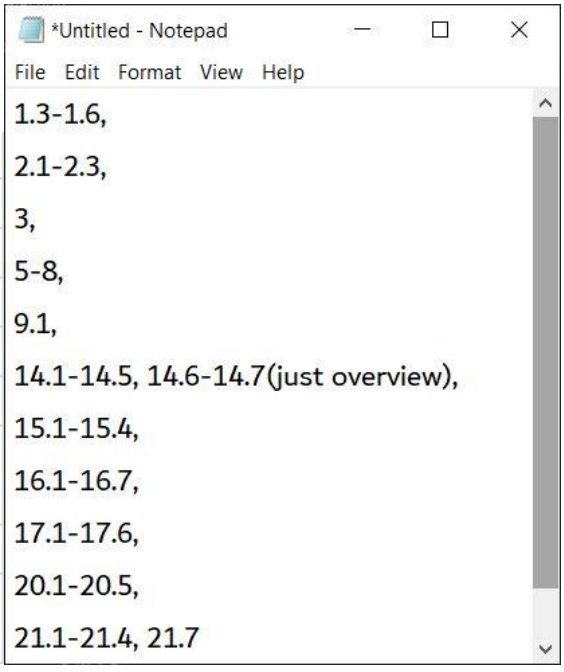
DBMS

GATE - 2023

Difficulty	Topic Name
moderate	ER-Diagrams
easy	Relational Schema- Key Constraints
moderate	Relational Algebra
moderate	Tuple Relational Calculus
moderate	SQL
easy	Functional Dependencies and Normalization
very easy	Transactions and Concurrency control
easy	Indexing and File Structures (B-Trees and B+ Trees)

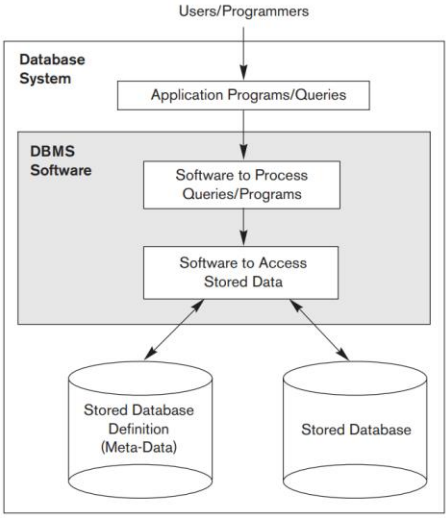


NOVEMBER 16, 2021

QUANTUM CITY
Nowhere

1. INTRODUCTION TO DATABASE

1) A database is a collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning.
2) A database represents some aspect of the real world, sometimes called the miniworld or the universe of discourse (UoD). Changes to the miniworld are reflected in the database.
3) A database management system (DBMS) is a computerized system that enables users to create and maintain a database.
   - **Defining** a database involves specifying the data types, structures, and constraints of the data to be stored in the database.
   - The database definition or descriptive information is also stored by the DBMS in the form of a database catalog or dictionary; it is called **meta-data**.
   - **Constructing** the database is the process of storing the data on some storage medium that is controlled by the DBMS.
   - **Manipulating** a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data.
   - **Sharing** a database allows multiple users and programs to access the database simultaneously.



Characteristics of the Database Approach :

- In **traditional file processing**, each user defines and implements the files needed for a specific software application as part of programming the application. For example, users are interested in data about students, each user maintains separate files—and programs to manipulate these files.
- In the **database approach**, a single repository maintains data that is defined once and then accessed by various users repeatedly through queries, transactions, and application programs.

1) Self-Describing Nature of a Database System :

In traditional file processing, data definition is typically part of the application programs themselves. Hence, these programs are constrained to work with only one specific database, whose structure is declared in the application programs.

DBMS software can access diverse databases by extracting the database definitions from the catalog and using these definitions.

The information stored in the catalog is called meta-data, and it describes the structure of the primary database. It is important to note that some newer types of database systems, known as NOSQL systems, do not require meta-data. Rather the data is stored as self-describing data that includes the data item names and data values together in one structure.

2) Insulation between Programs and Data, and Data Abstraction :

In traditional file processing, the structure of data files is embedded in the application programs, so any changes to the structure of a file may require changing all programs that access that file. By contrast, DBMS access programs do not require such changes in most cases. The structure of data files is stored in the DBMS catalogue separately from the access programs. We call this property program-data independence.

User application programs can operate on the data by invoking these operations through their names and arguments, regardless of how the operations are implemented. This may be termed program-operation independence.

3) Support of Multiple Views of the Data :

A database typically has many types of users, each of whom may require a different perspective or view of the database. A view may be a subset of the database or it may contain virtual data that is derived from the database files but is not explicitly stored.

For example, one user of the database may be interested only in accessing and printing the transcript of each student.

4) Sharing of Data and Multiuser Transaction Processing :

A multiuser DBMS, as its name implies, must allow multiple users to access the database at the same time. The DBMS must include concurrency control software to ensure that several users trying to update the same data do so in a controlled manner so that the result of the updates is correct.

people whose jobs involve the day-to-day use of a large database :

1) Database Administrators : Administering these resources is the responsibility of the database administrator (DBA). The DBA is responsible for authorizing access to the database, coordinating and monitoring its use, and acquiring software and hardware resources as needed.
2) Database Designers : Database designers are responsible for identifying the data to be stored in the database and for choosing appropriate structures to represent and store this data.
3) End Users : End users are the people whose jobs require access to the database for querying, updating, and generating reports; the database primarily exists for their use.

A typical DBMS provides multiple facilities to access a database. Naive end users need to learn very little about the facilities provided by the DBMS; they simply have to understand the user interfaces of the mobile apps or standard transactions designed and implemented for their use. Casual users learn only a few facilities that they may use repeatedly. Sophisticated users try to learn most of the DBMS facilities in order to achieve their complex requirements. Standalone users typically become very proficient in using a specific software package.

4) System Analysts and Application Programmers (Software Engineers) : System analysts determine the requirements of end users, especially naive and parametric end users, and develop specifications for standard canned transactions that meet these requirements. Application programmers implement these specifications as programs; then they test, debug, document, and maintain these canned transactions.

In addition to those who design, use, and administer a database, others are associated with the design, development, and operation of the DBMS software and system environment. These persons are typically not interested in the database content itself.

1) DBMS system designers and implementers design and implement the DBMS modules and interfaces as a software package.
2) Tool developers design and implement tools—the software packages that facilitate database modelling and design, database system design, and improved performance. Tools are optional packages that are often purchased separately.
3) Operators and maintenance personnel (system administration personnel) are responsible for the actual running and maintenance of the hardware and software environment for the database system.

Advantage of Using the DBMS Approach :

1) Controlling Redundancy : This redundancy in storing the same data multiple times leads to several problems. First, there is the need to perform a single logical update—such as entering data on a new student—multiple times: once for each file where student data is recorded. This leads to duplication of effort.

Second, storage space is wasted when the same data is stored repeatedly, and this problem may be serious for large databases. Third, files that represent the same data may become inconsistent.

Ideally, we should have a database design that stores each logical data item—such as a student's name or birth date—in only one place in the database. This is known as data normalization.

2) Restricting Unauthorized Access : For example, financial data such as salaries and bonuses is often considered confidential, and only authorized persons are allowed to access such data. In addition, some users may only be permitted to retrieve data, whereas others are allowed to retrieve and update.
3) Providing Persistent Storage for Program Objects : A complex object in C++ can be stored permanently in an object-oriented DBMS. Such an object is said to be persistent, since it survives the termination of program execution and can later be directly retrieved by another program. Traditional database systems often suffered from the so-called impedance mismatch problem.
4) Providing Storage Structures and Search Techniques for Efficient Query Processing : The DBMS must provide specialized data structures and search techniques to speed up disk search for the desired records.

Indexes are typically based on tree data structures or hash data structures that are suitably modified for disk search. In order to process the database records needed by a particular query, those records must be copied from disk to main memory. Therefore, the DBMS often has a buffering or caching module that maintains parts of the database in main memory buffers.

5) Providing Backup and Recovery : If the computer system fails in the middle of a complex update transaction, the recovery subsystem is responsible for making sure that the database is restored to the state it was in before the transaction started executing.
6) Providing Multiple User Interfaces : Because many types of users with varying levels of technical knowledge use a database, a DBMS should provide a variety of user interfaces.

7) Representing Complex Relationships among Data : A DBMS must have the capability to represent a variety of complex relationships among the data, to define new relationships as they arise, and to retrieve and update related data easily and efficiently.

8) Enforcing Integrity Constraints : Every section record must be related to a course record. This is known as a **referential integrity constraint**. Another type of constraint specifies uniqueness on data item values, such as every course record must have a unique value for Course_number. This is known as a key or **uniqueness constraint**.

9) Permitting Inferencing and Actions Using Rules and Triggers : In today's relational database systems, it is possible to associate triggers with tables. A trigger is a form of a rule activated by updates to the table, which results in performing some additional operations to some other tables, sending messages, and so on.

2. Database System Concepts and Architecture

In a basic client/server DBMS architecture, the system functionality is distributed between two types of modules.

1) A client module is typically designed so that it will run on a mobile device, user workstation, or personal computer (PC). Typically, application programs and user interfaces that access the database run in the client module.
2) A server module, typically handles data storage, access, search, and other functions.

Data Models, Schemas, and Instances : **Data abstraction** generally refers to the suppression of details of data organization and storage, and the highlighting of the essential features for an improved understanding of data. One of the main characteristics of the database approach is to support data abstraction so that different users can perceive data at their preferred level of detail. A **data model**—a collection of concepts that can be used to describe the structure of a database—provides the necessary means to achieve this abstraction. By structure of a database we mean the data types, relationships, and constraints that apply to the data. Most data models also include a set of **basic operations** for specifying retrievals and updates on the database.

1) **Categories of Data Models** : **High-level or conceptual data models** provide concepts that are close to the way many users perceive data, whereas **low-level or physical data models** provide concepts that describe the details of how data is stored on the computer storage media, typically magnetic disks. Concepts provided by physical data models are generally meant for computer specialists, not for end users. Between these two extremes is a class of **representational (or implementation) data models**, which provide concepts that may be easily understood by end users but that are not too far removed from the way data is organized in computer storage. Representational data models hide many details of data storage on disk but can be implemented on a computer system directly.
   A) ***Conceptual data models*** use concepts such as entities, attributes, and relationships.
      i) An **entity** represents a real-world object or concept, such as an employee or a project from the miniworld that is described in the database.
      ii) An **attribute** represents some property of interest that further describes an entity, such as the employee's name or salary.
      iii) A **relationship** among two or more entities represents an association among the entities, for example, a works-on relationship between an employee and a project.
   B) **Representational data models :** These include the widely used relational data model, as well as the so-called legacy data models—the network and hierarchical models—that have been widely used in the past.
   C) **Physical data models** describe how data is stored as files in the computer by representing information such as record formats, record orderings, and access paths.
      i) **access path** is a search structure that makes the search for particular database records efficient, such as indexing or hashing.
      ii) An **index** is an example of an access path that allows direct access to data using an index term or a keyword.
   D) self-describing data models. The data storage in systems based on these models combines the description of the data with the data values themselves. object data model as an example of a new family of higher-level implementation data models that are closer to conceptual data models.

2) **Schemas, Instances, and Database State** : The description of a database is called the **database schema**. We call each object in the schema—such as STUDENT or COURSE—a **schema construct**.
   The data in the database at a particular moment in time is called a **database state or snapshot**. It is also called the current **set of occurrences or instances** in the database.

Figure 2.1
Schema diagram for the database in Figure 1.2.

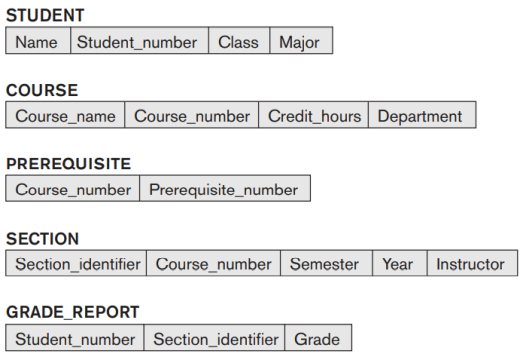
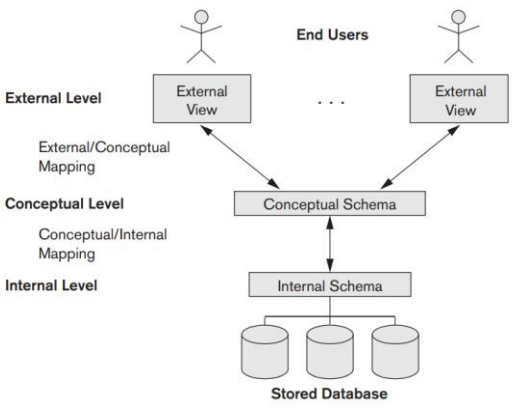


Figure 2.2
The three-schema architecture.



When we define a new database, we specify its database schema only to the DBMS. At this point, the corresponding **database state** is the *empty state* with no data. We get the *initial state* of the database when the database is first **populated or loaded** with the *initial data.*

Valid state—that is, a state that satisfies the structure and constraints specified in the schema.

The DBMS stores the descriptions of the schema constructs and constraints—also called the **meta-data.**

Schema evolution—Adding new schema construct to the database.

Note : The schema is sometimes called the intension, and a database state is called an extension of the schema.

3) **Three-Schema Architecture and Data Independence** :
   A) The **internal level (physical)** has an internal schema : internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.
   B) The **conceptual level (logical)** has a conceptual schema : The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. It uses representational data models.
   C) The **external or view level** includes a number of external schemas or user views : external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. It uses high-level conceptual data model. Example, application program.

The processes of transforming requests and results between levels are called **mappings**.

4) **Data Independence** : Capacity to change the schema at one level of a database system without having to change the schema at the next higher level.
   A) **Logical data independence** is the capacity to change the conceptual schema without having to change external schemas or application programs.
   B) **Physical data independence** is the capacity to change the internal schema without having to change the conceptual schema. Hence, the external schemas need not be changed as well.
5) **Database languages and Interfaces** :
   A) ***DBMS Languages*** :
      i) In many DBMSs where no strict separation of levels is maintained, one language, called the **data definition language (DDL)**, is used by the DBA and by database designers to define both schemas.
      ii) **storage definition language (SDL)**, is used to specify the internal schema. The mappings between the two schemas may be specified in either one of these languages.
      iii) **view definition language (VDL)**, to specify user views and their mappings to the conceptual schema, but in most DBMSs the DDL is used to define both conceptual and external schemas.
      iv) Typical manipulations include *retrieval, insertion, deletion, and modification* of the data. The DBMS provides a set of operations or a language called the **data manipulation language (DML)** for these purposes.
         a) A **high-level or nonprocedural DML** can be used on its own to specify complex database operations concisely. Many DBMSs allow high-level DML statements either to be entered interactively from a display monitor or terminal or to be embedded in a general-purpose programming language.
         b) A **low level or procedural DML** must be embedded in a general-purpose programming language. This type of DML typically retrieves individual records or objects from the database and processes each separately.

When it needs to use programming language constructs, such as looping, to retrieve and process each record from a set of records. Low-level DMLs are also called record-at-a-time DMLs because of this property.

High-level DMLs, such as SQL, can specify and retrieve many records in a single DML statement; therefore, they are called set-at-a-time or set-oriented DMLs. A query in a high-level DML often specifies which data to retrieve rather than how to retrieve it; therefore, such languages are also called **declarative.**

6) **DBMS Interfaces** :
   A) Menu-based Interfaces for Web Clients or Browsing.
   B) Apps for Mobile Devices.
   C) Forms-based Interfaces.
   D) Graphical User Interfaces.
   E) Natural Language Interfaces
   F) Keyword-based Database Search.
   G) Speech Input and Output
   H) Interfaces for Parametric Users
   I) Interfaces for the DBA

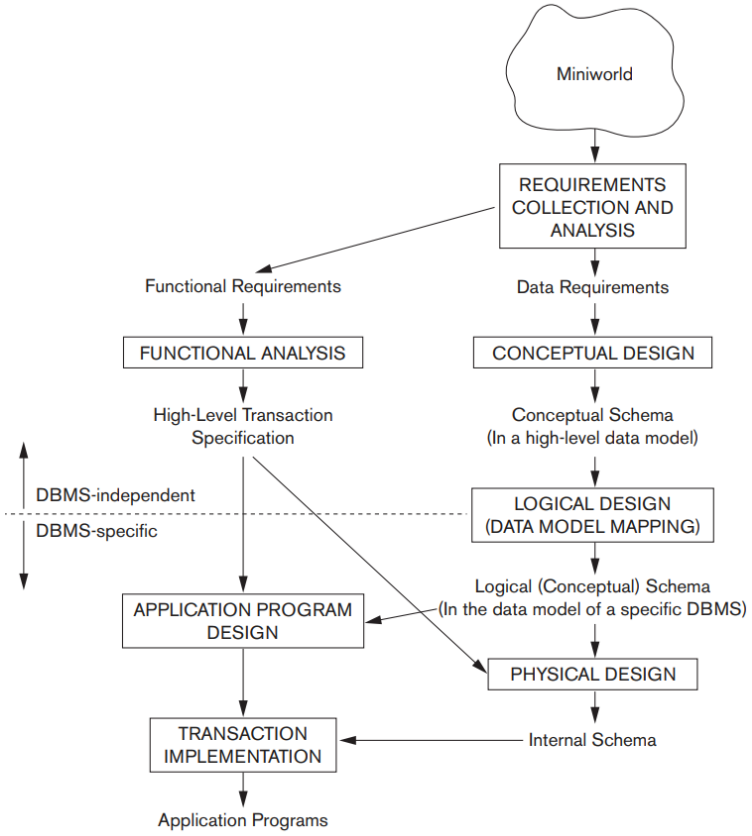
3. Data Modelling Using the Entity– Relationship (ER) Model

We present the modelling concepts of the entity–relationship (ER) model, which is a popular high-level conceptual data model.

1) Using High-Level Conceptual Data Models for Database Design : (5 steps)

Step 1 : The first step shown is requirements collection and analysis. The database designers interview prospective database users to understand and document their **data requirements**. The result of this step is a concisely written set of users' requirements.

Step 2 : In parallel with specifying the data requirements, it is useful to specify the known functional requirements of the application. These consist of the user defined operations (or transactions) that will be applied to the database, including both retrievals and updates.



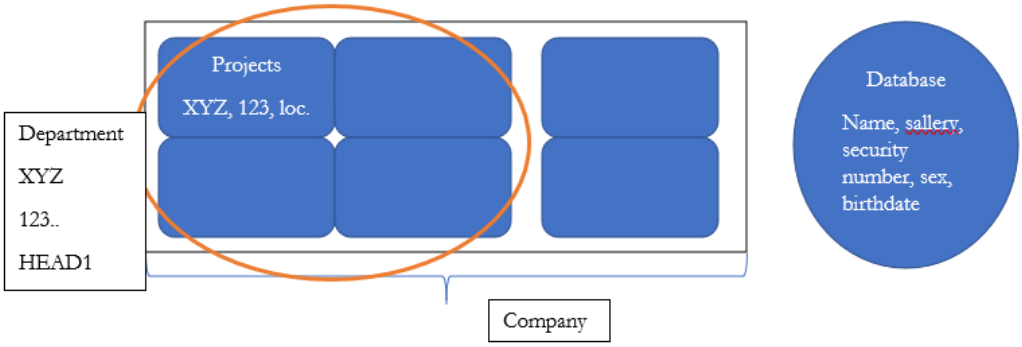
Step 3 : The next step is to create a conceptual schema for the database, using a high-level conceptual data model. This step is called conceptual design. The conceptual schema is a concise description of the data requirements of the users and includes detailed descriptions of the entity types, relationships, and constraints.

During or after the conceptual schema design, the basic data model operations can be used to specify the high-level user queries and operations identified during functional analysis.

Step 4 : The next step in database design is the actual implementation of the database, using a commercial DBMS. Most current commercial DBMSs use an implementation data model—such as the relational (SQL) model—so the conceptual schema is transformed from the high-level data model into the implementation data model. This step is called logical design or data model mapping; its result is a database schema in the implementation data model of the DBMS.

Step 5: The last step is the physical design phase, during which the internal storage structures, file organizations, indexes, access paths, and physical design parameters for the database files are specified.

2) A Sample Database Application :



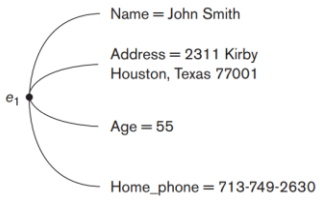
The database will store each employee's name, Social Security number, address, salary, sex (gender), and birth date. An employee is assigned to one department, but may work on several projects.

It is required to keep track of the current number of hours per week that an employee works on each project, as well as the direct supervisor of each employee (who is another employee).

Figure 3.2 shows how the schema for this database application can be displayed by means of the graphical notation known as ER diagrams. ER diagram is used in conceptual data model.

3) **Entity Types, Entity Sets, Attributes, and Keys** : The ER model describes data as entities, relationships, and attributes.

Entities and Their Attributes. The basic concept that the ER model represents is an entity, which is a thing or object in the real world with an independent existence. Each entity has attributes—the particular properties that describe it. For example, an EMPLOYEE entity may be described by the employee's name, age, address, salary, and job.



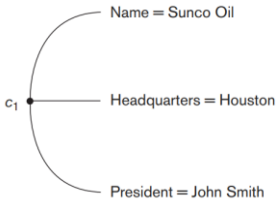


Figure 3.3
Two entities, EMPLOYEE e_1, and COMPANY c_1, and their attributes.

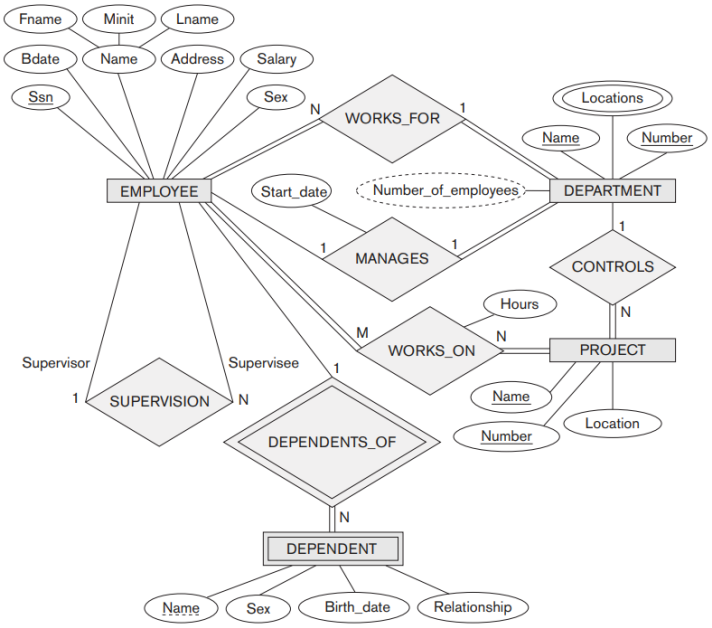


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 3.14.

Several types of attributes occur in the ER model: simple versus composite, single-valued versus multivalued, and stored versus derived.

A) **Composite versus Simple (Atomic) Attributes** : Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings. Attributes that are not divisible are called simple or atomic attributes.

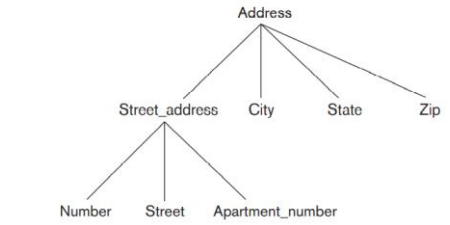


Figure 3.4
A hierarchy of composite attributes.

{Address_phone({Phone(Area_code,Phone_number)},Address(Street_address (Number,Street,Apartment_number),City,State,Zip))}

Figure 3.5
A complex attribute: Address_phone.

B) **Single-Valued versus Multivalued Attributes.** : Most attributes have a single value for a particular entity; such attributes are called single-valued. For example, Age is a single-valued attribute of a person.
One person may not have any college degrees, another person may have one, and a third person may have two or more degrees; therefore, different people can have different numbers of values for the College_degrees attribute. Such attributes are called multivalued.

C) **Stored versus Derived Attributes**. : In some cases, two (or more) attribute values are related—for example, the Age and Birth_date attributes of a person.
The Age attribute is hence called a derived attribute and is said to be derivable from the Birth_date attribute, which is called a stored attribute.

D) **NULL Values** : An address of a single-family home would have NULL for its Apartment_number attribute, and a person with no college degree would have NULL for College_degrees.

E) **Complex Attributes**. : Components of a composite attribute between parentheses ( ) and separating the components with commas, and by displaying multivalued attributes between braces { }. Such attributes are called complex attributes. For example, if a person can have more than one residence and each residence can have a single address and multiple phones, an attribute Address_phone for a person can be specified. Both Phone and Address are themselves composite attributes.

Entity Types, Entity Sets, Keys, and Value Sets : An **entity type** defines a collection (or set) of entities that have the same attributes. The collection of all entities of a particular entity type in the database at any point in time is called an entity set or entity collection; the entity set is usually referred to using the same name as the entity type, even though they are two separate concepts. For example, EMPLOYEE refers to both a type of entity as well as the current collection of all employee entities in the database. In short Entity type name is "name" EMPLOYEE, Entity is e1 having attributes NAME AGE SALARY, Entity set is set of all entities.

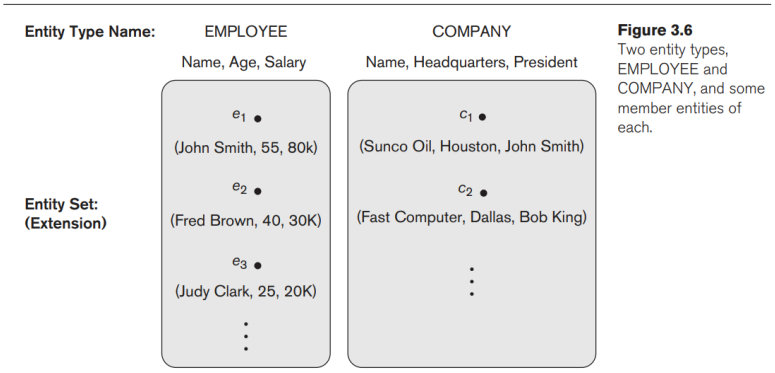


Figure 3.6
Two entity types, EMPLOYEE and COMPANY, and some member entities of each.

