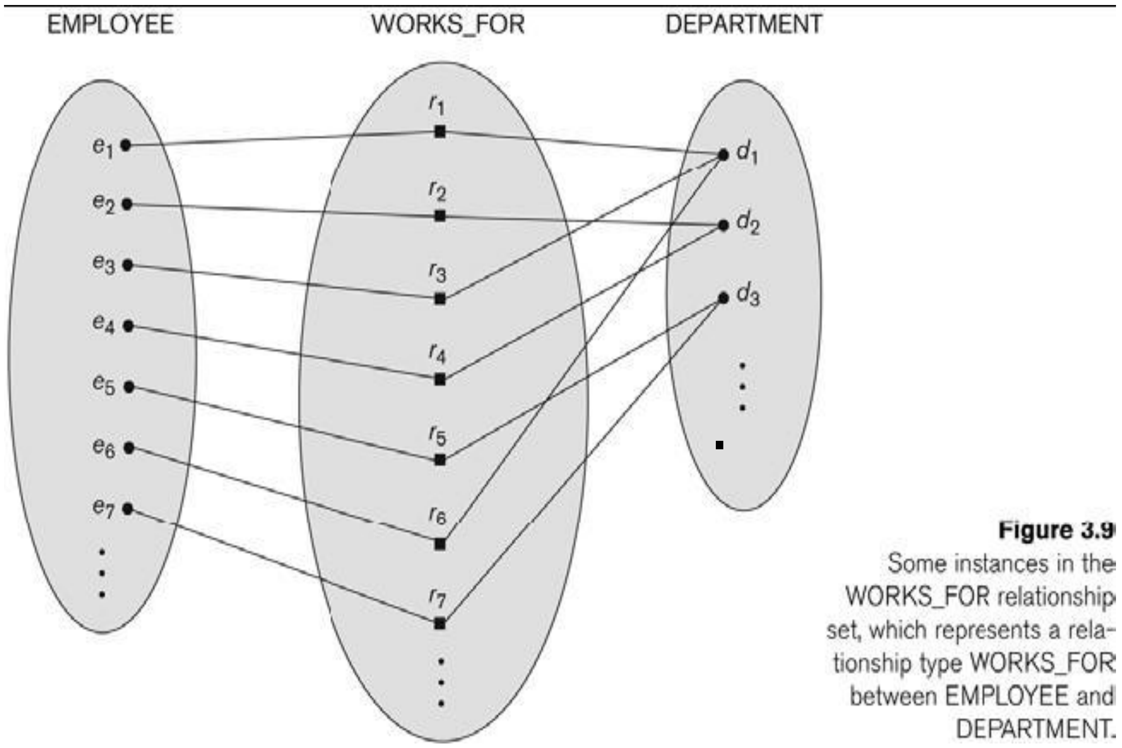
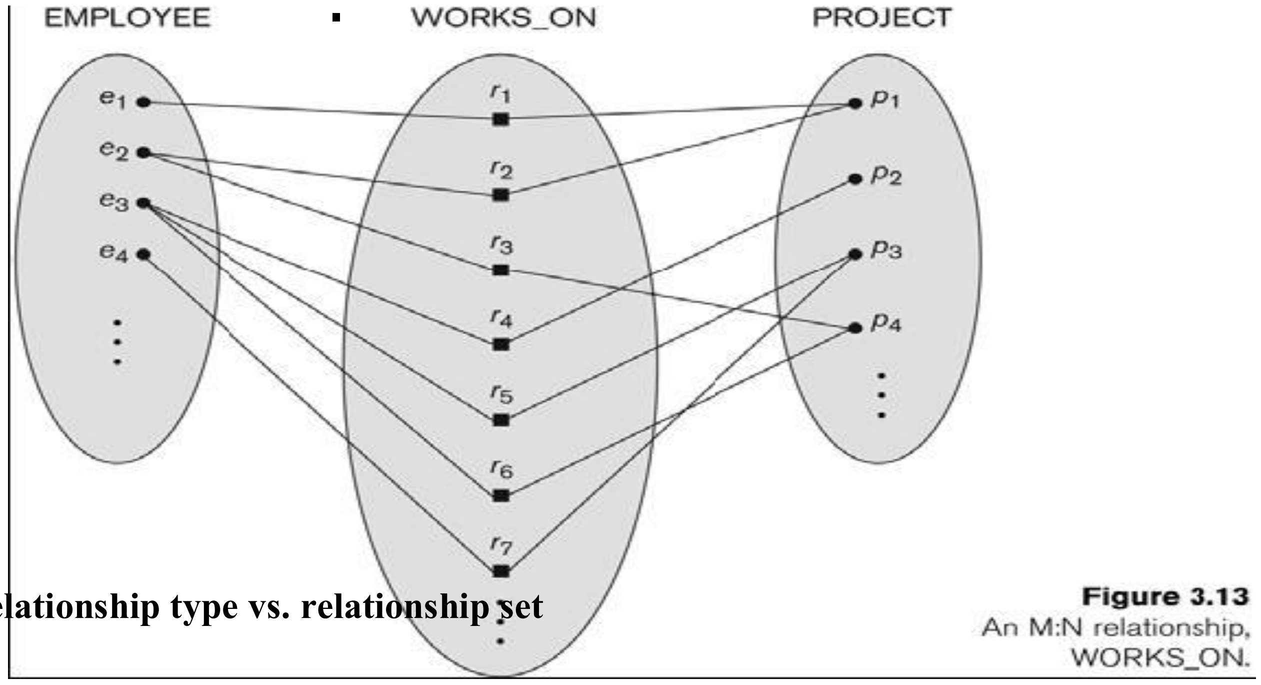


Figure 3.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Relationship instances of the M:N WORKS ON relationship between EMPLOYEE and PROJECT



Relationship type vs. relationship set

Figure 3.13
An M:N relationship, WORKS_ON.

Relationship Type:

Is the schema description of a relationship. Identifies the relationship name and the participating entity types. Also identifies certain relationship constraints.

Relationship Set:

The current set of relationship instances represented in the database. The current *state of a relationship type.* Previous figures displayed the relationship sets

Each instance in the set relates individual participating entities – one from each participating entity type.

In ER diagrams, we represent the *relationship type as follows:*

Diamond-shaped box is used to display relationship type.
Connected to the participating entity types via straight lines.

Refining the PANY databaseschemaby introducing relationships

By examining the requirements, six relationship types are identified.
All are *binaryrelationships(degree 2)*

Listed below with their participating entity types:

WORKS_FOR (between EMPLOYEE, DEPARTMENT)

MANAGES (also between EMPLOYEE,

DEPARTMENT) CONTROLS (between DEPARTMENT,

PROJECT) WORKS_ON (between EMPLOYEE,

PROJECT) SUPERVISION (between EMPLOYEE (as

subordinate), EMPLOYEE (as supervisor))

DEPENDENTS_OF (between EMPLOYEE, DEPENDENT)

ER DIAGRAM – Relationship Types are: WORKS_FOR, MANAGES, WORKS_ON, CONTROLS, SUPERVISION, DEPENDENTS_OF

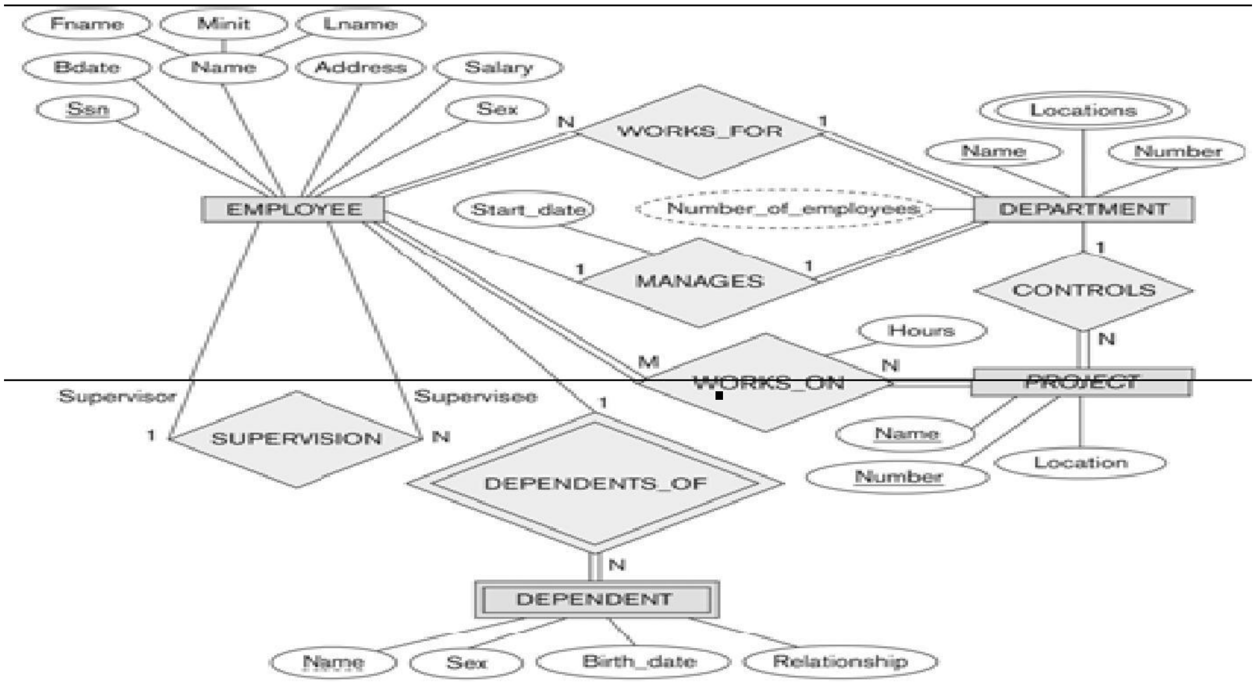


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

Relationship Types

In the refineddesign, some attributes from the initial entity types are refined into relationships:

Manager of DEPARTMENT -> MANAGES

Works_on of EMPLOYEE -> WORKS_ON

Department of EMPLOYEE -> WORKS_FOR etc

In general, more than one relationship type can exist between the same participating entity types MANAGES and WORKS_FOR are distinct relationship types between EMPLOYEE and DEPARTMENT

Different meanings and different relationship instances.

Recursive Relationship Type

An relationship type whosewith the same participating entity type in distinct roles

Example: In the SUPERVISION relationship EMPLOYEE participates twice in two distinct roles:

supervisor (or boss) role

supervisee (or subordinate) role

Each relationship instance relates two distinct EMPLOYEE entities:

One employee in *supervisor role*

One employee in *supervisee role*

Weak Entity Types

An entity that does not have key attribute. A weak entity must participate in an identifying relationship type with an owner or identifying entity type.

Entities are identified by the binationof: A partial key of

the weak entity type

The particular entity they are re ated to in the identifying entity type.

Example:

A DEPENDENT entityis identified by the dependent's first name, and the specific

EMPLOYEE with whom the dependent is related.

Name of DEPENDENT is the partial key.

DEPENDENT is a weak entity type.

EMPLOYEE is its identifying entity type via the identifying relationship type DEPENDENT_OF

Constraints on Relationships

Constraints on Relationship Types

(Also known as ratio constraints)

Cardinality Ratio (specifies *maximum participation)*

One-to-one (1:1)

One-to-many (1:N) or Many-to-one (N:1) Many-to-many (M:N)

Existence Dependency Constraint (specifies *minimum* participation) (also called participation constraint)

zero (optional participation, not existence-dependent)

one or more (mandatory participation, existence-dependent)

Many-to-one (N:1) Relationship

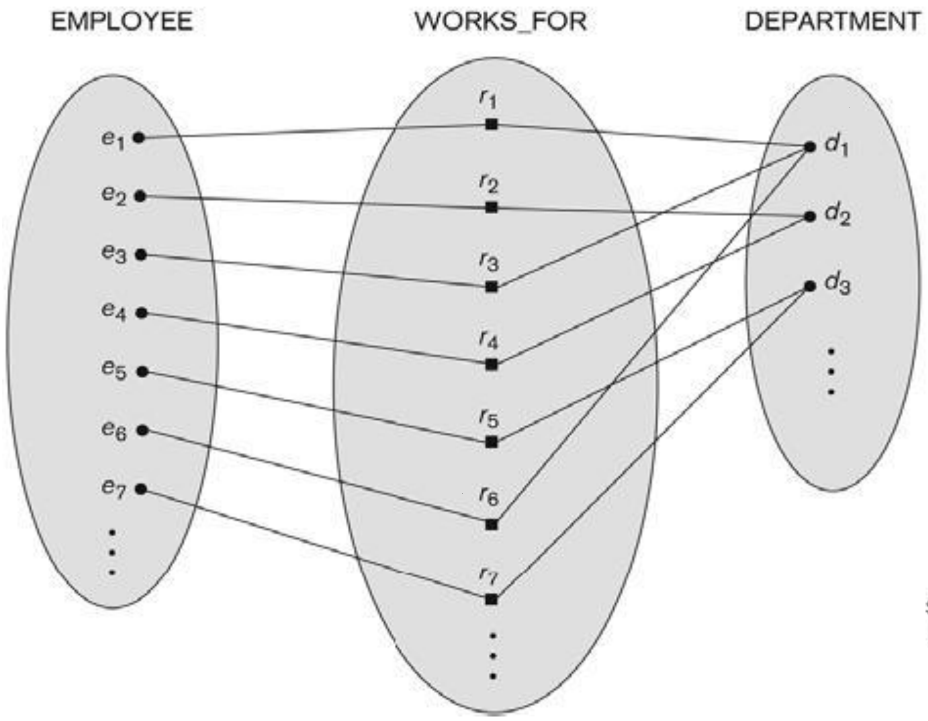


Figure 3.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

Many-to-many (M:N) Relationship

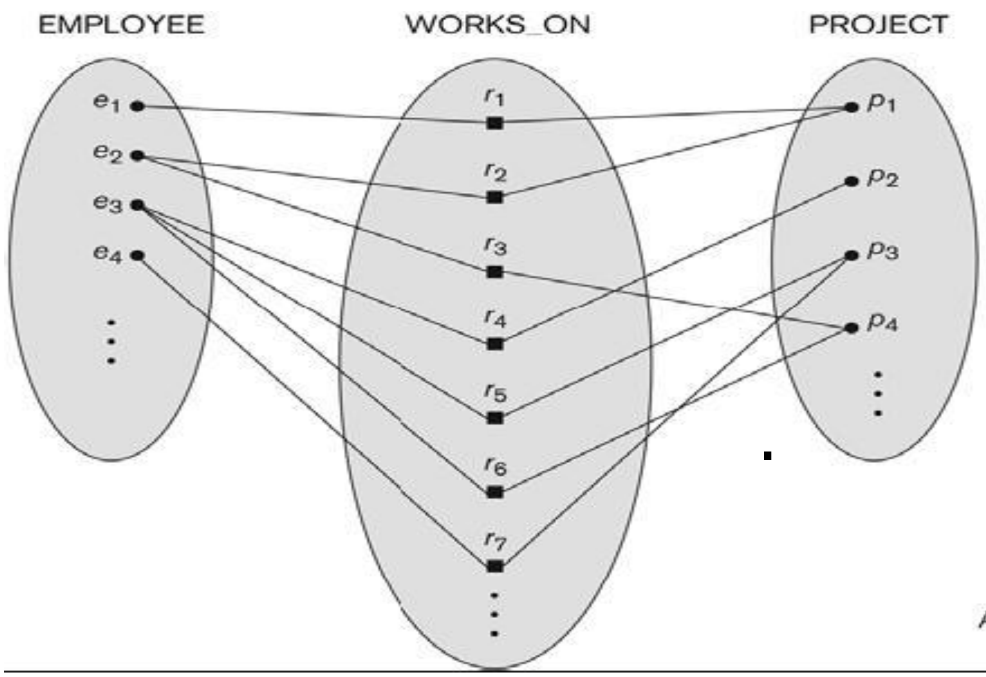


Figure 3.13
An M:N relationship, WORKS_ON.

Displaying a recursive relationship

In a recursive relationship type.

Both participations are same entity type in different roles.

For example, SUPERVISION relationships between EMPLOYEE (in role of supervisor orboss) and (another) EMPLOYEE (in role of subordinate or worker). In following figure, first role participation labeled with 1 and second role participation labeled with 2.

In ER diagram, need to display role names to distinguish participations.

A Recursive Relationship Supervision

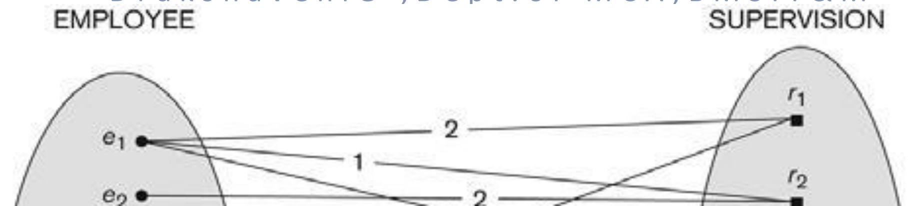


Figure 3.11
A recursive relationship SUPERVISION between EMPLOYEE in the *supervisor* role (1) and EMPLOYEE in the *subordinate*

▪

Recursive Relationship Type is: SUPERVISION
(participation role names are shown)

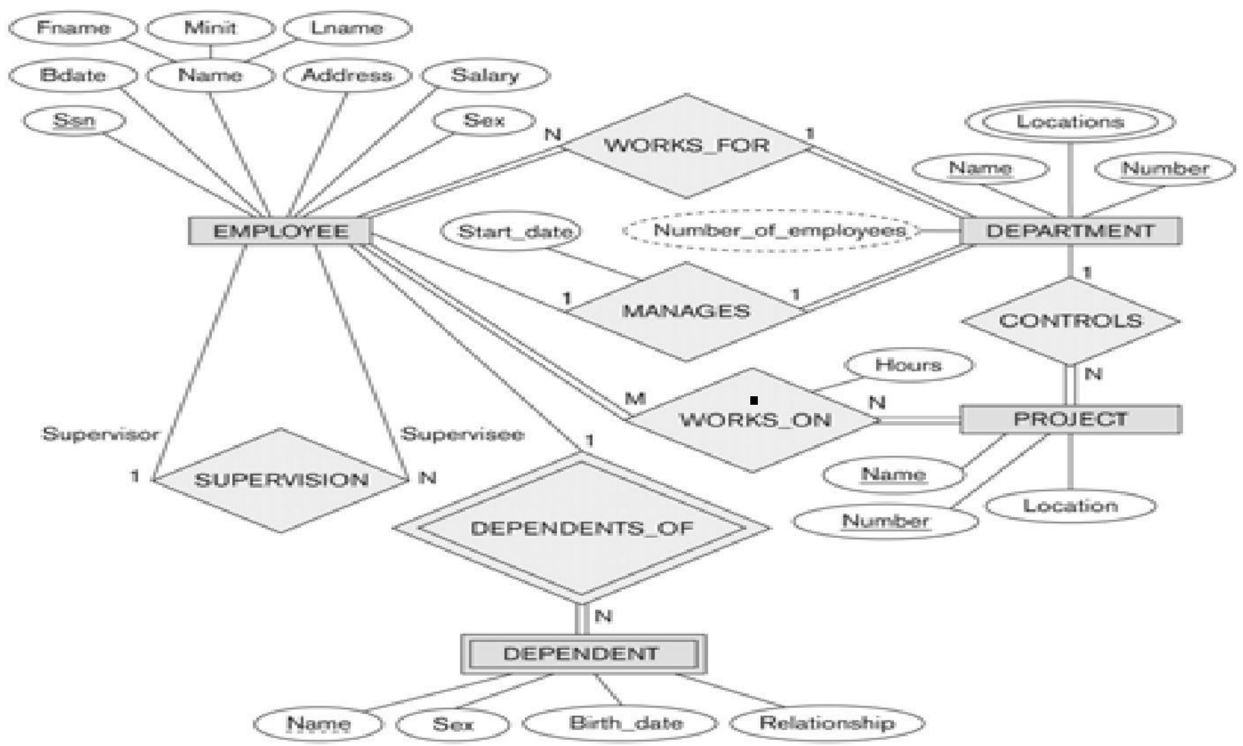


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter.

BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT

(Affiliated to the Visvesvaraya Technological University, Belagavi)

Department of Master of Computer Applications

Subject: Database Management System

Prepared by: Drakshaveni G
Assistant Professor
Dept.of MCA
BMSIT&M

Module -2

The Relational Data Model and Relational Database

Relational Model Concepts

The relational Model of Data is based on the concept of a Relation. A Relation is a mathematical concept based on the ideas of sets. The strength of the relational approach to data management comes from the formal foundation provided by the theory of relations. The model was first proposed by Dr. E.F. Codd of IBM in 1970 in the following paper: "A Relational Model for Large Shared Data Banks," Communications of the ACM, June 1970.

Informal Definitions

RELATION:

A Relation is table of values. A relation may be thought of as a set of rows. A relation may alternately be though of as a set of columns. Each row represents a fact that corresponds to a real-world entity or relationship. Each row has a value of an item or set of items that uniquely identifies that row in the table. Sometimes row-ids or sequential numbers are assigned to identify the rows in the table. Each column typically is called by its column name or column header or attribute name.

Formal definitions

A **Relation** may be defined in multiple ways. The **Schema** of a Relation: *R* (A1, A2, ...An) Relation schema *R* is defined over **attributes** A1, A2, ...An.

For Example -

CUSTOMER (Cust-id, Cust-name, Address, Phone#)

Here, CUSTOMER is a relation defined over the four attributes Cust-id, Cust-name, Address, Phone#, each of which has a **domain** or a set of valid values. For example, the domain of Cust-id is 6 digit numbers.

A tuple is an ordered set of values.Each value is derived from an appropriate domain. Each row in the CUSTOMER table may be referred to as a tuple in the table and would consist of four values.

<632895, "John Smith", "101 Main St. Atlanta, GA 30332", "(404) 894-2000">

is a tuple belonging to the CUSTOMER relation.

A relation may be regarded as a set of tuples (rows). Columns in a table are also called attributes of the relation.

A domain has a logical definition: e.g.,

"USA_phone_numbers" are the set of 10 digit phone numbers valid in the U.S.

A domain may have a data-type or a format defined for it. The USA_phone_numbers may have a format: (ddd)-ddd-dddd where each d is a decimal digit. E.g., Dates have various formats such as monthname, date, year or yyyy-mm-dd, or dd mm,yyyy etc.

An attribute designates the role played by the domain. E.g., the domain Date may be used to define attributes "Invoice-date" and "Payment-date".

The relation is formed over the cartesian product of the sets; each set has values from a domain; that domain is used in a specific role which is conveyed by the attribute name.

For example, attribute Cust-name is defined over the domain of strings of 25 characters. The role these strings play in the CUSTOMER relation is that of the name of customers.

Formally,

Given R(A1, A2, ..., An)

r(R) ⊂ dom (A1) X dom (A2) X ....X dom(An)

R: schema of the relation

r of R: a specific "value" or population of R.

R is also called the intension of a relation r is also called the extension of a relation

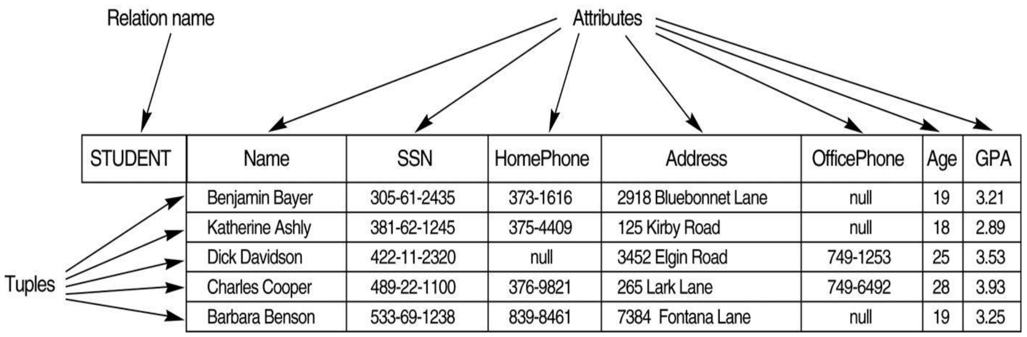
Let S1 = {0,1}

Let S2 = {a,b,c}

Let R ⊂ S1 X S2

Then for example: r(R) = {<0,a> , <0,b> , <1,c> } is one possible "state" or "population" or "extensi on" r of the relation R, defined over domains S1 and S2. It has three tuples.

Example



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25

Characteristics of Rela tions

Ordering of tuples in a relation r(R): The tuples are not considered t o be ordered, even though they appear to be in the tabular form.

Ordering of attributes in a relation schema R (and of values within ea ch tuple): We will consider the attribut es in R(A1, A2, ..., An) and the values in t=<v 1, v2, ..., vn> to be ordered .

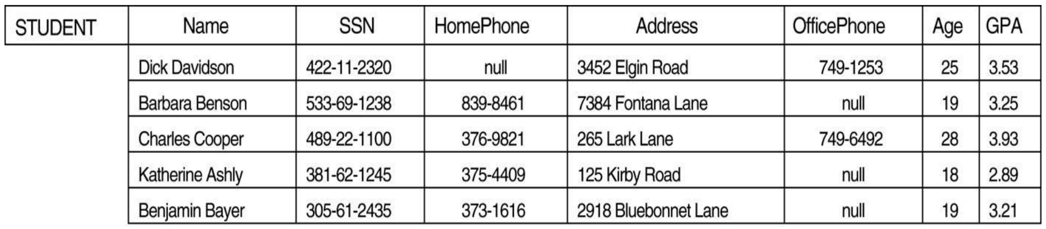
(However, a more ge neral alternative definition of relation does no t require this ordering).

Values in a tuple: All va lues are considered atomic (indivisible). A special null value is used to represent values that are unknown or inapplicable to certain tu ples.

Notation:

We refer to component v alues of a tuple t by t[Ai] = vi (the value of attribute Ai for tuple t).

Similarly, t[Au, Av, ..., Aw] refers to the subtuple of t containing t he values of attributes Au, Av, ..., Aw, respectively.



STUDENT	Name	SSN	HomePhone	Address	OfficePhone	Age	GPA
	Dick Davidson	422-11-2320	null	3452 Elgin Road	749-1253	25	3.53
	Barbara Benson	533-69-1238	839-8461	7384 Fontana Lane	null	19	3.25
	Charles Cooper	489-22-1100	376-9821	265 Lark Lane	749-6492	28	3.93
	Katherine Ashly	381-62-1245	375-4409	125 Kirby Road	null	18	2.89
	Benjamin Bayer	305-61-2435	373-1616	2918 Bluebonnet Lane	null	19	3.21

Relational Integrity Constraints

Constraints are conditio ns that must hold on all valid relation instances. There are three main types of constraints:

1. Key constraints
2. Entity integrity constra ints
3. Referential integrity co nstraints

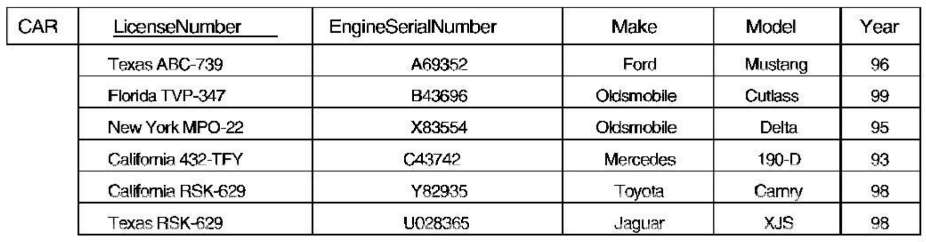
Superkey of R: A set of attributes SK of R such that no two tuples in any valid relation instance r(R) w ill have the same value for SK. That is, fo any distinct tuples t1 and t2 in r(R), t 1[SK] ≠ t2[SK].

Key of R: A "minimal" superkey; that is, a superkey K such that re moval of any attribute from K results i n a set of attributes that is not a superkey.

Example: The CAR relat ion schema:
CAR(State, Reg#, Serial No, Make, Model, Year)
has two keys Key1 = {S tate, Reg#}, Key2 = {SerialNo}, which are al so superkeys.
{SerialNo, Make} is a superkey but not a key.

If a relation has several candidate keys, one is chosen arbitrarily to b e the primary key. The primary key attr ibutes are underlined.

Figure 7.4 The CAR relation with two candidate keys: LicenseNumber and EngineSerialNumber.



CAR	LicenseNumber	EngineSerialNumber	Make	Model	Year
	Texas ABC-739	A69352	Ford	Mustang	96
	Florida TVP-347	B43696	Oldsmobile	Cutlass	99
	New York MPO-22	X83554	Oldsmobile	Delta	95
	California 432-TFY	C43742	Mercedes	190-D	93
	California RSK-629	Y82935	Toyota	Camry	98
	Texas RSK-629	U028365	Jaguar	XJS	98

Entity Integrity

Relational Database Schema: A set S of relation schemas that belong to the same database. S is the name o f the database.

S = {R1, R2, ..., Rn}

Entity Integrity: The prim ary key attributes PK of each relation schema R in S cannot have null values in any tuple of r(R). This is because primary key valu es are used to identify the individual tu ples.

t[PK] ≠ null for any tuple t in r(R)

Note: Other attributes of R may be similarly constrained to disallow nul l values, even though they are not members of the primary key.

Referential Integrity

The initial design is typic ally not complete. Some aspects in the requirem ents will be represented as relationshi ps.

ER model has three main concepts:
Entities (and their entity types and entity sets)

Attributes (simple, composite, multi valued)

Relationships (and their relationship types and relationship sets)

Referential Integrity Constraint

Statement of the constraint

The value in the foreign key column (or columns) FK of the the **referencing relation** R_1 can be either:
(1) a value of an existing primary key value of the corresponding primary key PK in the **referenced relation** R_2,, or..
(2) a null.
In case (2), the FK in R_1 should not be a part of its own primary key.

Other Types of Constraints

Semantic Integrity Constraints:

It is based on application semantics and cannot be expressed by the model per se E.g., "the max. no. of hours per employee for all projects he or she works on is 56 hrs per week"

A constraint specification language may have to be used to express these

SQL-99 allows triggers and ASSERTIONS to allow for some of these.

Figure 7.5 Schema diagram for the COMPANY relational database schema; the primary keys are underlined.

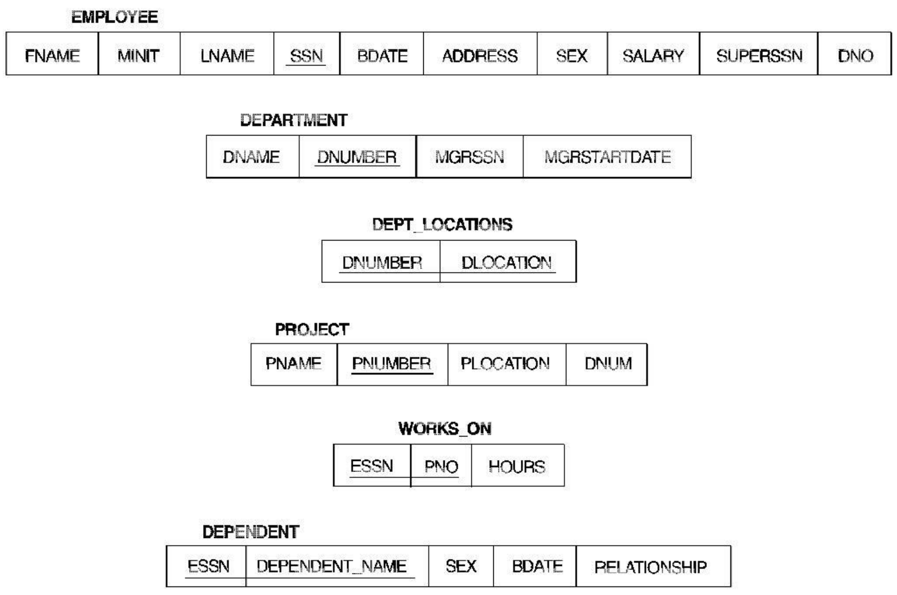


Figure 7.7 Referential integrity constraints displayed on the COMPANY relational database schema diagram.

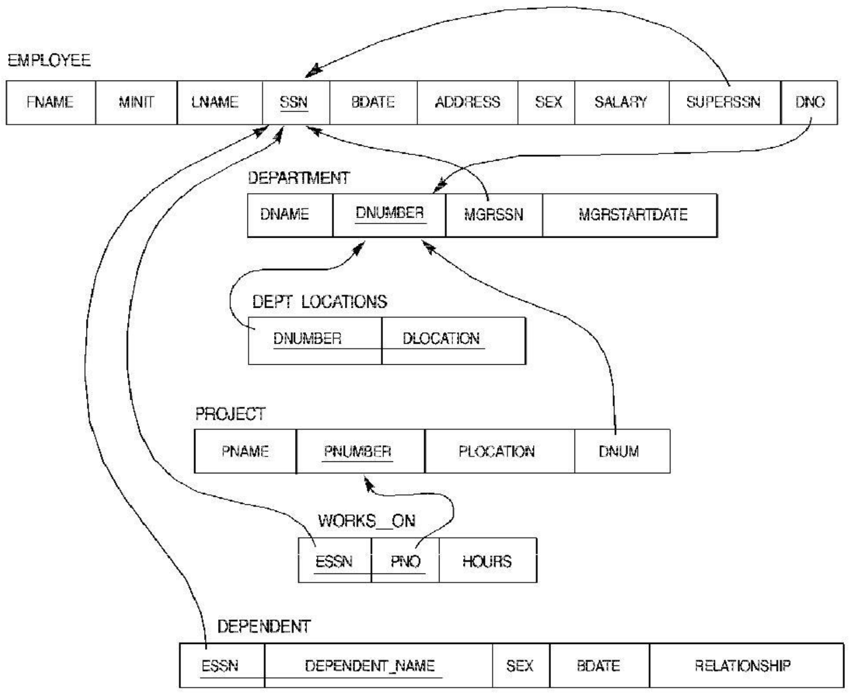
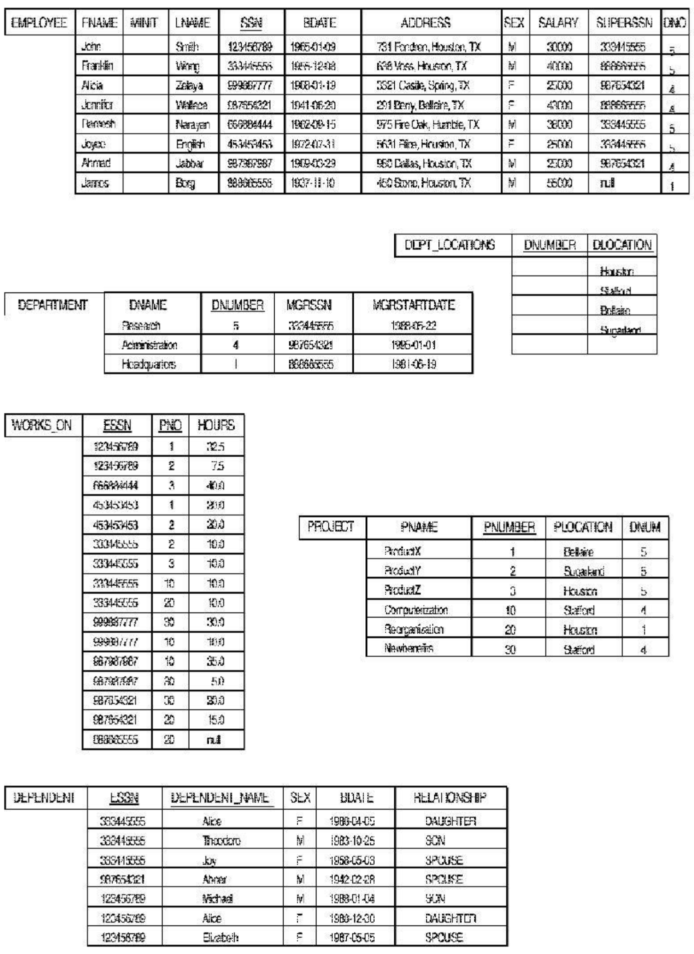


Figure 7.6 One possible relational database state corresponding to the COMPANY schema.



EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John		Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin		Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia		Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer		Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh		Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce		English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad		Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James		Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS	DNUMBER	DLOCATION
		Houston
		Stafford
		Bellaire
		Sugarland

DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE
	Research	5	333445555	1988-05-22
	Administration	4	987654321	1995-01-01
	Headquarters	1	888665555	1981-06-19

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	SEX	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

Update Operations on Relations

1. INSERT a tuple
2. DELETE a tuple
3. MODIFY a tuple

Update Operations on Relations

Integrity constraints should not be violated by the update operations. Several update operations may have to be grouped together. Updates may propagate to cause other updates automatically. This may be necessary to maintain integrity constraints. In case of integrity violation, several actions can be taken:

1. Cancel the operation that causes the violation (REJECT option)
2. Perform the operation but inform the user of the violation
3. Trigger additional updates so the violation is corrected (CASCADE option, SET NULL option)
4. Execute a user-specified error-correction routine

The Relational Algebra and Relational Calculus

Introduction

Relational Algebra is a procedural language used for manipulating relations. The relational model gives the structure for relations so that data can be stored in that format but relational algebra enables us to retrieve information from relations. Some advanced SQL queries requires explicit relational algebra operations, most commonly outer join.

Relations are seen as sets of tuples, which means that no duplicates are allowed. SQL behaves differently in some cases. Remember the SQL keyword distinct. SQL is declarative, which means that you tell the DBMS what you want.

Set operations

Relations in relational algebra are seen as sets of tuples, so we can use basic set operations.

Review of concepts and operations from set theory

Set
Element
No duplicate elements
No order among the elements
Subset
Proper subset (with fewer elements) Superset
Union
Intersection
Set Difference
Cartesian product

Relational Algebra

Relational Algebra consists of several groups of operations

Unary Relational Operations

SELECT (symbol: s (sigma))

PROJECT (symbol: ∏ (pi))

RENAME (symbol: ρ (rho))

Relational Algebra Operations From Set Theory

UNION (U), INTERSECTION (∩), DIFFERENCE (or MINUS, –)

CARTESIAN PRODUCT (x)

Binary Relational Operations

JOIN (several variations of JOIN exist)

DIVISION

Additional Relational Operations

OUTER JOINS, OUTER UNION
AGGREGATE FUNCTI ONS

Unary Relational Oper ations

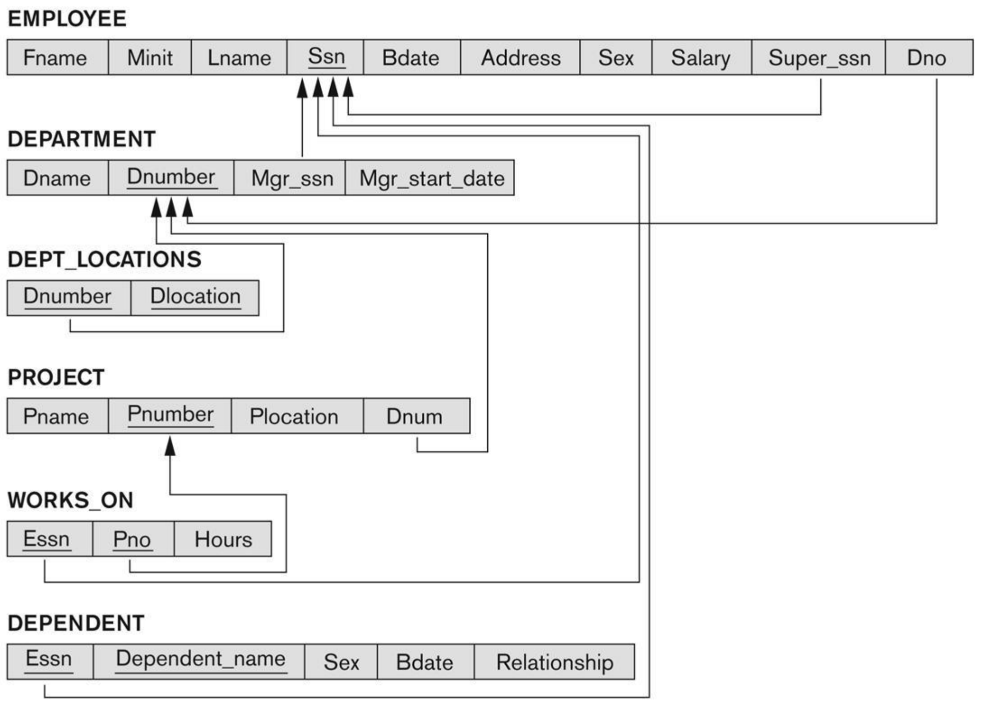
SELECT (symbol: s (sigma))
PROJECT (symbol: ∏ (pi))
RENAME (symbol: ρ (rho))

SELECT

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 ac ts as a filter and keeps only those tup les that satisfy the qualifying condition. Tuples satisfying the condition are selected wh ereas the other tuples are discarded (filtered out)

Database State for COM PANY

Figure 5.7
Referential integrity constraints displayed on the COMPANY relational database schema.



- **Examples:**
  - Select the EMPLOYEE tuples whose department number is 4:

$\sigma_{DNO = 4}$ (EMPLOYEE)

  - Select the employee tuples whose salary is greater than $30,000:

$\sigma_{SALARY > 30,000}$ (EMPLOYEE)

– In general, the *select* operation is denoted by $\sigma_{<\text{selection condition}>}(R)$

where the symbol σ (sigma) is used to denote the *select* operator

the selection condition is a Boolean (conditional) expression specified on the attributes of relation R

tuples that make the condition **true** are selected

(appear in the result of the operation)

tuples that make the condition **false** are filtered out

(discarded from the result of the operation)

The Boolean expression specified in <selection condition> is made up of a number of clauses of the form:

<attribute name><comparison op><constant value>

or

<attribute name><comparison op><attribute name>

Where <attribute name> is the name of an attribute of R, <comparison op> id normally one of the operations {=,>,>=,<,<=,!=}

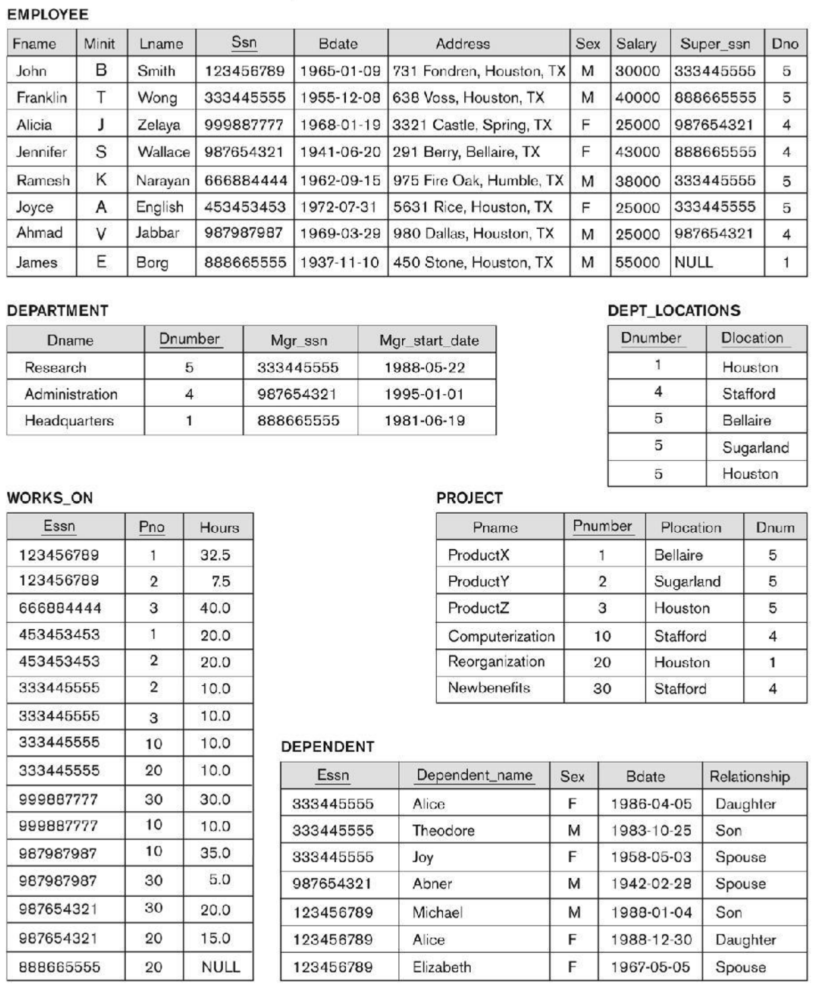
Clauses can be arbitrarily connected by the Boolean operators **and**, **or** and **not**

- **For example,** To select the tuples for all employees who either work indepartment 4 and make over $25000 per year, or work in department 5 and make over $30000, the select operation should be:

σ **(DNO=4 AND Salary>25000) OR (DNO=5 AND Salary>30000)** **(EMPLOYEE)**

The following query results refer to this database

Figure 5.6
One possible database state for the COMPANY relational database schema.