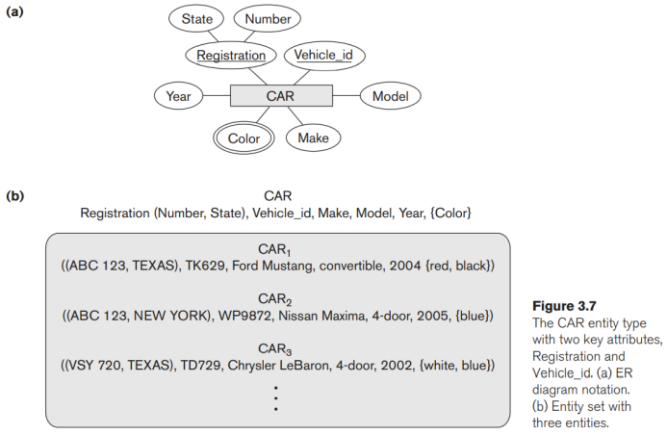


Figure 3.7
The CAR entity type with two key attributes, Registration and Vehicle_id. (a) ER diagram notation. (b) Entity set with three entities.

Composite attributes are attached to their component attributes by straight lines. Multivalued attributes are displayed in double ovals.

A) **Key Attributes of an Entity Type**. : An entity type usually has one or more attributes whose values are distinct for each individual entity in the entity set. Such an attribute is called a **key attribute**, and its values can be used to identify each entity uniquely. For example, the Name attribute is a key of the COMPANY entity type in Figure 3.6 because no two companies are allowed to have the same name. each key attribute has its name underlined inside the oval.

The Registration attribute is an example of a composite key formed from two simple component attributes, State and Number, neither of which is a key on its own. An entity type may also have no key, in which case it is called a weak entity type.

B) **Value Sets (Domains) of Attributes** : Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity.
we can specify the value set of the Age attribute of EMPLOYEE to be the set of integer numbers between 16 and 70.

- Mathematically, an attribute A of entity set E whose value set is V can be defined as a function from E to the power set P(V) of V:

A : E → P(V)

The power set P(V) of a set V is the set of all subsets of V.

We refer to the value of attribute A for entity e as A(e).

For a composite attribute A, the value set V is the power set of the Cartesian product of P(V1), P(V2), . . . , P(Vn), where V1, V2, . . . , Vn are the value sets of the simple component attributes that form A: V = P(P(V1) × P(V2) × . . . × P(Vn)).

4) Relationship Types, Relationship Sets, Roles, and Structural Constraints : There are several implicit relationships among the various entity types. In fact, whenever an attribute of one entity type refers to another entity type, some relationship exists. For example, the attribute Manager of DEPARTMENT refers to an employee who manages the department.

A) **Relationship Types, Sets, and Instances** : A relationship type R among n entity types E1, E2, . . . , En defines a set of associations—or a relationship set—among entities from these entity types.

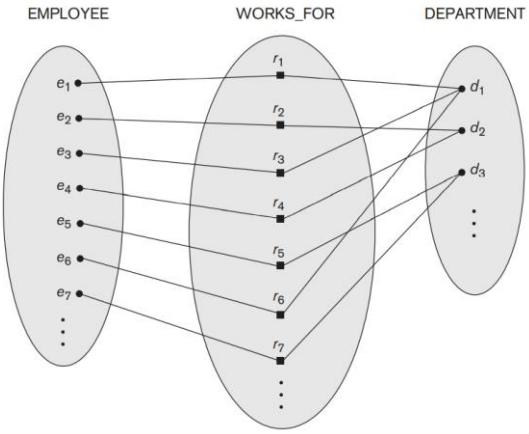


Figure 3.9
Some instances in the WORKS_FOR relationship set, which represents a relationship type WORKS_FOR between EMPLOYEE and DEPARTMENT.

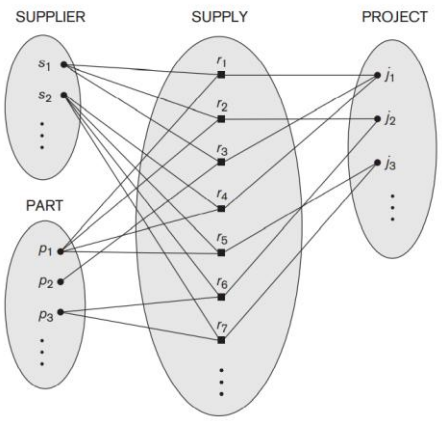


Figure 3.10
Some relationship instances in the SUPPLY ternary relationship set.

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

B) **Relationship Degree, Role Names, and Recursive Relationships** : The degree of a relationship type is the number of participating *entity types.* Hence, the WORKS_FOR relationship is of degree two. A relationship type of degree two is called binary, and one of degree three is called ternary.

The **role name** signifies the role that a participating entity from the entity type plays in each relationship instance, and it helps to explain what the relationship means. For example, in the WORKS_FOR relationship type, EMPLOYEE plays the role of employee or worker and DEPARTMENT plays the role of department or employer.

In some cases the same entity type participates more than once in a relationship type in different roles. In such cases the role name becomes essential for distinguishing the meaning of the role that each participating entity plays. Such relationship types are called **recursive relationships or self-referencing relationships.**

Constraints on Binary Relationship Types : The cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in. For example, in the WORKS_FOR binary relationship type, DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N, meaning that each department can be related to (that is, employs) any number of employees (N), but an employee can be related to (work for) at most one department (1).

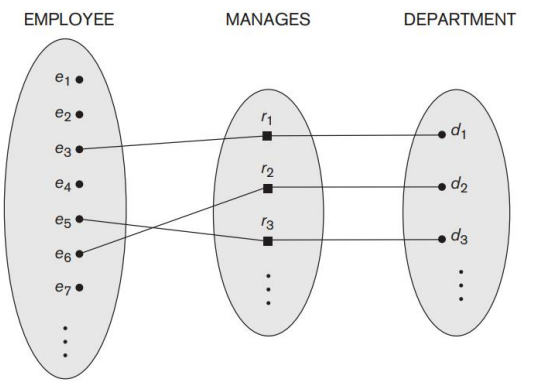


Figure 3.12
A 1:1 relationship, MANAGES.

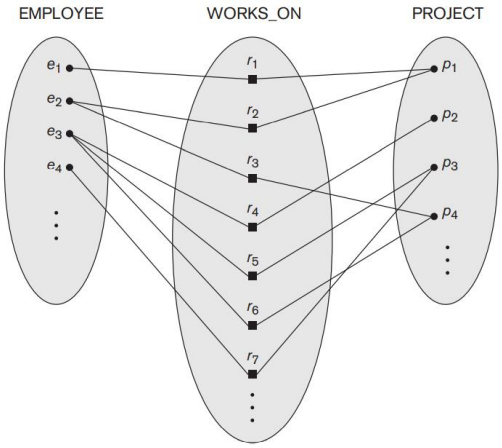


Figure 3.13
An M:N relationship, WORKS_ON.

The **participation constraint** specifies whether the existence of an entity depends on its being related to another entity via the relationship type. (Two types Total and Partial).

If a company policy states that every employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS_FOR relationship instance (Figure 3.9). Thus, the participation of EMPLOYEE in WORKS_FOR is called total participation, meaning that every entity in the total set of employee entities must be related to a department entity via WORKS_FOR. **Total participation is also called existence dependency**.

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

In ER diagrams, total participation (or existence dependency) is displayed as a double line connecting the participating entity type to the relationship, whereas partial participation is represented by a single line.

Relational Attributes :

i) Notice that attributes of 1:1 relationship types can be migrated to one of the participating entity types.
ii) For a 1:N relationship type, a relationship attribute can be migrated only to the entity type on the N-side of the relationship. (Same for N:1 but opposite)
iii) For M:N (many-to-many) relationship types, some attributes may be determined by the combination of participating entities in a relationship instance, not by any single entity.

Weak Entity Types : Entity types that do not have key attributes of their own are called **weak entity** types. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with one of their attribute values. We call this other entity type the **identifying or owner entity type.** A weak entity type normally has a **partial key**, which is the attribute that can uniquely identify weak entities that are related to the same owner entity.

Note : In ER diagrams, both a weak entity type and its identifying relationship are distinguished by surrounding their boxes and diamonds with double lines (see Figure 3.2). The partial key attribute is underlined with a dashed or dotted line.

- The identifying entity type is also sometimes called the parent entity type or the dominant entity type.
- The weak entity type is also sometimes called the child entity type or the subordinate entity type.

- The partial key is sometimes called the **discriminator**.

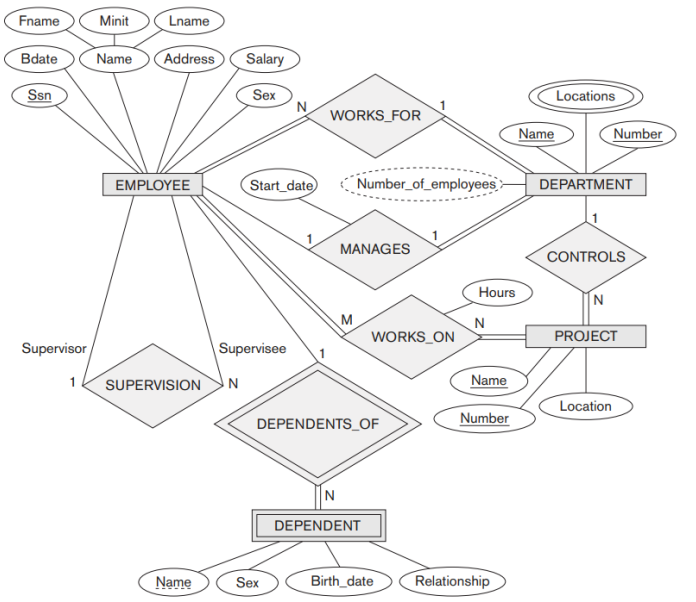


Figure 3.2
An ER schema diagram for the COMPANY database. The diagrammatic notation is introduced gradually throughout this chapter and is summarized in Figure 3.14.

- MANAGES, which is a 1:1(one-to-one) relationship type between EMPLOYEE and DEPARTMENT. EMPLOYEE participation is partial. DEPARTMENT participation is not clear from the requirements. We question the users, who say that a department must have a manager at all times, which implies total participation.[13] The attribute Start_date is assigned to this relationship type.
- WORKS_FOR, a 1:N (one-to-many) relationship type between DEPARTMENT and EMPLOYEE. Both participations are total.
- CONTROLS, a 1:N relationship type between DEPARTMENT and PROJECT. The participation of PROJECT is total, whereas that of DEPARTMENT is determined to be partial, after consultation with the users indicates that some departments may control no projects.
- SUPERVISION, a 1:N relationship type between EMPLOYEE (in the supervisor role) and EMPLOYEE (in the supervisee role). Both participations are determined to be partial, after the users indicate that not every employee is a supervisor and not every employee has a supervisor.
- WORKS_ON, determined to be an M:N (many-to-many) relationship type with attribute Hours, after the users indicate that a project can have several employees working on it. Both participations are determined to be total.
- DEPENDENTS_OF, a 1:N relationship type between EMPLOYEE and DEPENDENT, which is also the identifying relationship for the weak entity type DEPENDENT. The participation of EMPLOYEE is partial, whereas that of DEPENDENT is total.

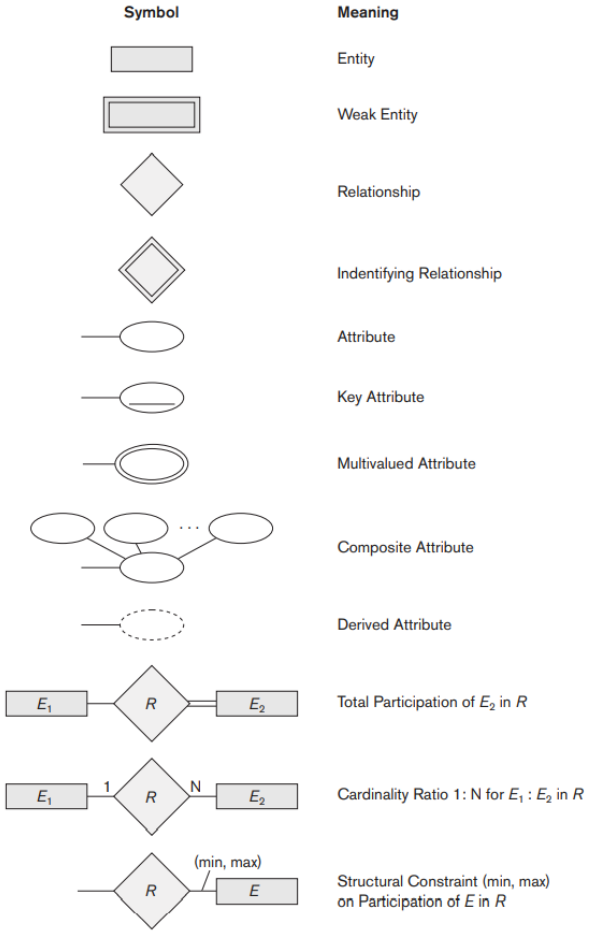


Figure 3.14
Summary of the notation for ER diagrams.

STUDENT

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

COURSE

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

SECTION

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

GRADE_REPORT

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

PREREQUISITE

Course_number	Prerequisite_number
CS3380	CS3320
CS3380	MATH2410
CS3320	CS1310

Figure 1.2
A database that stores student and course information.

Specialization is the process of classifying a class of objects into more specialized subclasses. **Generalization** is the inverse process of generalizing several classes into a higher-level abstract class that includes the objects in all these classes. We call the relationship between a subclass and its superclass an **IS-A-SUBCLASS-OF** relationship, or simply an **IS-A** relationship. Denoted by triangle.

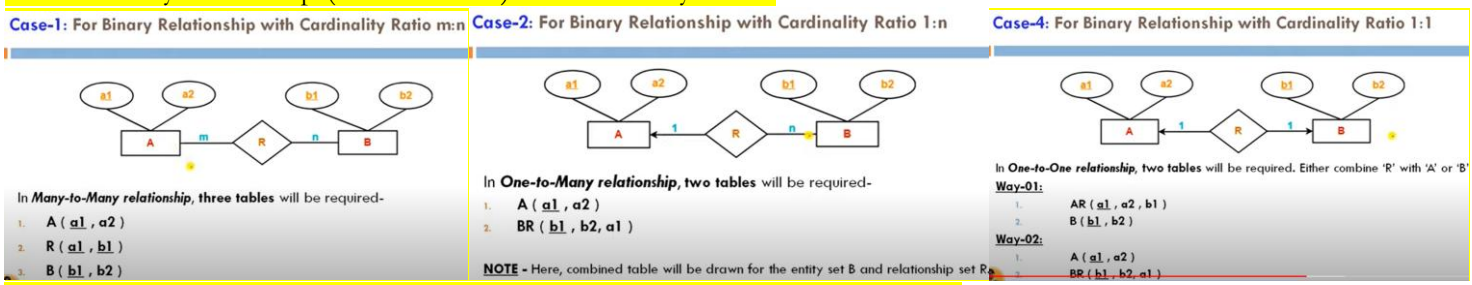
⇨ **Finding Minimum table from ER diagram**.

Rules :

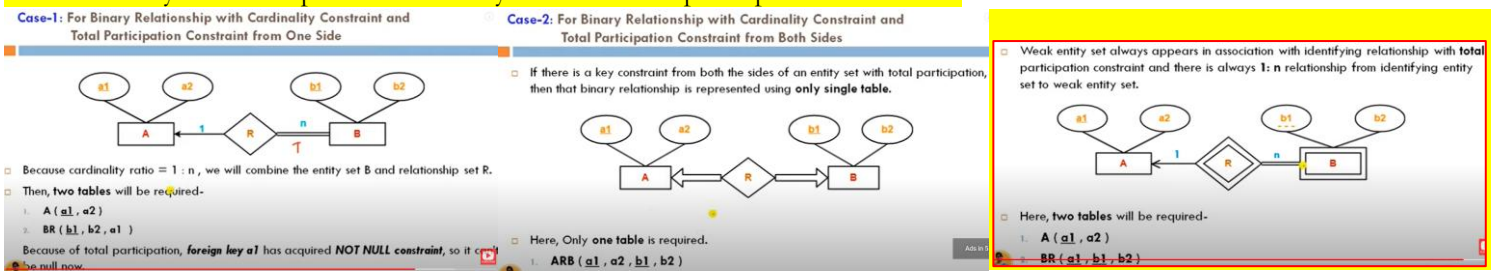
- ❖ A strong entity (single rectangle) set with only simple attributes will require only one table in relational model.
- ❖ A strong entity set with any number of composite attributes will require only one table in relational model. Only simple attributes of the composite attributes are taken into account and not the composite attributes itself.
- ❖ A strong entity set with any number of multivalued attributes (two circles) will require two table in rational model. Primary attribute will be in both tables.

❖ A relationship set will require one table in the relational model. Relationship set means figure 3.14 and 11th row diagram. Remember relation can also have attributes also known as **descriptive attributes**. For example, figure 3.2 WORKS_ON have hours.

❖ For Binary relationships(BR short form) with cardinality ratios :

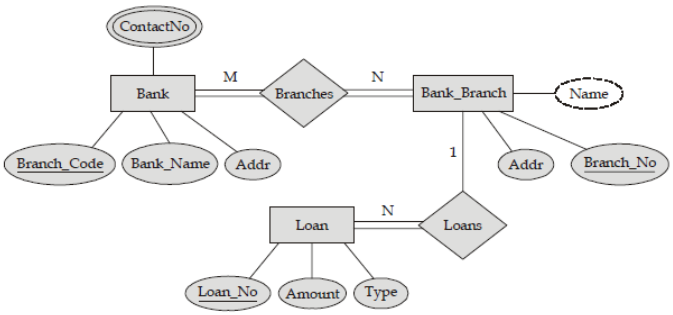


❖ For Binary relationships with cardinality constraint and participation constraint :



In all images where primary keys of one node is included in others tables it is considered as foreign keys that is why underline is not there. In ER diagram, single line represents many relation and line containing arrow represents one relation.

Example :



how many tables are required for this relational database that satisfy 1NF.

Answer :

1. Bank {Branch_code, Contact_No }

   where {Branch_code, Contact_No} is primary key.

2. Bank {Branch_code , Bank_Name, Addr}

3. Branches {Branch_No, Branch_code} where {Branch_No, Branch_code} is primary key.

4. Bank_Branch {Addr, Branch_No, Name}.

5. Loan taken {Loan No, Amount, Type, Branch_No}.

   where Loan_No is primary key.

- Here we can't merge branches relation and Bank_Branch entity because foreign key "Code" is not the candidate key of bank entity, so we cannot combined these two.

We cannot combine Bank and Branches because key of Branches is Branch_code, Branch_no and key of Bank is contact no, branch_code so both keys differ by contactNo so we need separate table for Braches. And we need separate table for Bank but here 1NF is not satisfied because of multivariable. So we split it into two tables. Finally have total of 5 table.

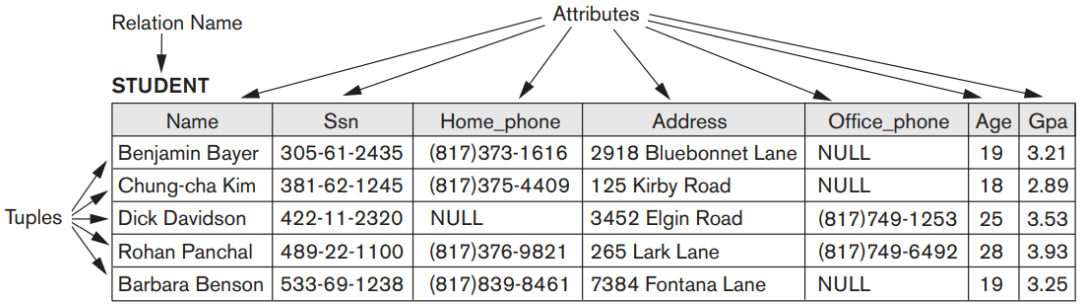
4. The Relational Data Model and Relational Database Constraints

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

1) **Domains, Attributes, Tuples, and Relations** : In relational scheme every tuple is divided in field called **domain**. A **domain** D is a set of atomic values. By atomic we mean that each value in the domain is indivisible as far as the formal relational model is concerned. Example : ■ Usa_phone_numbers. The set of ten-digit phone numbers valid in the United States.

The preceding is called logical definitions of domains. A data type or format is also specified for each domain. For example, the data type for the domain Usa_phone_numbers can be declared as a character string of the form (ddd)ddd-dddd, where each d is a numeric (decimal) digit and the first three digits form a valid telephone area code.

A **relation schema** R, denoted by R(A1, A2, … , An), is made up of a **relation name** R and a list of attributes, A1, A2, … , An. Each **attribute** Ai is the name of a role played by some domain D in the relation schema R. D is called the **domain** of Ai and is denoted by **dom(Ai)**. A relation schema is used to describe a relation; R is called the name of this relation. The **degree** (or arity) of a relation is the number of attributes n of its relation schema.



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

Figure 5.1
The attributes and tuples of a relation STUDENT.

2) **Relational Model Notation**: We will use the following notation in our presentation :

■ A relation schema R of degree n is denoted by R(A1, A2, … , An).

■ The uppercase letters Q, R, S denote relation names.

■ The lowercase letters q, r, s denote relation states.

■ The letters t, u, v denote tuples.

■ In general, the name of a relation schema such as STUDENT also indicates the current set of tuples in that relation—the current relation state—whereas STUDENT(Name, Ssn, …) refers only to the relation schema.

■ An attribute A can be qualified with the relation name R to which it belongs by using the dot notation R.A—for example, STUDENT.Name or STUDENT.Age. This is because the same name may be used for two attributes in different relations. However, all attribute names in a particular relation must be distinct.

■ An n-tuple t in a relation r(R) is denoted by t = , where vi is the value corresponding to attribute Ai. The following notation refers to component values of tuples:

Both t[Ai] and t.Ai (and sometimes t[i]) refer to the value vi in t for attribute Ai.

Both t[Au, Aw, … , Az] and t.(Au, Aw, … , Az), where Au, Aw, … , Az is a list of attributes from R, refer to the subtuple of values from t corresponding to the attributes specified in the list.

As an example, consider the tuple t = <'Barbara Benson', '533-69-1238', '(817)839-8461', '7384 Fontana Lane', NULL, 19, 3.25> from the STUDENT relation in Figure 5.1; we have t[Name] = <'Barbara Benson'>, and t[Ssn, Gpa, Age] = <'533-69-1238', 3.25, 19>.

3) **Relational Model Constraints and Relational Database Schemas** : In this section, we discuss the various restrictions on data that can be specified on a relational database in the form of constraints. Constraints on databases can generally be divided into three main categories:

1. Constraints that are inherent in the data model. We call these **inherent model-based constraints or implicit constraints**.

2. Constraints that can be directly expressed in the schemas of the data model, typically by specifying them in the DDL (data definition language, see Section 2.3.1). We call these **schema-based constraints or explicit constraints**.

3. Constraints that cannot be directly expressed in the schemas of the data model, and hence must be expressed and enforced by the application programs or in some other way. We call these application-based or **semantic constraints or business rules.**

The one we discussed was **inherent model-based constraints.** In part two in image given just before this 3rd part.

A) The schema-based constraints include domain constraints, key constraints, constraints on NULLs, entity integrity constraints, and referential integrity constraints.

a) Domain Constraints : We have already discussed the ways in which domains can be specified in Relational Model notation.

b) Key Constraints and Constraints on NULL Values : Suppose that we denote one such subset of attributes by SK; then for any two distinct tuples t1 and t2 in a relation state r of R, we have the constraint that: t1[SK] ≠ t2[SK]. Any such set of attributes SK is called a superkey of the relation schema R.

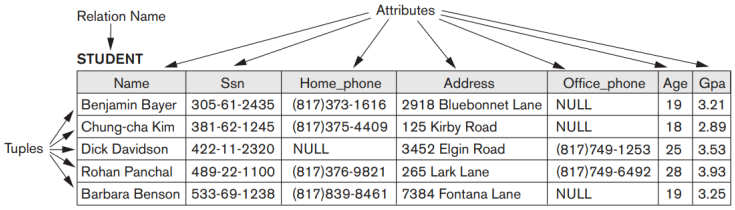
- A key k of a relation schema R is a superkey of R with the additional property that removing any attribute A from K leaves a set of attributes K' that is not a superkey of R any more. Hence, a key satisfies two properties:
  1. Two distinct tuples in any state of the relation cannot have identical values for (all) the attributes in the key. This uniqueness property also applies to a superkey.
  2. It is a minimal superkey—that is, a superkey from which we cannot remove any attributes and still have the uniqueness constraint hold. This minimality property is required for a key but is optional for a superkey.

Consider the STUDENT relation of Figure 5.1. The attribute set {Ssn} is a key of STUDENT because no two student tuples can have the same value for Ssn. Any set of attributes that includes Ssn—for example, {Ssn, Name, Age}—is a superkey. However, the superkey {Ssn, Name, Age} is not a key of STUDENT because removing Name or Age or both from the set still leaves us with a superkey.

CAR

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

Figure 5.4
The CAR relation, with two candidate keys: License_number and Engine_serial_number.



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

Figure 5.1
The attributes and tuples of a relation STUDENT.

Note : The choice of one to become the primary key is somewhat arbitrary; however, it is usually better to choose a primary key with a single attribute or a small number of attributes. The other candidate keys are designated as unique keys and are not underlined.

Relational Databases and Relational Database Schemas : A **relational database schema** S is a set of relation schemas S = {R1, R2, … , Rm} and a set of **integrity constraints** IC. A **relational database state** DB of S is a set of relation states DB = {r1, r2, … , rm} such that each ri is a state of Ri and such that the ri relation states satisfy the integrity constraints specified in IC. Figure 5.5 shows a relational database schema that we call COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}

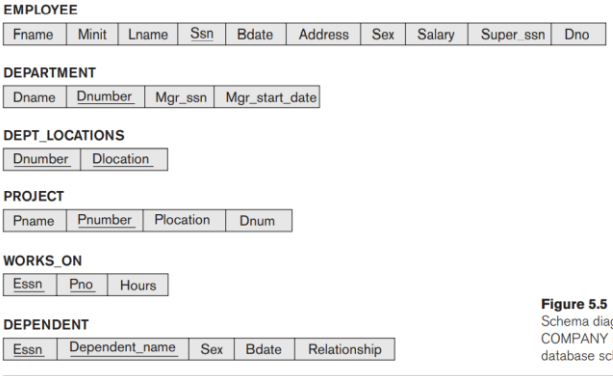


Figure 5.5
Schema diagram for the COMPANY relational database schema.

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

A database state that does not obey all the integrity constraints is called not valid, and a state that satisfies all the constraints in the defined set of integrity constraints IC is called a valid state.

Integrity constraints are specified on a database schema and are expected to hold on every valid database state of that schema. In addition to domain, key, and NOT NULL constraints, two other types of constraints are considered part of the relational model: entity integrity and referential integrity.

Entity Integrity, Referential Integrity, and Foreign Keys : The **entity integrity** constraint states that no primary key value can be NULL. 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. For example, in Figure 5.6, the attribute Dno of EMPLOYEE gives the department number for which each employee works; hence, its value in every EMPLOYEE tuple must match the Dnumber value of some tuple in the DEPARTMENT relation.