EMPLOYEE

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
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

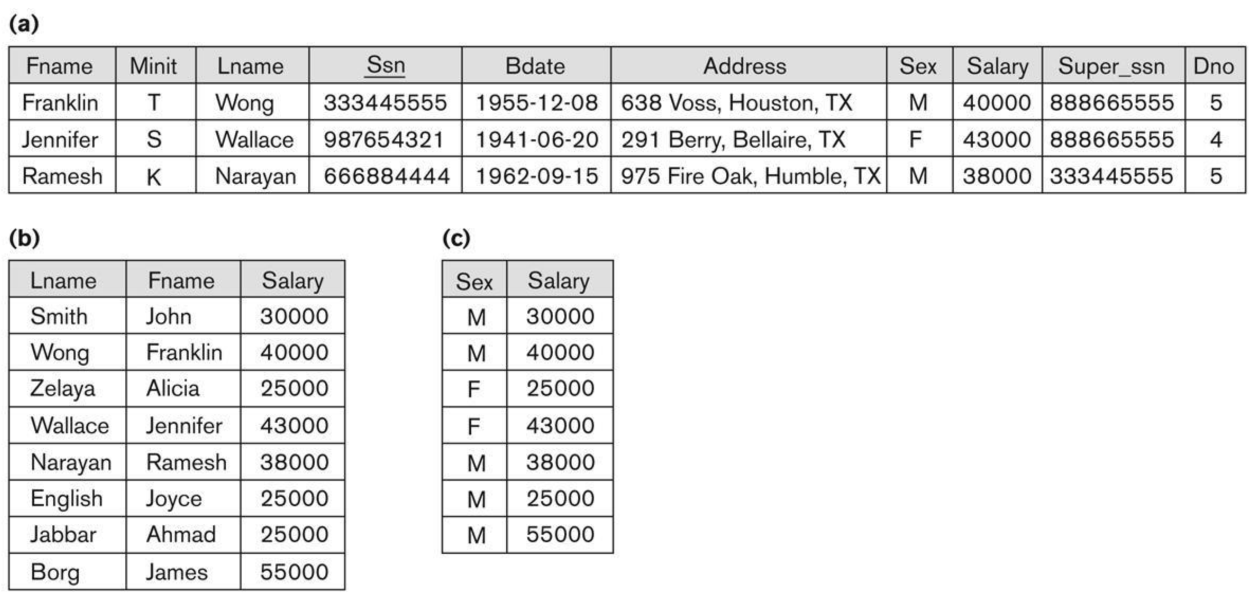
Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	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

Examples of applying S ELECT and PROJECT operations

Figure 6.1
Results of SELECT and PROJECT operations. (a) $\sigma_{(\text{Dno}=4 \textbf{ AND } \text{Salary}>25000) \text{ OR } (\text{Dno}=5 \textbf{ AND } \text{Salary}>30000)}(\text{EMPLOYEE})$. (b) $\pi_{\text{Lname, Fname, Salary}}(\text{EMPLOYEE})$. (c) $\pi_{\text{Sex, Salary}}(\text{EMPLOYEE})$.



(a)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
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

(b)

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

(c)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

SELECT Operation P roperties

– **SELECT s is commutative:**

$$\sigma_{<\text{condition1}>}(\sigma_{<\text{condition2}>}(R)) = \sigma_{<\text{condition2}>}(\sigma_{<\text{condition1}>}(R))$$

– **A cascade of SELECT operations may be repl aced by asingle sele ction with a conjunction of all the conditions:**

$$\sigma_{<\text{cond1}>}(\sigma_{<\text{cond2}>}(\sigma_{<\text{cond3}>}(R))) = \sigma_{<\text{cond1}> \text{ AND } <\text{cond2}> \text{ AND } <\text{cond3}>}(R)$$

PROJECT

PROJECT Operation is denoted by p (pi)

If we are interested in on ly certain attributes of relation, we use PROJEC T

This operation keeps certain *columns* (attributes) from a relation and disc ards the other columns.

PROJECT creates a vertical partitioning

The list of specified columns (attributes) is kept in each tu ple.

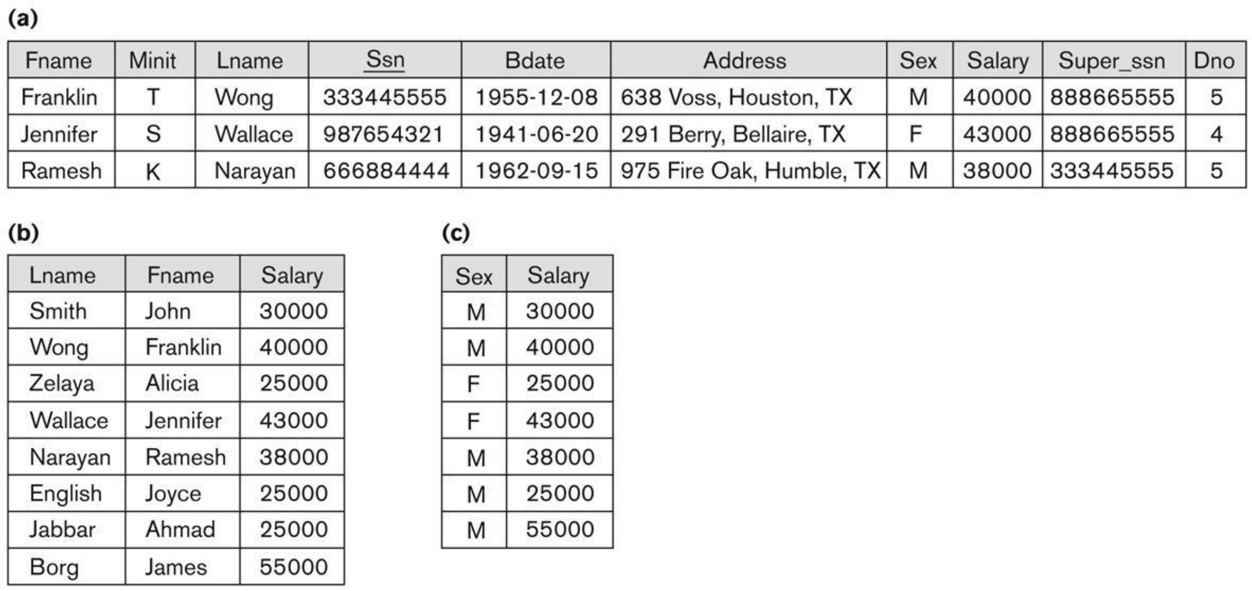
The other attributes in each tuple are discarded.

Example: To list each emp loyee's first and last name and salary, the follo wing is used:

$\prod_{\text{LNAME, FNAME,SALARY}}$(EMPL OYEE)

Examples of applying S ELECT and PROJECT operations

Figure 6.1
Results of SELECT and PROJECT operations. (a) $\sigma_{(\text{Dno}=4\ \textbf{AND}\ \text{Salary}>25000)\ \text{OR}\ (\text{Dno}=5\ \textbf{AND}\ \text{Salary}>30000)}$(EMPLOYEE). (b) $\pi_{\text{Lname, Fname, Salary}}$(EMPLOYEE). (c) $\pi_{\text{Sex, Salary}}$(EMPLOYEE).



(a)

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
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

(b)

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

(c)

Sex	Salary
M	30000
M	40000
F	25000
F	43000
M	38000
M	25000
M	55000

Single expression versus sequence of relational operations

We may want to apply se veral relational algebra operations one after the other.
Either we can write the operations as a single relational algebra expressio n by nesting the operations,
or
We can apply one operat ion at a time and create intermediate result relat ons.
In the latter case, we mus t give names to the relations that hold the inter
mediate results.

To retrieve the first name , last name, and salary of all employees who wo rk in department number 5, we must apply a select and a project operation We can write a *single relational algebra expression* as follows:

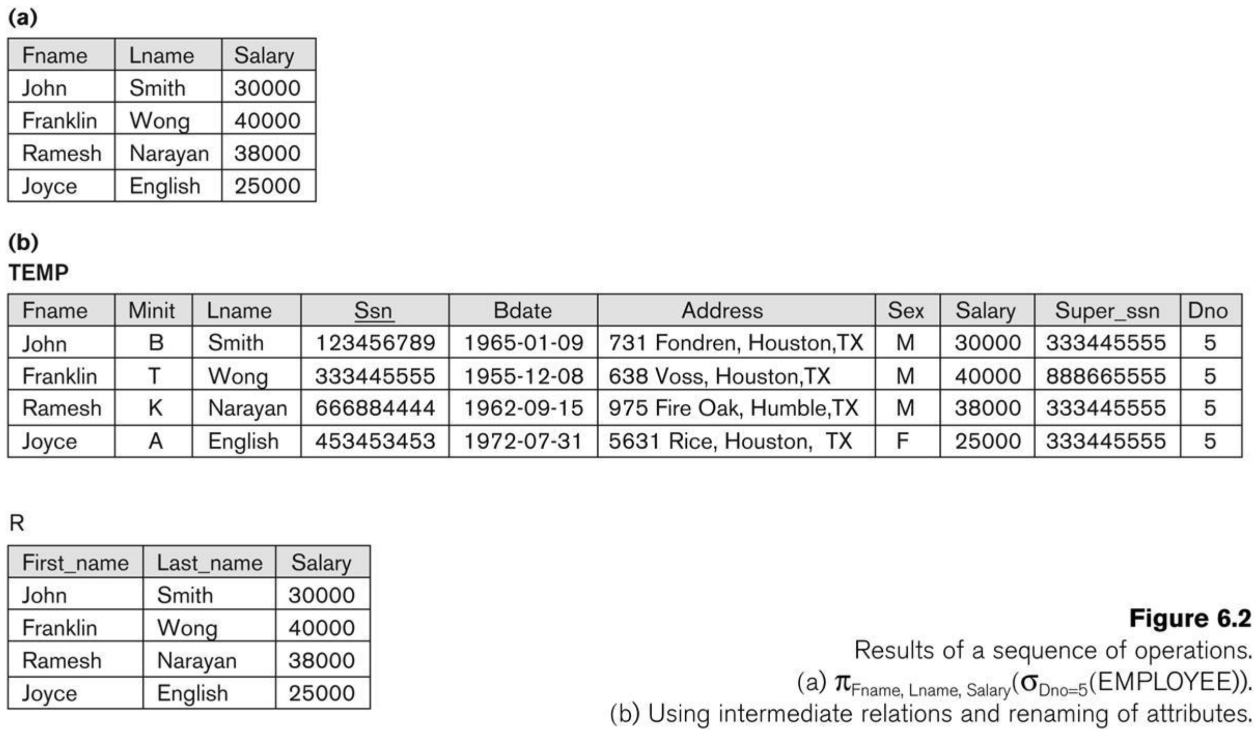
$\prod_{\text{FNAME, LNAME, SALARY}}(\sigma_{\text{DNO=5}}(\text{EMPLOYEE}))$

OR We can explicitly sh ow the *sequence of operations*, giving a name to each intermediate relation:

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

$\text{RESULT} \leftarrow \prod_{\text{FN AME, LNAME, SALARY}} (\text{DEP5\_EMPS})$

Example of applying multiple operations and RENAME



(a)

Fname	Lname	Salary
John	Smith	30000
Franklin	Wong	40000
Ramesh	Narayan	38000
Joyce	English	25000

(b)
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

Figure 6.2
Results of a sequence of operations.
(a) $\pi_{\text{Fname, Lname, Salary}}(\sigma_{\text{Dno=5}}(\text{EMPLOYEE}))$.
(b) Using intermediate relations and renaming of attributes.

RENAME

The RENAME operator i s denoted by ρ (rho)

In some cases, we may w ant to *rename* the attributes of a relation or the relation name or both

Useful when a query requires multiple operations

Necessary in some cases (see JOIN operation later)

RENAME operation – w hich can rename either the relation name or the attribute names, or both

The general RENAME operation ρ can be expressed by any of the following forms:

$\rho_S(R)$ changes:

the *relation name* only to S

$\rho_{(B1, B2, \ldots, Bn)}(R)$ changes:

the *column (attribute) names* only to B1, B1, …..Bn

$\rho_{S\ (B1, B2, \ldots, Bn)}(R)$ changes both:

the relation name to S, *and*

the column (attribute) names to B1, B1, …..Bn

Relational Algebra Operations from Set Theory

- Union
- Intersection
- Minus
- Cartesian Product

UNION

It is a Binary operation, denoted by ∪

The result of R È S, is a relation that includes all tuples that are either in R or in S or in both R and S

Duplicate tuples are eliminated

The two operand relations R and S must be "type compatible" (or UNION compatible)

R and S must have same number of attributes

Each pair of corresponding attributes must be type compatible (have same or compatible domains)

Example:

To retrieve the social security numbers of all employees who either *work indepartment 5* (RESULT1 below) or *directly supervise an employee who works in department 5* (RESULT2 below)

$DEP5\_EM\ PS \leftarrow s_{DNO=5}\ (EMPLOYEE)$

$RESULT1 \leftarrow p_{\ SSN}(DEP5\_EMPS)$

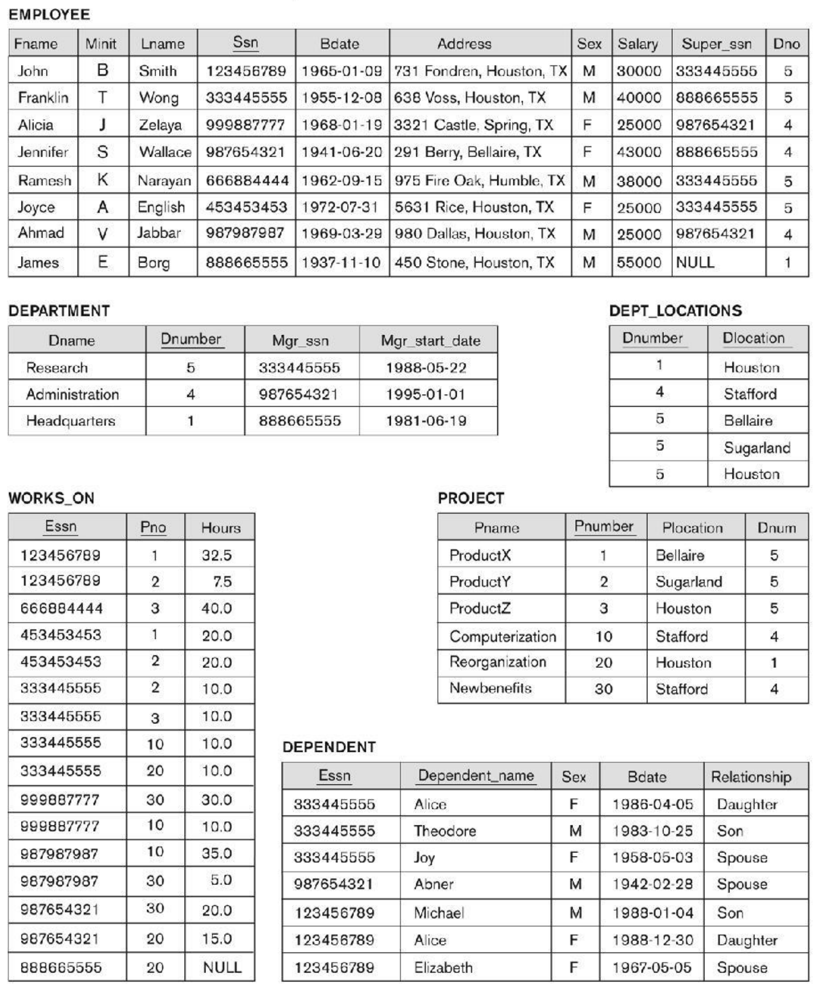
$RESULT2 \leftarrow p_{SUPERSSN}(DEP5\_EMPS)$

$RESULT \leftarrow RESULT1 \cup RESULT2$

The union operation prod uces the tuples that are in either RESULT1 or R ESULT2 or both.

The following query results r efer to this database state.

Figure 5.6
One possible database state for the COMPANY relational database schema.



EMPLOYEE

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
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

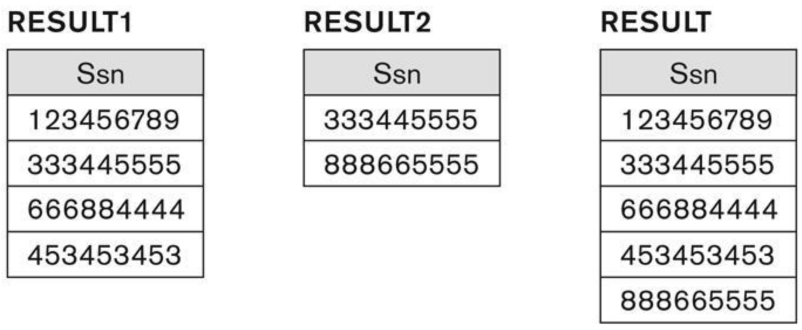
DEPENDENT

Essn	Dependent_name	Sex	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

Example of the result of a UNION operation

UNION Example

Figure 6.3
Result of the UNION operation RESULT ← RESULT1 ∪ RESULT2.



RESULT1

Ssn
123456789
333445555
666884444
453453453

RESULT2

Ssn
333445555
888665555

RESULT

Ssn
123456789
333445555
666884444
453453453
888665555

INTERSECTION

INTERSECTION is denoted by ∩

The result of the operation R ∩ S, is a relation that includes all tuples that are in both R and S

The attribute names in the result will be the same as the attribute names in R

The two operand relations R and S must be "type compatible"

SET DIFFERENCE

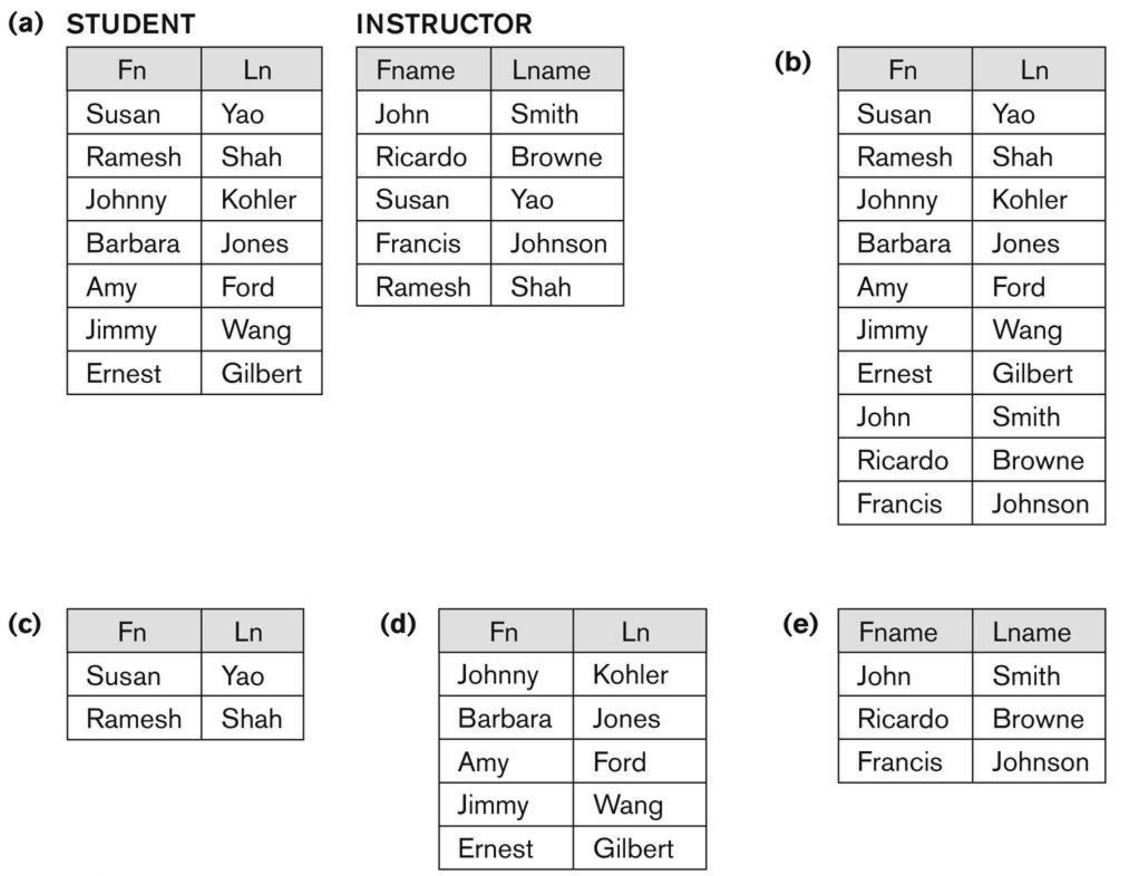
SET DIFFERENCE (also called MINUS or EXCEPT) is denoted b y –

The result of R – S, is a relation that includes all tuples that are in R but not in S

The attribute names in the result will be the same as the attribute names in R

The two operand relations R and S must be "type compatible"

Example to illustrate th e result of UNION, INTERSECT, and DIFFERENCE



(a) 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

(b)

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

(c)

Fn	Ln
Susan	Yao
Ramesh	Shah

(d)

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

(e)

Fname	Lname
John	Smith
Ricardo	Browne
Francis	Johnson

Figure 6.4
The set operations UNION, INTERSECTION, and MINUS. (a) Two union-compatible relations. (b) STUDENT ∪ INSTRUCTOR. (c) STUDENT ∩ INSTRUCTOR. (d) STUDENT – INSTRUCTOR. (e) INSTRUCTOR – STUDENT.

Some properties of UN ION, INTERSECT, and DIFFERENC E

Notice that both union an d intersection are *commutative* operations; that

is R È S = S È R, an d R Ç S = S Ç R

Both union and intersection can be treated as n-ary operations applicable to any number of relations as bo th are *associative* operations; that is

R È (S È T) = (R È S) È T

(R Ç S) Ç T = R Ç (S Ç T)

The minus operation is n ot commutative; that is, in general

$R - S \neq S - R$

CARTESIAN PRODUCT

CARTESIAN PRODUCT Operation

This operation is used to combine tuples from two relations in a combinatorial fashion.

Denoted by R(A1, A2, . . ., An) x S(B1, B2, . . ., Bm)

Result is a relation Q with degree **n + m** attributes:

Q(A1, A2, . . ., An, B1, B2, . . ., Bm), in that order.

The resulting relation state has one tuple for each 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 x S will have $\mathbf{n_R * n_S}$ tuples.

The two operands do NOT have to be "type compatible"

Generally, CROSS PRODUCT is not a meaningful operation

Can become meaningful when followed by other operations

Example (not meaningful):

FEMALE_EMPS $\leftarrow \sigma_{SEX='F'}$(EMPLOYEE)

EMPNAMES $\leftarrow \prod_{FNAME,\ LNAME,\ SSN}$ (FEMALE_EMPS)

EMP_DEPENDENTS ← EMPNAMES x DEPENDENT

The following query re sults refer to this database state

Dat abase Management System

Example of applying C ARTESIAN PRODUCT

Example of applying C ARTESIAN PRODUCT

To keep only combinations where the DEPENDENT is related to the EM PLOYEE, we add a SELECT opera tion as follows Add:

ACTUAL_DEPS $\leftarrow \sigma_{SSN=ESSN}$**(EMP_DEPENDENTS)**
RESULT $\leftarrow \prod_{FNAME,\ L\ NAME,\ DEPENDENT\_NAME}$ **(ACTUAL_DEPS)**

Binary Relational Oper ations

- **Division**
- **Join**

Database Management System

Division

Interpretation of the division operation A/B:

- Divide the attributes of A into 2 sets: A1 and A2.