Specify a referential integrity constraint between the two relation schemas R1 and R2. A set of attributes FK in relation schema R1 is a foreign key of R1 that references relation R2 if it satisfies the following rules:
1. The attributes in FK have the same domain(s) as the primary key attributes PK of R2; the attributes FK are said to reference or refer to the relation R2.
2. A value of FK in a tuple t1 of the current state r1(R1) either occurs as a value of PK for some tuple t2 in the current state r2(R2) or is NULL. In the former case, we have t1[FK] = t2[PK], and we say that the tuple t1 references or refers to the tuple t2.
If these two conditions hold, a referential integrity constraint from R1 to R2 is said to hold.
Notice that a foreign key can refer to its own relation.
We can diagrammatically display referential integrity constraints by drawing a directed arc from each foreign key to the relation it references. For clarity, the arrowhead may point to the primary key of the referenced relation.

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

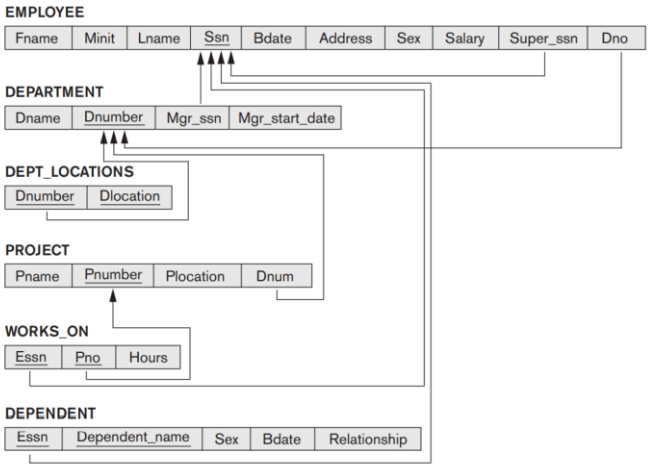
PROJECT

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

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



- Another class of general constraints, sometimes called semantic integrity constraints, are not part of the DDL and have to be specified and enforced in a different way. Examples of such constraints are the salary of an employee should not exceed the salary of the employee's supervisor and the maximum number of hours an employee can work on all projects per week is 56. Such constraints can be specified and enforced within the application programs that update the database.
- The types of constraints we discussed so far may be called state constraints because they define the constraints that a valid state of the database must satisfy. Another type of constraint, called transition constraints, can be defined to deal with state changes in the database.
- The Transaction Concept : A transaction is an executing program that includes some database operations, such as reading from the database, or applying insertions, deletions, or updates to the database. At the end of the transaction, it must leave the database in a valid or consistent state that satisfies all the constraints specified on the database schema.

5. Relational Algebra and Relational Calculus

The inputs and outputs of a query are relations. A query is evaluated using instances of each input relation and it produces an instance of the output relation. Relational algebra is one of the two formal query languages associated with the relational model. A fundamental property is that every operator in the algebra accepts (one or two) relation instances as arguments and returns a relation instance as the result.

1) **Selection (Sigma) and Projection (Pi)** :

A) **Selection** : The expression $\sigma_{rating>8}$ (S2) evaluates to the relation shown in Figure 4.4. The subscript rating> 8 specifies the selection criterion to be applied while retrieving tuples. The selection operator **"σ"** specifies the tuples to retain through a selection condition.

sid	*sname*	*rating*	*age*
28	yuppy	9	35.0
58	Rusty	10	35.0

Figure 4.4 $\sigma_{rating>8}(S2)$

sname	*rating*
yuppy	9
Lubber	8
guppy	5
Rusty	10

Figure 4.5 $\pi_{sname,rating}(S2)$

age
35.0
55.5

Figure 4.6 $\pi_{age}(S2)$

sname	*rating*
yuppy	9
Rusty	10

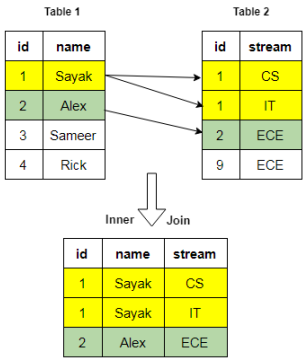
Figure 4.7 $\pi_{sname,rating}(\sigma_{rating>8}(S2))$

B) The projection operator **π** allows us to extract columns from a relation; for example, we can find out all sailor names and ratings by using **π**. The expression $\pi_{sname,rafing}$(S2) evaluates to the relation shown in Figure 4.5. The subscript sname,rating specifies the fields to be retained; the other fields are 'projected out.'

The expression $\pi_{sname,rating}$ (($\sigma_{rating>8}$ (S2)) produces the result shown in Figure 4.7. It is obtained by applying the selection to S2 (to get the relation shown in Figure 4.4) and then applying the projection.

2) **Sets Operation** : A) UNION, B) INTERSECTION, C) SET DIFFERENCE,
D) CROSS PRODUCT : The CARTESIAN PRODUCT creates tuples with the combined attributes of two relations.

3) **Joins** : The JOIN operation, denoted by |><|, is used to combine related tuples from two relations into single "longer" tuples. Example : →



EMP_DEPENDENTS ← EMPNAMES × DEPENDENT

ACTUAL_DEPENDENTS ← $\sigma_{Ssn=Essn}$(EMP_DEPENDENTS)

These two operations can be replaced with a single JOIN operation as follows:

ACTUAL_DEPENDENTS ← EMPNAMES |><|$_{Ssn=Essn}$ DEPENDENT

In JOIN, only combinations of tuples satisfying the join condition appear in the result, whereas in the CARTESIAN PRODUCT all combinations of tuples are included in the result

A) **Condition Join** : The JOIN operation can be specified as a CARTESIAN PRODUCT operation followed by a SELECT operation.

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

C) **Natural Join** : One of each pair of attributes with identical values is superfluous, a new operation called NATURAL JOIN—denoted by *—was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition. The standard definition of NATURAL JOIN requires that the two join attributes (or each pair of join attributes) have the same name in both relations. If this is not the case, a renaming operation is applied first. Natural join of two tables is just projected, filtered cartesian product.

Employee

Name	EmpId	GrpName
Jens	1234	A
Marcos	1235	B
Shant	1236	A
Lukas	1237	B

Group

GrpName	Manager
A	Jens
B	Lukas
C	Hans

Employee ⋈ Group

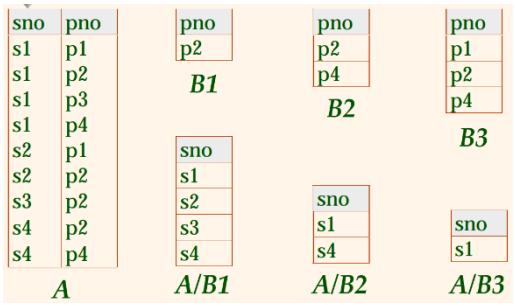
Name	EmpId	GrpName	Manager
Jens	1234	A	Jens
Marcos	1235	B	Lukas
Shant	1236	A	Jens
Lukas	1237	B	Lukas

Notice that if no combination of tuples satisfies the join condition, the result of a JOIN is an empty relation with zero tuples and in case if no common attribute present it gives cross product. In general, if R has nR tuples and S has nS tuples, the result of a JOIN operation R |><|$_{<Join Condition>}$ S will have between zero and nR * nS tuples. The expected size of the join result divided by the maximum size nR * nS leads to a ratio called join selectivity, which is a property of each join condition.

- $R \bowtie_{RA<5} S$ is equivalent to $\sigma_{RA<5}(R \bowtie S)$ This is not equal as first may contains duplicate and second one removes duplicate in brackets only and then apply the condition R.A<5.

- It has been shown that the **set of relational algebra operations {σ, π, ∪, ρ, –, ×}** is a complete set; that is, any of the other original relational algebra operations can be expressed as a sequence of operations from this set. For example, the INTERSECTION operation can be expressed by using UNION and MINUS as follows: R ∩ S ≡ (R ∪ S) – ((R – S) ∪ (S – R)).

4) **Division Operator :** The DIVISION operation, denoted by ÷, is useful for a special kind of query that sometimes occurs in database applications. An example is Retrieve the names of employees who work on all the projects that 'John Smith' works on.



The result of DIVISION is a relation T(Y) that includes a tuple t if tuples tR appear in R with tR [Y] = t, and with tR [X] = tS for every tuple tS in S. This means that, for a tuple t to appear in the result T of the DIVISION, the values in t must appear in R in combination with every tuple in S.

The DIVISION operation can be expressed as a sequence of π, ×, and – operations as follows: $A/B = \{\langle x\rangle \mid \exists\langle x,y\rangle \in A \ \forall\langle y\rangle \in B\}$

T1 ← πY(R) T2 ← πY((S × T1) – R) T ← T1 – T2

R ÷ S = $\prod_{R-S}(r) - \prod_{R-S}(r)[(\prod_{R-S}(r) * s) - \prod_{R-S,S}(r)]$

Table 8.1 Operations of Relational Algebra

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

- Tuples without a matching (or related) tuple are eliminated from the JOIN result. Tuples with NULL values in the join attributes are also eliminated. This type of join, where tuples with no match are eliminated, is known as an **inner join**.
- A set of operations, called **outer joins**, were developed for the case where the user wants to keep all the tuples in R, or all those in S, or all those in both relations in the result of the JOIN, regardless of whether or not they have matching tuples in the other relation.
  - ➔ 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.
  - ➔ RIGHT OUTER JOIN, denoted by ⟖, keeps every tuple in the second, or right, relation S in the result of R ⟖ S.
  - ➔ 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.

NOTE :

1) We cannot do aggregate operation like sum, multiplication, division, finding transitive closure using relational algebra.
2) Result of inner query is non empty because even no record satisfies where condition of inner query count (*) produces one record.

The Tuple Relational Calculus : A calculus expression specifies what is to be retrieved rather than how to retrieve it. Therefore, the relational calculus is considered to be a **nonprocedural language**. This differs from relational algebra, where we must write a sequence of operations to specify a retrieval request in a particular order of applying the operations; thus, it can be considered as a **procedural** way of stating a query.

- It has been shown that any retrieval that can be specified in the basic relational algebra can also be specified in relational calculus, and vice versa; in other words, the expressive power of the languages is identical. This led to the definition of the concept of a relationally complete language.
- The result of such a query is the set of all tuples t that evaluate COND(t) to TRUE. These tuples are said to satisfy COND(t). For example, to find all employees whose salary is above $50,000, we can write the following tuple calculus expression:
  {t | EMPLOYEE(t) AND t.Salary>50000}

Some special cases of this transformation can be stated as follows, where the ≡ symbol stands for equivalent to:
(∀x) (P(x)) ≡ NOT (∃x) (NOT (P(x)))
(∃x) (P(x)) ≡ NOT (∀x) (NOT (P(x)))
(∀x) (P(x) AND Q(x)) ≡ NOT (∃x) (NOT (P(x)) OR NOT (Q(x)))
(∀x) (P(x) OR Q(x)) ≡ NOT (∃x) (NOT (P(x)) AND NOT (Q(x)))
(∃x) (P(x)) OR Q(x)) ≡ NOT (∀x) (NOT (P(x)) AND NOT (Q(x)))
(∃x) (P(x) AND Q(x)) ≡ NOT (∀x) (NOT (P(x)) OR NOT (Q(x)))

Notice also that the following is TRUE, where the ⇒ symbol stands for implies:
(∀x)(P(x)) ⇒ (∃x)(P(x))
NOT (∃x)(P(x)) ⇒ NOT (∀x)(P(x))

- A safe expression in relational calculus is one that is guaranteed to yield a finite number of tuples as its result; otherwise, the expression is called unsafe.
  For example, the expression {t | NOT (EMPLOYEE(t))} is unsafe because it yields all tuples in the universe that are not EMPLOYEE tuples, which are infinitely numerous.

The Domain Relational Calculus : Domain calculus differs from tuple calculus in the type of variables used in formulas: Rather than having variables range over tuples, the variables range over single values from domains of attributes.
Example :
We will use lowercase letters l, m, n, … , x, y, z for domain variables.
Query 0. List the birth date and address of the employee whose name is 'John B. Smith'.
Q0: {u, v | (∃q) (∃r) (∃s) (∃t) (∃w) (∃x) (∃y) (∃z) (EMPLOYEE(qrstuvwxyz) AND q='John' AND r='B' AND s='Smith')}
We need ten variables for the EMPLOYEE relation, one to range over each of the domains of attributes of EMPLOYEE in order. Of the ten variables q, r, s, … , z, only u and v are free, because they appear to the left of the bar and hence should not be bound to a quantifier.

Questions :

1) Given the basic ER and relational models, which of the following is INCORRECT?
An attribute of an entity can has more than one value
An attribute of an entity can be composite
In a row of a relational table, an attribute can have more than one value
In a row of a relational table, an attribute can have exactly one value or a NULL value
Answer : Option C

2) Suppose (A, B) and (C,D) are two relation schemas. Let r1 and r2 be the corresponding relation instances. B is a foreign key that refers to C in r2. If data in r1 and r2 satisfy referential integrity constraints, which of the following is ALWAYS TRUE?

(A) $\Pi_B(r_1) - \Pi_C(r_2) = \varnothing$

(B) $\Pi_C(r_2) - \Pi_B(r_1) = \varnothing$

(C) $\Pi_B(r_1) = \Pi_C(r_2)$

(D) $\Pi_B(r_1) - \Pi_C(r_2) \neq \varnothing$ Answer : A

3) Which of the following tuple relational calculus expression(s) is/are equivalent to (∀x) (P(x))

I. $\neg\exists\, t \in r(P(t))$
II. $\exists\, t \notin r(P(t))$
III. $\neg\exists\, t \in r(\neg P(t))$
IV. $\exists\, t \notin r(\neg P(t))$

Answer : Option III

4) A Relation R with FD set {A->BC, B->A, A->C, A->D, D->A}. How many candidate keys will be there in R?
Answer : Simple candidate key means single attributed key. As (A)+ = {A, B, C, D}, (B)+ = {B,A, C, D}, (C)+ = {C} and (D)+ = {D, A, B, C}. So, A, B and D are candidate keys which are simple as well. So, correct option is 3.

5) Consider the following tables T1 and T2. In table T1 P is the primary key and Q is the foreign key referencing R in table T2 with on-delete cascade and on-update cascade. In table T2, R is the primary key and S is the foreign key referencing P in table T1 with on-delete set NULL and on-update cascade. In order to delete record ⟨3,8⟩ from the table T1, the number of additional records that need to be deleted from table T1 is

T1

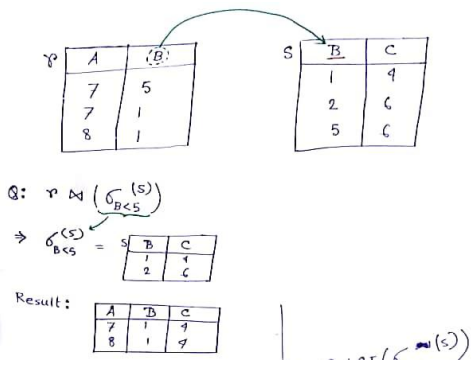
P	Q
2	2
3	8
7	3
5	8
6	9
8	5
9	8

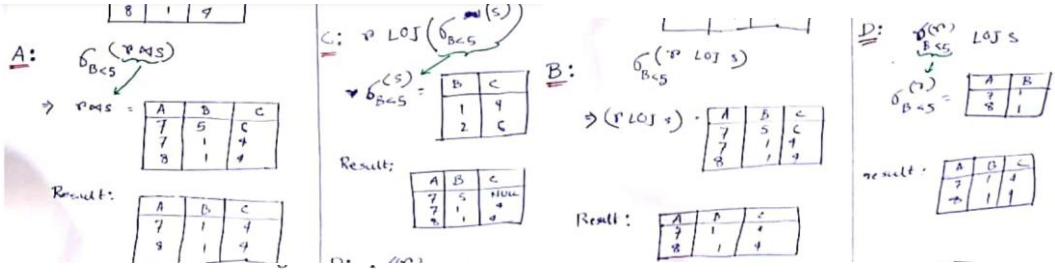
T2

R	S
2	2
8	3
3	2
9	7
5	7
7	2

Answer : As Q refers to R so, deleting 8 from Q won't be an issue, however S refers P. But as the relationship given is on delete set NULL, 3 will be deleted from T1 and the entry in T2 having 3 in column S will be set to NULL. So, no more deletions. Answer is 0.

6) Consider the relations r(A,B) and s(B,C), where s.B is a primary key and r.B is a foreign key referencing s.B. Consider the query Q:r⋈(σB<5(s))
Let LOJ denote the natural left outer-join operation. Assume that r and s contain no null values. Which of the following is NOT equivalent to Q?
σB<5(r⋈s)
σB<5(rLOJs)
rLOJ(σB<5(s))
σB<5(r)LOJs
Answer : C, Here By applying we will have same number of entries in table as the LHS of LOJ so in third option LOJ is applied with r in last which may cause some entries of 5 or greater than 5 be present in table with null values on RHS.





(a)	Entity integrity	(i)	enforces some specific business rule that do not fall into entity or domain
(b)	Domain integrity	(ii)	Rows can't be deleted which are used by other records
(c)	Referential integrity	(iii)	enforces valid entries for a column
(d)	Userdefined integrity	(iv)	No duplicate rows in a table

Code :

	(a)	(b)	(c)	(d)
(1)	(iii)	(iv)	(i)	(ii)
(2)	(iv)	(iii)	(ii)	(i)
(3)	(iv)	(ii)	(iii)	(i)
(4)	(ii)	(iii)	(iv)	(i)

7) Answer : (2)

8) Let pk(R) denotes primary key of relation R. A many-to-one relationship that exists between two relations R1 and R2 can be expressed as follows :

pk(R2) → pk(R1)
pk(R1) → pk(R2)
pk(R2) → R1 ∩ R2
pk(R1) → R1 ∩ R2

Answer : (B) Consider the following case
A many-to-one relationship set exists between entity sets students and course (a relation r with primary key roll_no).
let it is decomposed in 2 relations r1(student with Pk roll_no) and r2 (course with Pk course name), Roll_no can uniquely identify course name but course name can not uniquely identify roll_no, so the functional dependency Pk(student) → Pk(course) indicates a many-to-one relationship between r1 and r2.

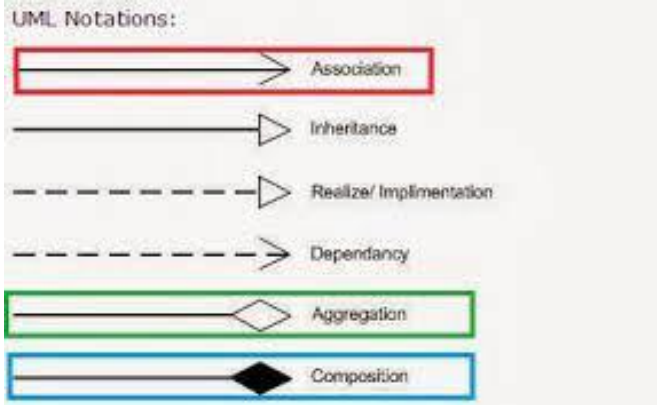
9) Let R(a, b, c) and S(d, e, f) be two relations in which d is the foreign key of S that refers to the primary key of R. Consider the following four operations R and S.

I. Insert into R
II. Insert into S
III. Delete from R
IV. Delete from S

Which of the following can cause violation of the referential integrity constraint above?
Answer : II and III

10) Properties of Relational Database :

- NULL values can be used to opt a tuple out of enforcement of a foreign key.
- The difference between the project operator (Π) in relational algebra and the SELECT keyword in SQL is that if the resulting table/set has more than one occurrence of the same tuple, then Π will return only one of them, while SQL SELECT will return all.
- Association is a relationship where all objects have their own lifecycle and there is no owner. Aggregation is a specialised form of Association where all objects have their own lifecycle, but there is ownership and child objects can not belong to another parent object. Aggregation implies a relationship where the child can exist independently of the parent. Example, class (parent) and student (child). Delete the class and the students still exist. Aggregation is an abstraction through which relationships are treated as higher-level entities which can participate in relationships with other entity sets or with other relationship-sets. Composition is again specialised form of Aggregation and we can call this as a "death" relationship. It is a strong type of Aggregation. Child object does not have its lifecycle and if parent object is deleted, all child objects will also be deleted.



- An update anomaly is when it is not possible to store information unless some other, unrelated information is stored as well.
- In relational database, NULL values can be used To allow duplicate tuples in the table by filling the primary key column(s) with NULL.

11) Every time the attribute A appears, it is matched with the same value of attribute B but not the same value of attribute C. Which of the following is true?

A-> (B,C)
A ->B, A->>C
A->B, C->>A
A->>B, B->C

Answer : Functional dependency is a relationship that exists when one attribute uniquely determines another attribute, therefore A→B. Multivalued dependency occurs when there are more than one independent multivalued attributes in a table. If A->> C is a dependency, it means for A, attribute C has more than one value. So, option (B) is correct.

12) Which of the following statements is FALSE about weak entity set?

Weak entities can be deleted automatically when their strong entity is deleted.
Weak entity set avoids the data duplication and consequent possible inconsistencies caused by duplicating the key of the strong entity.
A weak entity set has no primary keys unless attributes of the strong entity set on which it depends are included.
Tuples in a weak entity set are not partitioned according to their relationship with tuples in a strong entity set.

Answer : (D) because tuples in a weak entity set are partitioned according to their relationship with tuples in a strong entity set.

13) Consider the following database table having A, B, C and D as its four attributes and four possible candidate keys (I, II, III and IV) for this table :

A	B	C	D
a1	b1	c1	d1
a2	b3	c3	d1
a1	b2	c1	d2

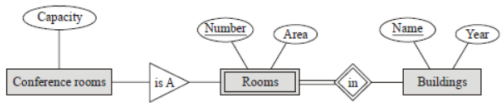
I : {B}
II : {B, C}
III : {A, D}
IV : {C, D}

If different symbols stand for different values in the table (e.g., d1 is definitely not equal to d2), then which of the above could not be the candidate key for the database table ?

Answer : © we can see that the functional dependency B -> {A, C, D} holds. Hence {B} is one candidate key. A,D -> {A, B, C, D} ; C,D -> {A, B, C, D} and B,C -> {A, B, C, D} also holds true. Therefore, these all can be candidate keys. But in B,C -> {A, B, C, D} , there exist a partial dependency of B->{A, B, C, D}. Hence {B, C} cannot be a candidate key. So, option (C) is correct.

14)

Consider the following ER design for a database that keeps track of buildings, rooms and in particular, conference rooms.



Suppose that we convert the above ER diagram into relations using the ER style translation for subclasses. What will we get for the conference rooms entity set?

A Conference room (capacity)

B Conference room (room number, capacity) Your answer is **IN-CORRECT**

C Conference room (building name, room number, capacity)

D Conference room (building name, room number, area, capacity) Correct Option

Answer : first weak entity and identifying relation will combine together so Rooms(number, area, name). Now, due to generalizable and specialization concept we say that conference rooms is a subclass of room so both will have same table. So, conference room(building name, room number, area, capacity)

15) Consider the following relation: R(U V W X Y Z) with FD set of relation R . {U → V, W → X, Y → Z}

What is the minimum number of relations required to decompose the relational into BCNF which satisfy lossless join and dependency preserving decomposition? _____

Answer : As you can see UWY is candidate key and relation contains W → X. Which is partial dependency so relation is not in 2NF to make it in BCNF we will decompose the relation UWY into three relation So decomposition R_1(UWY), R_2(UV), R_3(WX) and R_4(YZ) is lossless and dependency preserving As we need to consider original relation, answer is 4.

Consider the following relations Enrollment and Course as given below:

Enrollment

Sid	Cid	Fee
S_1	C_1	1000
S_1	C_2	2000
S_1	C_3	3000
S_2	C_2	2000

Course

Cid	Cname	Instructor
C_1	DB	James
C_2	CN	Jacob
C_3	CO	Mathew
C_2	OS	David

If following relation algebra query is executed over above database table then number of records that present in result of the query are _____.

$\pi_{sid, cid}$ (Enrollment) / π_{cid} ($\sigma_{instructor = \text{"Samuel"}}$ (Course))

2 — Correct Option

Solution :
2
There is no instructor with the name "Samuel" in course.
So, all distinct sid's will be returned which are S_1 and S_2 only.

Your Answer is 4

Consider following relations:

$P(\underline{A}, B, C)$ and $Q(A, \underline{E}, F)$

P has 1000 records and Q has 2000 records. The non-null attribute 'A' in Q is referencing to attribute 'A' in P. Let X be the minimum number of records in $R_1 \bowtie R_2$ and Y be the maximum number of records in $R_1 \bowtie R_2$ then Y − X is _____.

0 — Correct Option

Solution :
0
'A' in Q is not key, so all value of A in Q may or may not be unique.
Therefore every entry of 'A' in Q will match with 'A' in P.
Hence maximum is 2000. But 'A' in Q is foreign key referencing 'A' in P.
So, minimum is also 2000
So, Y − X = 2000 − 2000 = 0

Your Answer is 999

16) Consider the following relational schema.

Students(rollno: integer, sname: string)
Courses(courseno: integer, cname: string)
Registration(rollno: integer, courseno: integer, percent: real)
Which of the following queries are equivalent to this query in English?
"Find the distinct names of all students who score more than 90% in the course numbered 107"

I) SELECT DISTINCT S.sname FROM Students as S, Registration as R WHERE R.rollno=S.rollno AND R.courseno=107 AND R.percent >90

II) Πsname(σcourseno=107∧percent>90(Registration⋈Students))

III) {T | ∃ S∈Students, ∃ R∈Registration(S.rollno=R.rollno∧R.courseno=107∧R.percent>90∧T.sname=S.sname)}

IV) {⟨SN⟩ | ∃ SR, RP(⟨SR,SN⟩∈Students∧⟨SR,107,RP⟩∈Registration∧RP>90)}

I, II, III and IV — *I, II and IV only*
I, II and III only — *II, III and IV only*

Answer : See each option carefully. All are correct.

17) Consider a relational table r with sufficient number of records, having attributes A1, A2,…, An and let 1 <= p <= n. Two queries Q1 and Q2 are given below.

Q1 : $\pi_{A1...A_n}(\sigma_{A_p=c}(r))$ where c is a const
Q2: $\pi_{A1...A_n}(\sigma_{c_1 \le A_p \le c_2}(r))$ where c_1 and c_2 are constants

The database can be configured to do ordered indexing on Ap or hashing on Ap. Which of the following statements is TRUE?

Ordered indexing will always outperform hashing for both queries — *Hashing will outperform ordered indexing on Q1, but not on Q2*
Hashing will always outperform ordered indexing for both queries — *Hashing will outperform ordered indexing on Q2, but not on Q1.*

Answer : here outperform means perform better. English is easy.
If records are accessed for a particular value from table, hashing will do better.
If records are accessed in a range of values, ordered indexing will perform better.

18) Which of the following query transformations (i.e., replacing the l.h.s. expression by the r.h.s. expression) is incorrect? R1 and R2 are relations. C1, C2 are selection conditions and A1, A2 are attributes of R1.

a. $\sigma C1(\sigma C2R1)) \rightarrow \sigma C2(\sigma C1(R1))$
b. $\sigma C1(\pi A1R1)) \rightarrow \pi A1(\sigma C1(R1))$
c. $\sigma C1(R1 \cup R2) \rightarrow \sigma C1(R1) \cup \sigma C1(R2)$
d. $\pi A1(\sigma C1(R1)) \rightarrow \sigma C1(\pi A1(R1))$

Answer : You're thinking were correct. Just read the question carefully. It says l.h.s expression by the r.h.s expression means l.h.s can be null, just like p→q. Imagine C1 is the condition on A2. In D option r.h.s. is null or not applicable because no A2 will be present after applying πA1.

SQL Questions **:**

1) Given the following statements:

S1: A foreign key declaration can always be replaced by an equivalent check assertion in SQL.

S2: Given the table R(a,b,c) where a and b together form the primary key, the following is a valid table definition.

```
CREATE TABLE S (
        a INTEGER,
        d INTEGER,
        e INTEGER,
        PRIMARY KEY (d),
        FOREIGN KEY (a) references R)
```

Which one of the following statements is CORRECT?

Answer : Both statements are false.

S1 : Using a check condition we can have the same effect as Foreign key while adding elements to the child table. But when we delete an element from the parent table the referential integrity constraint is no longer valid. So, a check constraint cannot replace a foreign key. So, we cannot replace it with a single check.

S2 : Foreign key in one table should uniquely identifies a row of another table. In above table definition, table S has a foreign key that refers to field 'a' of R. The field 'a' in table S doesn't uniquely identify a row in table R.

2)

Let R and S be relational schemes such that $R = \{a, b, c\}$ and $S = \{c\}$. Now consider the following queries on the database:

1. $\pi_{R-S}(r) - \pi_{R-S}\left(\pi_{R-S}(r) \times s - \pi_{R-S,S}(r)\right)$
2. $\{t \mid t \in \pi_{R-S}(r) \wedge \forall u \in s\left(\exists v \in r\left(u = v[S] \wedge t = v\left[R - S\right]\right)\right)\}$
3. $\{t \mid t \in \pi_{R-S}(r) \wedge \forall v \in r\left(\exists u \in s\left(u = v[S] \wedge t = v\left[R - S\right]\right)\right)\}$
4.
```
Select R.a,R.b
    From R,S
    Where R.c = S.c
```

Which of the above queries are equivalent?

A. 1 and 2
B. 1 and 3
C. 2 and 4
D. 3 and 4

1. $\pi_{R-S}(r) - \pi_{R-S}\left(\pi_{R-S}(r) \times s - \pi_{R-S,S}(r)\right)$
   $= \pi_{a,b}(r) - \pi_{a,b}\left(\pi_{a,b}(r) \times s - \pi_R(r)\right)$
   $= (r/s)$
2. Expanding logically the statement means to select $t(a, b)$ from r such that for all tuples u in s, there is a tuple v in r, such that $u = v[S]$ and $t = v[R - S]$. This is just equivalent to (r/s)
3. Expanding logically the statement means that select $t(a, b)$ from r such that for all tuples v in r, there is a tuple u in s, such that $u = v[S]$ and $t = v[R - S]$. This is equivalent to saying to select (a, b) values from r, where the c value is in some tuple of s.
4. This selects (a, b) from all tuples from r which has an equivalent c value in s.

, 1 and 2 are equivalent.

r

a	b	c
Arj	TY	12
Arj	TY	14
Cell	TR	13
Tom	TW	12
Ben	TE	14

s

c
12
14

1. will give $\langle Arj, TY\rangle$.
2. will give $\langle Arj, TY\rangle$.
3. will not return any tuple as the c value 13, is not in s.
4. will give $\langle Arj, TY\rangle, \langle Arj, TY\rangle, \langle Tom, TW\rangle, \langle Ben, TE\rangle$.

3) Consider the relation account (customer, balance) where customer is a primary key and there are no null values. We would like to rank customers according to decreasing balance. The customer with the largest balance gets rank 1. ties are not broke but ranks are skipped: if exactly two customers have the largest balance they each get rank 1 and rank 2 is not assigned

Query1:

```
select Acustomer, count(Bcustomer) from account A, account B where Abalance <=Bbalance  group by Acustomer
```

Query2:

```
select Acustomer, 1+count(Bcustomer)  from account A, account B  where Abalance < Bbalance group by Acustomer
```

Consider these statements about Query1 and Query2.

1. Query1 will produce the same row set as Query2 for some but not all databases.

2. Both Query1 and Query2 are correct implementation of the specification

3. Query1 is a correct implementation of the specification but Query2 is not

4. Neither Query1 nor Query2 is a correct implementation of the specification

5. Assigning rank with a pure relational query takes less time than scanning in decreasing balance order assigning ranks using ODBC.

Which two of the above statements are correct?

Answer : 1 AND 4. Both Query1 and Query2 are not correct implementations because: Assume that we have a table with n customers having the same balance. In that case Query1 will give rank n to each customer. But according to the question the rank assigned should be 1. And Query2 will return an empty result set (as it will never return rank 1). So, statement 4 is correct. For the same reason Query1 is wrong though it is true if we assume the relation set is empty.

4) Given relations r(w, x) and s(y, z), the result of "SELECT DISTINCT w, x FROM r, s" is guaranteed to be same as r, provided

A)r has no duplicates and s is non-empty

B)r and s have no duplicates

C)s has no duplicates and r is non-empty

D)r and s have the same number of tuples

Answer : The query selects all attributes of r. Since we have distinct in query, result can be equal to r only if r doesn't have duplicates. If we do not give any attribute on which we want to join two tables, then the queries like above become equivalent to Cartesian product. Cartesian product of two sets will be empty if any of the two sets is empty. So, s should have atleast one record to get all rows of r.

5) SELECT sch-name, COUNT (*) FROM School C, Enrolment E, ExamResult R WHERE E.school-id = C.school-id AND E.examname = R.examname AND E.erollno = R.erollno AND R.marks = 100 AND S.school-id IN (SELECT school-id FROM student GROUP BY school-id HAVING COUNT (*) > 200) GROUP By school-id

Answer : Syntex Error. In outer SQL query in SELECT sch-name is used where as in GROUP BY clause, school-id is used, that should be same as in SELECT clause.

6) An SQL does not remove duplicates like relational algebra projection, we have to remove it using distinct. An SQL will work slowly but surely if there are no indexes. An SQL does not permit 2 attributes of same name in a relation.

7) Given the following schema:

employees(emp-id, first-name, last-name, hire-date, dept-id, salary)

departments(dept-id, dept-name, manager-id, location-id)

You want to display the last names and hire dates of all latest hires in their respective departments in the location ID 1700. You issue the following query:

```
SQL>SELECT last-name, hire-date
   FROM employees
   WHERE (dept-id, hire-date) IN
   (SELECT dept-id, MAX(hire-date)
   FROM employees JOIN departments USING(dept-id)
   WHERE location-id =1700
   GROUP BY dept-id);
```

What is the outcome?

It executes but does not give the correct result

It executes and gives the correct result.

It generates an error because of pairwise comparison.

It generates an error because of the GROUP BY clause cannot be used with table joins in a sub-query.

Answer : This inner query will give the max hire date of each department whose location_id =1700 and outer query will give the last name and hire-date of all those employees who joined on max hire date. Answer should come to (B) no errors.

8) SQL allows tuples in relations, and correspondingly defines the multiplicity of tuples in the result of joins. Which one of the following queries always gives the same answer as the nested query shown below:

```
select * from R where a in (select S.a from S)
```

select R. from R, S where R.a=S.a (D)*

select distinct R. from R,S where R.a=S.a*

select R. from R,(select distinct a from S) as S1 where R.a=S1.a*

select R. from R,S where R.a=S.a and is unique R*

Answer :

R

A	B	C
1	2	3
1	2	3
7	8	9
7	8	9

S

A	X	Z
1	2	3
3	5	7
7	6	5
7	6	5

select * from R where a in (select S.a from S)

This query will result in same table as R. Because * means all attribute of R because from R. This same as **R.***. R.a means attribute now solve it is very easy. C is the answer.

9) Consider the relation "enrolled(student, course)" in which (student, course) is the primary key, and the relation "paid(student, amount)" where student is the primary key. Assume no null values and no foreign keys or integrity constraints. Given the following four queries:

```
Query1: select student from enrolled where
        student in (select student from paid)
Query2: select student from paid where
        student in (select student from enrolled)
Query3: select E.student from enrolled E, paid P
         where E.student = P.student
Query4:  select student from paid where exists
        (select * from enrolled where enrolled.student
         = paid.student)
```

Which one of the following statements is correct?

All queries return identical row sets for any database
Query2 and Query4 return identical row sets for all databases but there exist databases for which Query1 and Query2 return different row sets.
There exist databases for which Query3 returns strictly fewer rows than Query2
There exist databases for which Query4 will encounter an integrity violation at runtime.

Answer : read the question twice you missed one point i.e. primary key of both relation. Primary key of the enrolled are student and course. So, two students can have same name but course must be different now make example on your own make duplicate entry in enrolled table and solve all queries. B is answer.

10) Consider the set of relations shown below and the SQL query that follows.

```
Students: (Roll_number, Name, Date_of_birth)
 Courses: (Course number, Course_name, Instructor)
 Grades: (Roll_number, Course_number, Grade)
select distinct Name
    from Students, Courses, Grades
    where Students. Roll_number = Grades.Roll_number
       and Courses.Instructor = Korth
       and Courses.Course_number = Grades.Course_number
       and Grades.grade = A
```

Which of the following sets is computed by the above query?

Names of students who have got an A grade in all courses taught by Korth
Names of students who have got an A grade in all courses
Names of students who have got an A grade in at least one of the courses taught by Korth
None of the above

Answer : You have selected option A but it is wrong as we are not comparing all student marks for particular student so all should not be there in query but if you look closely course number = course number belong to grade means any course number other than korth will be there but at least one course of korth should be there as to satisfy second and condition so option c is correct.

11) In an inventory management system implemented at a trading corporation, there are several tables designed to hold all the information. Amongst these, the following two tables hold information on which items are supplied by which suppliers, and which warehouse keeps which items along with the stock-level of these items. Supply = (supplierid, itemcode) Inventory = (itemcode, warehouse, stocklevel) For a specific information required by the management, following SQL query has been written

```
Select distinct STMP.supplierid
From Supply as STMP
Where not unique (Select ITMP.supplierid
          From Inventory, Supply as ITMP
          Where STMP.supplierid = ITMP.supplierid
          And ITMP.itemcode = Inventory.itemcode
          And Inventory.warehouse = 'Nagpur');
```

For the warehouse at Nagpur, this query will find all suppliers who

do not supply any item
supply exactly one item
supply one or more items
supply two or more items

Answer : Not unique means it should not be unique and it means more than one so two items is base case as 2 is greater than 1. See unique is itself one so two must be number of items.

6. Basics of Functional Dependencies and Normalization for Relational Databases

- So far, we have assumed that attributes are grouped to form a relation schema by using the common sense of the database designer or by mapping a database schema design from a conceptual data model such as the ER or enhanced-ER (EER) data model.
- In this chapter we discuss some of the theory that has been developed with the goal of evaluating relational schemas for design quality—that is, to measure formally why one set of groupings of attributes into relation schemas is better than another.

1) Informal Design Guidelines for Relation Schemas :

Before discussing the formal theory of relational database design, we discuss four informal guidelines that may be used as measures to determine the quality of relation schema design:

- Making sure that the semantics of the attributes is clear in the schema
- Reducing the redundant information in tuples
- Reducing the NULL values in tuples
- Disallowing the possibility of generating spurious tuples.

Whenever we group attributes to form a relation schema, we assume that attributes belonging to one relation have certain real-world meaning and a proper interpretation associated with them. The semantics of a relation refers to its meaning resulting from the interpretation of attribute values in a tuple.

The ease with which the meaning of a relation's attributes can be explained is an informal measure of how well the relation is designed.

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. Intuitively, if a relation schema corresponds to one entity type or one relationship type, it is straightforward to explain its meaning. Otherwise, if the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained.

Examples of Violating Guideline 1.:

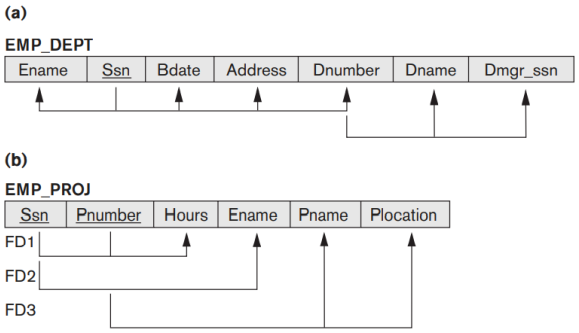


Figure 14.3
Two relation schemas suffering from update anomalies.
(a) EMP_DEPT and
(b) EMP_PROJ.

Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

Redundancy Redundancy

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

Figure 14.4
Sample states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 14.2. These may be stored as base relations for performance reasons.

They violate Guideline 1 by mixing attributes from distinct real-world entities: EMP_DEPT mixes attributes of employees and departments, and EMP_PROJ mixes attributes of employees and projects and the WORKS_ON relationship. Hence, they fare poorly against the above measure of design quality. They may be used as views, but they cause problems when used as base relations, as we discuss in the following section.

One goal of schema design is to minimize the storage space used by the base relations (and hence the corresponding files). Grouping attributes into relation schemas has a significant effect on storage space. For example, compare the space used by the two base relations EMPLOYEE and DEPARTMENT in Figure 14.2 with that for an EMP_DEPT base relation in Figure 14.4, which is the result of applying the NATURAL JOIN operation to EMPLOYEE and DEPARTMENT.

Storing natural joins of base relations leads to an additional problem referred to as update anomalies. These can be classified into insertion anomalies, deletion anomalies, and modification anomalies.

Insertion Anomalies. Insertion anomalies can be differentiated into two types, illustrated by the following examples based on the EMP_DEPT relation:

- To insert a new employee tuple into EMP_DEPT, we must include either the attribute values for the department that the employee works for, or NULLs (if the employee does not work for a department as yet).

■ It is difficult to insert a new department that has no employees as yet in the EMP_DEPT relation. The only way to do this is to place NULL values in the attributes for employee. This violates the entity integrity for EMP_DEPT because its primary key Ssn cannot be null.

Deletion Anomalies. The problem of deletion anomalies is related to the second insertion anomaly situation just discussed. 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 inadvertently 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; otherwise, the database will become inconsistent.

It is easy to see that these three anomalies are undesirable and cause difficulties to maintain consistency of data as well as require unnecessary updates that can be avoided; hence, we can state the next guideline as follows.

Guideline 2. Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correctly.

NULL Values in Tuples :

In some schema designs we may group many attributes together into a "fat" relation. If many of the attributes do not apply to all tuples in the relation, we end up with many NULLs in those tuples. This can waste space at the storage level and may also lead to problems with understanding the meaning of the attributes and with specifying JOIN operations at the logical level.

Guideline 3. As far as possible, avoid placing attributes in a base relation whose values may frequently be NULL. If NULLs are unavoidable, make sure that they apply in exceptional cases only and do not apply to a majority of tuples in the relation.

Using space efficiently and avoiding joins with NULL values are the two overriding criteria that determine whether to include the columns that may have NULLs in a relation or to have a separate relation for those columns (with the appropriate key columns).

Generation of Spurious Tuples :

Suppose that we used EMP_PROJ1 and EMP_LOCS as the base relations instead of EMP_PROJ. This produces a particularly bad schema design because we cannot recover the information that was originally in EMP_PROJ from EMP_PROJ1 and EMP_LOCS. If we attempt a NATURAL JOIN operation on EMP_PROJ1 and EMP_LOCS, the result produces many more tuples than the original set of tuples in EMP_PROJ.

Additional tuples that were not in EMP_PROJ are called spurious tuples because they represent spurious information that is not valid. The spurious tuples are marked by asterisks (*) in Figure 14.6.

	Ssn	Pnumber	Hours	Pname	Plocation	Ename
	123456789	1	32.5	ProductX	Bellaire	Smith, John B.
*	123456789	1	32.5	ProductX	Bellaire	English, Joyce A.
	123456789	2	7.5	ProductY	Sugarland	Smith, John B.
*	123456789	2	7.5	ProductY	Sugarland	English, Joyce A.
*	123456789	2	7.5	ProductY	Sugarland	Wong, Franklin T.
	666884444	3	40.0	ProductZ	Houston	Narayan, Ramesh K.
*	666884444	3	40.0	ProductZ	Houston	Wong, Franklin T.
*	453453453	1	20.0	ProductX	Bellaire	Smith, John B.
	453453453	1	20.0	ProductX	Bellaire	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Smith, John B.
	453453453	2	20.0	ProductY	Sugarland	English, Joyce A.
*	453453453	2	20.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	2	10.0	ProductY	Sugarland	Smith, John B.
*	333445555	2	10.0	ProductY	Sugarland	English, Joyce A.
	333445555	2	10.0	ProductY	Sugarland	Wong, Franklin T.
*	333445555	3	10.0	ProductZ	Houston	Narayan, Ramesh K.
	333445555	3	10.0	ProductZ	Houston	Wong, Franklin T.
	333445555	10	10.0	Computerization	Stafford	Wong, Franklin T.
*	333445555	20	10.0	Reorganization	Houston	Narayan, Ramesh K.
	333445555	20	10.0	Reorganization	Houston	Wong, Franklin T.

* * *

Figure 14.6
Result of applying NATURAL JOIN to the tuples in EMP_PROJ1 and EMP_LOCS of Figure 14.5 just for employee with Ssn = "123456789". Generated spurious tuples are marked by asterisks.

Guideline 4. Design relation schemas so that they can be joined with equality conditions on attributes that are appropriately related (primary key, foreign key) pairs in a way that guarantees that no spurious tuples are generated. Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations because joining on such attributes may produce spurious tuples.

Summery : *The problems we pointed out, which can be detected without additional tools of analysis, are as follows:*

■ *Anomalies that cause redundant work to be done during insertion into and modification of a relation, and that may cause accidental loss of information during a deletion from a relation*

■ *Waste of storage space due to NULLs and the difficulty of performing selections, aggregation operations, and joins due to NULL values*

■ *Generation of invalid and spurious data during joins on base relations with matched attributes that may not represent a proper (foreign key, primary key) relationship.*

2) Definition of Functional Dependency : Suppose that our relational database schema has n attributes A1, A2, … , An; let us think of the whole database as being described by a single universal relation schema R = {A1, A2, … , An}.

Definition. *A functional dependency, denoted by X → Y, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation state r of R. The constraint is that, for any two tuples t1 and t2 in r that have t1[X] = t2[X], they must also have t1[Y] = t2[Y].*

This means that the values of the Y component of a tuple in r depend on, or are determined by, the values of the X component; alternatively, the values of the X component of a tuple uniquely (or functionally) determine the values of the Y component. We also say that there is a functional dependency from X to Y, or that Y is functionally dependent on X. The abbreviation for functional dependency is FD or f.d. The set of attributes X is called the left-hand side of the FD, and Y is called the right-hand side.

Note the following:

■ If a constraint on R states that there cannot be more than one tuple with a given X-value in any relation instance r(R)—that is, X is a candidate key of R—this implies that X → Y for any subset of attributes Y of R (because the key constraint implies that no two tuples in any legal state r(R) will have the same value of X). If X is a candidate key of R, then X → R.

■ If X → Y in R, this does not say whether or not Y → X in R.

A functional dependency is a property of the semantics or meaning of the attributes. Relation extensions r(R) that satisfy the functional dependency constraints are called legal relation states (or legal extensions) of R. It is, however, sufficient to demonstrate a single counterexample to disprove a functional dependency. For example, because 'Smith' teaches both 'Data Structures' and 'Database Systems,' we can conclude that Teacher does not functionally determine Course.

TEACH

Teacher	Course	Text
Smith	Data Structures	Bartram
Smith	Data Management	Martin
Hall	Compilers	Hoffman
Brown	Data Structures	Horowitz

Figure 14.7
A relation state of TEACH with a *possible* functional dependency TEXT → COURSE. However, TEACHER → COURSE, TEXT → TEACHER and COURSE → TEXT are ruled out.

Figure 14.8
A relation *R*(A, B, C, D) with its extension.

A	B	C	D
a1	b1	c1	d1
a1	b2	c2	d2
a2	b2	c2	d3
a3	b3	c4	d3

Figure 14.8. Here, the following FDs may hold because the four tuples in the current extension have no violation of these constraints: B → C; C → B; {A, B} → C; {A, B} → D; and {C, D} → B. However, the following do not hold because we already have violations of them in the given extension: A → B (tuples 1 and 2 violate this constraint); B → A (tuples 2 and 3 violate this constraint); D → C (tuples 3 and 4 violate it).

3) Normal Forms Based on Primary Keys :

Most practical relational design projects take one of the following two approaches:

■ Perform a conceptual schema design using a conceptual model such as ER or EER and map the conceptual design into a set of relations.

■ Design the relations based on external knowledge derived from an existing implementation of files or forms or reports.

Normalization of data can be considered a process of analysing the given relation schemas based on their FDs and primary keys to achieve the desirable properties of (1) minimizing redundancy and (2) minimizing the insertion, deletion, and update anomalies discussed in Section 14.1.2.

The normalization procedure provides database designers with the following:

■ A formal framework for analysing relation schemas based on their keys and on the functional dependencies among their attributes

■ A series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desired degree Definition.

The normal form of a relation refers to the highest normal form condition that it meets, and hence indicates the degree to which it has been normalized.

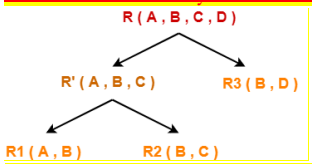
These would include two properties:

■ The nonadditive join or lossless join property, which guarantees that the spurious tuple generation problem discussed in Section 14.1.4 does not occur with respect to the relation schemas created after decomposition.

■ The dependency preservation property, which ensures that each functional dependency is represented in some individual relation resulting after decomposition The nonadditive join property is extremely critical and must be achieved at any cost, whereas the dependency preservation property, although desirable, is sometimes sacrificed.

⇨ **Decomposition of relation** (lossless and lossy) :

Decomposition is a process of dividing a single relation into two or more sub relations.
1) lossless join decomposition
2) Lossy join decomposition
Consider a relation R is decomposed into two sub relations R1 and R2. Then, **If all the following conditions satisfy, then the decomposition is lossless.**
If any of these conditions fail, then the decomposition is lossy or not lossless.
Condition 1 : R1 ∪ R2 = R
Condition 2 : R1 ∩ R2 ≠ ∅
Condition 3 : R1 ∩ R2 = Super key of R1 or R2. Super key means it can determine all the attributes of particular relation through functional dependencies.
Thumb rule : Any relation can be decomposed only into two sub relations at a time.



Here we first check for 1st level decomposition and then for both leaves.

People do not use Higher Normal forms, The reason is that the constraints on which they are based are rare and hard for the database designers and users to understand or to detect. Designers and users must either already know them or discover them as a part of the business.

Definition. Denormalization is the process of storing the join of higher normal form relations as a base relation, which is in a lower normal form.
Definition. ***A superkey of a relation schema R = {A1, A2, … , An} is a set of attributes S ⊆ R with the property that no two tuples t1 and t2 in any legal relation state r of R will have t1[S] = t2[S]. A key K is a superkey with the additional property that removal of any attribute from K will cause K not to be a superkey anymore.***
The difference between a key and a superkey is that a key has to be minimal; that is, if we have a key K = {A1, A2, … , Ak} of R, then K − {Ai} is not a key of R for any Ai, 1 ≤ i ≤ k. In Figure 14.1, {Ssn} is a key for EMPLOYEE, whereas {Ssn}, {Ssn, Ename}, {Ssn, Ename, Bdate}, and any set of attributes that includes Ssn are all superkeys.
If a relation schema has more than one key, each is called a candidate key. A set of minimal attributes(s) that can identify each tuple uniquely in the given relation is called as a candidate key. **OR** A minimal super key is called as a candidate key. One of the candidate keys is arbitrarily designated to be the primary key, and the others are called secondary keys. In a practical relational database, each relation schema must have a primary key. If no candidate key is known for a relation, the entire relation can be treated as a default superkey. In Figure 14.1, {Ssn} is the only candidate key for EMPLOYEE, so it is also the primary key.
Number of possible candidate keys with n attributes is C(n, floor(n/2)).
Definition. An attribute of relation schema R is called a prime attribute of R if it is a member of some candidate key of R. An attribute is called nonprime if it is not a prime attribute—that is, if it is not a member of any candidate key. In Figure 14.1, both Ssn and Pnumber are prime attributes of WORKS_ON, whereas other attributes of WORKS_ON are nonprime.

⇨ ***Finding superkeys and candidate keys*** :

- Determine all essential attributes of the given relation.
- Essential attributes are those attributes which are not present on RHS of any functional dependency.
- Essential attributes are always a part of every candidate key.
- This is because they can not be determined by other attributes.

Example : Let R = (A, B, C, D, E, F) be a relation scheme with the following dependencies-
C → F
E → A
EC → D
A → B
Answer: essential attributes are C,E because none of them are in RHS. next we see if both or combination of both can determine all other attributes. and yes they can therefore, CE are candidate keys. total keys are 6 and CE are essential means they are part of every keys. 4 attributes are non-essential means they may or may not be part of keys so there is 2^4 possible super keys are possible.

1st Normal Forms : 1NF disallows having a set of values, a tuple of values, or a combination of both as an attribute value for a single tuple. In other words, 1NF disallows relations within relations or relations as attribute values within tuples.

(a)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations

(b)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocations
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c)

DEPARTMENT

Dname	Dnumber	Dmgr_ssn	Dlocation
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

Figure 14.9
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Sample state of relation DEPARTMENT. (c) 1NF version of the same relation with redundancy.

(a)

EMP_PROJ

Ssn	Ename	Projs	
		Pnumber	Hours

(b)

EMP_PROJ

Ssn	Ename	Pnumber	Hours
123456789	Smith, John B.	1	32.5
		2	7.5
666884444	Narayan, Ramesh K.	3	40.0
453453453	English, Joyce A.	1	20.0
		2	20.0
333445555	Wong, Franklin T.	2	10.0
		3	10.0
		10	10.0
		20	10.0
999887777	Zelaya, Alicia J.	30	30.0
		10	10.0
987987987	Jabbar, Ahmad V.	10	35.0
		30	5.0
987654321	Wallace, Jennifer S.	30	20.0
		20	15.0
888665555	Borg, James E.	20	NULL

(c)

EMP_PROJ1

Ssn	Ename

EMP_PROJ2

Ssn	Pnumber	Hours

Figure 14.10
Normalizing nested relations into 1NF. (a) Schema of the EMP_PROJ relation with a *nested relation* attribute PROJS. (b) Sample extension of the EMP_PROJ relation showing nested relations within each tuple. (c) Decomposition of EMP_PROJ into relations EMP_PROJ1 and EMP_PROJ2 by propagating the primary key.