- Divide the attributes of B into 2 sets: B2 and B3.

- Where the sets A2 and B2 have the same attributes.

- For each set of values in B2:

  - Search in A2 for the sets of rows (having the same A1 values) whose A2 values (taken together) form a set which is the same as the set of B2's.

  - For all the set of rows in A which satisfy the above search, pick out their A1 values and put them in the answer.

JOIN

JOIN Operation (denoted by)

The sequence of **CARTESIAN PRODECT** followed by **SELEC T** is used quite commonly to identify and select related tuples from two rel ations

This operation is very important for any relational database with more than a single relation, be cause it allows us *combine related tuples* from various relations

The general form of a join operation on two relations R(A1, A2, . . ., An) and S(B1, B2, . . ., Bm) is:

R <join condition>S

where R and S can be any relations that result from general *relational algebraexpressions*.

Example: Suppose that w e want to retrieve the name of the manager of each department.

Dat abase Management System

To get the manager’s nam e, we need to combine each DEPARTMENT t uple with the EMPLOYEE tuple whos e SSN value matches the MGRSSN value in the department tuple.

DEPT_MGR ← DEPARTMENT $_{\text{MGRSSN=SSN}}$ EMPLOYEE

The following query re sults refer to this database state

Example of applying th e JOIN operation

DEPT_MGR ← DEPA RTMENT MGRSSN=SSN EMPLOYEE

Dat abase Management System

The general case of JOIN operation is called a Theta-join:

R *theta* S

The join condition is called *theta*

Theta can be any general boolean expression on the attributes of R and S; forexample:

R.Ai<S.Bj AND (R.Ak=S.Bl OR R.Ap<S.Bq)

EQUIJOIN

The most common use of join involves join conditions with *equality com parisons* only Such a join, where the only comparison operator used is =, is called an EQUIJOIN.

The JOIN seen in the previous example was an EQUIJOIN

NATURAL JOIN

Another variation of JOI N called NATURAL JOIN — denoted by *

It was created to get rid of the second (superfluous) attribute in a EQUIJOIN condition.

Another example: Q ← R (A,B,C,D) * S(C,D,E)

The implicit join condition includes *each pair* of attributes with t he same name, “AND”ed t ogether:

R.C=S.C AND R.D = S.D

Result keeps only one attribute of each such pair:

Q(A,B,C, D,E)

Dat abase Management System

Example: To apply a nat ural join on the DNUMBER attributes of DEPARTMENT and DEPT_LOCATION S, it is sufficient to write:

DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS
Only attribute with the same name is DNUMBER

An implicit join condition is created based on this attribute:

DEPARTMENT.DNUMBER=DEPT_LOCATIONS.DNUMBER

The following query resu lts refer to this database state

Example of NATURAL JOIN operation

Complete Set of Relat ional Operations

The set of operations incl uding SELECT σ, PROJECT ∏, UNION ∪, DIFFERENCE

- , RENAME ρ, and CAR TESIAN PRODUCT X is called a *complete se* because any other relational algebra e xpression can be expressed by a combination of these five operations.

For example:

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

$$R_{\text{<join con dition>}} S = \sigma_{\text{<join condition>}}(R \times S)$$

Dat abase Management System

Recap of Relational Alg ebra Operations

NATURAL JOIN

Example: To apply a nat ural join on the DNUMBER attributes of DEPARTMENT and DEPT_LOCATION S, it is sufficient to write:

DEPT_LOCS ← DEPARTMENT * DEPT_LOCATIONS

Only attribute with the same name is DNUMBER

An implicit join condition is created based on this attribute:

DEPARTMENT.DNUMBE R=DEPT_LOCATIONS.DNUMBER

Database Management System

Aggregate Functions and Grouping

A type of request that cannot be expressed in the basic relational algebra is to specify mathematical aggregate functions on collections of values from the database.

Examples of such functions include retrieving the average or total salary of all employees or the total number of employee tuples.

Common functions applied to collections of numeric values include

SUM, AVERAGE, MAXIMUM, and MINIMUM.

The COUNT function is used for counting tuples or values.

Use of the Aggregate Functional operation ζ

ζ MAX Salary (EMPLOYEE) retrieves the maximum salary value from the EMPLOYEE relation

ζ MIN Salary (EMPLOYEE) retrieves the minimum Salary value from the EMPLOYEE relation

ζ SUM Salary (EMPLOYEE) retrieves the sum of the Salary from the EMPLOYEE relation

ζCOUNT SSN, AVERAGE Salary (EMPLOYEE) computes the count (number) of employees and their average salary

Additional Relational Operations

Outer Join

The OUTER JOIN Operation

Database Management System

In NATURAL JOIN and EQUIJOIN, tuples without a *matching* (or *related*) tuple are eliminated from the join result

Tuples with null in the join attributes are also eliminated

This amounts to loss of information.

A set of operations, called OUTER joins, can be used when we want 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.

The **left outer join** operation keeps every tuple in the first or left relation R in R S; if no matching tuple is found in S, then the attributes of S in the join result are filled or "padded" with null values.

A similar operation, right outer join, keeps every tuple in the second or right relation S in the result of R S.

A third operation, full outer join, denoted by keeps all tuples in both the left and the right relations when no matching tuples are found, padding them with null values as needed.

Left Outer Join

E.g. List all employees and t he department they manage, *if* they manage a de partment.

Outer join

Left ou ter,rightouter and full outer join

Examples of Queries in Relational Algebra

Q1: Retrieve the name and address of all employees who work for the 'Research' department.

RESEARCH_DEPT $\leftarrow \sigma_{\text{DNAME='Research'}}$ (DEPARTMENT)

RESEARCH_EMPS $\leftarrow$ (RESEARCH_DEPT $_{\text{DNUMBER= DNOEMPLOYEE}}$ EMPLOYEE)

RESULT $\leftarrow \prod_{\text{FNAME, LNAME, ADDRESS}}$ (RESEARCH_EMPS)

Q6: Retrieve the names of employees who have no dependents.

ALL_EMPS $\leftarrow \prod_{\text{SSN}}$(EMPLOYEE)

EMPS_WITH_DEPS(SSN) $\leftarrow \prod_{\text{ESSN}}$(DEPENDENT)

EMPS_WITHOUT_DEPS ← (ALL_EMPS - EMPS_WITH_DEPS)

RESULT ←$\prod_{\text{LNAME, FNAME}}$ (EMPS_WITHOUT_DEPS *

EMPLOYEE)

PREPARED BY: NAMRATHA
K

Page 35

MODULE 3

SQL The Relational Database Standard

4.1 Data Definition, Constraints, and Schema Changes in SQL2

Structured Query Language (SQL) was designed and implemented at IBM Research.
Created in late 70's, under the name of SEQUEL
A standard version of SQL (ANSI 1986), is called SQL86 or SQL1.
A revised version of standard SQL, called SQL2 (or SQL92).
SQL are going to be extended with objectoriented and other recent database concepts.
Consists of

A Data Definition Language (DDL) for declaring database schemas
Data Manipulation Language (DML) for modifying and quer ying database instances

In SQL, relation, tuple, and attribute are called *table*, *row* , and *columns* respectively.
The SQL commands for data definition are *CREATE*, *ALTER* , and *DROP* .
The *CREATE TABLE* Command is used to specify a new table by giving it a name and specifying its attributes (columns) and constraints.
Data types available for attributes are:

- o *Numeric* integer, real (formated, such as *DECIMAL(10,2)*)
- o *CharacterString* fixedlength and varyinglength
- o *BitString* fixedlength, varyinglength
- o *Date* in the form YYYYMMDD
- o *Time* in the form HH:MM:SS
- o *Timestamp* includes both the *DATE* and *TIME* fields
- o *Interval* to increase/decrease the value of date, time, or timestamp

4.2 Basic Queries in SQL

SQL allows a table (relation) to have two or more tuples that are identical in all their attributes values. Hence, an **SQL table** is not a set of tuple, because a set does not allow two identical members; rather it is a multiset of tuples.
A basic query statement in SQL is the *SELECT* statement.
The *SELECT* statement used in SQL has no relationship to the *SELECT* operation of relational algebra.

The SELECT Statement

The syntax of this command is:

SELECT <attribute list>
FROM <table list>
WHERE <Condition>;

Some example:

59

Query 0: Retrieve the birthday and address of the employee(s) whose name is =John B. Smith‘

Q0: SELECT BDATE, ADDRESS

FROM EMPLOYEE

WHERE FNAME = =John‘ AND MINIT ==B‘ AND LNAME = =SMITH‘

Query 1: Retrieve the name and address of all employee who work for the =Research‘ Dept.

Q1: SELECT FNAME, LNAME, ADDRESS

FROM EMPLOYEE, DEPARTMENT

WHERE DNAME = =Resear ch‘ AND DNUMBER = DNO

Query 2: For every project located in =Stafford‘, list the project number, the controlling department number, and the department manager‘s last name, address, and birthdate.

Q2: SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE

FROM PROJECT, DEPARTMENT, EMPLOYEE

WHERE DNUM=DNUMBER AND MGRSSN=SSN AND PLOCATION = =Stafford‘

Dealing with Ambiguous Attribute Names and Renaming (Aliening)

Ambiguity in the case where attributes are same name need to qualify the attribute using DOT separator

e.g., WHERE DEPARTMENT.DNUMBER=EMPLOYEE.DNUMBER

More

Ambiguity in the case of queries that refer to the same relation twice

Query 8: For each employee, retrieve the employee‘s first and last name and the first and last name of his or her immediate supervisor

Q8: SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME

FROM EMPLOYEE AS E, EMPLOYEE AS S

60

WHERE E.SUPERSSN=S.SSN

Unspecified WHEREClause and Use of Asterisk (*)

A missing WHEREclause indicates no conditions, which means all tuples are selected

In case of two or more table, then all possible tuple combinations are selected

Example: Q10: Select all EMPLOYEE SSNs , and all combinations of EMPLOYEE SSN and DEPARTMENT DNAME

SELECT SSN, DNAME

FROM EMPLOYEE, DEPARTMENT

More

To retrieve all the attributes, use * in SELECT clause

Retrieve all employees working for Dept. 5

SELECT *

FROM EMPLOYEE

WHERE DNO=5

Substring Comparisons, Arithmetic Operations, and Ordering

like, binary operator for comparing strings
%, wild card for strings
_, wild card for characters
||, concatenate operation for strings

(name like ‘ %a_‘) is true for all names having =a‘ as second letter from the end.

Partial strings are specified by using '

```
   SELECT   FNAME, LNAME
   FROM     EMPLOYEE
WHERE    FNAME LIKE '%Mc%';
```

In order to list all employee who were born during 1960s we have the followings:

```
   SELECT   FNAME, LNAME
.  FROM     EMPLOYEE
WHERE    BDATE LIKE '6_______';
```

61

SQL also supports addition, subtraction, multiplication and division (denoted by +, , *, and /, respectively) on numeric values or attributes with numeric domains.

Examples: Show the resulting salaries if every employee working on the 'ProductX' project is given a 10 percent raise.

```
SELECT   FNAME, LNAME, 1.1*SALARY
FROM     EMPLOYEE, WORKS_ON, PROJECT
WHERE    SSN=ESSN AND PNO=PNUMBER AND PNAME='ProductX';
```

Retrieve all employees in department number 5 whose salary between $30000 and $40000.

```
SELECT   *
FROM     EMPLOYEE
WHERE    (SALARY BETWEEN 30000 AND 40000) AND DNO=5;
```

It is possible to order the tuples in the result of a query.

```
SELECT   DNAME, LNAME, FNAME, PNAME
FROM     DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT
WHERE    DNUMBER=DNO AND SSN=ESSN AND PNO=PNUMBER
ORDER BY DNAME, LNAME, FNAME;
```

The default order is in ascending order, but user can specif y

```
ORDER BY DNAME DESC, LNAME ASC, FNAME, ASC;
```

Tables as Sets in SQL

SQL treats table as a **multiset,** which means duplicate tuples are OK

SQL does not delete duplicate because Duplicate elimination is an expensive operation (sort and delete) user may be interested in the result of a query in case of aggregate function, we do not want to eliminate duplicates

To eliminate duplicate, use DISTINCT

examples

Q11: Retrieve the salary of ever y employee , and (Q!2) all distinct salary values

Q11: SELECT ALL SALARY

FROM EMPLOYEE

Q12: SELECT DISTINCT SALARY

62

FROM   EMPLOYEE

4.3 More Complex SQL Queries

Complex SQL queries can be formulated by composing nested SELECT/FROM/WHERE clauses within the WHEREclause of another query

Example: Q4:   Make a list of Project numbers for projects that involve an employee whose last name is =Smith‘, either as a worker or as a manger of the department that controls the project

Q4   SELECT DISTINCT PNUMBER

FROM PROJECT

WHERE PNUMBER IN (SELECT PNUMBER

FROM PROJECT, DEPARTMENT, EMPLOYEE

WHERE DNUM=DNUMBER AND   MGRSSN=SSN AND LNAME==Smith‘

OR   PNUMBER IN (SELECT PNO

FROM WORKS_ON, EMPLOYEE

WHERE ESSN=SSN AND LNAME==Smith‘)

IN operator and set of unioncompatible tuples

Example:

SELECT DISTINCT ESSN

FROM WORKS_ON

WHERE (PNO, HOURS) IN (SELECT PNO, HOURS

FROM   WORKS_ON

WHERE SSN==123456789‘

ANY, SOME and >, <=,<>,etc.

The keyword ALL

In addition to the IN operator, a number of other comparison operators can be used to compare a single value v to a set of multiset V.

ALL V returns TRUE if v is greater than all the value in the set

Select the name of employees whose salary is greater than the salary of all the employees in department 5

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE SALARY > ALL (SELECT SALARY
FROM EMPLOYEE
WHERE DNO=5);
```

Ambiguity in nested query

```
SELECT E.FNAME, E.LNAME
FROM EMPLOYEE AS E
WHERE E.SSN IN (SELECT ESSN
FROM DEPENDENT
WHERE  ESSN=E.SSN  AND  E.FNAM=DEPENDENT_NAME  AND
SEX=E.SEX
```

Correlated Nested Query

Whenever a condition in the WHEREclause of a nested quer y references some attributes of a relation declared in the outer query, the two queries are said to be correlated. The result of a correlated nested query is different for each tuple (or combination of tuples) of the r elation(s) the outer query.

In general, any   nested query involving the = or comparison operator IN can always be rewritten as a single block query

```
SELECT E.FNAME, E.LNAME

FROM EMPLOYEE E, DEPENDENT D

WHERE E.SSN=D.ESSN AND E.SEX=D.SEX AND E.FNAME =D.DEPENDENT=NAME
```

Query 12: Retrieve the name of each emplo yee who has a dependent with the same first name as the employee.

```
Q12:   SELECT  E.FNAME, E.LNAME
FROM  EMPLOYEE AS E
WHERE  E.SSN IN (SELECT  ESSN
FROM  DEPENDENT
WHERE  ESSN=E.SSN AND
E.FNAME=DEPENDENT_NAME)
```

In Q12, the nested query has a different result for each tuple in the outer query.

The original SQL as specified for SYSTEM R also had a CONTAINS comparison operator, which is used in conjunction with nested correlated queries This operator was dropped from the language, possibly because of the difficulty in implementing it efficiently Most implementations of SQL do not have this operator The CONTAINS operator compar es two sets of values , and returns TRUE if one set contains all values in the other set (reminiscent of the division operation of algebra).

Query 3: Retrieve the name of each employee who works on all the projects controlled by department number 5.

```
Q3: SELECT FNAME, LNAME

FROM EMPLOYEE WHERE ( (SELECT PNO FROM WORKS_ON WHERE SSN=ESSN)

CONTAINS (SELECT PNUMBER FROM PROJECT WHERE DNUM=5) )
```

In Q3, the second nested query, which is not correlated with the outer query, retrieves the project numbers of all projects controlled by department 5.

The first nested query, which is correlated, retrieves the project numbers on which the employee works, which is different for each employee tuple because of the correlation.

THE EXISTS AND UNIQUE FUNCTIONS IN SQL

EXISTS is used to check whether the result of a correlated nested query is empty (contains no tuples) or not We can formulate Query 12 in an alternative form that uses EXISTS as Q12B below.

Query 12: Retrieve the name of each employee who has a dependent with the same first name as the employee.

```
SELECT E.FNAME, E.LNAME
FROM  EMPLOYEE E
WHERE EXISTS (SELECT *
FROM DEPENDENT
```

```
WHERE  E.SSN=ESSN  AND  SEX=E.SEX  AND
E.FNAME=DEPENDENT_NAME
```

Query 6: Retrieve the names of employees who have no dependents.

```
Q6:  SELECT  FNAME, LNAME
FROM  EMPLOYEE
WHERE  NOT EXISTS   (SELECT  *
FROM   DEPENDENT
WHERE  SSN=ESSN)
```

In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If none exist , the EMPLOYEE tuple is selected EXISTS is necessary for the expressive power of SQL

EXPLICIT SETS AND NULLS IN SQL

It is also possible to use an explicit (enumerated) set of values in the WHEREclause rather than a nested query Quer y 13: Retrieve the social security numbers of all employees who work on project number 1, 2, or 3.

Retrieve SSNs of all employees who work on project number 1,2,3

```
SELECT DISTINCT ESSN
FROM  WORKS_ON
WHERE PNO IN (1,2,3)
```

Null example

SQL allows queries that check if a value is NULL (missing or undefined or not applicable) SQL uses IS or IS NOT to compare NULLs because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate .

Retrieve the names of all employees who do not have supervisors

```
SELECT FNAME, LNAME

FROM EMPLOYEE

WHERE SUPERSSN IS NULL
```

Note: If a join condition is specified, tuples with NULL values for the join attributes are not included in the result

Join Revisit

Retrieve the name and address of every employee who works for =Search‘ department

```
SELECT FNAME, LNAME, ADDRESS
FROM (EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER)
WHERE DNAME==Search‘
```

Aggregate Functions

Include COUNT, SUM, MAX, MIN, and AVG

Query 15: Find the sum of the salaries of all employees the =Research‘ dept, and the max salary, the min salary, and average:

```
SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY) AVG(SALARY)

FROM EMPLOYEE

WHERE DNO=FNUMBER AND DNAME==RSEARCH‘
```

Query 16: Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department.

```
Q16:  SELECT MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM  EMPLOYEE, DEPARTMENT
WHERE  DNO=DNUMBER AND DNAME='Research'
```

Queries 17 and 18: Retrieve the total number of employees in the company (Q17), and the number of employees in the 'Research' department (Q18).

```
Q17:  SELECT  COUNT (*)
FROM  EMPLOYEE
Q18:  SELECT  COUNT (*)
FROM  EMPLOYEE, DEPARTMENT
WHERE  DNO=DNUMBER AND DNAME='Research'
```

Example of grouping

In many cases, we want to apply the aggregate functions to subgroups of tuples in a relation Each subgroup of tuples consists of the set of tuples that have the same value for the grouping attribute(s)

The function is applied to each subgroup independently

SQL has a GROUP BYclause for specif ying the grouping attributes, which must also appear in the SELECTclause

For each project, select the project number, the project name, and the number of employees who work on that projet

SELECT PNUMBER, PNAME, COUNT(*)

FROM PROJECT, WORKS_ON

WHERE PNUMBER=PNO

GROUP BY PNUMBER, PNAME

In Q20, the EMPLOYEE tuples are divided into groupseach group having the same value for the grouping attribute DNO
The COUNT and AVG functions are applied to each such group of tuples separately.The SELECTclause includes only the grouping attribute and the functions to be applied on each group of tuples. A join condition can be used in conjunction with grouping

Query 21: For each project, retrieve the project nu mber, project name, and the nu mber of emplo yees who work on that project.
Q21: SELECT PNUMBER, PNAME, COUNT (*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME
In this case, the grouping and functions are applied after the joining of the two relations
THE HAVINGCLAUSE:

Sometimes we want to retrieve the values of these functions for only those groups that satisfy certain conditions. The HAVINGclause is used for specifying a selection condition on groups (rather than on individual tuples)

Query 22: For each project on which more than two employees work , retrieve the project number, project name, and the number of employees who work on that project.

Q22: SELECT PNUMBER, PNAME, COUNT (*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME
HAVING COUNT (*) > 2

Questions

1. Explain how groupBy clause works?What is the difference between WHERE and Having?
2. How does SQL inplement the entity integrity constraints of relational Data Model?Explain with an example?
3. With respect to SQL, explain with example.
4. Explain IN and EXISTS operations with an example
5. Explain UNIQUE and INTERSECTION operations with an example

SQL 2 The Relational Database Standard

5.1 Update Statements in SQL

5.2 Views in SQL

5.3 Additional features

5.4 Database Programming

5.5 Embedded SQL

5.6 Dynamic SQL

5.7 Database stored procedures and SQL/PSM

SQL The Relational Database Standard

5.1 Update Statements in SQL

The Insert Command

INSERT INTO EMPLOYEE

VALUES (=Richard‘,‘K‘,‘Marini‘,653298653‘,‘30dec52‘,98 Oak Forest, Katy, TX‘,‘M‘,37000,‘987654321‘,4)

More on Insert

Use explicit attribute names:

INSERT INTO EMPLOYEE (FNAME, LNAME,SSN)

VALUES (=Richard‘,‘Marini‘, =653298653‘

The DELECT Command

DELETE FROM EMPLOYEE

WHERE LNAME==Brown‘

The UPDATE Command

Used to modify values of one or more selected tuples

Change the location and controlling department number of project number 10 to =Bellaire‘ and 5 respectively

UPDATE PROJECT

SET PLOCATION = = Bellaire‘, DNUM=5

Where PNUMBER=10;

5.2 Views in SQL

A view refers to a single table that is derived from other tables

CREATE VIEW WORKS_ON1

71

AS SELECT FNAME, LNAME, PNAME, HOURS

FROM EMPLOYEE, PROJECT, WORKS_ON WHERE SSN=ESSN AND PNO=PNUMBER

More on View

CREATE VIEW DEPT_INFO(DEPT_NAME, NO_OF_EMPLS, TOTAL_SAL)

AS SELECT DNAME, COUNT(*), SUM(SALARY)

FROM DEPARTMENT, EMPLOYEE

WHERE DNUMBER=DNO

GROUP BY DNAME

More on view

Treat WORKS_ON1 like a base table as follows

SELECT FNAME, LNAME

FROM WORKS_ON1

WHERE PNMAE==PROJECTX‘

Main advantage of view:

Simplify the specification of commonly used queries

More on View

A View is always up to date;

A view is realized at the time we specify(or execute) a quer y on the view

DROP VIEW WORKS_ON1

Updating of Views

Updating the views can be complicated and ambiguous

In general, an update on a view on defined on a single table w/o any aggregate functions can be mapped to an update on the base table

72

More on Views

We can make the following observations:

A view with a single defining table is updatable if we view contain PK or CK of the base table

View on multiple tables using joins are not updatable

View defined using grouping/aggregate are not updatable

Specifying General Constraints

Users can specify certain constraints such as semantics constraints

CREATE ASSERTION SALARY_CONSTRAINT

CHECK (NOT EXISTS (SELECT * FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D

WHERE E.SALARY > M. SALARY **AND** E.DNO=D.NUMBER **AND** D.MGRSSN=M.SSN))

5.3 Additional features

Granting and revoking privileges

Embedding SQL statements in a general purpose languages (C, C++, COBOL, PASCAL)

SQL can also be used in conjunction with a general purpose programming language, such as PASCAL, COBOL, or PL/I. The programming language is called the host language. The embedded SQL statement is distinguished from programming language statements by prefixing it with a special character or command so that a preprocessor can extract the SQL statements. In PL/I the keywords EXEC SQL precede any SQL statement. In some implementations, SQL statements are passed as parameters in procedure calls. We will use PASCAL as the host programming language, and a "$" sign to identify SQL statements in the program. Within an embedded SQL command, we may refer to program variables, which are prefixed by a "%" sign. The programmer should declare program variables to match the data types of the database attributes that the program will process.These program variables may or may not have names that are identical to their corresponding attributes.

Example: Write a program segment (loop) that reads a social security number and prints out some information from the corresponding EMPLOYEE tuple

```
E1:  LOOP:= 'Y';
while LOOP = 'Y' do
begin
writeln('input social security number:');
```

73

```
readln(SOC_SEC_NUM);
$SELECT FNAME, MINIT, LNAME, SSN, BDATE,
ADDRESS, SALARY
INTO %E.FNAME, %E.MINIT, %E.LNAME, %E.SSN,
%E.BDATE, %E.ADDRESS, %E.SALARY
FROM EMPLOYEE
WHERE SSN=%SOC_SEC_NUM ;
writeln( E.FNAME, E.MINIT, E.LNAME, E.SSN,
E.BDATE, E.ADDRESS, E.SALARY);
writeln('more social security numb ers (Y or N)? ');
readln(LOOP)
end;
```

In E1, a single tuple is selected by the embedded SQL quer y; that is why we are able to assign its attribute values directly to program variables. In general, an SQL query can retrieve many tuples. The concept of a cursor is used to allow tupleatatime processing by the PASCAL programCURSORS: We can think of a cursor as a pointer that points to a single tuple (row) from the result of a query.The cursor is declared when the SQL query command is specified. A subsequent OPEN cursor command fetches the query result and sets the cursor to a position before the first row in the result of the query; this becomes the current row for the cursor. Subsequent FETCH commands in the program advance the cursor to the next row and copy its attribute values into PASCAL program variables specified in the FETCH command. An implicit variable SQLCODE communicates to the program the status of SQL embedded commands. An SQLCODE of 0 (zero) indicates successful execution. Different codes are returned to indicate exceptions and errors. A special END_OF_CURSOR code is used to terminate a loop over the tuples in a query result. A CLOSE cursor command is issued to indicate that we are done with the result of the query
When a cursor is defined for rows that are to be updated the clause FOR UPDATE OF must be in the cursor declaration, and a list of the names of any attributes that will be updated follows.The condition WHERE CURRENT OF cursor specifies that the current tuple is the one to be updated (or deleted)

Example: Write a program segment that reads (inputs) a department name, then lists the names of employees who work in that department, one at a time. The program reads a raise amount for each employee and updates the employee's salary by that amount.

```
E2:  writeln('enter the department name:'); readln(DNAME);
$SELECT DNUMBER INTO %DNUMBER
FROM DEPARTMENT
WHERE DNAME=%DNAME;
$DECLARE EMP CURSOR FOR
SELECT SSN, FNAME, MINIT, LNAME, SALARY
FROM EMPLOYEE
WHERE DNO=%DNUMBER
FOR UPDATE OF SALARY;
$OPEN EMP;
$FETCH EMP INTO %E.SSN, %E.FNAME, %E.MINIT,
%E.LNAME, %E.SAL;
```

```
while SQLCODE = 0 do
begin
writeln('employee name: ', E.FNAME, E.MINIT, E.LNAME);
writeln('enter raise amount: ');  readln(RAISE);
$UPDATE EMPLOYEE  SET SALARY = SALARY + %RAISE
WHERE CURRENT OF EMP;
$FETCH EMP INTO %E.SSN, %E.FNAME, %E.MINIT,
%E.LNAME, %E.SAL;
end;
$CLOSE CURSOR EMP;
```

5.4 Database Programming

Objective:
- To access a database from an application program (as opposed to interactive interfaces)

Why?
- An interactive interface is convenient but not sufficient
  - A majority of database operations are made thru application progr ams (increasingly thru web applications)

Embedded commands:
- Database commands are embedded in a general-purpose programming language

Library of database functions:
- Available to the host language for database calls; known as an *API*
  - *API* standards for Application Program Interf ace

A brand new, full-fledged language
- Minimizes impedance mismatch

Impedance Mismatch

Incompatibilities between a host programming language and the database model, e.g.,
- type mismatch and incompatibilities; requires a new binding for each language
- set vs. record-at-a-time processing
  - need special iterators to loop over quer y results and manipulate individual values

Client program *opens a connection* to the database server

Client program *submits queries to and/or updates* the database

When database access is no longer needed, client program *closes (terminates) the connection*

5.5 Embedded SQL

Most SQL statements can be embedded in a general-purpose *host* programming language such as COBOL, C, Java

75

An embedded SQL statement is distinguished from the host language statements by enclosing it between EXEC SQL or EXEC SQL BEGIN and a matching END-EXEC or EXEC SQL END (or semicolon)

Syntax may vary with language

Shared variables (used in both languages) usually prefixed with a colon (:) in SQL

Variables inside DECLARE are shared and can appear (while prefixed by a colon) in SQL statements

SQLCODE is used to communicate errors/exceptions between the database and the program

int loop;

EXEC SQL BEGIN DECLARE SECTION;

varchar dname[16], fname[16], …;

char ssn[10] , bdate[11], …;

int dno, dnumber, SQLCODE, …;

EXEC SQL END DECLARE SECTION;

Connection (multiple connections are possible but only one is active)

CONNECT TO server-name AS connection-name

AUTHORIZATION user-account-info;

Change from an active connection to another one

SET CONNECTION connection-name;

Disconnection

DISCONNECT connection-name;

loop = 1;

while (loop) {

prompt (Enter SSN: , ssn);

EXEC SQL

select FNAME, LNAME, ADDRESS, SALARY

into :fname, :lname, :address, :salary

from EMPLOYEE where SSN == :ssn;

if (SQLCODE == 0) printf(fname, …);

else printf(SSN does not exist: , ssn);

prompt(More SSN? (1=yes, 0=no): , loop);

END-EXEC

}A cursor (iterator) is needed to process multiple tuples
FETCH commands move the cursor to the *next* tuple
CLOSE CURSOR indicates that the processing of query results has been completed
Objective:

5.6 Dynamic SQL

Composing and executing new (not previously compiled) SQL statements at run-time

a program accepts SQL statements from the keyboard at run-time
a point-and-click operation translates to certain SQL query
Dynamic update is relatively simple; dynamic query can be complex
because the type and number of retrieved attributes are unknown at compile time

EXEC SQL BEGIN DECLARE SECTION;

varchar sqlupdatestring[256];

EXEC SQL END DECLARE SECTION;

…prompt (Enter update command: , sqlupdatestring);

EXEC SQL PREPARE sqlcommand FROM :sqlupdatestring;

EXEC SQLSQLJ: a standard for embedding SQL in Java
An SQLJ translator converts SQL statements into Java
These are executed thru the *JDBC* interface
Certain classes have to be imported
E.g., java.sql

77

EXECUTE sqlcommand;

Environment record :
    Keeps track of database connections
Connection record :
    Keep tracks of info needed for a particular connection
Statement record :
    Keeps track of info needed for one SQL statement
Description record :
    Keeps track of tuples
Load SQL/CLI libraries
Declare record handle variables for the above components (called: SQLHSTMT, SQLHDBC, SQLHENV, SQLHDEC)
Set up an environment record using SQLAllocHandle
Set up a connection record using SQLAllocHandle
Set up a statement record using SQLAllocHandle
Prepare a statement using SQL/CLI function SQLPrepare
Bound parameters to program variables
Execute SQL statement via SQLExecute
Bound query columns to a C variable via SQLBindCol
Use SQLFetch to retrieve column values into C variables

5.7 Database stored procedures and SQL/PSM

Persistent procedures/functions (modules) are stored locally and executed by the database server
    As opposed to execution by clients
Advantages:
    If the procedure is needed by many applications, it can be invoked by any of them (thus reduce duplications)
    Execution by the server reduces communication costs
    Enhance the modeling power of views
Disadvantages:
    Every DBMS has its own syntax and this can make the system less portable

A stored procedure

CREATE PROCEDURE procedur e-name (params)

local-declarations

procedure-body;

A stored function

78

CREATE FUNCTION fun-name (params) RETRUNS return-type

local-declarations

function-body;

Calling a procedure or function

CALL procedure-name/fun-name (arguments);

SQL/PSM:
Part of the SQL standard for writing persistent stored modules
SQL + stored procedures/functions + additional programming constructs
E.g., branching and looping statements
Enhance the power of SQL

```
CREATE FUNCTION DEPT_SIZE (IN deptno INTEGER)

RETURNS VARCHAR[7]

DECLARE TOT_EMPS INTEGER;

SELECT COUNT (*) INTO TOT_EMPS

FROM SELECT EMPLOYEE WHERE DNO = deptno;

IF TOT_EMPS > 100 THEN RETURN  HUGE

ELSEIF TOT_EMPS > 50 THEN RETURN  LARGE

ELSEIF TOT_EMPS > 30 THEN RETURN  MEDIUM

ELSE RETURN  SMALL

ENDIF;
```

Questions

Questions

1. List the approaches to DB Programming. Main issues involved in DB Programming?
2. What is Impedance Mismatch problem? Which of the three programming approaches minimizes this problem
3. How are Triggers and assertions defined in SQL?Explain
4. A explain the syntax of a SELECT statement in SQL.write the SQL query for the following relation algebra expression.
5. Explain the drop command with an example
6. How is a view created and dropped? What problems are associated with updating of views?
7. What is embedded SQL? With an example explain how would you Connect to a database, fetch records and display. Also explain the concept of stored procedure in brief.
8. Explain insert, delete and update statements in SQL with example.
9. Write a note on aggregate functions in SQL with examples.

Department of Master of Computer Applications

Subject: Database Management System

Prepared by: Drakshaveni G
Assistant Professor
Dept.of MCA
BMSIT&M

Module -4

MODEL-4 Data Base design-1

6.1 Informal design guidelines for relation schemas

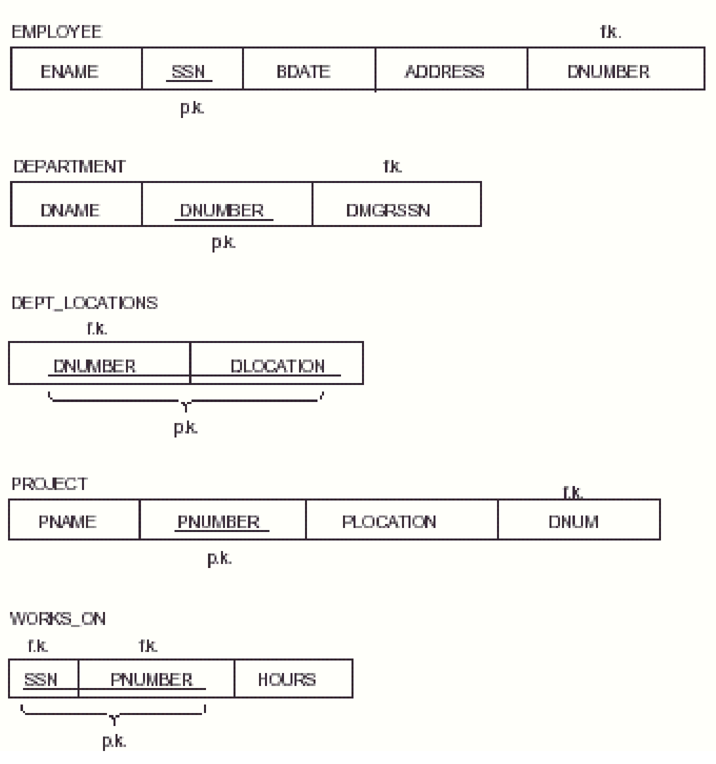
The four informal measures of quality for relation schema

Semantics of the attributes
Reducing the redundant values in tuples
Reducing the null values in tuples
Disallowing the possibility of generating spurious tuples

6.1.1 Semantics of relations attributes

Specifies how to interpret the attributes values stored in a tuple of the relation. In other words, how the attribute value in a tuple relate to one another.

Figure 14.1 Simplified version of the COMPANY relational database schema.



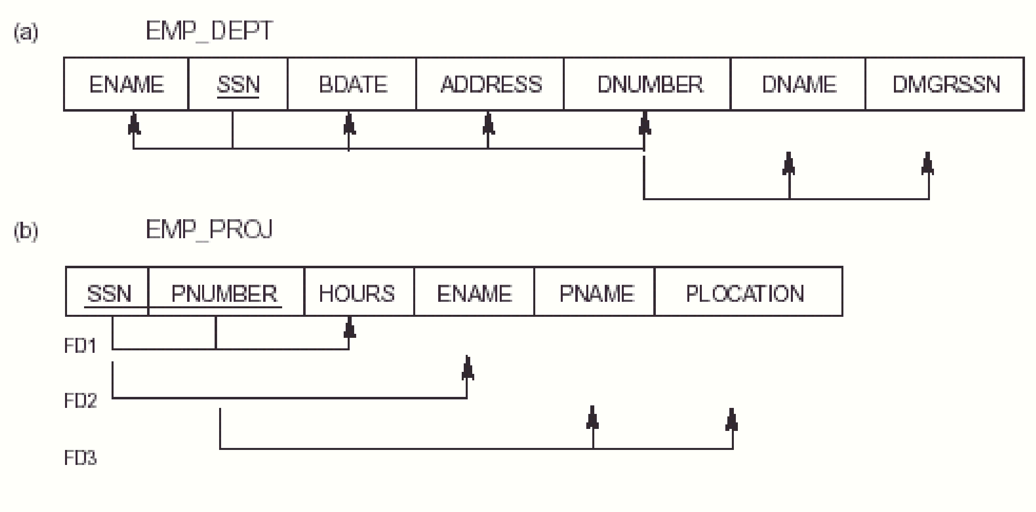
Guideline 1 : Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship types into a single relation.

Reducing redundant values in tuples. Save storage space and avoid update anomalies.

82

Insertion anomalies.
Deletion anomalies.
Modification anomalies.

Figure 14.3 Two relation schemas and their functional dependencies. Both suffer from update anomalies. (a) The EMP_DEPT relation schema. (b) The EMP_PROJ relation schema.



Insertion Anomalies
To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for that department that the employee works for, or nulls.

It's difficult to insert a new department that has no employee as yet in the EMP_DEPT relation. The only way to do this is to place null values in the attributes for employee. This causes a problem because SSN is the primary key of EMP_DEPT, and each tuple is supposed to represent an employee entity - not a department entity.

Deletion Anomalies
If we delete from EMP_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database.

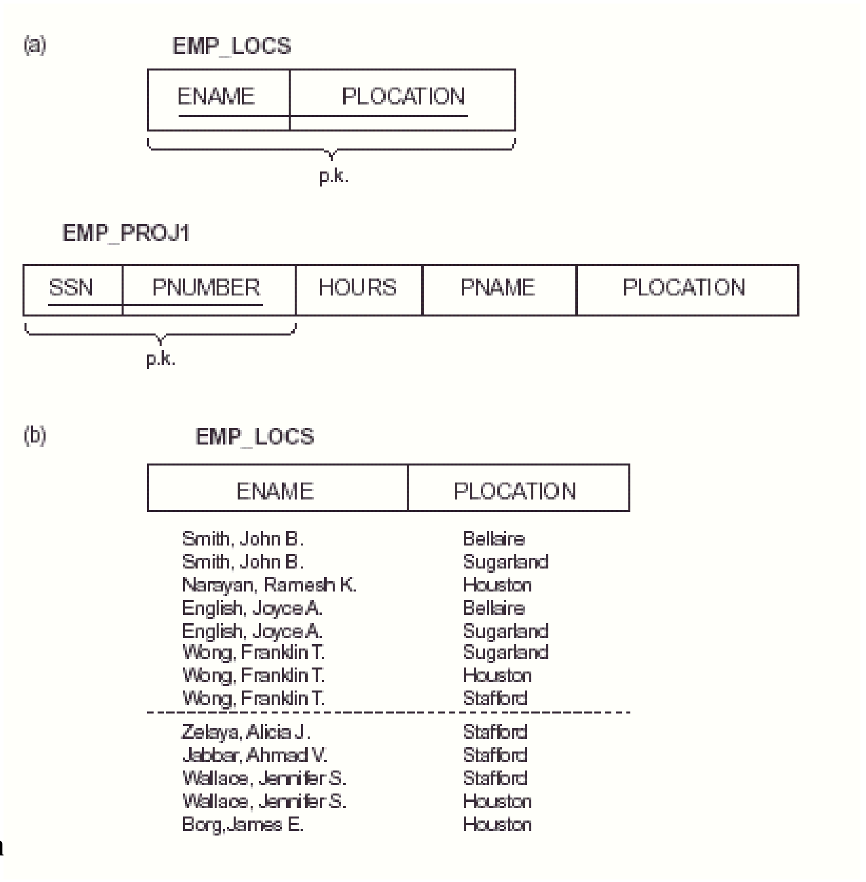
Modification Anomalies
In EMP_DEPT, if we change the value of one of the attributes of a particular department- say the manager of department 5- we must update the tuples of all employees who work in that department.

Guideline 2: Design the base relation schemas so that no insertion, deletion, or modification anomalies occur. Reducing the null values in tuples. e.g., if 10% of employees have offices, it is better to have a separate relation, EMP_OFFICE, rather than an attribute OFFICE_NUMBER in EMPLOYEE.

Guideline 3 : Avoid placing attributes in a base relation whose values are mostly null. Disallowing spurious tuples.

Spurious tuples - tuples that are not in the original r elation but generated by natural join of decomposed subrelations.

Example: decompose EMP_PROJ into EMP_LOCS and EMP_PROJ1.



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

Fig. 14.5a

Guideline 4 : Design relation schemas so that they can be naturally JOINed on primary keys or foreign keys in a way that guar antees no spurious tuples are generated.

6.2 A functional dependency (FD) is a constraint between two sets of attributes from the database. It is denoted by

X Y

We say that " *Y*is **functionally dependent** on *X*". Also, X is called the left-hand side of the FD. Y is called the right-hand side of the FD.

A functional dependency is a property of the semantics or meaning of the attributes, i.e., a property of the relation schema. They must hold on all relation states (extensions) of R. Relation extensions r(R). A FD X *Y* is a full *functional dependency* if removal of any attribute from X means that the dependen[redacted] does not hold any mor e; otherwise, it is a *partial functional dependency.*

Examples:

1. SSN ENAME
2. PNU[redacted]BER {PNAME, PLOCATION}
3. {SSN, PNU[redacted]BER} HOURS [redacted]

FD is property of the relation schema R, not of a particular relation state/instance

Let R be a relation schema, where X R [redacted] and Y R [redacted]

t_1, t_2 r[redacted] $t_1[X] = t_2[X]$ [redacted] $t_1[Y] = t_2[Y]$

The FD X Y holds on R if and only if for all possible relations r(R), whenever two tuples of r agree on th[redacted]attributes of X, they also agree on the attributes of Y.

the single arrow denotes "functional dependency"
X Y can also [redacted]read as "*X*determines *Y*"
th[redacted]ouble arrow denotes "logical implication" [redacted]

6.2.1 Inference Rules

IR1. Reflexivity e.g. *X X* [redacted]

a formal statement of *trivial dependencies* ; useful for derivations

IR2. Augmentation e.g. *X Y XZ Y*

if a dependen[redacted]olds[redacted]then [redacted]can freely expand its left hand side

IR3. Transitivity e.g. *X Y*, *Y Z X Z*

the "most p[redacted]werful[redacted]infere[redacted]ce ru[redacted] useful in multi-step derivations

Armstrong inference rules are sound

meaning that given a set of functional dependencies F specified on a relation schema R, any dependency that we can infer from F by using IR1 through IR3 holds ever y relation state r of R that specifies the dependencies in F. In other words, rules can be used to derive precisely the closure or no additional FD can be derived.

complete

85

meaning that using IR1 through IR3 r epeatedly to infer dependencies until no more dependencies can be inferred results in the complete set of all possible dependencies that can be inferred from F. In other words, given a set of FDs, all implied FDs can be derived using these 3 rules.

Closure of a Set of Functional Dependencies

Given a set X of FDs in relation R, the set of all FDs that are implied by X is called the closure of X, and is denoted X^+.

Algorithms for determining X^+