As we can see, this is not in 1NF because Dlocations is not an atomic attribute, as illustrated by the first tuple in Figure 14.9(b). There are two ways we can look at the Dlocations attribute:

■ The domain of Dlocations contains atomic values, but some tuples can have a set of these values. In this case, Dlocations is not functionally dependent on the primary key Dnumber.

■ The domain of Dlocations contains sets of values and hence is nonatomic. In this case, Dnumber → Dlocations because each set is considered a single member of the attribute domain.

There are three main techniques to achieve first normal form for such a relation:

1. Remove the attribute Dlocations that violates 1NF and place it in a separate relation DEPT_LOCATIONS along with the primary key Dnumber of DEPARTMENT.
2. Expand the key so that there will be a separate tuple in the original DEPARTMENT relation for each location of a DEPARTMENT, as shown in Figrure 14.9(c). In this case, the primary key becomes the combination {Dnumber, Dlocation}. This solution has the disadvantage of introducing redundancy in the relation and hence is rarely adopted.
3. If a maximum number of values is known for the attribute—for example, if it is known that at most three locations can exist for a department—replace the Dlocations attribute by three atomic attributes: Dlocation1, Dlocation2, and Dlocation3. This solution has the disadvantage of introducing NULL values (spurious semantics).

First normal form also disallows multivalued attributes that are themselves composite. These are called nested relations because each tuple can have a relation within it. Figure 14.10 shows how the EMP_PROJ relation could appear if nesting is allowed.
EMP_PROJ(Ssn, Ename, {PROJS(Pnumber, Hours)})

To normalize this into 1NF, we remove the nested relation attributes into a new relation and propagate the primary key into it; the primary key of the new relation will combine the partial key with the primary key of the original relation. Decomposition and primary key propagation yield the schemas EMP_PROJ1 and EMP_PROJ2, as shown in Figure 14.10(c).

Example :

1) CANDIDATE (Ssn, Name, {JOB_HIST (Company, Highest_position, {SAL_HIST (Year, Max_sal)})})

Solution : The first normalization using internal partial keys Company and Year, respectively, results in the following 1NF relations:
CANDIDATE_1 (Ssn, Name)
CANDIDATE_JOB_HIST (Ssn, Company, Highest_position)
CANDIDATE_SAL_HIST (Ssn, Company, Year, Max-sal)

2) consider the following non-1NF relation: PERSON (Ss#, {Car_lic#}, {Phone#})

Solution : PERSON_IN_1NF (Ss#, Car_lic#, Phone#) <= **WRONG**

⇨ **Correct one** : P1(Ss#, Car_lic#) and P2(Ss#, Phone#).

When images, videos, text are stored in a relation, the entire object or file is treated as an atomic value, which is stored as a BLOB (binary large object) or CLOB (character large object) data type using SQL. For practical purposes, the object is treated as an atomic, single-valued attribute and hence it maintains the 1NF status of the relation.

Second Normal Form :

Second normal form (2NF) is based on the concept of full functional dependency. A functional dependency X → Y is a full functional dependency if removal of any attribute A from X means that the dependency does not hold anymore; that is, for any attribute A ε X, (X − {A}) does not functionally determine Y.

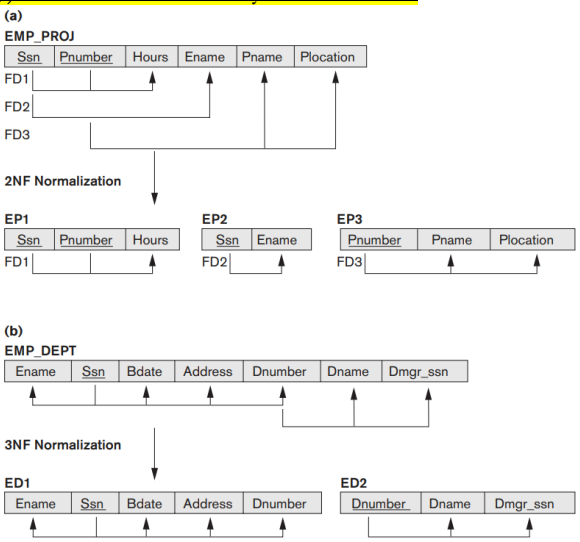
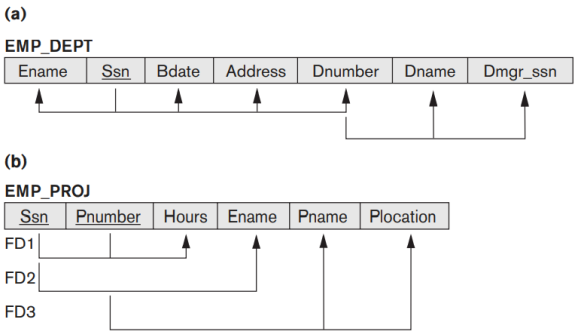


Figure 14.11
Normalizing into 2NF and 3NF. (a) Normalizing EMP_PROJ into 2NF relations. (b) Normalizing EMP_DEPT into 3NF relations.

Figure 14.3
Two relation schemas suffering from update anomalies. (a) EMP_DEPT and (b) EMP_PROJ.



In Figure 14.3(b), {Ssn, Pnumber} → Hours is a full dependency (neither Ssn → Hours nor Pnumber → Hours holds). However, the dependency {Ssn, Pnumber} → Ename is partial because Ssn → Ename holds.

Definition: A relation schema R is in 2NF if every nonprime attribute A in R is fully functionally dependent on the primary key of R.

The test for 2NF involves testing for functional dependencies whose left-hand side attributes are part of the primary key. If the primary key contains a single attribute, the test need not be applied at all. The EMP_PROJ relation in Figure 14.3(b) is in 1NF but is not in 2NF. The nonprime attribute Ename violates 2NF because of FD2, as do the nonprime attributes Pname and Plocation because of FD3. Each of the functional dependencies FD2 and FD3 violates 2NF because Ename can be functionally determined by only Ssn, and both Pname and Plocation can be functionally determined by only Pnumber. Attributes Ssn and Pnumber are a part of the primary key {Ssn, Pnumber} of EMP_PROJ, thus violating the 2NF test.

If a relation schema is not in 2NF, it can be second normalized or 2NF normalized into a number of 2NF relations in which nonprime attributes are associated only with the part of the primary key on which they are fully functionally dependent. Therefore, the functional dependencies FD1, FD2, and FD3 in Figure 14.3(b) lead to the decomposition of EMP_PROJ into the three relation schemas EP1, EP2, and EP3 shown in Figure 14.11(a), each of which is in 2NF.

Third Normal Forms :

Third normal form (3NF) is based on the concept of transitive dependency. A functional dependency X → Y in a relation schema R is a transitive dependency if there exists a set of attributes Z in R that is neither a candidate key nor a subset of any key of R, 11 and both X → Z and Z → Y hold. The dependency Ssn → Dmgr_ssn is transitive through Dnumber in EMP_DEPT in Figure 14.3(a), because both the dependencies Ssn → Dnumber and Dnumber → Dmgr_ssn hold and Dnumber is neither a key itself nor a subset of the key of EMP_DEPT. Intuitively, we can see that the dependency of Dmgr_ssn on Dnumber is undesirable in EMP_DEPT since Dnumber is not a key of EMP_DEPT.

Definition. *According to Codd's original definition, a relation schema R is in 3NF if it satisfies 2NF and no nonprime attribute of R is transitively dependent on the primary key.*

The relation schema EMP_DEPT in Figure 14.3(a) is in 2NF, since no partial dependencies on a key exist. However, EMP_DEPT is not in 3NF because of the transitive dependency of Dmgr_ssn (and also Dname) on Ssn via Dnumber. We can normalize EMP_DEPT by decomposing it into the two 3NF relation schemas ED1 and ED2 shown in Figure 14.11(b).

Table 14.1 Summary of Normal Forms Based on Primary Keys and Corresponding Normalization

Normal Form	Test	Remedy (Normalization)
First (1NF)	Relation should have no multivalued attributes or nested relations.	Form new relations for each multivalued attribute or nested relation.
Second (2NF)	For relations where primary key contains multiple attributes, no nonkey attribute should be functionally dependent on a part of the primary key.	Decompose and set up a new relation for each partial key with its dependent attribute(s). Make sure to keep a relation with the original primary key and any attributes that are fully functionally dependent on it.
Third (3NF)	Relation should not have a nonkey attribute functionally determined by another nonkey attribute (or by a set of nonkey attributes). That is, there should be no transitive dependency of a nonkey attribute on the primary key.	Decompose and set up a relation that includes the nonkey attribute(s) that functionally determine(s) other nonkey attribute(s).

Intuitively, we can see that any functional dependency in which the left-hand side is part (a proper subset) of the primary key, or any functional dependency in which the left-hand side is a nonkey attribute, is a problematic FD.

General Definitions of Second and Third Normal Forms :

Task: In general, we want to design our relation schemas so that they have neither partial nor transitive dependencies because these types of dependencies cause the update anomalies.

As a general definition of prime attribute, an attribute that is part of any candidate key will be considered as prime. Partial and full functional dependencies and transitive dependencies will now be considered with respect to all candidate keys of a relation.

1) ***General Definition of Second Normal Form***:

Definition. A relation schema R is in second normal form (2NF) if every nonprime attribute A in R is not partially dependent on any key of R.

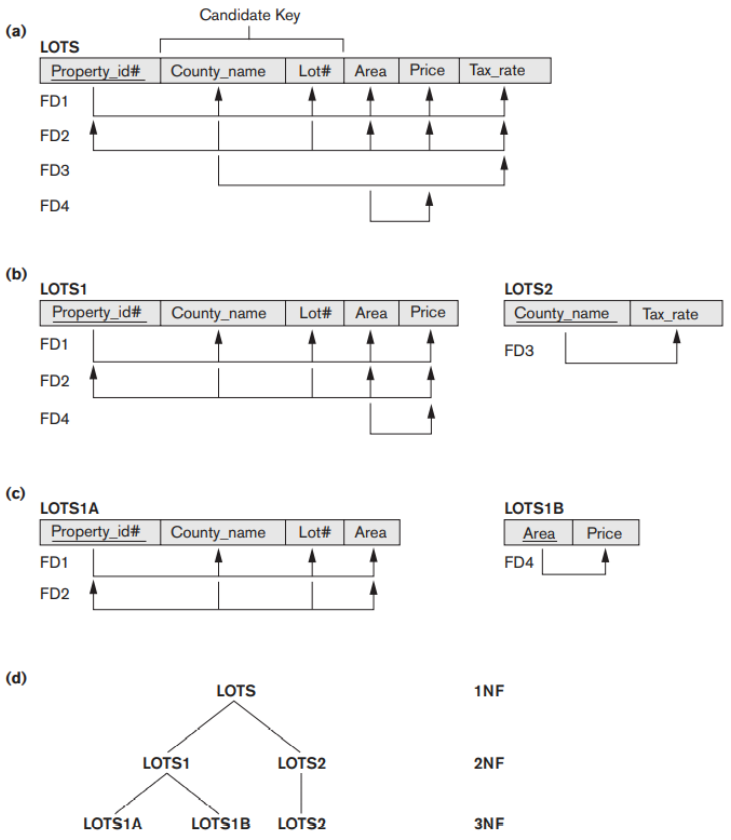
The LOTS relation schema violates the general definition of 2NF because Tax_rate is partially dependent on the candidate key {County_name, Lot#}, due to FD3. To normalize LOTS into 2NF, we decompose it into the two relations LOTS1 and LOTS2, shown in Figure 14.12(b). We construct LOTS1 by removing the attribute Tax_rate that violates 2NF from LOTS and placing it with County_name (the left-hand side of FD3 that causes the partial dependency) into another relation LOTS2. Both LOTS1 and LOTS2 are in 2NF. Notice that FD4 does not violate 2NF and is carried over to LOTS1.

2) ***General Definition of Third Normal Form*** **:**

Definition. A relation schema R is in third normal form (3NF) if, whenever a nontrivial functional dependency X → A holds in R, either (a) X is a superkey of R, or (b) A is a prime attribute of R.

According to this definition, LOTS2 (Figure 14.12(b)) is in 3NF. However, FD4 in LOTS1 violates 3NF because Area is not a superkey and Price is not a prime attribute in LOTS1. To normalize LOTS1 into 3NF, we decompose it into the relation schemas LOTS1A and LOTS1B shown in Figure 14.12(c). We construct LOTS1A by removing the attribute Price that violates 3NF from LOTS1 and placing it with Area (the left-hand side of FD4 that causes the transitive dependency) into another relation LOTS1B. Both LOTS1A and LOTS1B are in 3NF.

Figure 14.12
Normalization into 2NF and 3NF. (a) The LOTS relation with its functional dependencies FD1 through FD4. (b) Decomposing into the 2NF relations LOTS1 and LOTS2. (c) Decomposing LOTS1 into the 3NF relations LOTS1A and LOTS1B. (d) Progressive normalization of LOTS into a 3NF design.



Two points are worth noting about this example and the general definition of 3NF:

■ LOTS1 violates 3NF because Price is transitively dependent on each of the candidate keys of LOTS1 via the nonprime attribute Area.

■ This general definition can be applied directly to test whether a relation schema is in 3NF; it does not have to go through 2NF first. In other words, if a relation passes the general 3NF test, then it automatically passes the 2NF test.

If we apply the above 3NF definition to LOTS with the dependencies FD1 through FD4, we find that both FD3 and FD4 violate 3NF by the general definition above because the LHS County_name in FD3 is not a superkey. Therefore, we could decompose LOTS into LOTS1A, LOTS1B, and LOTS2 directly. Hence, the transitive and partial dependencies that violate 3NF can be removed in any order.

⇨ A relation schema R violates the general definition of 3NF if a functional dependency X → A holds in R that meets either of the two conditions, namely (a) and (b). The first condition "catches" two types of problematic dependencies:

■ A nonprime attribute determines another nonprime attribute. Here we typically have a transitive dependency that violates 3NF.

■ A proper subset of a key of R functionally determines a nonprime attribute. Here we have a partial dependency that violates 2NF.

⇨ Alternative Definition. A relation schema R is in 3NF if every nonprime attribute of R meets both of the following conditions:

■ It is fully functionally dependent on every key of R.

■ It is non-transitively dependent on every key of R.

Boyce-Codd Normal Form :

Boyce-Codd normal form (BCNF) was proposed as a simpler form of 3NF, but it was found to be stricter than 3NF. That is, every relation in BCNF is also in 3NF; however, a relation in 3NF is not necessarily in BCNF.

Definition. A relation schema R is in BCNF if whenever a nontrivial functional dependency X → A holds in R, then X is a superkey of R.

The formal definition of BCNF differs from the definition of 3NF in that clause (b) of 3NF, which allows f.d.'s having the RHS as a prime attribute, is absent from BCNF. That makes BCNF a stronger normal form compared to 3NF. In our example, FD5 violates BCNF in LOTS1A because Area is not a superkey of LOTS1A. We can decompose LOTS1A into two BCNF relations LOTS1AX and LOTS1AY, shown in Figure 14.13(a). This decomposition loses the functional dependency FD2 because its attributes no longer coexist in the same relation after decomposition.

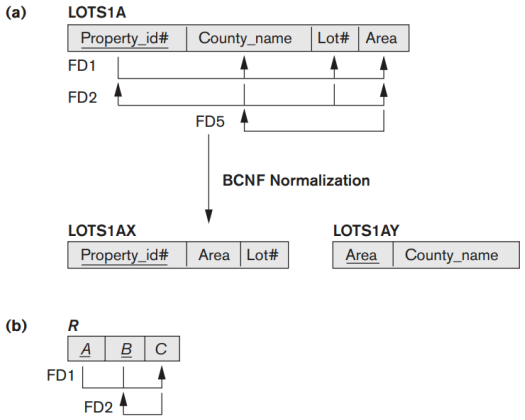


Figure 14.13
Boyce-Codd normal form. (a) BCNF normalization of LOTS1A with the functional dependency FD2 being lost in the decomposition. (b) A schematic relation with FDs; it is in 3NF, but not in BCNF due to the f.d. $C \rightarrow B$.

TEACH

Student	Course	Instructor
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe
Narayan	Operating Systems	Ammar

Figure 14.14
A relation TEACH that is in 3NF but not BCNF.

As another example, consider Figure 14.14, which shows a relation TEACH with the following dependencies:

FD1: {Student, Course} → Instructor

FD2: Instructor → Course

Note that {Student, Course} is a candidate key for this relation and that the dependencies shown follow the pattern in Figure 14.13(b), with Student as A, Course as B, and Instructor as C. Hence this relation is in 3NF but not BCNF.

1. R1 (Student, Instructor) and R2(Student, Course)
2. R1 (Course, Instructor) and R2(Course, Student)
3. R1 (Instructor, Course) and R2(Instructor, Student)

All three decompositions lose the functional dependency FD1. The question then becomes: Which of the above three is a desirable decomposition?

A simple test comes in handy to test the binary decomposition of a relation into two relations:

NJB (Nonadditive Join Test for Binary Decompositions). A decomposition D = {R1, R2} of R has the lossless (nonadditive) join property with respect to a set of functional dependencies F on R if and only if either

- The FD ((R1 ∩ R2) → (R1 − R2)) is in F+, or
- The FD ((R1 ∩ R2) → (R2 − R1)) is in F+

Hence, the proper decomposition of TEACH into BCNF relations is: TEACH1 (Instructor, Course) and TEACH2 (Instructor, Student)

Note : If relation is 3NF and not BCNF then below conditions are true.

- *Proper subset of candidate keys → proper subset of other candidate keys* must be present.
- There will be overlapped candidate keys means consider candidate key (AB, BC) here B is overlapped key.

If relation is 3NF and no overlapped candidate keys satisfy then *proper subset of candidate keys → proper subset of other candidate keys* this FD does not exists. This also implies that R is also BCNF.

Multivalued Dependency and Fourth Normal Form :

The relation EMP is an all-key relation (with key made up of all attributes) and therefore has no f.d.'s and as such qualifies to be a BCNF relation.

(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	Part_name	Proj_name
Smith	Bolt	ProjX
Smith	Nut	ProjY
Adamsky	Bolt	ProjY
Walton	Nut	ProjZ
Adamsky	Nail	ProjX
Adamsky	Bolt	ProjX
Smith	Bolt	ProjY

(d) R_1

Sname	Part_name
Smith	Bolt
Smith	Nut
Adamsky	Bolt
Walton	Nut
Adamsky	Nail

R_2

Sname	Proj_name
Smith	ProjX
Smith	ProjY
Adamsky	ProjY
Walton	ProjZ
Adamsky	ProjX

R_3

Part_name	Proj_name
Bolt	ProjX
Nut	ProjY
Bolt	ProjY
Nut	ProjZ
Nail	ProjX

Figure 14.15
Fourth and fifth normal forms.
(a) The EMP relation with two MVDs: Ename →→ Pname and Ename →→ Dname.
(b) Decomposing the EMP relation into two 4NF relations EMP_PROJECTS and EMP_DEPENDENTS.
(c) The relation SUPPLY with no MVDs is in 4NF but not in 5NF if it has the JD(R_1, R_2, R_3).
(d) Decomposing the relation SUPPLY into the 5NF relations R_1, R_2, R_3.

As illustrated by the EMP relation, some relations have constraints that cannot be specified as functional dependencies and hence are not in violation of BCNF. To address this situation, the concept of multivalued dependency (MVD) was proposed and, based on this dependency, the fourth normal form was defined.

Whenever X →→ Y holds, we say that X multidetermines Y. Because of the symmetry in the definition, whenever X →→ Y holds in R, so does X →→ Z. Hence, X →→ Y implies X →→ Z and therefore it is sometimes written as X →→ Y|Z.

An MVD X →→ Y in R is called a trivial MVD if (a) Y is a subset of X, or (b) X ∪ Y = R. For example, the relation EMP_PROJECTS in Figure 14.15(b) has the trivial MVD Ename →→ Pname and the relation EMP_DEPENDENTS has the trivial MVD Ename →→ Dname. An MVD that satisfies neither (a) nor (b) is called a nontrivial MVD. A trivial MVD will hold in any relation state r of R; it is called trivial because it does not specify any significant or meaningful constraint on R.

Notice that relations containing nontrivial MVDs tend to be all-key relations—that is, their key is all their attributes taken together. We now present the definition of fourth normal form (4NF), which is violated when a relation has undesirable multivalued dependencies and hence can be used to identify and decompose such relations.

Definition. A relation schema R is in 4NF with respect to a set of dependencies F (that includes functional dependencies and multivalued dependencies) if, for every nontrivial multivalued dependency X →→ Y in F+, X is a superkey for R.

We can state the following points:

■ An all-key relation is always in BCNF since it has no FDs.

■ An all-key relation such as the EMP relation in Figure 14.15(a), which has no FDs but has the MVD Ename →→ Pname | Dname, is not in 4NF.

■ A relation that is not in 4NF due to a nontrivial MVD must be decomposed to convert it into a set of relations in 4NF.

■ The decomposition removes the redundancy caused by the MVD.

The process of normalizing a relation involving the nontrivial MVDs that is not in 4NF consists of decomposing it so that each MVD is represented by a separate relation where it becomes a trivial MVD.

3) Relational Database Design Algorithms and Further Dependencies :

1) Further Topics in Functional Dependencies: Inference Rules, Equivalence, and Minimal Cover :

A) Inference Rules for Functional Dependencies :

Definition: An FD X → Y is inferred from or implied by a set of dependencies F specified on R if X → Y holds in every legal relation state r of R; that is, whenever r satisfies all the dependencies in F, X → Y also holds in r.

B) We consider some of these inference rules next. We use the notation **F |=X → Y** to denote that the functional dependency X → Y is inferred from the set of functional dependencies F.

Armstrong's axioms. (Sound and Complete)

IR1 (reflexive rule) : If X ⊇ Y, then X →Y.

IR2 (augmentation rule) : {X → Y} |=XZ → YZ.

IR3 (transitive rule): {X → Y, Y → Z} |=X → Z.

Note : By **sound** : we mean 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 in every relation state r of R that satisfies the dependencies in F.

By **complete**, we mean that using IR1 through IR3 repeatedly 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.

- Formally, a functional dependency X → Y is trivial if X ⊇ Y; otherwise, it is nontrivial.
  There are three other inference rules that follow from IR1, IR2 and IR3. They are as follows:
  IR4 (decomposition, or projective, rule): {X → YZ} |=X → Y.
  IR5 (union, or additive, rule): {X → Y, X → Z} |=X → YZ.
  IR6 (pseudotransitive rule): {X → Y, WY → Z} |=WX → Z.
- **Definition.** For each such set of attributes X, we determine the set X+ of attributes that are functionally determined by X based on F; X+ is called the **closure of X under F.**
  Example :
  CLASS (Classid, Course#, Instr_name, Credit_hrs, Text, Publisher, Classroom, Capacity).
  Let F, the set of functional dependencies for the above relation include the following f.d.s:
  FD1: Sectionid → Course#, Instr_name, Credit_hrs, Text, Publisher, Classroom, Capacity;
  FD2: Course# → Credit_hrs;
  FD3: {Course#, Instr_name} → Text, Classroom;
  FD4: Text → Publisher
  FD5: Classroom → Capacity

⇨ *Answer :* { Classid } + = { Classid , Course#, Instr_name, Credit_hrs, Text, Publisher, Classroom, Capacity } = CLASS
{ Course#} + = { Course#, Credit_hrs}
{ Course#, Instr_name } + = { Course#, Credit_hrs, Text, Publisher, Classroom, Capacity }

C) Definition. A set of functional dependencies F is said to ***cover*** another set of functional dependencies E if every FD in E is also in F+; that is, if every dependency in E can be inferred from F; alternatively, we can say that E is covered by F.
Definition. Two sets of functional dependencies E and F are equivalent if E+ = F+. Therefore, ***equivalence*** means that every FD in E can be inferred from F, and every FD in F can be inferred from E; that is, E is equivalent to F if both the conditions—E covers F and F covers E—hold.

D) Minimal Sets of Functional Dependencies :

Informally, a minimal cover of a set of functional dependencies E is a set of functional dependencies F that satisfies the property that every dependency in E is in the closure F+ of F. In addition, this property is lost if any dependency from the set F is removed; F must have no redundancies in it, and the dependencies in F are in a standard form.

Definition: An attribute in a functional dependency is considered an extraneous attribute if we can remove it without changing the closure of the set of dependencies. Formally, given F, the set of functional dependencies, and a functional dependency X → A in F, attribute Y is extraneous in X if Y ⊂ X, and F logically implies (F − (X → A) ∪ { (X − Y) → A }).

We can formally define a set of functional dependencies F to be minimal if it satisfies the following conditions:

1. Every dependency in F has a single attribute for its right-hand side.
2. We cannot replace any dependency X → A in F with a dependency Y → A, where Y is a proper subset of X, and still have a set of dependencies that is equivalent to F.
3. We cannot remove any dependency from F and still have a set of dependencies that is equivalent to F.

Definition. A minimal cover of a set of functional dependencies E is a minimal set of dependencies (in the standard canonical form5 and without redundancy) that is equivalent to E. We can always find at least one minimal cover F for any set of dependencies E using Algorithm 15.2.

Algorithm 15.2. Finding a Minimal Cover F for a Set of Functional Dependencies E

Input: A set of functional dependencies E.

Note: Explanatory comments are given at the end of some of the steps. They follow the format: (**comment**).

1. Set $F := E$.
2. Replace each functional dependency $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ in F by the n functional dependencies $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_n$. (*This places the FDs in a canonical form for subsequent testing*)
3. For each functional dependency $X \rightarrow A$ in F
   for each attribute B that is an element of X
   if $\{ \{F - \{X \rightarrow A\}\} \cup \{ (X - \{B\}) \rightarrow A\} \}$ is equivalent to F
   then replace $X \rightarrow A$ with $(X - \{B\}) \rightarrow A$ in F.
   (*This constitutes removal of an extraneous attribute B contained in the left-hand side X of a functional dependency $X \rightarrow A$ when possible*)
4. For each remaining functional dependency $X \rightarrow A$ in F
   if $\{F - \{X \rightarrow A\}\}$ is equivalent to F,
   then remove $X \rightarrow A$ from F. (*This constitutes removal of a redundant functional dependency $X \rightarrow A$ from F when possible*)

Next, we provide a simple algorithm to determine the key of a relation:

Algorithm 15.2(a). Finding a Key K for R Given a Set F of Functional Dependencies

Input: A relation R and a set of functional dependencies F on the attributes of R.

1. Set $K := R$.
2. For each attribute A in K
   {compute $(K - A)^+$ with respect to F;
   if $(K - A)^+$ contains all the attributes in R, then set $K := K - \{A\}$ };

Example 1:

Let the given set of FDs be E: {B → A, D → A, AB → D}. We have to find the minimal cover of E.

■ All above dependencies are in canonical form (that is, they have only one attribute on the right-hand side), so we have completed step 1 of Algorithm 15.2 and can proceed to step 2. In step 2 we need to determine if AB → D has any redundant (extraneous) attribute on the left-hand side; that is, can it be replaced by B → D or A → D?

■ Since B → A, by augmenting with B on both sides (IR2), we have BB → AB, or B → AB (i). However, AB → D as given (ii).

■ Hence by the transitive rule (IR3), we get from (i) and (ii), B → D. Thus AB → D may be replaced by B → D.

■ We now have a set equivalent to original E, say E': {B → A, D → A, B → D}. No further reduction is possible in step 2 since all FDs have a single attribute on the left-hand side.

■ In step 3 we look for a redundant FD in E'. By using the transitive rule on B → D and D → A, we derive B → A. Hence B → A is redundant in E' and can be eliminated.

■ Therefore, the minimal cover of E is F: {B → D, D → A}. The reader can verify that the original set F can be inferred from E; in other words, the two sets F and E are equivalent.

Example 2:

Let the given set of FDs be G: {A → BCDE, CD → E}.

■ Here, the given FDs are NOT in the canonical form. So we first convert them into: E: {A → B, A→ C, A→ D, A→ E, CD → E}.

■ In step 2 of the algorithm, for CD → E, neither C nor D is extraneous on the left-hand side, since we cannot show that C → E or D → E from the given FDs. Hence we cannot replace it with either.

■ In step 3, we want to see if any FD is redundant. Since A→ CD and CD → E, by transitive rule (IR3), we get A→ E. Thus, A→ E is redundant in G.

■ So we are left with the set F, equivalent to the original set G as: {A → B, A→ C, A→ D, CD → E}. F is the minimum cover. As we pointed out in footnote 6, we can combine the first three FDs using the union rule (IR5) and express the minimum cover as:

Minimum cover of G, F: {A → BCD, CD → E}.

Properties of Relational Decompositions :

1) Relation Decomposition and Insufficiency of Normal Forms : The relational database design algorithms that we present in Section 15.3 start from a single universal relation schema R = {A1, A2, … , An} that includes all the attributes of the database. We implicitly make the universal relation assumption, which states that every attribute name is unique.
Using the functional dependencies, the algorithms decompose the universal relation schema R into a set of relation schemas D = {R1, R2, … , Rm} that will become the relational database schema; D is called a decomposition of R.

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

This is called the attribute preservation condition of a decomposition.
Another goal is to have each individual relation Ri in the decomposition D be in BCNF or 3NF.

Note : Every relation with two attributes is automatically is in BCNF.

2) Dependency Preservation Property of a Decomposition : It would be useful if each functional dependency X → Y specified in F either appeared directly in one of the relation schemas Ri in the decomposition D or could be inferred from the dependencies that appear in some Ri. Informally, this is the dependency preservation condition.

Definition. Given a set of dependencies F on R, the projection of F on Ri, denoted by πRi (F) where Ri is a subset of R, is the set of dependencies X → Y in F+ such that the attributes in X ∪ Y are all contained in Ri. Hence, the projection of F on each relation schema Ri in the decomposition D is the set of functional dependencies in F+, the closure of F, such that all the left- and right-hand-side attributes of those dependencies are in Ri. We say that a decomposition D = {R1, R2, … , Rm} of R is dependency-preserving with respect to F if the union of the projections of F on each Ri in D is equivalent to F; that is, ((πR1 (F)) ∪ K ∪ (πRm(F)))+ = F+.

Claim 1. It is always possible to find a dependency-preserving decomposition D with respect to F such that each relation Ri in D is in 3NF.

3) Nonadditive (Lossless) Join Property of a Decomposition : Another property that a decomposition D should possess is the nonadditive join property, which ensures that no spurious tuples are generated when a NATURAL JOIN operation is applied to the relations resulting from the decomposition.

Definition. Formally, a decomposition D = {R1, R2, … , Rm} of R has the lossless (nonadditive) join property with respect to the set of dependencies F on R if, for every relation state r of R that satisfies F, the following holds, where * is the NATURAL JOIN of all the relations in D: *(πR1 (r), … , πRm(r)) = r.

Claim 2 (Preservation of Nonadditivity in Successive Decompositions). If a decomposition D = {R1, R2, … , Rm} of R has the nonadditive (lossless) join property with respect to a set of functional dependencies F on R, and if a decomposition Di = {Q1, Q2, … , Qk} of Ri has the nonadditive join property with respect to the projection of F on Ri, then the decomposition D2 = {R1, R2, … , Ri−1, Q1, Q2, … , Qk, Ri+1, … , Rm} of R has the nonadditive join property with respect to F.

4) Algorithms for Relational Database Schema Design : Algorithm 15.4 yields a decomposition D of R that does the following: ■ Preserves dependencies ■ Has the nonadditive join property ■ Is such that each resulting relation schema in the decomposition is in 3NF.

Example :

Example 1 of Algorithm 15.4. Consider the following universal relation:
U (Emp_ssn, Pno, Esal, Ephone, Dno, Pname, Plocation)

Emp_ssn, Esal, and Ephone refer to the Social Security number, salary, and phone number of the employee. Pno, Pname, and Plocation refer to the number, name, and location of the project. Dno is the department number.
The following dependencies are present:
FD1: Emp_ssn → {Esal, Ephone, Dno}
FD2: Pno → { Pname, Plocation}
FD3: Emp_ssn, Pno → {Esal, Ephone, Dno, Pname, Plocation}

By virtue of FD3, the attribute set {Emp_ssn, Pno} represents a key of the universal relation. Hence F, the set of given FDs, includes {Emp_ssn → Esal, Ephone, Dno; Pno → Pname, Plocation; Emp_ssn, Pno → Esal, Ephone, Dno, Pname, Plocation}.

By applying the minimal cover Algorithm 15.2, in step 3 we see that Pno is an extraneous attribute in Emp_ssn, Pno → Esal, Ephone, Dno. Moreover, Emp_ssn is extraneous in Emp_ssn, Pno → Pname, Plocation. Hence the minimal cover consists of FD1 and FD2 only (FD3 being completely redundant) as follows (if we group attributes with the same left-hand side into one FD):

Minimal cover G: {Emp_ssn → Esal, Ephone, Dno; Pno → Pname, Plocation}
The second step of Algorithm 15.4 produces relations R1 and R2 as:
R1 (Emp_ssn, Esal, Ephone, Dno)

R2 (Pno, Pname, Plocation)

In step 3, we generate a relation corresponding to the key {Emp_ssn, Pno} of U. Hence, the resulting design contains:

R1 (Emp_ssn, Esal, Ephone, Dno)

R2 (Pno, Pname, Plocation)

R3 (Emp_ssn, Pno)

This design achieves both the desirable properties of dependency preservation and nonadditive join.

Example 2 of Algorithm 15.4 (Case X). Assume that this relation is given as a universal relation
U (Property_id, County, Lot#, Area) with the following functional dependencies:
FD1: Property_id → Lot#, County, Area
FD2: Lot#, County → Area, Property_id
FD3: Area → County

For ease of reference, let us abbreviate the above attributes with the first letter for each and represent the functional dependencies as the set

F: { P → LCA, LC → AP, A → C }

The universal relation with abbreviated attributes is U (P, C, L, A). If we apply the minimal cover Algorithm 15.2 to F, (in step 2) we first represent the set F as

F: {P → L, P → C, P → A, LC → A, LC → P, A → C}

In the set F, P → A can be inferred from P → LC and LC → A; hence P → A by transitivity and is therefore redundant. Thus, one possible minimal cover is

Minimal cover GX: {P → LC, LC → AP, A → C}

In step 2 of Algorithm 15.4, we produce design X (before removing redundant relations) using the above minimal cover as Design X: R1 (P, L, C), R2 (L, C, A, P), and R3 (A, C)

In step 4 of the algorithm, we find that R3 is subsumed by R2 (that is, R3 is always a projection of R2 and R1 is a projection of R2 as well). Hence both of those relations are redundant. Thus the 3NF schema that achieves both of the desirable properties is (after removing redundant relations)

Design X: R2 (L, C, A, P).

Example 3,
F: {P → C, P → A, P → L, LC → A, LC → P, A → C}

The FD LC → A may be considered redundant because LC → P and P → A implies LC → A by transitivity. Also, P → C may be considered to be redundant because P → A and A → C implies P → C by transitivity. This gives a different minimal cover as

Minimal cover GY: { P → LA, LC → P, A → C }

The alternative design Y produced by the algorithm now is

Design Y: S1 (P, A, L), S2 (L, C, P), and S3 (A, C)

Note that this design has three 3NF relations, none of which can be considered as redundant by the condition in step 4. All FDs in the original set F are preserved.

Note : It is important to note that the theory of nonadditive join decompositions is based on the assumption that no NULL values are allowed for the join attributes.

5) ***About Nulls, Dangling Tuples, and Alternative Relational Designs*** : We must carefully consider the problems associated with NULLs when designing a relational database schema. There is no fully satisfactory relational design theory as yet that includes NULL values. A related problem is that of dangling tuples, which may occur if we carry a decomposition too far.

Table 15.1 Summary of the Algorithms Discussed in This Chapter

Algorithm	Input	Output	Properties/Purpose	Remarks
15.1	An attribute or a set of attributes X, and a set of FDs F	A set of attributes in the closure of X with respect to F	Determine all the attributes that can be functionally determined from X	The closure of a key is the entire relation
15.2	A set of functional dependencies F	The minimal cover of functional dependencies	To determine the minimal cover of a set of dependencies F	Multiple minimal covers may exist—depends on the order of selecting functional dependencies
15.2a	Relation schema R with a set of functional dependencies F	Key K of R	To find a key K (that is a subset of R)	The entire relation R is always a default superkey
15.3	A decomposition D of R and a set F of functional dependencies	Boolean result: yes or no for nonadditive join property	Testing for nonadditive join decomposition	See a simpler test NJB in Section 14.5 for binary decompositions
15.4	A relation R and a set of functional dependencies F	A set of relations in 3NF	Nonadditive join and dependency-preserving decomposition	May not achieve BCNF, but achieves *all* desirable properties and 3NF
15.5	A relation R and a set of functional dependencies F	A set of relations in BCNF	Nonadditive join decomposition	No guarantee of dependency preservation
15.6	A relation R and a set of functional and multivalued dependencies	A set of relations in 4NF	Nonadditive join decomposition	No guarantee of dependency preservation

(a)
EMPLOYEE

Ename	Ssn	Bdate	Address	Dnum
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	NULL
Benitez, Carlos M.	888664444	1963-01-09	7654 Beech, Houston, TX	NULL

DEPARTMENT

Dname	Dnum	Dmgr_ssn
Research	5	333445555
Administration	4	987654321
Headquarters	1	888665555

(b)

Ename	Ssn	Bdate	Address	Dnum	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

(c)

Ename	Ssn	Bdate	Address	Dnum	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX	NULL	NULL	NULL
Benitez, Carlos M.	888665555	1963-01-09	7654 Beech, Houston, TX	NULL	NULL	NULL

Figure 15.2
Issues with NULL-value joins. (a) Some EMPLOYEE tuples have NULL for the join attribute Dnum. (b) Result of applying NATURAL JOIN to the EMPLOYEE and DEPARTMENT relations. (c) Result of applying LEFT OUTER JOIN to EMPLOYEE and DEPARTMENT.

(a) EMPLOYEE_1

Ename	Ssn	Bdate	Address
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX
Narayan, Ramesh K.	666884444	1962-09-15	975 Fire Oak, Humble, TX
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX
Berger, Anders C.	999775555	1965-04-26	6530 Braes, Bellaire, TX
Benitez, Carlos M.	888665555	1963-01-09	7654 Beech, Houston, TX

(b) EMPLOYEE_2

Ssn	Dnum
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1
999775555	NULL
888664444	NULL

(c) EMPLOYEE_3

Ssn	Dnum
123456789	5
333445555	5
999887777	4
987654321	4
666884444	5
453453453	5
987987987	4
888665555	1

Figure 15.3
The dangling tuple problem.
(a) The relation EMPLOYEE_1 (includes all attributes of EMPLOYEE from Figure 15.2(a) except Dnum).
(b) The relation EMPLOYEE_2 (includes Dnum attribute with NULL values).
(c) The relation EMPLOYEE_3 (includes Dnum attribute but does not include tuples for which Dnum has NULL values).

Questions on functional dependencies :

1) Let r be a relation instance with schema R = (A, B, C, D). We define r1=πA,B,C(R) and r2=πA,D(r). Let s=r1∗r2 where ∗ denotes natural join. Given that the decomposition of r into r1 and r2 is lossy, which one of the following is TRUE?

$s \subset r$ $r \subset s$

$r \cup s = r$ $r * s = s$

Answer :

Answer is **C** $r \subset s$.

r

A	B	C	D
1	2	3	3
1	5	3	4

r1

A	B	C
1	2	3
1	5	3

r2

A	D
1	3
1	4

s = r1 * r2

A	B	C	D
1	**2**	**3**	**3**
1	2	3	4
1	5	3	4
1	**5**	**3**	**4**

All the rows of r are in s (marked bold). So, $r \subset s$.

And one more result $r * s = r$.

2) Consider the following functional dependencies in a database:

Data_of_Birth → Age
Age → Eligibility
Name → Roll_number
Roll_number → Name
Course_number → Course_name
Course_number → Instructor
(Roll_number, Course_number) → Grade
The relation (Roll_number, Name, Date_of_birth, Age) is:

Answer : 1st step : candidate key are dataofbirth, roll number, name
FD 1st contains part of candidate key i.e. data of birth implies age which is non-prime. Means not in 2NF.

3) Relation R with an associated set of functional dependencies, F is decomposed into BCNF. The redundancy (arising out of functional dependencies) in the resulting set relations is.

Answer : Zero, If a relational schema is in BCNF then all redundancy based on functional dependency has been removed, although other types of redundancy may still exist.

4) With regard to the expressive power of the formal relational query languages, which of the following statements is true?

Relational algebra is more powerful than relational calculus *Relational algebra has the same power as safe relational calculus*
Relational algebra has the same power as relational calculus *None of the above*

Answer : C, it is possible to write syntactically correct relational calculus queries that have infinite number of answers. Such queries are unsafe. Queries that have an finite number of answers are safe relational calculus queries.

5) Relation R is decomposed using a set of functional dependencies, F, and relation S is decomposed using another set of functional dependencies, G. One decomposition is definitely BCNF, the other is definitely 3NF, but it is not known which is which. To make a guaranteed identification, which one of the following tests should be used on the decompositions? (Assume that the closures of F and G are available).

Dependency-preservation *BCNF definition*
Lossless-join *3NF definition*

Answer : C

a. False. BCNF may or may not satisfy Dependency preservation, 3NF always does. But we can't make any guaranteed decision, regarding BCNF if it satisfies Dependency preservation
b. False. Both are lossless.
c. True. Using this we can always decide between BCNF & 3NF.
d. False. Every BCNF relation is also 3NF trivially. It's like egg yolk is BCNF and egg white is 3NF.

6) From the following instance of a relation scheme R (A, B, C), we can conclude that :

A	B	C
1	1	1
1	1	0
2	3	2
2	3	2

A functionally determines B and B function-ally determines C
A functionally determines B and B does not functionally determine C

B does not functionally determine C
A does not functionally determine B and B does not functionally determine C
Answer : C, Don't make mistake by choosing option B it is wrong because instance is given not whole relation so there might be a chance that A->B will not work.

7) Consider a schema R(A,B,C,D) and functional dependencies A->B and C->D. Then the decomposition of R into R1(AB) and R2(CD) is

dependency preserving and lossless join
dependency preserving but not lossless join
lossless join but not dependency preserving
not dependency preserving and not lossless join

Answer : Lossless-Join Decomposition:
Decomposition of R into R1 and R2 is a lossless-join decomposition if at least one of the following functional dependencies are in F+ (Closure of functional dependencies)

$R1 \cap R2 \rightarrow R1$
OR
$R1 \cap R2 \rightarrow R2$

In the above question R(A, B, C, D) is decomposed into R1 (A, B) and R2(C, D), and $R1 \cap R2$ is empty. So, the decomposition is not lossless.

8) Suppose the adjacency relation of vertices in a graph is represented in a table Adj(X,Y). Which of the following queries cannot be expressed by a relational algebra expression of constant length?

List of all vertices adjacent to a given vertex
List all vertices which belong to cycles of less than three vertices
List all vertices which have self loops
List all vertices reachable from a given vertex

Answer : D, (A) This is simple query as we need to find (X, Y) for a given X. (B) This is also simple as need to find (X, X) (C) :-> Cycle < 3 . Means cycle of length 1 & 2. Cycle of length 1 is easy., Same as self loop. Cycle of length 2 is also not too hard to compute. Though it'll be little complex, will need to do like (X,Y) & (Y, X) both present & X != Y,. We can do this with constant RA query. (D) :-> This is most hard part. Here we need to find closure of vertices. This will need kind of loop. If the graph is like skewed tree, our query must loop for O(N) Times. We can't do with constant length query here. Answer is :-> D

9) R(A,B,C,D) is a relation. Which of the following does not have a lossless join, dependency preserving BCNF decomposition?

A->B, B->CD
AB->C, C->AD
A->B, B->C, C->D
A ->BCD

Answer : We know that for lossless decomposition common attribute should be candidate key in one of the relation. A) A->B, B->CD R1(AB) and R2(BCD) B is the key of second and hence decomposition is lossless. B) A->B, B->C, C->D R1(AB) , R2(BC), R3(CD) B is the key of second and C is the key of third, hence lossless. C) AB->C, C->AD R1(ABC), R2(CD) C is key of second, but C->A violates BCNF condition in ABC as C is not a key. We cannot decompose ABC further as AB->C dependency would be lost. D) A ->BCD Already in BCNF. Therefore, Option C AB->C, C->AD is the answer.

10) Consider an Entity-Relationship (ER) model in which entity sets E1 and E2 are connected by an m : n relationship R12, E1 and E3 are connected by a 1 : n (1 on the side of E1 and n on the side of E3) relationship R13. E1 has two single-valued attributes a11 and a12 of which a11 is the key attribute. E2 has two single-valued attributes a21 and a22 is the key attribute. E3 has two single-valued attributes a31 and a32 of which a31 is the key attribute. The relationships do not have any attributes. If a relational model is derived from the above ER model, then the minimum number of relations that would be generated if all the relations are in 3NF is _____.

2
4
3
5

Answer : C, Make ER diagram and do the changes as needed, then find dependency preserving condition and lossless properties as shown in about questions.
The relation is still in 3NF as for every functional dependency X -> A, one of the following holds
1) X is a superkey or
2) A-X is prime attribute

11) Consider the relation X(P, Q, R, S, T, U) with the following set of functional dependencies

F = {
{P, R} → {S,T},
{P, S, U} → {Q, R}
}

Which of the following is the trivial functional dependency in F+ is closure of F?

{P,R}→{S,T}
{P,R}→{R,T}
{P,S}→{S}

{P,S,U}→{Q}
Answer : C, A functional dependency X -> Y is trivial if Y is a subset of X. Only S is subset of PS.

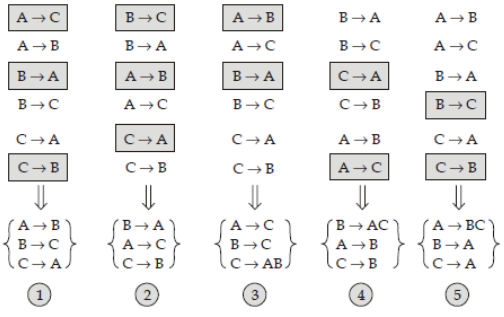
12) Which option is true about the SQL query given below?
SELECT firstName, lastName
FROM Employee
WHERE lastName BETWEEN 'A%' AND 'D%';
It will display all the employees having last names starting with the alphabets 'A' till 'D' inclusive of A and exclusive of D.
It will throw an error as BETWEEN can only be used for Numbers and not strings.
It will display all the employees having last names starting from 'A' and ending with 'D'.
It will display all the employees having last names in the range of starting alphabets as 'A' and 'D' excluding
Answer : A

13) An attribute A of datatype varchar (20) has value 'Ram' and the attribute B of datatype char (20) has value 'Sita' in oracle. The attribute A has _____ memory spaces and B has _____ memory spaces.
20,20 *3,4*
3,20 *20,4*
Answer : varchar will acquire the exact memory of attribute and it varies from tuple to tuple while char will acquire memory space which is define at the time of table creation it is fixed: varchar(20) 'Ram' will take 3 and 'Sita' will take 20 character space in memory. So, option (B) is correct.

14) Consider the following database table: Create table test(one integer, two integer, primary key(one), unique(two), check(one >= 1 and <= 10), check(two >= 1 and <= 5)); How many data records/tuples atmost can this table containt?
5 *15*
10 *50*
Answer : check(one >= 1 and <= 10), check(two >= 1 and <= 5).
Here second constraint will decide the no of tuples(record). Or we can say that the common condition will dominate. i.e. check(two >= 1 and <= 5) 5 tuples. So, option (A) is correct.

15) A relation r(A,B) in a relational database has 1200 tuples. The attribute A has integer values ranging from 6 to 20, and the attribute B has integer values ranging from 1 to 20. Assume that the attributes A and B are independently distributed. The estimated number of tuples in the output of σ(A>10)∨(B=18)(r) is _____.
820 *960*
1200 *1000*
Answer : A
Probability of first condition satisfies, P(A) = 10/15 Probability of second condition satisfies, P(B) = 1/20 Probability of both condition satisfy, P(A∩B) = 10/15*1/20 Because of independent. So, either first or second condition satisfy P(A∪B) = P(A) + P(B) - P(A∩B) = 10/15 + 1/20 - 10/15*1/20 = 0.6833 Therefore, estimated number of tuples in the output, = 1200*0.6833 = 820

16) Consider the following FD set {A → BC, B → AC, C → AB}. The number of different minimal covers possible for the above FD set _____.



Answer : do it yourself every question.

Questions on Normal Forms :

1) Consider a relational table with a single record for each registered student with the following attributes.
1. Registration_Num: Unique registration number of each registered student
2. UID: Unique identity number, unique at the national level for each citizen
3. BankAccount_Num: Unique account number at the bank. A student can have multiple accounts or join accounts. This attribute stores the primary account number.

4. Name: Name of the student
5. Hostel_Room: Room number of the hostel

Which one of the following option is INCORRECT?

BankAccount_Num is candidate key

Registration_Num can be a primary key

UID is candidate key if all students are from the same country

If S is a superkey such that S∩UID is NULL then S∪UID is also a superkey

Answer : Second and third option is trivial. BankAccount_Num is not candidate key as it cannot identify raw uniquely. Because two student can have joint account means different name but same BankAccount_Num. Forth option is correct as S is super key so any combination with other element of the relation will be superkey.

2) Consider the following entity relationship diagram (ERD), where two entities E1 and E2 have a relation R of cardinality 1 : m. The attributes of E1 are A11, A12 and A13 where A11 is the key attribute. The attributes of E2 are A21, A22 and A23 where A21 is the key attribute and A23 is a multi-valued attribute. Relation R does not have any attribute. A relational database containing minimum number of tables with each table satisfying the requirements of the third normal form (3NF) is designed from the above ERD. The number of tables in the database is

Answer : You have selected 2 but there is multivalued attribute so we make another table like we do in 1NF we built another table with all other same attribute but we separate multivalued attribute. So, answer is 3 table.

3) DBMS provides the facility of accessing data from a database through

DDL

DML

DBA

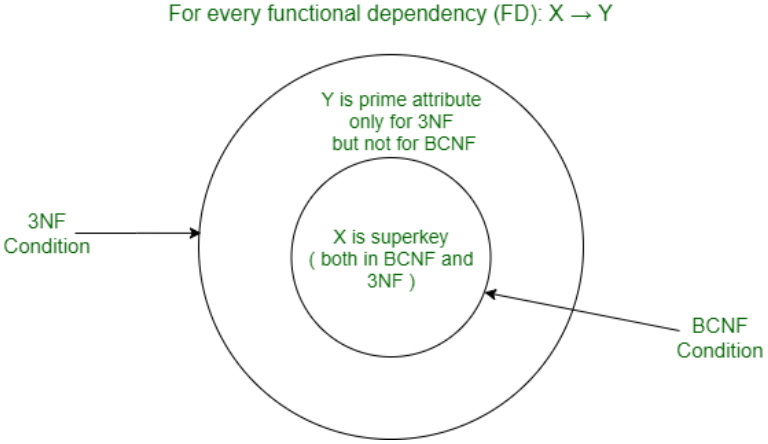
Schema

Answer : DML stands for Data Manipulation Language. DML statement is used to insert, update or delete the records.
DDL statements are used to create database, schema, constraints, users, tables etc.
Data se chedchad karna means DML and data ko banana is DDL.
Functions of a DBA include: Schema definition, Storage structure and access-method definition, Schema and physical-organization modification, Granting of authorization for data access, Routine maintenance.
The term "schema" refers to the organization of data as a blueprint of how the database is constructed (divided into database tables in the case of relational databases).

Facts very important for GATE :

1) Every table with two single-valued attributes is in 1NF, 2NF, 3NF and BCNF.
2) 3NF ensures lossless decomposition while BCNF does not.
3) Closure of every candidate key is all attribute of that relation.
4) Data which improves the performance and accessibility of the database are called indexes because index is a type of any data which improves the lookup table.
5) A **safe** tuple relational calculus expression is the one which surely generates finite results.
6) More than one Foreign key is possible if there is a requirement.
7) Foreign key act upon columns and used to identify tuples.
8) Select * from student, department → Is the form of cartesian product.
9) A relation is in 3NF if at least one of the following condition holds in every non-trivial function dependency X → Y:

X is a super key. (This condition is must for BCNF relations.).
Y is a prime attribute (each element of Y is part of some candidate key).
But, a relation is in BCNF iff, X is superkey for every functional dependency (FD) X → Y in given relation. Therefore, BCNF relations are subset of 3NF relations. Means every BCNF relation is 3NF but converse may not true.



7. DISK STORAGE, BASIC FILE STRUCTURES

This chapter and the next deal with the organization of databases in storage and the techniques for accessing them efficiently using various algorithms, some of which require auxiliary data structures called **indexes**.

The collection of data that makes up a computerized database must be stored physically on some computer storage medium.
■*Primary storage*. This category includes storage media that can be operated on directly by the computer's central processing unit (CPU), such as the computer's main memory and smaller but faster cache memories.
■ *Secondary storage*. The primary choice of storage medium for online storage of enterprise databases has been magnetic disks. However, flash memories are becoming a common medium of choice for storing moderate amounts of permanent data. When used as a substitute for a disk drive, such memory is called a solid-state drive (SSD).
■ *Tertiary storage*. Optical disks (CD-ROMs, DVDs, and other similar storage media) and tapes are removable media, Data in secondary or tertiary storage cannot be processed directly by the CPU; first it must be copied into primary storage and then processed by the CPU.

1) Memory Hierarchies and Storage Devices :

At the primary storage level, the memory hierarchy includes, at the most expensive end, cache memory, which is a static RAM (random access memory). Cache memory is typically used by the CPU to speed up execution of program instructions using techniques such as prefetching and pipelining. The next level of primary storage is DRAM (dynamic RAM), which provides the main work area for the CPU for keeping program instructions and data.