```
X+ := X;
repeat
oldX+ := X+
for each FD Y Z in F do
if Y   X+ then X+ := X+   Z;
until oldX+ = X+;
```

Example:

A BC

E CF

B E

CD EF

Compute $\{A, B\}^+$ of the set of attributes under this set of FDs.

Solution:

Step1: $\{A, B\}^+ := \{A, B\}$.

Go round the inner loop 4 time, once for each of the given FDs.
On the first iteration, for A BC
A {A, B} +
{A, B} +:= {A, B, C}.

86

Step2: On the second iteration, for E CF, {A, B, C}

Step3 :On the third iteration, for B E
B {A, B,C} +
{A, B} +:= {A, B, C, E}.

Step4: On the fourth iteration, for CD EF remains unchanged.

Go round the inner loop 4 times again. On the first iteration result does not change; on the second it expands to {A,B,C,E,F}; On the third and forth it does not change.

Now go round the inner loop 4 times. Closure does not change and so the whole process terminates, with
{A,B}+= {A,B,C,E,F}

Example.

F = { SSN ENAME, PNUMBER {PNAME, PLOCATION}, {SSN,PNUMBER}
HOURS }

{SSN}+= {SSN, ENAME}

{PNUMBER}+= ?

{SSN,PNUMBER}+= ?

6.3 Normalization

The purpose of normalization.

The problems associated with redundant data.
The identification of various types of update anomalies such as insertion, deletion, and modification anomalies.
How to recognize the appropriateness or quality of the design of relations.
The concept of functional dependency, the main tool for measuring the appropriateness of attribute groupings in relations.
How functional dependencies can be used to group attributes into relations that are in a known normal form.
How to define normal forms for relations.
How to undertake the process of normalization.
How to identify the most commonly used normal forms, namely first (1NF), second (2NF), and third (3NF) normal forms, and Boyce-Codd normal form (BCNF).
How to identify fourth (4NF), and fifth (5NF) normal forms.

Main objective in developing a logical data model for relational database systems is to create an accurate representation of the data, its relationships, and constraints. To achieve this objective, we must identify a suitable set of relations. A technique for producing a set of relations with desirable properties, given the data requirements of an enterprise

NORMAL FORMS

A relation is defined as a *set of tuples*. By definition, all elements of a set are distinct; hence, 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.

Any set of attributes of a relation schema is called a **superkey**. Every relation has at least one superkey—the set of all its attributes. A **key** is a *minimal superkey*, i.e., a superkey from which we cannot remove any attribute and still have the uniqueness constraint hold.

In general, a relation schema may have more than one key. In this case, each of the keys is called a **candidate key.** It is common to designate one of the candidate keys as the **primary key** of the relation. A **foreign key** is a key in a relation R but it's not a key (just an attribute) in other relation R' of the same schema.

Integrity Constraints

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.

The **referential integrity constraint** is specified between two relations and is used to maintain the consistency among tuples of 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 that relation.

An attribute of a relation schema R is called a **prime attribute** of the relation R if it is a member of *any key* of the relation 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.

The goal of normalization is to create a set of relational tables that are free of redundant data and that can be consistently and corr ectly modified. This means that all tables in a relational database should be in the in the third normal form (3 NF).

Normalization of data can be looked on as a process during which unsatisfactory relation schemas are decomposed by breaking up their attributes into smaller relation schemas that possess desirable properties. One objective of the original normalization process is to ensure that the update anomalies such as insertion, deletion, and modification anomalies do not occur.

The most commonly used normal forms

- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form

Other Normal Forms

- Fourth Normal Form
- Fifth Normal Form
- Domain Key Normal Form

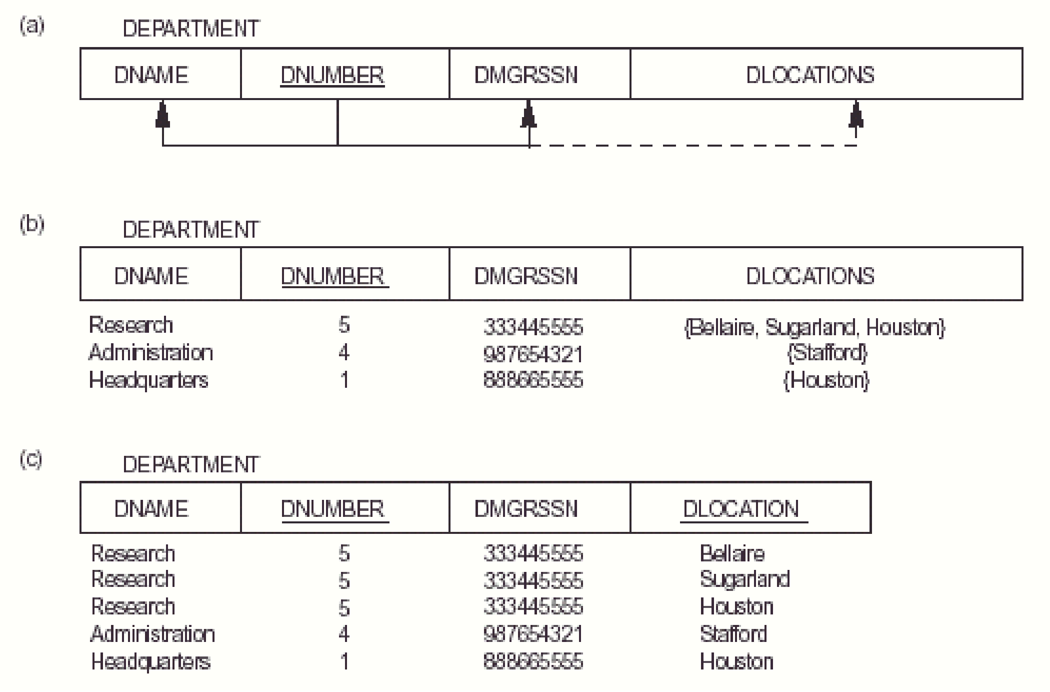
6.3.1 First Normal Form (1NF)

First normal form is now considered to be part of the formal definition of a relation; historically, it was defined to disallow multivalued attributes, composite attributes, and their combinations. It states that the domains of attributes 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.

Practical Rule: "Eliminate Repeating Groups," i.e., make a separate table for each set of related attributes, and give each table a primary key.

Formal Definition: A relation is in first normal form (1NF) if and only if all underlying simple domains contain atomic values only.

Figure 14.8 Normalization into 1NF. (a) Relation schema that is not in 1NF. (b) Example relation instance. (c) 1NF relation with redundancy.



(a) DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS

(b) DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c) DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATION
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

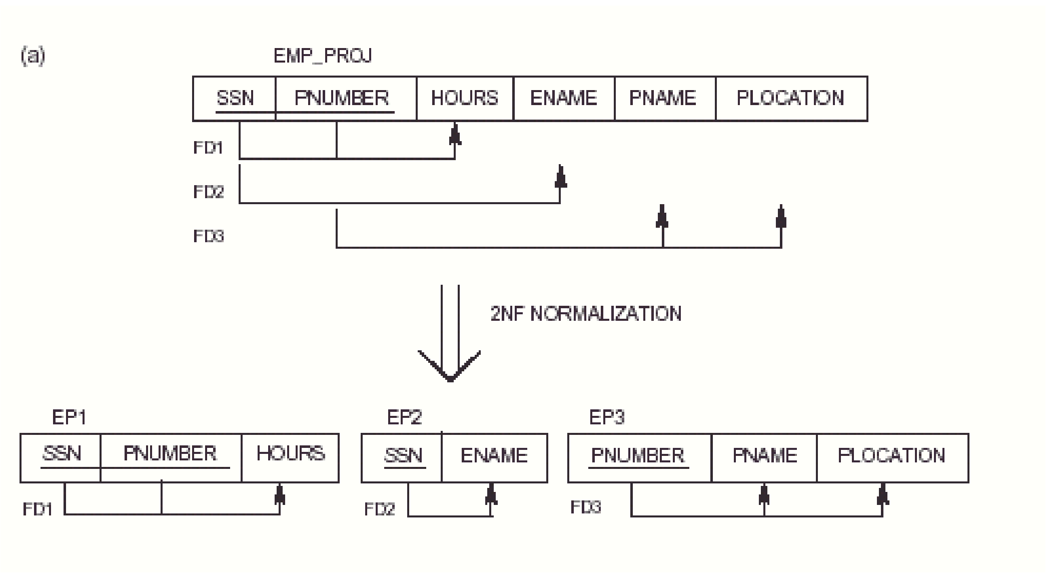
89

6.3.2 Second Normal Form (2NF)

Second normal form is based on the concept of fully functional dependency. A functional X Y is a fully functional dependency is removal of any attribute A from X means that the dependency [redacted] does not hold any more. A relation schema is in 2NF if every nonprime attribute in relation is fully functionally dependent on the primary key of the relation. It also can be restated as: a relation schema is in 2NF if every nonprime attribute in relation is not par tially dependent on any key of the relation.

Practical Rule: "Eliminate Redundant Data," i.e., if an attribute depends on only part of a multivalued key, remove it to a separate table.

Formal Definition: A relation is in second normal form (2NF) if and only if it is in 1NF and every nonkey attribute is fully dependent on the primar y key.



6.3.3 Third Normal Form (3NF)

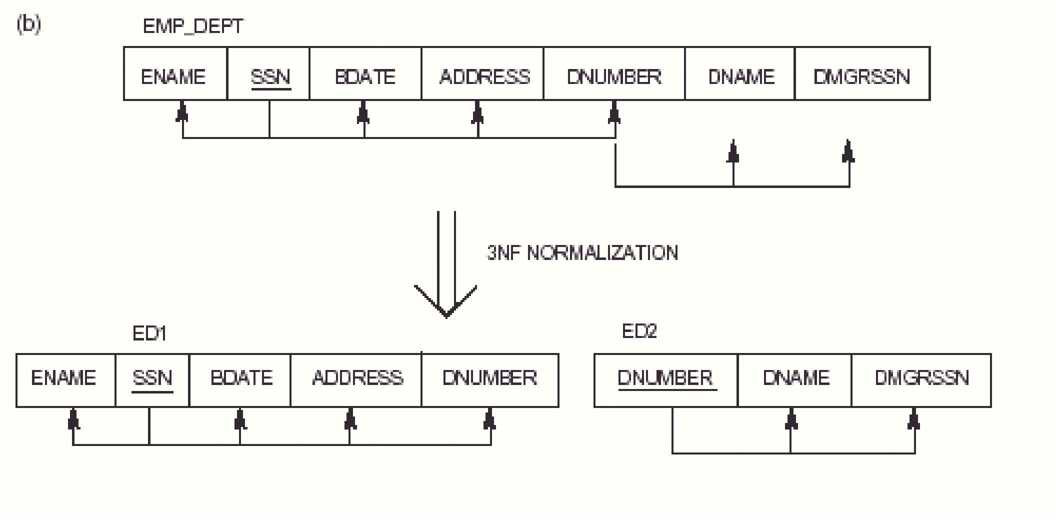
Third normal form is based on the concept of transitive dependency. A functional dependency X Y in a relation is a transitive dependency if there is a set of attributes Z that is not a subset o[redacted]y key of the relation, and both X Z and Z Y hold. In other words, a relation is in 3NF if, whenever a functional dependency [redacted] [redacted]

X A holds in the relation, either (a) X is a superkey of the relation, or (b) A is a prime att[redacted]ute of the relation.

Practical Rule: "Eliminate Columns not Dependent on Key," i.e., if attributes do not contribute to a description of a key, remove them to a separate table.

90

Formal Definition: A relation is in third normal form (3NF) if and only if it is in 2NF and every nonkey attribute is nontransitively dependent on the primary key.



1NF: R is in 1NF iff all domain values are atomic.

2NF: R is in 2 NF iff R is in 1NF and every nonkey attribute is fully dependent on the key.

3NF: R is in 3NF iff R is 2NF and every nonkey attribute is non-transitively dependent on the key.

6.4 Boyce-Codd Normal Form (BCNF)

A relation schema R is in Boyce-Codd Normal Form (BCNF) if whenever a FD X -> A holds in R, then X is a superkey of R

Each normal form is strictly stronger than the previous one:
Every 2NF relation is in 1NF Every 3NF relation is in 2NF
Every BCNF relation is in 3NF
There exist relations that are in 3NF but not in BCNF

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

Additional criteria may be needed to ensure the the set of relations in a relational database are satisfactory.

Figure 14.12 Boyce-Codd normal form. (a) BCNF normalization with the dependency of FD2 being "lost" in the decomposition. (b) A relation R in 3NF but not in BCNF.

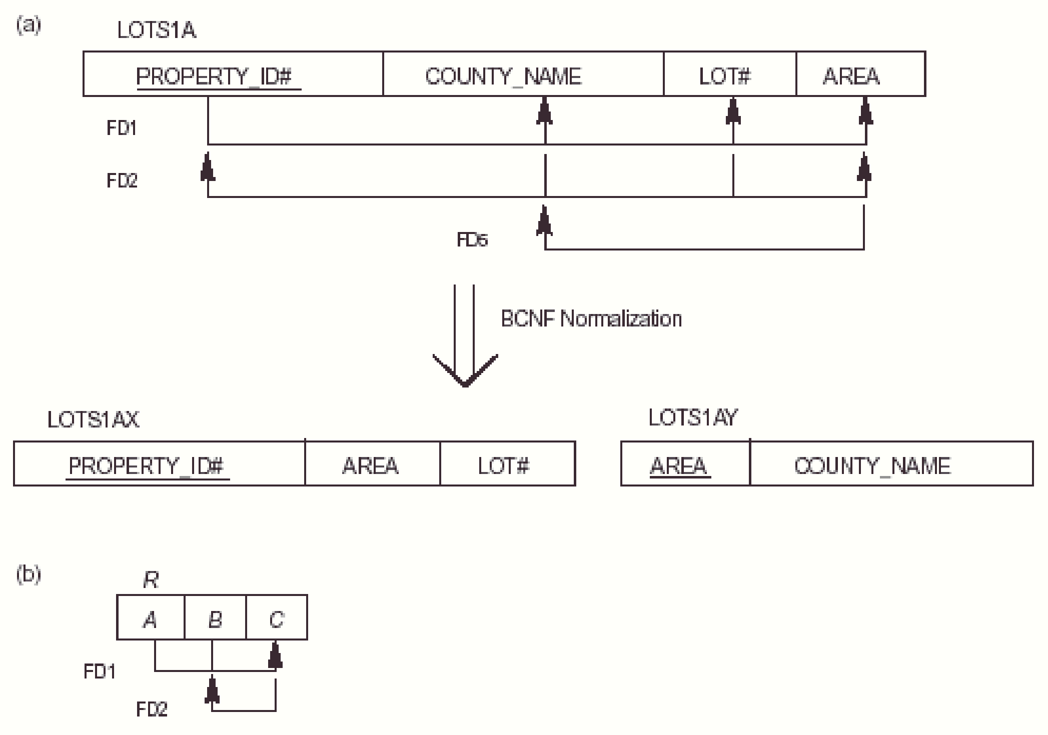


Figure 14.13 A relation TEACH that is in 3NF but not in BCNF.

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

92

If X Y is non-trivial then X is a super key

STREET CITY ZIP

{CITY,STREET } ZIP

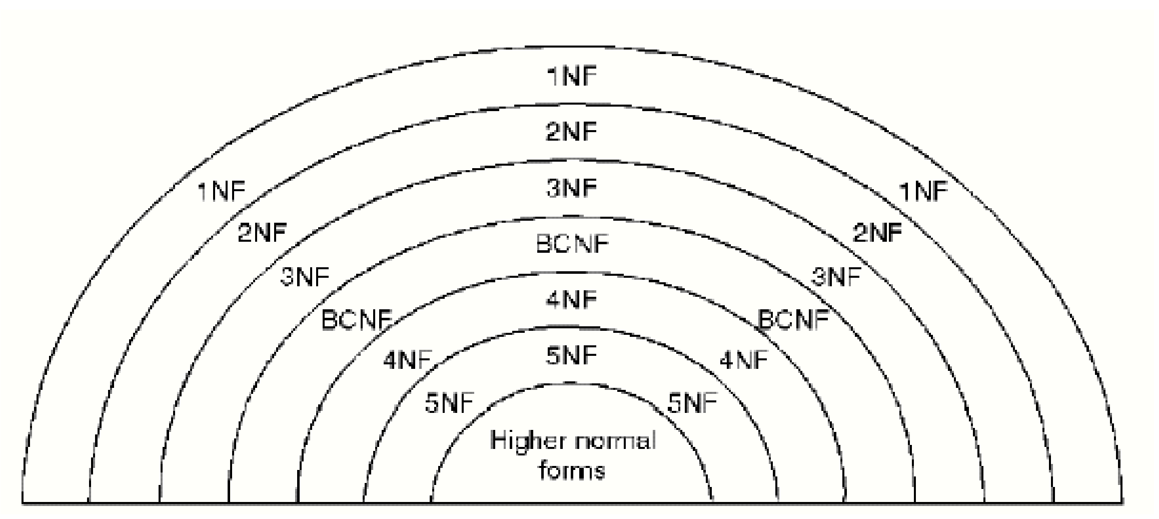
ZIP CITY

Insertion anomaly: the city of a zip code can‘t be stored, if the street is not given

Normalization

STREET ZIP   ZIP CITY

Relationship Between Normal Forms



Questions

1. What is the need for normalization? Explain the first,second and third normal forms with examples.
2. Explain informal design guidelines for relation schemas.
3. A What is functional dependency?write an algorithm to find a minimal cover for a set of functional dependencies.
4. What is the need for normalization ?explain second normal form
5. Which normal form is based on the concept of transitive dependency? Explain with an example the decomposition into 3NF
6. Explain multivalued dependency. Explain 4NF with an example.
7. Explain any Two informal quality measures employed for a relation schema Design?
8. Consider the following relations: Car_sale(car_no,date-sold,salemanno,commission%,discount).assume a car can be sold by multiple salesman and hence primar y key is {car-no,salesman} additional dependencies are: Date-sold discount and salesmanno commision Yes this relation is in 1NF
9. Discuss the minimal sets of FD‘S?

UNIT 7

Data base design 2

Subject Code : **10CS54** IA Marks : 25 No. of Lecture Hours/Week : 04
Exam Hours : 03 Total No. of Lecture Hours : 52 Exam Marks : 100

UNIT-7 Data base design 2

7.1 Properties of relational decomposition

Normalization Algorithms based on FDs to synthesize 3NF and BCNF describe two desirable properties (known as properties of decomposition).

Dependency Preservation Property

Lossless join property

Dependency Preservation Property enables us to enforce a constraint on the original relation from corresponding instances in the smaller relations.

Lossless join property enables us to find any instance of the original relation from corresponding instances in the smaller relations (Both used by the design algorithms to achieve desirable decompositions).
A property of decomposition, which ensures that no spurious rows are generated when relations are reunited through a natural join operation.

7.2 Algorithm s for Relational Database Schema Design

Individual relations being in higher normal do not guarantee a good deign Database schema must posses additional properties to guarantee a good design.

Relation Decomposition and Insufficiency of Normal Forms

Suppose R = { A_1, A_2, …, A_n} that includes all the attributes of the database. R is a universal relation schema, which states that every attribute name is unique. Using FDs, the algorithms decomposes the universal relation schema R into a set of relation schemas
D = {R_1, R_2, …, R_n} that will become the relational database schema; D is called a decomposition of R. Each attribute in R will appear in at least one relation schema R_i in the decomposition so that no attributes are lost; we have

$$\bigcup_{i=1}^{m} R_i = R$$

This is called attribute preservation condition of a decomposition.

7.2.1 Decomposition and Dependency Preservation

We want to preserve dependencies because each dependencies in F represents a constraint on the database.

We would like to check easily that updates to the database do not result in illegal relations being created.

It would be nice if our design allowed us to check updates without having to compute natural joins. To know whether joins must be computed, we need to determine what functional dependencies may be tested by checking each relation individually.

Let F be a set of functional dependencies on schema R. Let D = {R1, R2, …, Rn} be a decomposition of R. Given a set of dependencies F on R, the projection of F on Ri, $_{R_i}$(F), where R$_i$ is a subset of R, is the set of all functional dependencies XY such that attributes in XY are all contained in Ri. Hence the projection of F on each relation schema Ri in the decomposition D is the set of FDs in F+, such that all their LHS and RHS attributes are in Ri. Hence, the projection of F on each relation schema Ri in the decomposition D is the set of functional dependencies in F+.

$((_{R1}(F)) \; (_{R2}(F)) \; \dots \; (_{Rm}(F)))^{+} = F^{+}$

i.e., the union of the dependencies that hold on each Ri belongs to D be equivalent to closure of F (all possible FDs)

/*Decompose relation, R, with functional dependencies, into relations, R_1,..., R_n, with associated functional dependencies,

F_1,..., F_k.

The decomposition is **dependency preserving iff**

$F^{+}=(F_1 \dots F_k)^{+}$*/

If each functional dependency specified in F either appeared directly in one of the relation schema R in the decomposition D or could be inferred from the dependencies that appear in some R.

7.2.2 Lossless-join Dependency

A property of decomposition, which ensures that no spurious rows are generated when relations are reunited through a natural join operation.

Lossless-join property refers to when we decompose a relation into two relations - we can rejoin the resulting relations to produce the original relation.

$$\boxed{R_1 \bowtie R_2 = R}$$

Decompose relation, R, with functional dependencies, F, into relations, R1 and R2, with attributes, A1 and A2, and associated functional dependencies, F1 and F2.

Decompositions are projections of relational schemas

R

A	B	C
a1	b1	c1
a2	b2	c2
a3	b1	c3

$\pi_{A,B}$

A	B
a1	b1
a2	b2
a3	b1

$\pi_{B,C}$

B	C
b1	c1
b2	c2
b1	c3

Old tables should be derivable from the newer ones through the natural join operation

$\pi_{A,B}(R) \bowtie \pi_{B,C}(R)$

A	B	C
a1	b1	c1
a2	b2	c2
a3	b1	c3
a1	b1	c3
a3	b1	c1

Wrong!

R_1, R_2 is a lossless join decomposition of R iff the attributes common to R_1 and R_2 contain a key for at least one of the involved relations

R

A	B	C
a1	b1	c1
a2	b2	c2
a3	b1	c1

A, B

A	B
a1	b1
a2	b2
a3	b1

B, C

B	C
b1	c1
b2	c2

$\pi_{A,B}(R) \bowtie \pi_{B,C}(R) = B$

The decomposition is **lossless iff** :

A A_1 A_2 $\backslash A_1 A_2$ is in F_+, or

A_1 A_2 $A_2 \backslash A_1$ is in F_+

However, sometimes there is the requirement to decompose a relation into more than two relations. Although rare, these cases are managed by join dependency and 5NF.

7.3 Multivalued Dependencies and Fourth Normal Form (4NF)

4NF associated with a dependency called **multi-valued dependency** (MVD). MVDs in a relation are due to first normal form (1NF), which disallows an attribute in a row from having a set of values.

MVD represents a dependency between attributes (for example, A, B, and C) in a relation, such that for each value of A there is a set of values for B, and a set of values for C. However, the set of values for B and C are independent of each other.
MVD between attributes A, B, and C in a relation using the following notation

A ***B*** ***(A multidetermines B)***

A ***C***

Formal Definition of Multivalued Dependency

A **multivalued dependency** (MVD) X Y specified on R, where X, and Y are both subsets of R and Z = (R – (X Y)) specifies the following restrictions on r(R)

$t_3[X]=t_4[X]=t_1[X]=t_2[X]$

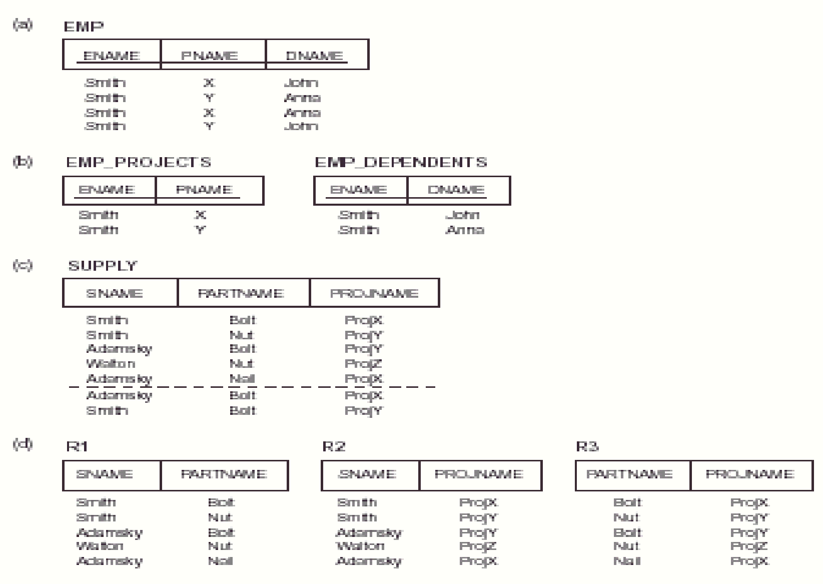
$t_3[Y] = t_1[Y]$ and $t_4[Y] = t_2[Y]$

$t_3[Z] = t_2[Z]$ and $t_4[Z] = t_1[Z]$

7.3.1 Fourth Normal Form (4NF)

A relation that is in Boyce-Codd Normal Form and contains no MVDs. BCNF to 4NF involves the removal of the MVD from the relation by placing the attribute(s) in a new relation along with a copy of the determinant(s).

A Relation is in 4NF if it is in 3NF and there is no multivalued dependencies.



(a) EMP

ENAME	PNAME	DNAME
Smith	X	John
Smith	Y	Anna
Smith	X	Anna
Smith	Y	John

(b) EMP_PROJECTS

ENAME	PNAME
Smith	X
Smith	Y

EMP_DEPENDENTS

ENAME	DNAME
Smith	John
Smith	Anna

(c) SUPPLY

SNAME	PARTNAME	PROJNAME
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

(d) R1

SNAME	PARTNAME
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R2

SNAME	PROJNAME
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R3

PARTNAME	PROJNAME
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

Figure 15.4 Fourth and fifth normal form. (a) The EMP relation with two MVDS: ENAME $\twoheadrightarrow$ PNAME and ENAME $\twoheadrightarrow$ DNAME. (b) Decomposing EMP into two relations in 4NF. (c) The relation SUPPLY with no MVDS satisfies 4NF but does not satisfy 5NF if the JD(R1, R2, R3) holds. (d) Decomposing SUPPLY into three 5NF relations.

7.4 Join Dependencies and 5 NF

A **join dependency** (**JD**), denoted by JD$\{R_1, R_2, \ldots, R_n\}$, specified on relation schema *R*, specifies a constraint on the states *r* of *R*. The constraint states that every legal state *r* of *R* should have a lossless join decomposition into $R_1, R_2, \ldots, R_n$; that is, for every such *r* we have
$* (_{R1}(r), (_{R2}(r) \ldots (_{Rn}(r)) = r$

Lossless-join property refers to when we decompose a relation into two relations - we can rejoin the resulting relations to produce the original relation. However, sometimes there is the requirement to decompose a relation into more than two relations. Although rare, these cases are managed by join dependency and 5NF.

5NF (or project-join normal form (PJNF))

A relation that has no join dependency.

100

7.5 Other dependencies:

7.5.1 Template Dependencies

The idea behind template dependencies is to specify a template—or example—that defines each constraint or dependency. There are two types of templates: tuple-generating templates and constraint-generating templates. A template consists of a number of hypothesis tuples that are meant to show an example of the tuples that may appear in one or more relations. The other part of the template is the template conclusion. For tuple-generating templates, the conclusion is a set of tuples that must also exist in the relations if the hypothesis tuples are there. For constraint-generating templates, the template conclusion is a condition that must hold on the hypothesis tuples.

7.5.2 Domain Key Normal Form

The idea behind **domain-key normal form (DKNF)** is to specify (theoretically, at least) the "ultimate normal form" that takes into account all possible types of dependencies and constraints.

A relation is said to be in **DKNF** if all constraints and dependencies that should hold on the relation can be enforced simply by enforcing the domain constraints and key constraints on the relation.

However, because of the difficulty of including complex constraints in a DKNF relation, its practical utility is limited, since it may be quite difficult to specify general integrity constraints. 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 f orm: "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 countr y of manufacture is Japan." There is no simplified way to represent such constraints short of writing a procedure (or general assertions) to test them.

Questions

Questions

1. Explain
   i. Inclusion dependency
   ii. ii) Domain Key Normal Form
2. Explain multivolume dependency and fourth normal form, with an example
3. Explain lossless join property