At the secondary and tertiary storage level, the hierarchy includes magnetic disks; mass storage in the form of CD-ROM (compact disk–read-only memory) and DVD (digital video disk or digital versatile disk) devices; and finally tapes at the least expensive end of the hierarchy.

Note : Volatile memory typically loses its contents in case of a power outage, whereas nonvolatile memory does not.

Flash Memory. Between DRAM and magnetic disk storage, another form of memory, flash memory, is becoming common, particularly because it is nonvolatile.
Flash memories are high-density, high-performance memories using EEPROM (electrically erasable programmable read-only memory) technology. The advantage of flash memory is the fast access speed; the disadvantage is that an entire block must be erased and written over simultaneously. And they are of two types NAND, NOR.
Optical Drives : CDs have a 700-MB capacity whereas DVDs have capacities ranging from 4.5 to 15 GB. CD-ROM(compact disk – read only memory) disks store data optically and are read by a laser. CD-ROMs contain pre-recorded data that cannot be overwritten.

The version of compact and digital video disks called CD-R (compact disk recordable) and DVD-R or DVD+R, which are also known as WORM (write-once-read-many) disks, are a form of optical storage used for archiving data; they allow data to be written once and read any number of times without the possibility of erasing.

Optical jukebox memories use an array of CD-ROM platters, which are loaded onto drives on demand.

Table 16.1 Types of Storage with Capacity, Access Time, Max Bandwidth (Transfer Speed), and Commodity Cost

Type	Capacity*	Access Time	Max Bandwidth	Commodity Prices (2014)**
Main Memory- RAM	4GB–1TB	30ns	35GB/sec	$100–$20K
Flash Memory- SSD	64 GB–1TB	50µs	750MB/sec	$50–$600
Flash Memory- USB stick	4GB–512GB	100µs	50MB/sec	$2–$200
Magnetic Disk	400 GB–8TB	10ms	200MB/sec	$70–$500
Optical Storage	50GB–100GB	180ms	72MB/sec	$100
Magnetic Tape	2.5TB–8.5TB	10s–80s	40–250MB/sec	$2.5K–$30K
Tape jukebox	25TB–2,100,000TB	10s–80s	250MB/sec–1.2PB/sec	$3K–$1M+

Each record is a collection of data values that can be interpreted as facts about entities, their attributes, and their relationships.

There are several primary file organizations, which determine how the file records are physically placed on the disk, and hence how the records can be accessed. A heap file (or unordered file) places the records on disk in no particular order by appending new records at the end of the file, whereas a sorted file (or sequential file) keeps the records ordered by the value of a particular field (called the sort key).

2) Secondary Storage Devices : Magnetic disks are used for storing large amounts of data. The device that holds the disks is referred to as a hard disk drive, or HDD. The most basic unit of data on the disk is a single bit of information.

Interfacing Disk Drives to Computer Systems. A disk controller, typically embedded in the disk drive, controls the disk drive and interfaces it to the computer system. One of the standard interfaces used for disk drives on PCs and workstations was called SCSI (small computer system interface). Today to connect HDDs, CDs, and DVDs to a computer, the interface of choice is SATA.

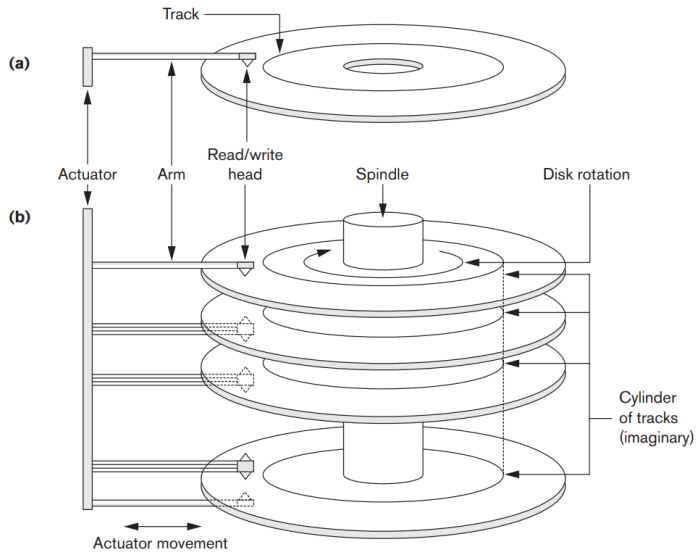


Figure 16.1
(a) A single-sided disk with read/write hardware. (b) A disk pack with read/write hardware.

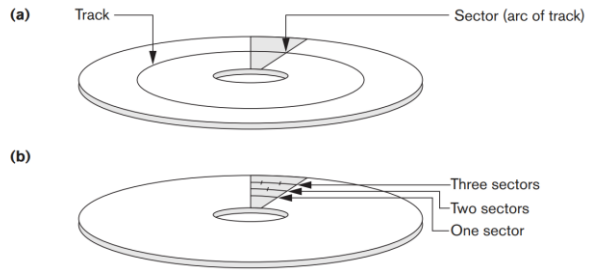


Figure 16.2
Different sector organizations on disk. (a) Sectors subtending a fixed angle. (b) Sectors maintaining a uniform recording density.

To transfer a disk block, given its address, the disk controller must first mechanically position the read/write head on the correct track. The time required to do this is called the seek time. Typical seek times are 5 to 10 msec on desktops and 3 to 8 msec on servers. Following that, there is another delay—called the rotational delay or latency—while the beginning of the desired block rotates into position under the read/write head. Finally, some additional time is needed to transfer the data; this is called the block transfer time. Here block means it is the area constrain by sectors and tracks.

Solid State Device (SSD) Storage : The recent trend is to use flash memories as an intermediate layer between main memory and secondary rotating storage in the form of magnetic disks (HDDs).

Magnetic Tape Storage Devices : Disks are random access secondary storage devices because an arbitrary disk block may be accessed at random once we specify its address. The main characteristic of a tape is its requirement that we access the data blocks in sequential order.

3) Buffering of Blocks :

While one buffer is being read or written, the CPU can process data in the other buffer because an independent disk I/O processor (controller) exists that, once started, can proceed to transfer a data block between memory and disk independent of and in parallel to CPU processing.

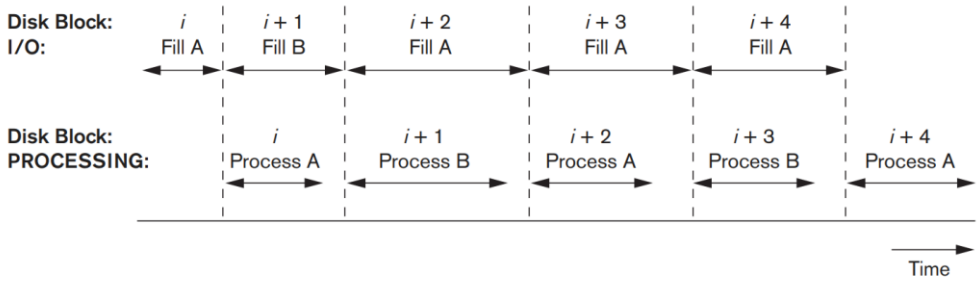


Figure 16.4
Use of two buffers, A and B, for reading from disk.

The CPU can start processing a block once its transfer to main memory is completed; at the same time, the disk I/O processor can be reading and transferring the next block into a different buffer. This technique is called **double buffering**.

A) Buffer management : Buffer manager is a software component of a DBMS that responds to requests for data and decides what buffer to use and what pages to replace in the buffer to accommodate the newly requested blocks. The buffer manager views the available main memory storage as a **buffer pool**, which has a collection of pages.

There are two kinds of buffer managers; the first kind controls the main memory directly, as in most RDBMSs. The second kind allocates buffers in virtual memory, which allows the control to transfer to the operating system (OS).

To enable its operation, the buffer manager keeps two types of information on hand about each page in the buffer pool:

1. A **pin-count**: the number of times that page has been requested, or the number of current users of that page. If this count falls to zero, the page is considered unpinned. Initially the pin-count for every page is set to zero. Incrementing the pin-count is called pinning. In general, a pinned block should not be allowed to be written to disk.
2. A **dirty bit**, which is initially set to zero for all pages but is set to 1 whenever that page is updated by any application program.

Buffer Replacement Strategies:

Least recently used (LRU): The strategy here is to throw out that page that has not been used (read or written) for the longest time.

Clock policy: This is a round-robin variant of the LRU policy. Imagine the buffers are arranged like a circle similar to a clock. Each buffer has a flag with a 0 or 1 value. Buffers with a 0 are vulnerable and may be used for replacement and their contents read back to

disk. Buffers with a 1 are not vulnerable. When a block is read into a buffer, the flag is set to 1. When the buffer is accessed, the flag is set to 1 also.
First-in-first-out (FIFO): Under this policy, the manager notes the time each page gets loaded into a buffer; but it does not have to keep track of the time pages are accessed.

Note : There are also situations when the DBMS has the ability to write certain blocks to disk even when the space occupied by those blocks is not needed. This is called *force-writing* and occurs typically when log records have to be written to disk ahead of the modified pages in a transaction for recovery purposes.

4) Placing File Records on Disk :

A) Records and Record Types : Data is usually stored in the form of records. Each record consists of a collection of related data values or items, where each value is formed of one or more bytes and corresponds to a particular field of the record. Records usually describe entities and their attributes. For example, an EMPLOYEE record. The data type of a field is usually one of the standard data types used in programming.

struct employee{ char name[30]; char ssn[9]; int salary; int job_code; char department[20]; } ;

A BLOB data item is typically stored separately from its record in a pool of disk blocks, and a pointer to the BLOB is included in the record.

B) A file is a sequence of records. In many cases, all records in a file are of the same record type. If every record in the file has exactly the same size (in bytes), the file is said to be made up of fixed-length records. If different records in the file have different sizes, the file is said to be made up of variable-length records.

File may have variable length records because of it contain variable length field for example name of employee, optional field, mixed field, repeating field. To determine the bytes within a particular record that represent each field, we can use special separator characters (such as ? or % or $).

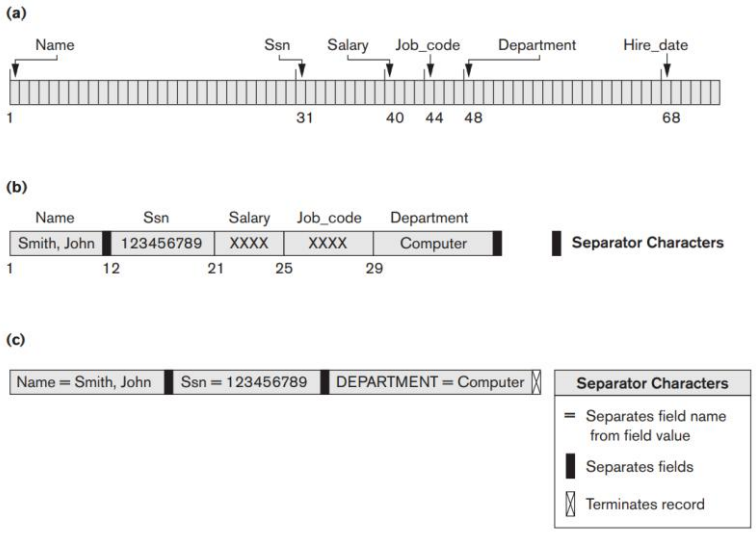


Figure 16.5
Three record storage formats. (a) A fixed-length record with six fields and size of 71 bytes. (b) A record with two variable-length fields and three fixed-length fields. (c) A variable-field record with three types of separator characters.

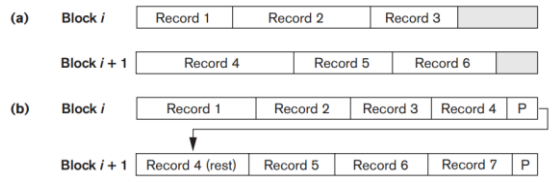


Figure 16.6
Types of record organization. (a) Unspanned. (b) Spanned.

C) Record Blocking and Spanned versus Unspanned Records : Suppose that the block size is B bytes. For a file of fixed-length records of size R bytes, with B ≥ R, we can fit bfr = $\lfloor B/R \rfloor$ records per block, where the $\lfloor (x) \rfloor$ (floor function) rounds down the number x to an integer. The value bfr is called the blocking factor for the file.

In general, R may not divide B exactly, so we have some unused space in each block equal to B − (bfr * R) bytes.

For variable-length records using spanned organization, each block may store a different number of records. In this case, the blocking factor bfr represents the average number of records per block for the file. We can use bfr to calculate the number of blocks b needed for a file of r records: b = $\lceil (r/bfr) \rceil$ blocks where the $\lceil (x) \rceil$ (ceiling function) rounds the value x up to the next integer.

D) Allocating File Blocks on Disk : There are several standard techniques for allocating the blocks of a file on disk. In *contiguous allocation*, the file blocks are allocated to consecutive disk blocks. This makes reading the whole file very fast using double buffering, but it makes expanding the file difficult. In *linked allocation*, each file block contains a pointer to the next file block. This makes it easy to expand the file but makes it slow to read the whole file. A combination of the two allocates *clusters* of consecutive disk blocks, and the clusters are linked. Clusters are sometimes called *file segments* or *extents*. Another possibility is to use *indexed allocation*, where one or more *index blocks* contain pointers to the actual file blocks. It is also common to use combinations of these techniques.

E) Header file : A file header or file descriptor contains information about a file that is needed by the system programs that access the file records. The header includes information to determine the disk addresses of the file blocks as well as to record format descriptions, which may include field lengths and the order of fields within a record for fixed-length unspanned records and field type codes, separator characters, and record type codes for variable-length records.

5) **Operations on Files** : Operations on files are usually grouped into retrieval operations and update operations. A file organization refers to the organization of the data of a file into records, blocks, and access structures; this includes the way records and blocks are placed on the storage medium and interlinked. An access method, on the other hand, provides a group of operations—such as those listed earlier—that can be applied to a file.

Some files may be static, meaning that update operations are rarely performed; other, more dynamic files may change frequently, so update operations are constantly applied to them. If a file is not updatable by the end user, it is regarded as a read-only file.

6) **Files of Unordered Records (Heap Files)** : In this simplest and most basic type of organization, records are placed in the file in the order in which they are inserted, so new records are inserted at the end of the file. Such an organization is called a heap or pile file.

Blocks before it finds the record. For a file of b blocks, this requires searching (b/2) blocks, on average. If no records or several records satisfy the search condition, the program must read and search all b blocks in the file.

To delete a record, a program must first find its block, copy the block into a buffer, delete the record from the buffer, and finally *rewrite the block* back to the disk. This leaves unused space in the disk block. Deleting a large number of records in this way results in wasted storage space. Another technique used for record deletion is to have an extra byte or bit, called a *deletion marker*, stored with each record.

7) **Files of Ordered Records (Sorted Files)** : We can physically order the records of a file on disk based on the values of one of their fields—called the *ordering field*. This leads to an ordered or *sequential file*. If the ordering field is also a *key field* of the file—a field guaranteed to have a unique value in each record—then the field is called the *ordering key* for the file.

Ordered files are rarely used in database applications unless an additional access path, called a *primary index*, is used; this results in an *indexed-sequential file*. This further improves the random-access time on the *ordering key field*. (We discuss indexes in Chapter 17.) If the ordering attribute is not a key, the file is called a *clustered file*.

Table 16.3 Average Access Times for a File of *b* Blocks under Basic File Organizations

Type of Organization	Access/Search Method	Average Blocks to Access a Specific Record
Heap (unordered)	Sequential scan (linear search)	$b/2$
Ordered	Sequential scan	$b/2$
Ordered	Binary search	$\log_2 b$

Indexing Structures for Files and Physical Database Design :

1) **Types of Single-Level Ordered Indexes** : The idea behind an ordered index is similar to that behind the index used in a textbook, which lists important terms at the end of the book in alphabetical order along with a list of page numbers where the term appears in the book. We can search the book index for a certain term in the textbook to find a list of addresses—page numbers in this case—and use these addresses to locate the specified pages first and then search for the term on each specified page. The alternative, if no other guidance is given, would be to sift slowly through the whole textbook word by word to find the term we are interested in; this corresponds to doing a linear search, which scans the whole file.

For a file with a given record structure consisting of several fields (or attributes), an index access structure is usually defined on a single field of a file, called an indexing field (or indexing attribute). The index typically stores each value of the index field along with a list of pointers to all disk blocks that contain records with that field value.

There are several types of ordered indexes.

A) A primary index is specified on the ordering key field of an ordered file of records.

B) if numerous records in the file can have the same value for the ordering field— another type of index, called a clustering index, can be used.

C) A third type of index, called a secondary index, can be specified on any nonordering field of a file. A data file can have several secondary indexes in addition to its primary access method.

a) ***Primary Indexes*** : A primary index is an ordered file whose records are of fixed length with two fields, and it acts like an access structure to efficiently search for and access the data records in a data file. The first field is of the same data type as the ordering key field—called the primary key—of the data file, and the second field is a pointer to a disk block (a block address). There is one index entry (or index record) in the index file for each block in the data file.

The first three index entries are as follows:

<K(1) = (Aaron, Ed), P(1) = address of block 1>

<K(2) = (Adams, John), P(2) = address of block 2>

<K(3) = (Alexander, Ed), P(3) = address of block 3>

The first record in each block of the data file is called the anchor record of the block.

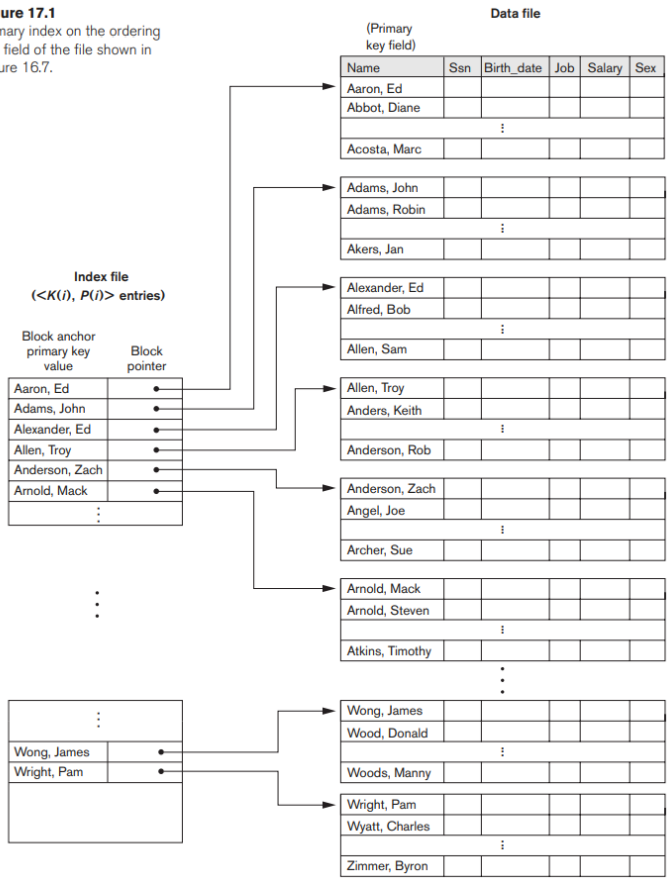


Figure 17.1
Primary index on the ordering key field of the file shown in Figure 16.7.

Example 1. Suppose that we have an ordered file with $r = 300{,}000$ records stored on a disk with block size $B = 4{,}096$ bytes.[5] File records are of fixed size and are unspanned, with record length $R = 100$ bytes. The blocking factor for the file would be $bfr = \lfloor (B/R) \rfloor = \lfloor (4{,}096/100) \rfloor = 40$ records per block. The number of blocks needed for the file is $b = \lceil (r/bfr) \rceil = \lceil (300{,}000/40) \rceil = 7{,}500$ blocks. A binary search on the data file would need approximately $\lceil \log_2 b \rceil = \lceil (\log_2 7{,}500) \rceil = 13$ block accesses.

Now suppose that the ordering key field of the file is $V = 9$ bytes long, a block pointer is $P = 6$ bytes long, and we have constructed a primary index for the file. The size of each index entry is $R_i = (9 + 6) = 15$ bytes, so the blocking factor for the index is $bfr_i = \lfloor (B/R_i) \rfloor = \lfloor (4{,}096/15) \rfloor = 273$ entries per block. The total number of index entries r_i is equal to the number of blocks in the data file, which is 7,500. The number of index blocks is hence $b_i = \lceil (r_i/bfr_i) \rceil = \lceil (7{,}500/273) \rceil = 28$ blocks. To perform a binary search on the index file would need $\lceil (\log_2 b_i) \rceil = \lceil (\log_2 28) \rceil = 5$ block accesses. To search for a record using the index, we need one additional block access to the data file for a total of 5 + 1 = 6 block accesses—an improvement over binary search on the data file, which required 13 disk block accesses. Note that the index with 7,500 entries of 15 bytes each is rather small (112,500 or 112.5 Kbytes) and would typically be kept in main memory thus requiring negligible time to search with binary search. In that case we simply make one block access to retrieve the record.

b) ***Clustering Indexes*** : A clustered index is an index which defines the physical order in which table records are stored in a database. Since there can be only one way in which records are physically stored in a database table, there can be only one clustered index per table. This differs from a primary index, which requires that the ordering field of the data file have a distinct value for each record.

A clustering index is another example of a nondense index because it has an entry for every distinct value of the indexing field, which is a nonkey by definition and hence has duplicate values rather than a unique value for every record in the file.

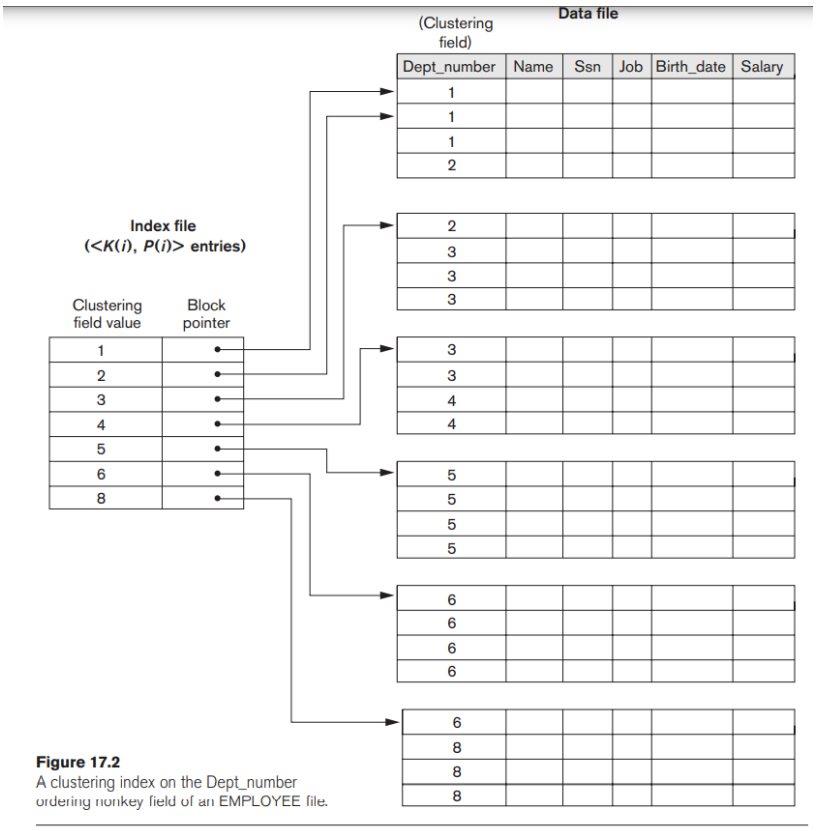


Figure 17.2
A clustering index on the Dept_number ordering nonkey field of an EMPLOYEE file.

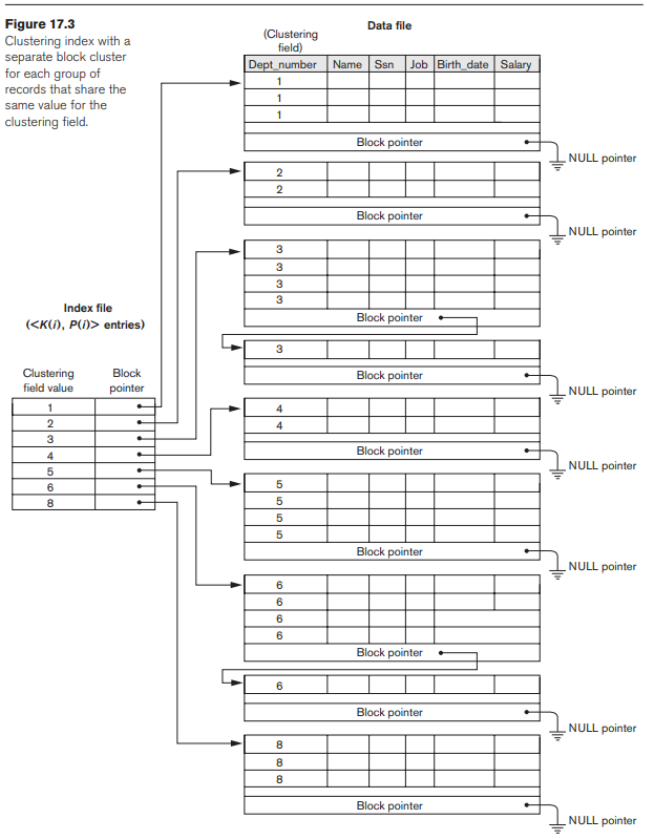
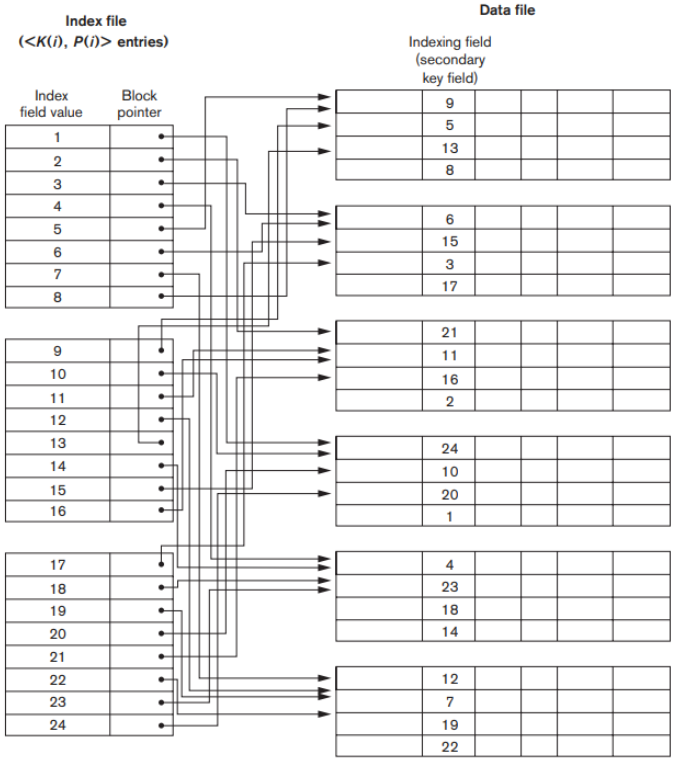


Figure 17.3
Clustering index with a separate block cluster for each group of records that share the same value for the clustering field.