4. what are the ACID Properties? Explain any One?
5. What is Serializibility?How can seriaizability?Justify your answer?

102

UNIT 8

Data base design 2

Subject Code : **10CS54** IA Marks : 25 No. of Lecture Hours/Week : 04
Exam Hours : 03 Total No. of Lecture Hours : 52 Exam Marks : 100

Transaction Processing Concepts

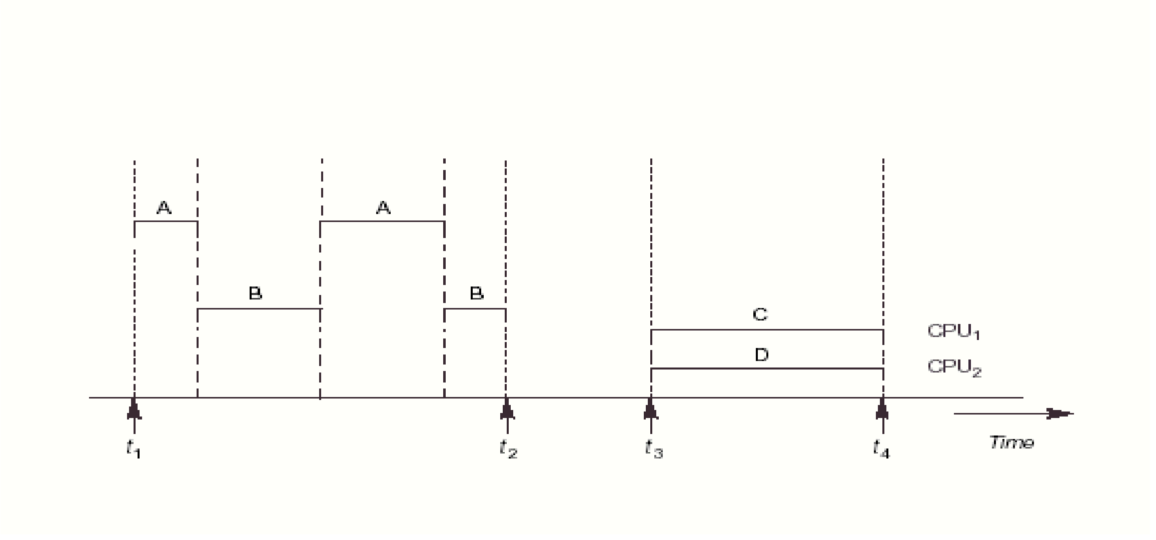
UNIT 8 Transaction Processing Concepts

8.1 Introduction to Transaction Processing

Single-User Versus Multiuser Systems

A DBMS is **single-user** if at most one user at a time can use the system, and it is **multiuser** if many users can use the system—and hence access the database—concurrently.
Most DBMS are multiuser (e.g., airline reservation system).
Multiprogramming operating systems allow the computer to execute multiple programs (or processes) at the same time (having one CPU, concurrent execution of processes is actually interleaved).
If the computer has multiple hardware processors (CPUs), *parallel processing* of multiple processes is possible.

Figure 19.1 Interleaved processing versus parallel processing of concurrent transactions.



8.2 Transactions, Read and Write Operations

A *transaction* is a logical unit of database processing that includes one or more database access operations (e.g., insertion, deletion, modification, or retrieval operations). The database operations that form a transaction can either be embedded within an application program or they can be specified interactively via a high-level query language such as SQL. One way of specifying the transaction boundaries is by specifying explicit **begin transaction** and **end transaction** statements in an application program; in this case, all database access operations between the two are considered as forming one transaction. A single application program may contain more than one transaction if it contains several transaction boundaries. If the database operations in a transaction do not update the database but only retrieve data, the transaction is called a **read-only transaction.**
Read-only transaction - do not changes the state of a database, only retrieves data.
The basic database access operations that a transaction can include are as follows:

104

- *read_item(X)*: reads a database item *X* into a program variable *X*.
- *write_item(X)*: writes the value of program variable *X* into the database item named *X*.

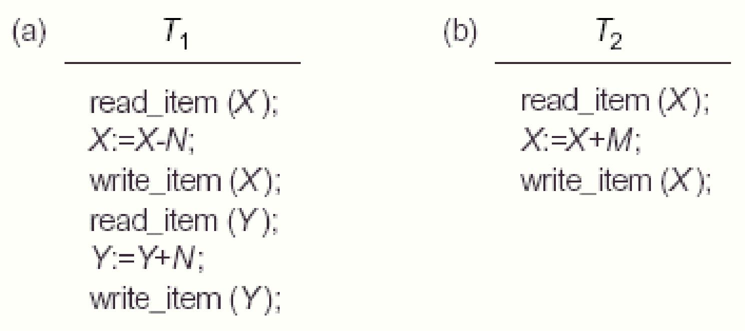
Executing a read_item(*X*) command includes the following steps:

3. Find the address of the disk block that contains item *X*.
4. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
5. Copy item *X* from the buffer to the program variable named *X*.

Executing a write_item(*X*) command includes the following steps:

6. Find the address of the disk block that contains item *X*.
7. Copy that disk block into a buffer in main memory (if that disk block is not already in some main memory buffer).
8. Copy item *X* from the program variable named *X* into its correct location in the buffer.
9. Store the updated block from the buffer back to disk (either immediately or at some later point in time).

Figure 19.2 Two sample transactions. (a) Transaction T_1. (b) Transaction T_2.



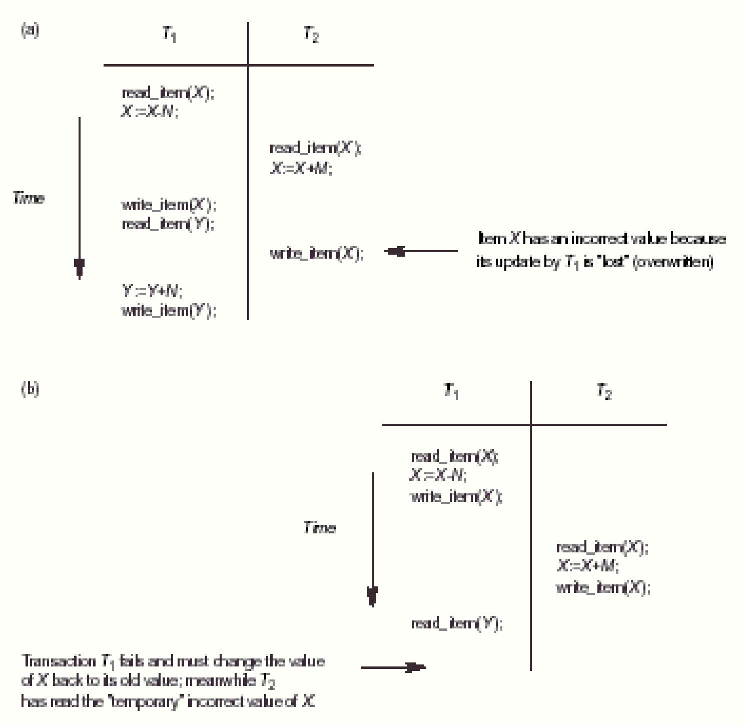
(a) T_1	(b) T_2
read_item (*X*);	read_item (*X*);
X:=*X*-*N*;	*X*:=*X*+*M*;
write_item (*X*);	write_item (*X*);
read_item (*Y*);	
Y:=*Y*+*N*;	
write_item (*Y*);	

8.3 Why Concurrency Control Is Needed

The Lost Update Problem.

This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect. Suppose that transactions T1 and T2 are submitted at approximately the same time, and suppose that their operations are interleaved then the final value of item *X* is incorrect, because T2 reads the value of *X before* T1 changes it in the database, and hence the updated value resulting from T1 is lost. For example, if *X* = 80 at the start (originally there were 80 reservations on the flight), *N* = 5 (T1 transfers 5 seat reservations from the flight corresponding to *X* to the flight corresponding to *Y*), and *M* = 4 (T2 reserves 4 seats on *X*), the final result should be *X* = 79; but in the interleaving of operations, it is *X* = 84 because the update in T1 that removed the five seats from *X* was *lost.*

Figure 19.3 Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem.


(a)
T_1
T_2
read_item(X);
X:=X-N;
read_item(X);
X:=X+M;
Time
write_item(X);
read_item(Y);
write_item(X);
Item X has an incorrect value because its update by T_1 is "lost" (overwritten)
Y:=Y+N;
write_item(Y);
(b)
T_1
T_2
read_item(X);
X:=X-N;
write_item(X);
Time
read_item(X);
X:=X+M;
write_item(X);
read_item(Y);
Transaction T_1 fails and must change the value of X back to its old value; meanwhile T_2 has read the "temporary" incorrect value of X

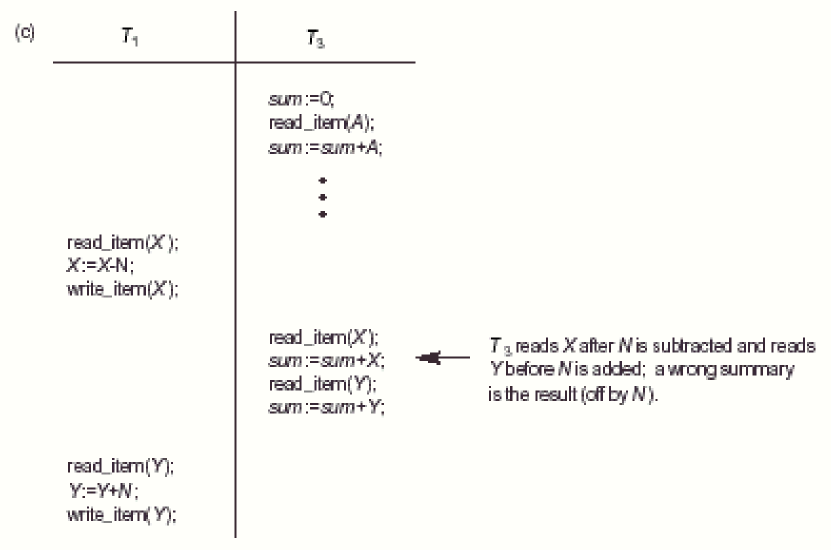
The Temporary Update (or Dirty Read) Problem.

This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value. Figure 19.03(b) shows an example where T1 updates item *X* and then fails before completion, so the system must change *X* back to its original value. Before it can do so, however, transaction T2 reads the "temporary" value of *X*,which will not be recorded permanently in the database because of the failure of T1. The value of item *X* that is read by T2 is called *dirty data,* because it has been

106

created by a transaction that has not completed and committed yet; hence, this problem is also known as the *dirty read problem.*

Figure 19.3 Some problems that occur when concurrent execution is uncontrolled. (c) The incorrect summary problem.



The Incorrect Summary Problem.

If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated. For example, suppose that a transaction T3 is calculating the total number of reservations on all the flights; meanwhile, transaction T1 is executing. If the interleaving of operations shown in Figure 19.03(c) occurs, the result of T3 will be off by an amount *N* because T3 reads the value of *X after N* seats have been subtracted from it but reads the value of *Y before* those *N* seats have been added to it.

Another problem that may occur is called **unrepeatable read,** where a transaction *T* reads an item twice and the item is changed by another transaction *T'* between the two reads. Hence, *T* receives *different values* for its two reads of the same item. This may occur, for example, if during an airline reservation transaction, a customer is inquiring 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.

8.4 Why Recovery Is Needed

Whenever a transaction is submitted to a DBMS for execution, the system is 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) the transaction has no effect whatsoever on the database or on any other tr ansactions. The DBMS must not permit some operations of a transaction *T* to be applied to the database while other operations of *T* are not. This may happen if a transaction **f ails** after executing some of its operations but before executing all of them.

Types of Failures

Failures are generally classified as transaction, system, and media failures. There are several possible reasons for a transaction to fail in the middle of execution:

1. *A computer failure (system crash):* A hardware, software, or network error occurs in the computer system during transaction execution. Hardware crashes are usually media failures—for example, main memory failure.
2. *A transaction or system error* Some operation in the transaction may cause it to fail, such as integer overflow or division by zero. Transaction failure may also occur because of erroneous parameter values or because of a logical programming error . In addition, the user may interrupt the transaction during its execution.
3. *Local errors or exception conditions detected by the transaction:* During transaction execution, certain conditions may occur that necessitate cancellation of the transaction. For example, data for the transaction may not be found. Notice that an exception condition , such as insufficient account balance in a banking database, may cause a transaction, such as a fund withdrawal, to be canceled. This exception should be programmed in the transaction itself, and hence would not be considered a failure.
4. *Concurrency control enforcement:* The concurrency control method (see Chapter 20) may decide to abort the transaction, to be restarted later, because it violates serializability (see Section 19.5) or because several transactions are in a state of deadlock.
5. *Disk failure:* Some disk blocks may lose their data because of a read or write malfunction or because of a disk read/write head crash. This may happen during a read or a write operation of the transaction.
6. *Physical problems and catastrophes:* This refers to an endless list of problems that includes power or air-conditioning failure, fire, theft, sabotage, overwriting disks or tapes by mistake, and mounting of a wrong tape by the operator.

Failures of types 1, 2, 3, and 4 are more common than those of types 5 or 6. Whenever a failure of type 1 through 4 occurs, the system must keep sufficient information to recover from the failure. Disk failure or other catastrophic failures of type 5 or 6 do not happen frequently; if they do occur, recovery is a major task.

The concept of transaction is fundamental to many techniques for concurrency control and recovery from failur es.

8.5 Transaction and System Concepts

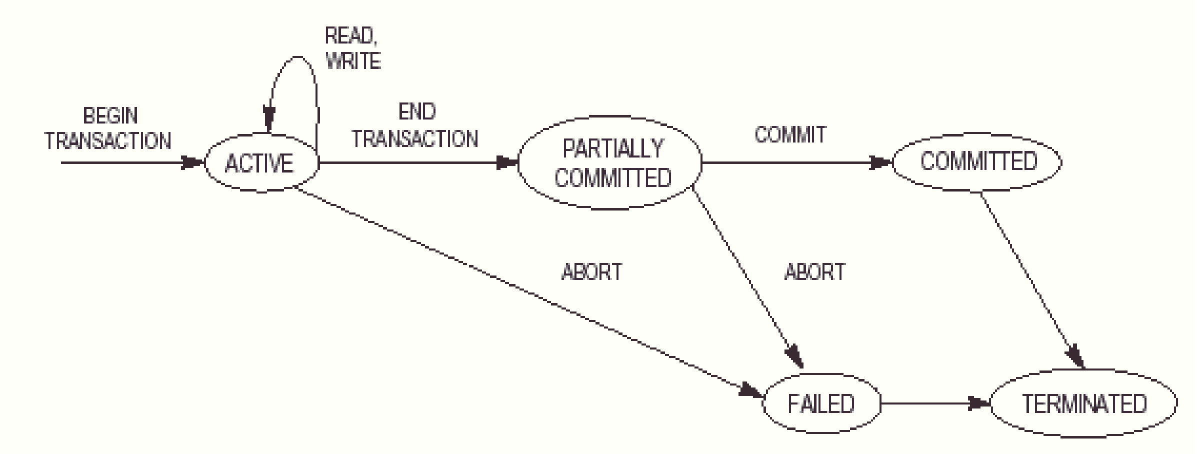
Transaction States and Additional Operations

A transaction is an atomic unit of work that is either completed in its entirety or not done at all. For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts (see below). Hence, the recovery manager keeps track of the following operations:

- BEGIN_TRANSACTION: This marks the beginning of transaction execution.
- READ or WRITE: These specify read or write operations on the database items that are executed as part of a transaction.
- END_TRANSACTION: This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates serializability (see Section 19.5) or for some other reason.
- COMMIT_TRANSACTION: This 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 (or ABORT) This 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 19.04 shows a state transition diagram that describes how a transaction moves through its execution states. A transaction goes into an **active state** immediately after it starts execution, where it can issue READ and WRITE operations. When the transaction ends, it moves to the **partially committed state.** At this point, some recovery protocols need to ensure that a system failure will not result in an inability to record the changes of the transaction permanently (usually by recording changes in the system log). Once this check is successful, the transaction is said to have reached its commit point and enters the **committed state.** Once a transaction is committed, it has concluded its execution successfully and all its changes must be recorded permanently in the database.

Figure 19.4 State transition diagram illustrating the states for transaction execution.



8.6 The System Log

To be able to recover from failures that affect transactions, the system maintains a *log* to keep track of all transactions that affect the values of database items.
Log records consists of the following information (*T* refers to a unique *transaction_id*):

1. [start_transaction,*T*]: Indicates that transaction *T* has started execution.
2. [write_item,*T*,*X*,*old_value*,*new_value*]: Indicates that transaction *T* has changed the value of database item *X* from *old_value* to *new_value*.
3. [read_item,*T*,*X*]: Indicates that transaction *T* has read the value of database item *X*.
4. [commit,*T*]: Indicates that transaction *T* has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
5. [abort,*T*]: Indicates that transaction *T* has been aborted.

8.7 Desirable Properties of Transactions

Transactions should posses the following (ACID) properties:

Transactions should possess several properties. These are often called the **ACID properties,** and they should be enforced by the concurrency control and recover y methods of the DBMS The following are the ACID properties:

1. **Atomicity:** A transaction is an atomic unit of processing; it is either performed in its entirety or not performed at all.

2. **Consistency preservation:** A transaction is consistency preserving if its complete execution take(s) the database from one consistent state to another.
3. **Isolation:** A transaction should appear as though it is being executed in isolation from other transactions. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.
4. **Durability or permanency:** The changes applied to the database by a committed transaction must persist in the database. These changes must not 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 ensure atomicity. If a transaction fails to complete for some reason, such as a system crash in the midst of transaction execution, the recovery technique must undo any effects of the transaction on the database.

8.8 Schedules and Recoverability

A **schedule** (or **history**) *S* of *n* transactions T1, T2, ..., Tn is an ordering of the operations of the transactions subject to the constraint that, for each transaction Ti that participates in *S*, the operations of Ti in *S* must appear in the same or der in which they occur in Ti. Note, however, that operations from other transactions Tj can be interleaved with the operations of Ti in *S*. For now, consider the order of operations in *S* to be a *total ordering*, although it is possible theoretically to deal with schedules whose operations form *partial orders*.

$$S_a: r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); w_1(Y);$$

Similarly, the schedule f or Figur e 19.03(b), which we call Sb, can be written as follows, if we assume that transaction T1 aborted after its read_item(*Y*) operation:

$$S_b: r_1(X); w_1(X); r_2(X); w_2(X); r_1(Y); a_1;$$

Two operations in a schedule are said to **conflict** if they satisfy all three of the following conditions:

1. they belong to different transactions;
2. they access the same item *X;* and
3. at least one of the operations is a write_item(*X*).

For example, in schedule , S_ae operations co$r_1(X)$ and $w_2(X)$' operations

$r_2(X)$ and $w_1(X)$,), and the operations w1(*X*) and w2(*X*). However, the operations r1(*X*) and r2(*X*) do not conflict, since they are both read operations; the operations w2(*X*) and w1(*Y*) do not conflict, because they operate on distinct data items *X* and *Y;* and the operations r1(*X*) and w1(*X*) do not conflict, because they belong to the same transaction.

A schedule *S* of *n* transactions T1, T2, ..., Tn, is said to be a **complete schedule** if the following conditions hold:

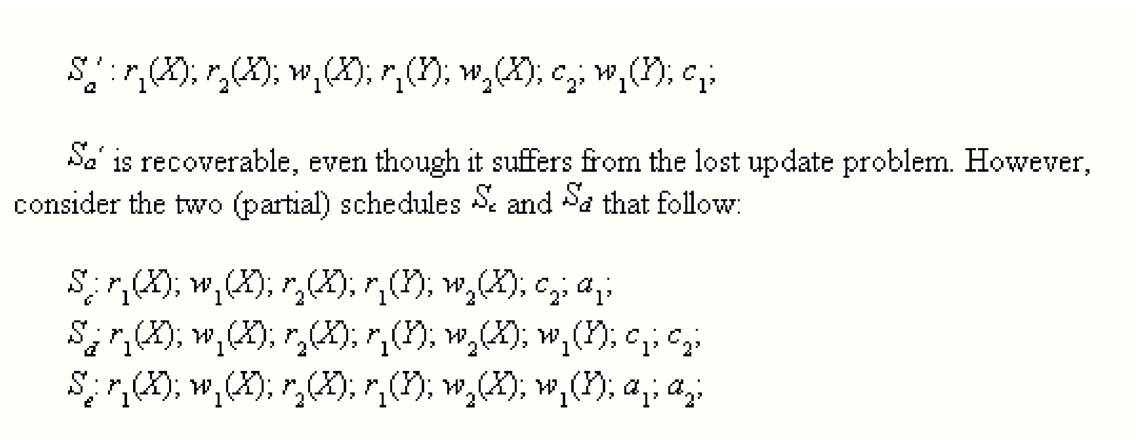
1. The operations in*S* are exactly those operations in T1, T2, ..., Tn, including a commit or abort operation as the last operation for each transaction in the schedule.
2. For any pair of operations from the same transaction Ti, their order of appearance in*S* is the same as their order of appearance in Ti.
3. For any two conflicting operations, one of the two must occur before the other in the schedule.

8.10 Characterizing Schedules Based on Recoverability

once a transaction *T* is committed, it should *never* be necessar y to roll back *T.* The schedules that theoretically meet this criterion are called *recoverable schedules* and those that do not are called **nonrecoverable,** and hence should not be permitted.

A schedule *S* is recoverable if no transaction *T* in *S* commits until all transactions *T'* that have written an item that *T* reads have committed. A transaction *T* **reads** from transaction *T*in a schedule *S* if some item *X* is first written by [redacted] and later read by *T.* In addition, [redacted] should not have been aborted before *T* reads item *X*, and there should be no transactions that write *X* after [redacted] writes it and before *T* reads it (unless those transactions, if any, have aborted before *T* reads *X*).

Consider the schedule [redacted] given below, which is the same as schedule exceS_a that two commit operations have been added to : S_a



S_a' : $r_1(X); r_2(X); w_1(X); r_1(Y); w_2(X); c_2; w_1(Y); c_1;$

S_a' is recoverable, even though it suffers from the lost update problem. However, consider the two (partial) schedules S_c and S_d that follow:

S_c: $r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); c_2; a_1;$
S_d: $r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); w_1(Y); c_1; c_2;$
S_e: $r_1(X); w_1(X); r_2(X); r_1(Y); w_2(X); w_1(Y); a_1; a_2;$

112

[redacted] is not recover able, because T2 reads item *X* from T1, and then T2 commits before T1 commits. If T1 aborts after the c2 operation in , th[redacted]the value of *X* that T2 read is no longer valid and T2 must be aborted *after* it had be[redacted] committed, leading to a schedule that is not recoverable. For the schedule to be recoverable, the c2 operation in must [redacted]postponed until after T1 commits. If T1 aborts instead of committing, then T2 should also abort as shown in Se, because the value of *X* it read is no longer valid.

In a recoverable schedule, no committed transaction ever needs to be rolled back. However, it is possible for a phenomenon known as **cascading rollback** (or **cascading abort**) to occur, where an *uncommitted* transaction has to be rolled back because it read an item from a transaction that failed.

Serializability of Schedules

If no interleaving of operations is permitted, there are only two possible arrangement for transactions T1 and T2.

1. Execute all the operations of T1 (in sequence) followed by all the operations of T2 (in sequence).
2. Execute all the operations of T2 (in sequence) followed by all the operations of T1

A schedule *S* is *serial* if, for every transaction *T* all the operations of *T* are executed consecutively in the schedule.

A schedule *S* of *n* transactions is *serializable* if it is equivalent to some serial schedule of the same *n* transactions.

113

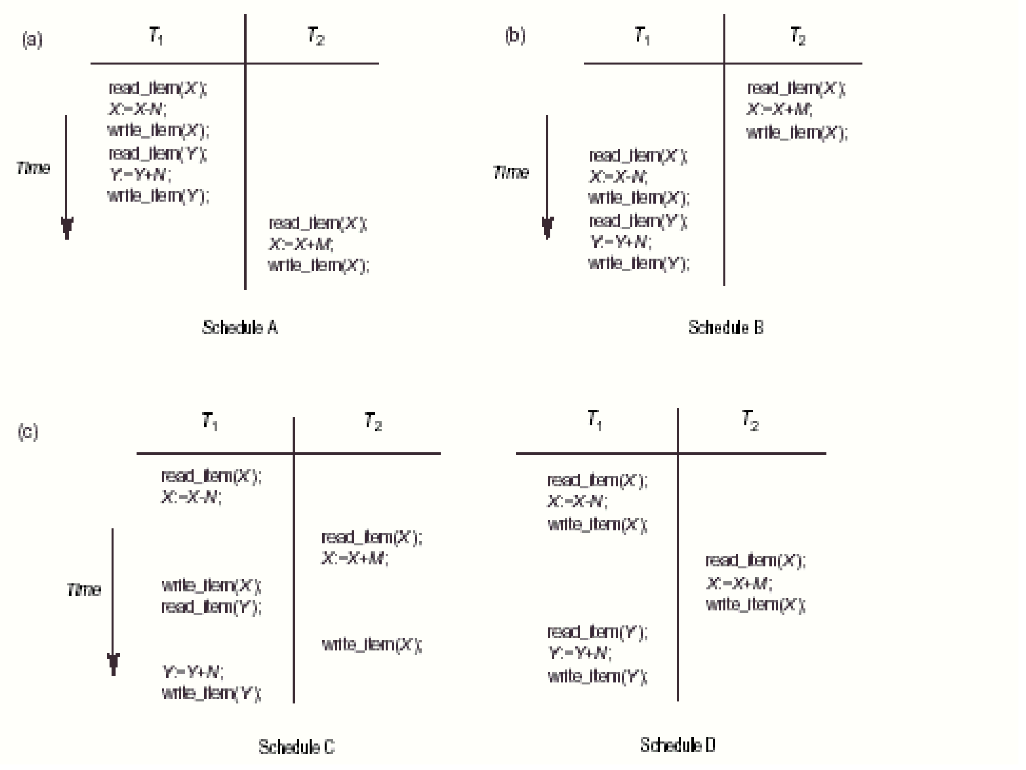


Figure 19.5 Examples of serial and nonserial schedules involving transactions T_1 and T_2. (a) Serial schedule A: T_1 followed by T_2. (b) Serial schedule B: T_2 followed by T_1. (c) Two nonserial schedules C and D with interleaving of operations.

8.11 Transaction Support in SQL

An SQL transaction is a logical unit of work (i.e., a single SQL statement).
The *access mode* can be specified as *READ ONLY* or *READ WRITE*. The default is *READ WRITE*, which allows update, insert, delete, and create commands to be executed.
The *diagnostic area size* option specifies an integer value *n*, indicating the number of conditions that can be held simultaneously in the diagnostic area.
The *isolation level* option is specified using the statement *ISOLATION LEVEL*
the default isolation level is *SERIALIZABLE*.

A sample SQL transaction might look like the following:

```
EXEC SQL WHENEVER SQLERROR GOTO UNDO;
EXEC SQL SET TRANSACTION
READ WRITE
DIAGNOSTICS SIZE 5
ISOLATION LEVEL SERIALIZABLE;

EXEC SQL INSERT INTO EMPLOYEE (FNAME, LNAME, SSN, DNO, SALARY)
VALUES ('Jabbar', 'Ahmad', '998877665', 2, 44 000);
EXEC SQL UPDATE EMPLOYEE
SET SALARY = SALARY * 1.1 WHERE DNO = 2;
EXEC SQL COMMIT;
```

114

```
GOTO THE_END;
UNDO:   EXEC SQL ROLLBACK;
THE_END:  . . . ;
```

115

Questions

1. Write a short Notes on

    i. 2PL Lock
    ii. Two-P Deadlock

2. Three phase Locking Techniques: Essential components
3. Explain properties of a transaction with state transition diagram.
4. What is a schedule? Explain with example serial, non serial and conflict serializable schedules.
5. Write short notes on
    1. Write ahead log protocol
    2. Time stamp Ordering
    3. Two phase locking protocol
6. Explain the problems that can occur whaen concurrent transaction are executed give examples

Subject: Database Management System

Prepared by: Drakshaveni G
Assistant Professor
Dept.of MCA
BMSIT&M

Module -5

-

Transaction Management

-

A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

Let’s take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

A’s Account

```
Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)
```

B’s Account

```
Open_Account(B)
Old_Balance = B.balance
New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)
```

ACID Properties

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain **A**tomicity, **C**onsistency, **I**solation, and **D**urability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

Atomicity − This property states that a transaction must be treated as an atomic unit, thatis, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

Consistency − The database must remain in a consistent state after any transaction. Notransaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

Durability − The database should be durable enough to hold all its latest updates even ifthe system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

Isolation − In a database system where more than one transaction are being executedsimultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

Serializability

When multiple transactions are being executed by the operating system in a multiprogramming environment, there are possibilities that instructions of one transactions are interleaved with some other transaction.

Schedule − A chronological execution sequence of a transaction is called a schedule. Aschedule can have many transactions in it, each comprising of a number of instructions/tasks.

Serial Schedule − It is a schedule in which transactions are aligned in such a way that onetransaction is executed first. When the first transaction completes its cycle, then the next transaction is executed. Transactions are ordered one after the other. This type of schedule is called a serial schedule, as transactions are executed in a serial manner.

In a multi-transaction environment, serial schedules are considered as a benchmark. The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed in a random fashion. This execution does no harm if two transactions are mutually independent and working on different segments of data; but in case these two transactions are working on the same data, then the results may vary. This ever-varying result may bring the database to an inconsistent state.

To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either serializable or have some equivalence relation among them.

Equivalence Schedules

An equivalence schedule can be of the following types −

Result Equivalence

If two schedules produce the same result after execution, they are said to be result equivalent. They may yield the same result for some value and different results for another set of values. That's why this equivalence is not generally considered significant.

View Equivalence

Two schedules would be view equivalence if the transactions in both the schedules perform similar actions in a similar manner.

For example −

If T reads the initial data in S1, then it also reads the initial data in S2.

If T reads the value written by J in S1, then it also reads the value written by J in S2.

If T performs the final write on the data value in S1, then it also performs the final write on the data value in S2.

Conflict Equivalence

Two schedules would be conflicting if they have the following properties −

Both belong to separate transactions.

Both accesses the same data item.

At least one of them is "write" operation.

Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if −

Both the schedules contain the same set of Transactions.

The order of conflicting pairs of operation is maintained in both the schedules.

Note − View equivalent schedules are view serializable and conflict equivalent schedules areconflict serializable. All conflict serializable schedules are view serializable too.

States of Transactions

A transaction in a database can be in one of the following states −
Active − In this state, the transaction is being executed. This is the initial state of everytransaction.

Partially Committed − When a transaction executes its final operation, it is said to be in apartially committed state.

Failed − A transaction is said to be in a failed state if any of the checks made by thedatabase recovery system fails. A failed transaction can no longer proceed further.

Aborted − If any of the checks fails and the transaction has reached a failed state, then therecovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts −

Re-start the transaction

Kill the transaction

Committed − If a transaction executes all its operations successfully, it is said to becommitted. All its effects are now permanently established on the database system.

CONCURRENCY CONTROL

In a multiprogramming environment where multiple transactions can be executed simultaneously, it is highly important to control the concurrency of transactions. We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions. Concurrency control protocols can be broadly divided into two categories −

- Lock based protocols
- Time stamp based protocols

Lock-based Protocols

Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it. Locks are of two kinds −

- **Binary Locks** − A lock on a data item can be in two states; it is either locked or unlocked.
- **Shared/exclusive** − This type of locking mechanism differentiates the locks based on theiruses. If a lock is acquired on a data item to perform a write operation, it is an exclusive lock. Allowing more than one transaction to write on the same data item would lead the database into an inconsistent state. Read locks are shared because no data value is being changed.

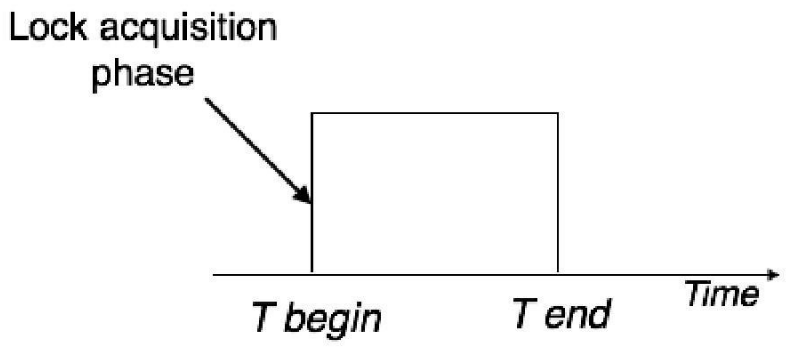
There are four types of lock protocols available −

Simplistic Lock Protocol

Simplistic lock-based protocols allow transactions to obtain a lock on every object before a 'write' operation is performed. Transactions may unlock the data item after completing the ‘write’ operation.

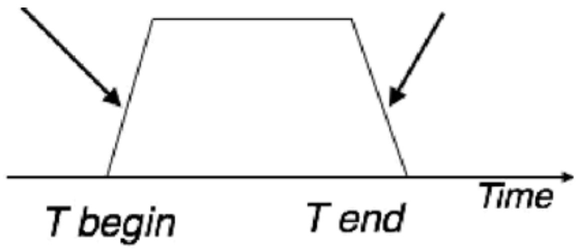
Pre-claiming Lock Protocol

Pre-claiming protocols evaluate their operations and create a list of data items on which they need locks. Before initiating an execution, the transaction requests the system for all the locks it needs beforehand. If all the locks are granted, the transaction executes and releases all the locks when all its operations are over. If all the locks are not granted, the transaction rolls back and waits until all the locks are granted.



Two-Phase Locking 2PL

This locking protocol divides the execution phase of a transaction into three parts. In the first part, when the transaction starts executing, it seeks permission for the locks it requires. The second part is where the transaction acquires all the locks. As soon as the transaction releases its first lock, the third phase starts. In this phase, the transaction cannot demand any new locks; it only releases the acquired locks.



Two-phase locking has two phases, one is **growing**, where all the locks are being acquired by the transaction; and the second phase is shrinking, where the locks held by the transaction are being released.

To claim an exclusive *write* lock, a transaction must first acquire a shared *read* lock and then upgrade it to an exclusive lock.

Strict Two-Phase Locking

The first phase of Strict-2PL is same as 2PL. After acquiring all the locks in the first phase, the transaction continues to execute normally. But in contrast to 2PL, Strict-2PL does not release a lock after using it. Strict-2PL holds all the locks until the commit point and releases all the locks at a time.

Strict-2PL does not have cascading abort as 2PL does.

Timestamp-based Protocols

The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Every transaction has a timestamp associated with it, and the ordering is determined by the age of the transaction. A transaction created at 0002 clock time would be older than all other transactions that come after it. For example, any transaction 'y' entering the system at 0004 is two seconds younger and the priority would be given to the older one.

In addition, every data item is given the latest read and write-timestamp. This lets the system know when the last 'read and write' operation was performed on the data item.

Timestamp Ordering Protocol

The timestamp -ordering protocol ensures serializability among transactions in their conflicting read and write operations. This is the responsibility of the protocol system that the conflicting pair of tasks should be executed according to the timestamp values of the transactions.

The timestamp of transaction T_i is denoted as TS(T_i).

Read time-stamp of data-item X is denoted by R-timestamp*X*.

Write time-stamp of data-item X is denoted by W-timestamp*X*.

Timestamp ordering protocol works as follows −

If a transaction Ti issues a read*X* operation −

If TS*Ti*< W-timestamp*X*

Operation rejected.

If TS*Ti*>= W-timestamp*X*

Operation executed.

All data-item timestamps updated.

If a transaction Ti issues a write*X* operation −

If TS*Ti*< R-timestamp*X*

Operation rejected.

If TS*Ti*< W-timestamp*X*

Operation rejected and Ti rolled back.

Otherwise, operation executed.

Thomas' Write Rule

This rule states if TS*Ti*< W-timestamp*X*, then the operation is rejected and T_i is rolled back.

Time-stamp ordering rules can be modified to make the schedule view serializable.

Instead of making T_i rolled back, the 'write' operation itself is ignored.

DBMS-DEADLOCK

Inamulti-processsystem,deadlockisanunwantedsituationthatarisesinasharedresourceenvironment,whereaprocessindefinitelywaitsforaresourcethatisheldbyanotherprocess.

Forexample,assumeasetoftransactions{T_0,T_1,T_2,...,T_n}.T_0needsaresourceXtocompleteitstask.ResourceXisheldbyT_1,a ndT$_1$iswaitingforaresourceY,whichisheldbyT_2.T_2iswaitingforresourceZ,whichisheldbyT_0.Thus,alltheprocesseswaitf oreachothertoreleaseresources.Inthissituation,noneoftheprocessescanfinishtheirtask.Thissituationisknownasadeadlock.

Deadlocksarenothealthyforasystem.Incaseasystemisstuckinadeadlock,thetransactionsinvolvedinthedeadlockareeitherroll edbackorrestarted.

DeadeockPrevention

Topreventanydeadlocksituationinthesystem,theDBMSaggressivelyinspectsalltheoperations,wheretransactionsareaboutto execute.TheDBMSinspectstheoperationsandanalyzesiftheycancreateadeadlocksituation.Ifitfindsthatadeadlocksituation mightoccur,thenthattransactionisneverallowedtobeexecuted.

Therearedeadlockpreventionschemesthatusetimestamporderingmechanismoftransactionsinordertopredetermineadeadlo cksituation.

Wait-DieScheme

Inthisscheme,ifatransactionrequeststolockaresource*dataitem*,whichisalreadyheldwithaconflictinglockbyanothertran saction,thenoneofthetwopossibilitiesmayoccur−

- IfTS(T_i)<TS(T_j)−thatisT_i,whichisrequestingaconflictinglock,isolderthanT_j−then$T_i$isallowedtowaituntilthedata-itemisavailable.
- IfTS(T_i)>TS(t_j)−thatis$T_i$isyoungerthanT_j−then$T_i$dies.$T_i$isrestartedlaterwitharandomdelaybutwiththesametim estamp.

Thisschemeallowstheoldertransactiontowaitbutkillstheyoungerone.

Wound-WaitScheme

Inthisscheme,ifatransactionrequeststolockaresource*dataitem*,whichisalreadyheldwithconflictinglockbysomeanothertrans action,oneofthetwopossibilitiesmayoccur−

- IfTS(T_i)<TS(T_j),then$T_i$forces$T_j$toberolledback−thatis$T_i$woundsT_j.$T_j$isrestartedlaterwitharandomdelaybutwiththesametimestamp.

- IfTS(T_i)>TS(T_j),then$T_i$isforcedtowaituntiltheresourceisavailable.

Thisscheme,allowstheyoungertransactiontowait;butwhenanoldertransactionrequestsanitemheldbyayoungerone,theoldertransactionforcestheyoungeronetoabortandreleasetheitem.

Inboththecases,thetransactionthatentersthesystematalaterstageisaborted.

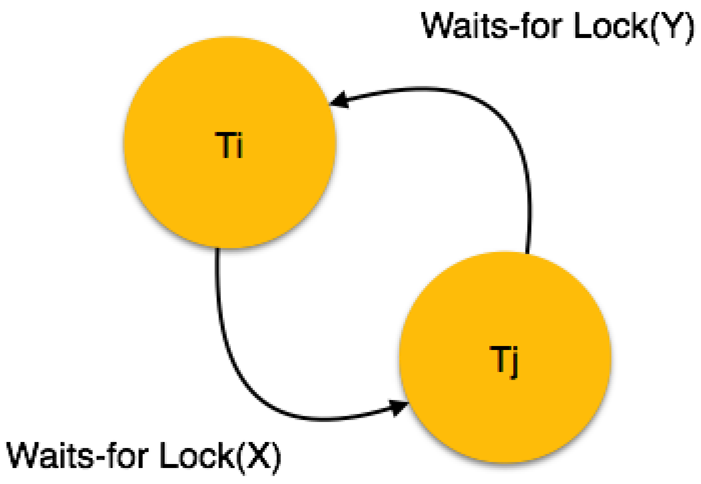
DeadeockAvoidance

Abortingatransactionisnotalwaysapracticalapproach.Instead,deadlockavoidancemechanismscanbeusedtodetectanydeadlocksituationinadvance.Methodslike"wait-forgraph"areavailablebuttheyaresuitableforonlythosesystemswheretransactionsarelightweighthavingfewerinstancesofresource.Inabulkysystem,deadlockpreventiontechniquesmayworkwell.

Wait-forGraph

Thisisasimplemethodavailabletotrackifanydeadlocksituationmayarise.Foreachtransactionenteringintothesystem,anodeiscreated.WhenatransactionT¡requestsforalockonanitem,sayX,whichisheldbysomeothertransactionTj,adirectededgeiscreatedfromT¡toTj.IfTjreleasesitemX,theedgebetweenthemisdroppedandT¡locksthedataitem.

Thesystemmaintainsthiswait-forgraphforeverytransactionwaitingforsomedataitemsheldbyothers.Thesystemkeepscheckingifthere'sanycycleinthegraph.



Here,wecanuseanyofthetwofollowingapproaches−

- First,donotallowanyrequestforanitem,whichisalreadylockedbyanothertransaction.Thisisnotalwaysfeasibleandmaycausestarvation,whereatransactionindefinitelywaitsforadataitemandcanneveracquireit.
- Thesecondoptionistorollbackoneofthetransactions.Itisnotalwaysfeasibletorollbacktheyoungertransaction,asitmaybeimportantthantheolderone.Withthehelpofsomerelativealgorithm,atransactionischosen,whichistobeaborted.Thistransactionisknownasthevictimandtheprocessisknownasvictimseeection.