Example 2. Suppose that we consider the same ordered file with r = 300,000 records stored on a disk with block size B = 4,096 bytes. Imagine that it is ordered by the attribute Zipcode and there are 1,000 zip codes in the file (with an average 300 records per zip code, assuming even distribution across zip codes.) The index in this case has 1,000 index entries of 11 bytes each (5-byte Zipcode and 6-byte block pointer) with a blocking factor bfri = $\lfloor (B/Ri) \rfloor = \lfloor (4{,}096/11) \rfloor$ = 372 index entries per block. The number of index blocks

is hence bi = ⌈(ri/bfri)⌉ = ⌈(1,000/372)⌉ = 3 blocks. To perform a binary search on the index file would need ⌈(log2 bi)⌉ = ⌈(log2 3)⌉ = 2 block accesses. Again, this index would typically be loaded in main memory (occupies 11,000 or 11 Kbytes) and takes negligible time to search in memory. One block access to the data file would lead to the first record with a given zip code.

Figure 17.4
A dense secondary index (with block pointers) on a nonordering key field of a file.



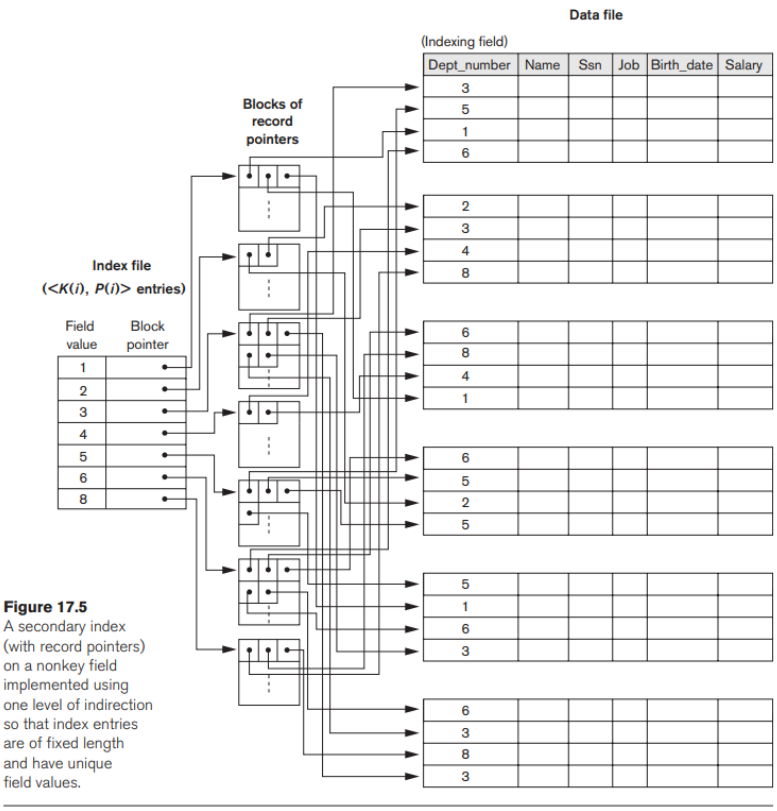


Figure 17.5
A secondary index (with record pointers) on a nonkey field implemented using one level of indirection so that index entries are of fixed length and have unique field values.

c) ***Secondary Indexes*** : A secondary index provides a secondary means of accessing a data file for which some primary access already exists. **The data file records could be ordered, unordered, or hashed.**

Example 3. Consider the file of Example 1 with r = 300,000 fixed-length records of size R = 100 bytes stored on a disk with block size B = 4,096 bytes. The file has b = 7,500 blocks, as calculated in Example 1. Suppose we want to search for a record with a specific value for the secondary key—a nonordering key field of the file that is V = 9 bytes long. Without the secondary index, to do a linear search on the file would require b/2 = 7,500/2 = 3,750 block accesses on the average. Suppose that we construct a secondary index on that nonordering key field of the file. As in Example 1, a block pointer is P = 6 bytes long, so each index entry is Ri = (9 + 6) = 15 bytes, and the blocking factor for the index is bfri = ⌊(B/Ri)⌋ = ⌊(4,096/15)⌋ = 273 index entries per block. In a dense secondary index such as this, the total number of index entries ri is equal to the number of records in the data file, which is 300,000. The number of blocks needed for the index is hence bi = ⌈(ri/bfri)⌉ = ⌈(300,000/273)⌉ = 1,099 blocks.

2) Multilevel Indexes : The idea behind a multilevel index is to reduce the part of the index that we continue to search by bfri, the blocking factor for the index, which is larger than 2. Hence, the search space is reduced much faster. The value bfri is called the fan-out of the multilevel index, and we will refer to it by the symbol fo.

Example 4. Suppose that the dense secondary index of Example 3 is converted into a multilevel index. We calculated the index blocking factor bfri = 273 index entries per block, which is also the fan-out fo for the multilevel index; the number of first-level blocks b1 = 1,099 blocks was also calculated. The number of second-level blocks will be b2 = ⌈(b1/fo)⌉ = ⌈(1,099/273)⌉ = 5 blocks, and the number of third level blocks will be b3 = ⌈(b2/fo)⌉ = ⌈(5/273)⌉ = 1 block. Hence, the third level is the top level of the index, and t = 3. To access a record by searching the multilevel index, we must access one block at each level plus one block from the data file, so we need t + 1 = 3 + 1 = 4 block accesses. Compare this to Example 3, where 12 block accesses were needed when a single-level index and binary search were used.

Figure 17.6
A two-level primary index resembling ISAM (indexed sequential access method) organization.

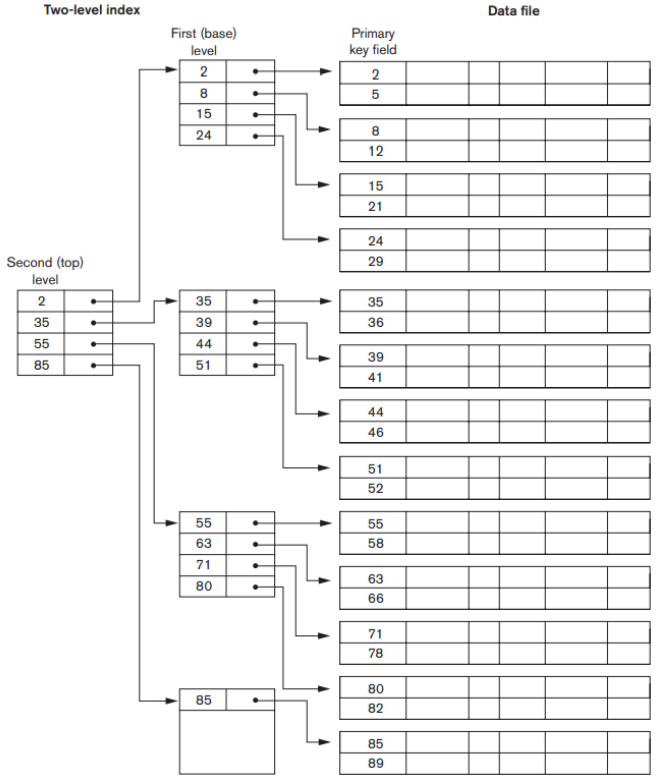


Table 17.1 Types of Indexes Based on the Properties of the Indexing Field

	Index Field Used for Physical Ordering of the File	Index Field Not Used for Physical Ordering of the File
Indexing field is key	Primary index	Secondary index (Key)
Indexing field is nonkey	Clustering index	Secondary index (NonKey)

Table 17.2 Properties of Index Types

Type of Index	Number of (First-Level) Index Entries	Dense or Nondense (Sparse)	Block Anchoring on the Data File
Primary	Number of blocks in data file	Nondense	Yes
Clustering	Number of distinct index field values	Nondense	Yes/no[a]
Secondary (key)	Number of records in data file	Dense	No
Secondary (nonkey)	Number of records[b] or number of distinct index field values[c]	Dense or Nondense	No

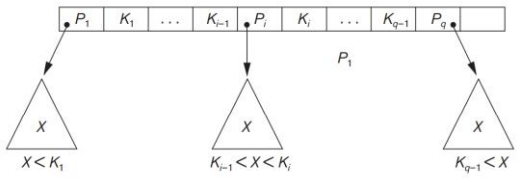
[a]Yes if every distinct value of the ordering field starts a new block; no otherwise.
[b]For option 1.
[c]For options 2 and 3.

3) Dynamic Multilevel Indexes Using B-Trees and B+-Trees :

A) Search Trees and B-Trees : A search tree is a special type of tree that is used to guide the search for a record, given the value of one of the record's fields

a) Search Trees : A search tree is slightly different from a multilevel index. A search tree of order p is a tree such that each node contains at most p − 1 search values and p pointers in the order , where q ≤ p. Each Pi is a pointer to a child node (or a NULL pointer), and each Ki is a search value from some ordered set of values.

Figure 17.8
A node in a search tree with pointers to subtrees below it.



[8]This restriction can be relaxed. If the index is on a nonkey field, duplicate search values may exist and the node structure and the navigation rules for the tree may be modified.

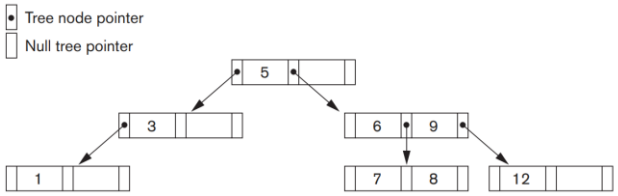


Figure 17.9
A search tree of order $p = 3$.

b) B-Trees : The B-tree has additional constraints that ensure that the tree is always balanced and that the space wasted by deletion, if any, never becomes excessive. Each internal node in the B-tree (Figure 17.10(a)) is of the form , P2, , … , , Pq> where q ≤ p. Each Pi is a tree pointer—a pointer to another node in the B-tree. Each Pri is a data pointer—a pointer to the record whose search key field value is equal to Ki (or to the data file block containing that record). Search key is the attribute or set of attributes used to lookup the record in file.

For B tree, block size ≥ order

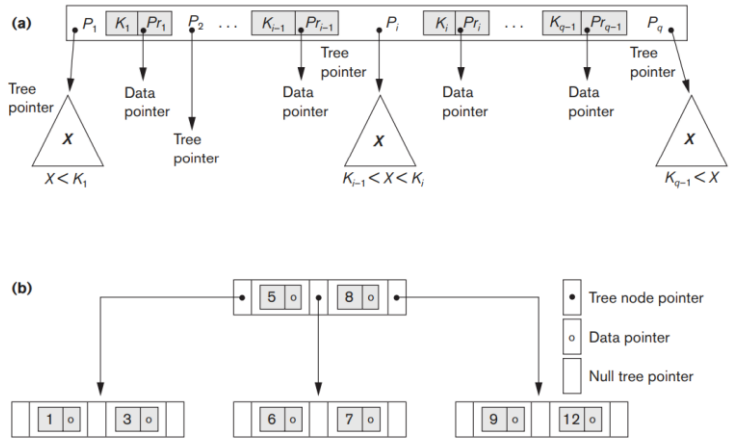


Figure 17.10
B-tree structures. (a) A node in a B-tree with $q - 1$ search values. (b) A B-tree of order $p = 3$. The values were inserted in the order 8, 5, 1, 7, 3, 12, 9, 6.

Example 6. To calculate the order p of a B^+-tree, suppose that the search key field is $V = 9$ bytes long, the block size is $B = 512$ bytes, a record pointer is $Pr = 7$ bytes, and a block pointer/tree pointer is $P = 6$ bytes. An internal node of the B^+-tree can have up to p tree pointers and $p - 1$ search field values; these must fit into a single block. Hence, we have:

$$(p * P) + ((p - 1) * V) \leq B$$
$$(p * 6) + ((p - 1) * 9) \leq 512$$
$$(15 * p) \leq 512$$

We can choose p to be the largest value satisfying the above inequality, which gives $p = 34$. This is larger than the value of 23 for the B-tree (it is left to the reader to compute the order of the B-tree assuming same size pointers), resulting in a larger fan-out and more entries in each internal node of a B^+-tree than in the corresponding B-tree. The leaf nodes of the B^+-tree will have the same number of values and pointers, except that the pointers are data pointers and a next pointer. Hence, the order p_{leaf} for the leaf nodes can be calculated as follows:

$$(p_{leaf} * (Pr + V)) + P \leq B$$
$$(p_{leaf} * (7 + 9)) + 6 \leq 512$$
$$(16 * p_{leaf}) \leq 506$$

It follows that each leaf node can hold up to $p_{leaf} = 31$ key value/data pointer combinations, assuming that the data pointers are record pointers.

c) B++ tree : In a B-tree, every value of the search field appears once at some level in the tree, along with a data pointer. In a B++tree, data pointers are stored only at the leaf nodes of the tree; hence, the structure of leaf nodes differs from the structure of internal nodes. Key means data values in one node. Order = key+1= child pointer. #Pointer is key + 1which is trivial. Always insert element at the leaf.

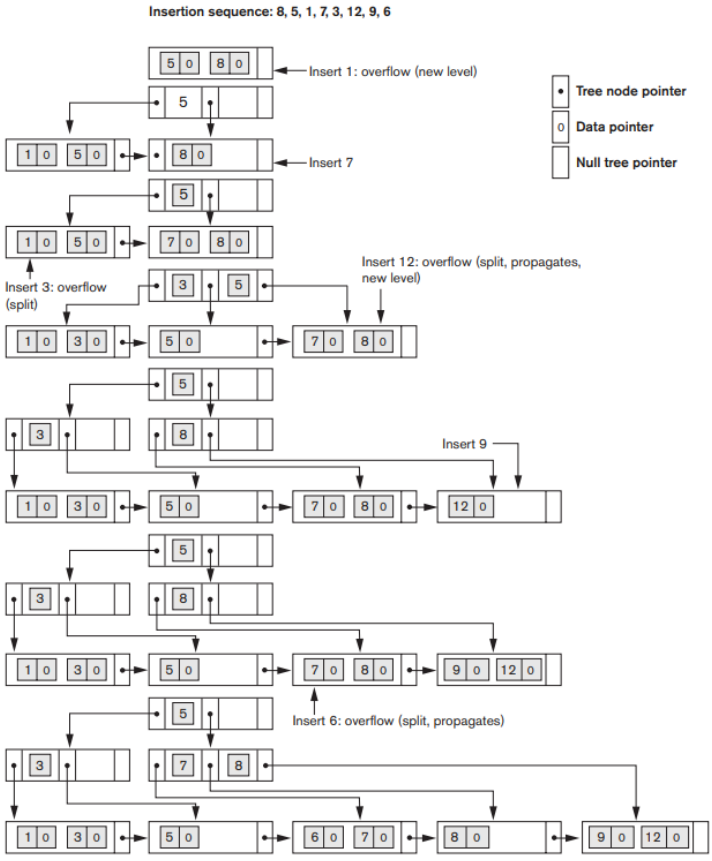
for internal node

(K*block pointer) + (k-1) . key field ⩽ Block Size

for leaf

K (record pointer + key field) + block pointer ⩽ Block size

Figure 17.12
An example of insertion in a B+-tree with $p = 3$ and $p_{leaf} = 2$.



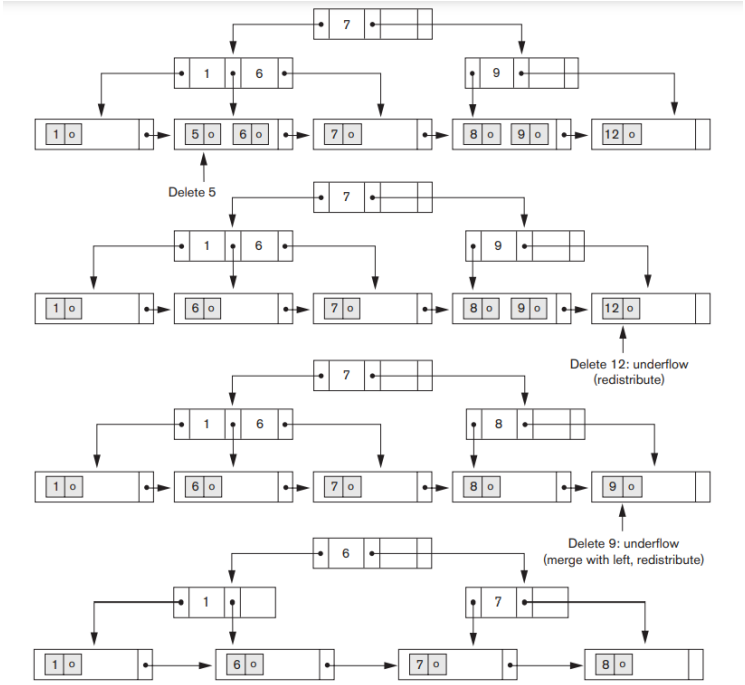


Figure 17.13
An example of deletion from a B+-tree.

Facts :

1) Range queries are faster in case of B+trees than in B-tree because in B+tree all keys are present in leaves and all leaves are connected so it is very easy to traverse through while linked link like structure for range queries.
2) If disk block allocated for B+tree and same size disk block allocated for B-tree then the I/O cost of B+tree <= I/O cost of B-tree b
3) ecause in B-tree we have record pointer for each key and block pointer as well where in B+tree we do not have record pointer. So, for given disk block size we can store more keys in B+tree than B-tree so the nodes and levels in B-tree will increase and thus I/O operation increase.
4) If number of keys in B+tree and B-tree are same then I/O cost in B+tree will increase as compared to B-tree because in B+tree values are repeated in nodes so for the same number of keys, levels will increase and thus increase in cost of I/O in case of B+tree.

4) Indexes on Multiple Keys : In our discussion so far, we have assumed that the primary or secondary keys on which files were accessed were single attributes (fields). In many retrieval and update requests, multiple attributes are involved.

For example, consider an EMPLOYEE file containing attributes Dno (department number), Age, Street, City, Zip_code, Salary and Skill_code, with the key of Ssn (Social Security number). Consider the query: List the employees in department number 4 whose age is 59.

A) Partitioned Hashing :

For example, consider the composite search key . If Dno and Age are hashed into a 3-bit and 5-bit address respectively, we get an 8-bit bucket address. Suppose that Dno = 4 has a hash address '100' and Age = 59 has hash address '10101'. Then to search for the combined search value, Dno = 4 and Age = 59, one goes to bucket address 100 10101; just to search for all employees with Age = 59, all buckets (eight of them) will be searched whose addresses are '000 10101', '001 10101', … and so on.

B) Grid Files : Our request for Dno = 4 and Age = 59 maps into the cell (1, 5) corresponding to the grid array. The records for this combination will be found in the corresponding bucket.

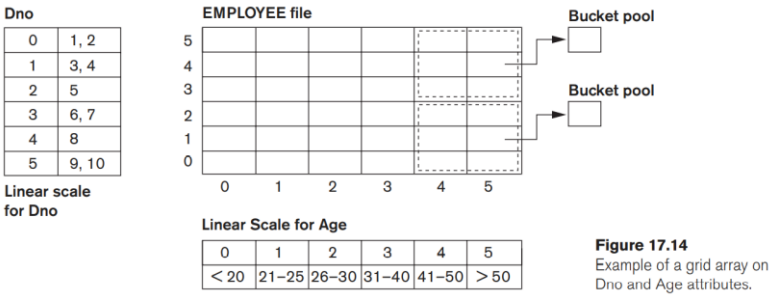
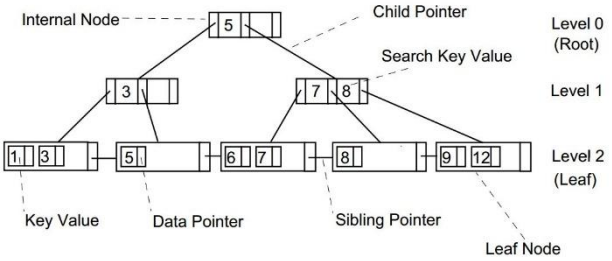
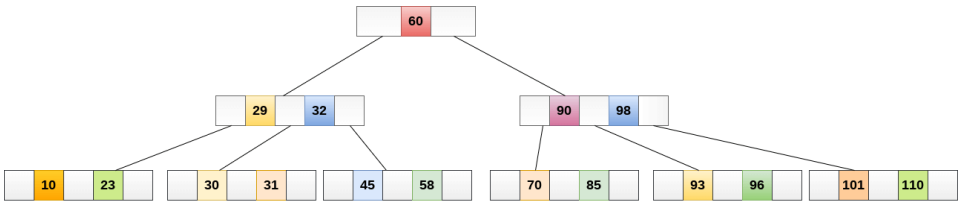


Figure 17.14
Example of a grid array on Dno and Age attributes.

∞.1) B Tree :

B Tree is a specialized m-way(maximum number of child pointers) tree that can be widely used for disk access. A B-Tree of order m can have at most m-1 keys and m children. One of the main reasons of using B tree is its capability to store large number of keys in a single node and large key values by keeping the height of the tree relatively small.



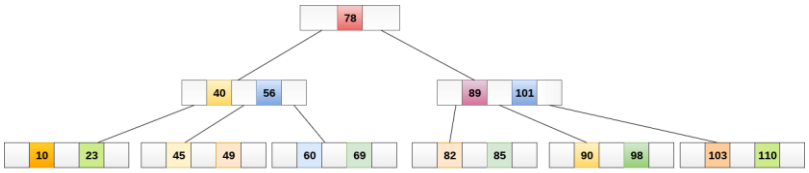


A B tree of order m contains all the properties of an M way tree. In addition, it contains the following properties.

1) Every node in a B-Tree contains at most m children.
2) Every node in a B-Tree except the root node and the leaf node contain at least m/2 children.
3) The root nodes must have at least 2 nodes.
4) All leaf nodes must be at the same level.

Operations :

Searching : (lg n)



1. Compare item 49 with root node 78. since 49 < 78 hence, move to its left sub-tree.
2. Since, 40<49<56, traverse right sub-tree of 40.
3. 49>45, move to right. Compare 49.
4. match found, return.

Inserting : O(t ·log tn), where the coefficient t is due to the operations performed for each node in the main memory. In insertion if block overflows than put medium of all elements as the parent node.

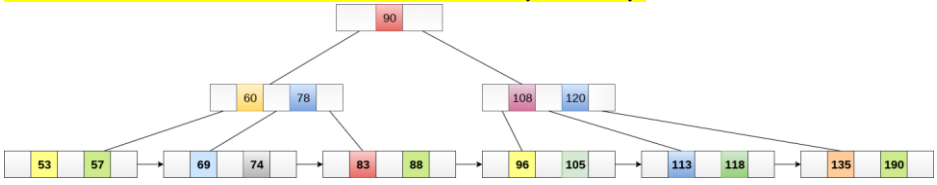
Deletion : O(lg n), Where n is total number of node.

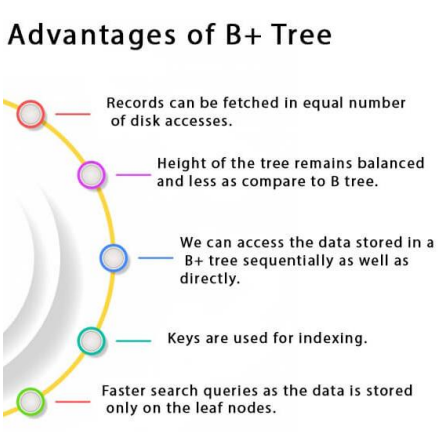
Advantages of B tree : Searching an un-indexed and unsorted database containing n key values needs O(n) running time in worst case. However, if we use B Tree to index this database, it will be searched in O(log n) time in worst case.

∞.2) B+ tree :

In B Tree, Keys and records pointer both can be stored in the internal as well as leaf nodes. Whereas, in B+ tree, records (data) can only be stored on the leaf nodes while internal nodes can only store the key values. Remember that node does not mean no of keys. Node means block.

B+ Tree are used to store the large amount of data which cannot be stored in the main memory. Due to the fact that, size of main memory is always limited, the internal nodes (keys to access records) of the B+ tree are stored in the main memory whereas, leaf nodes are stored in the secondary memory.

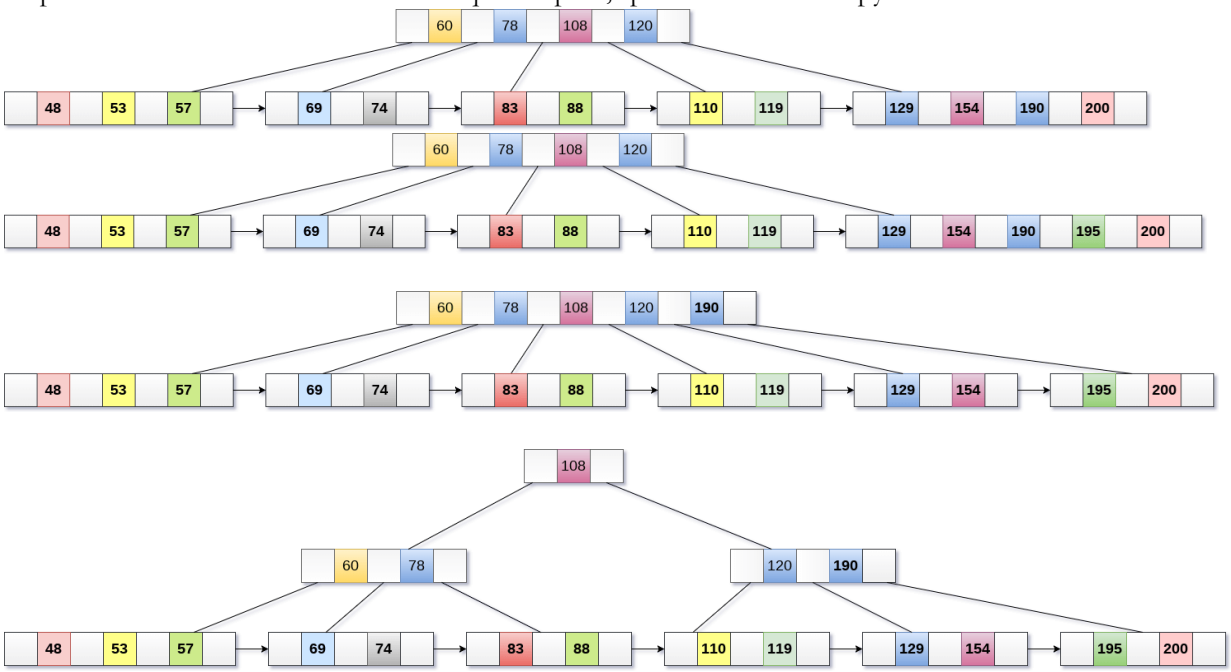




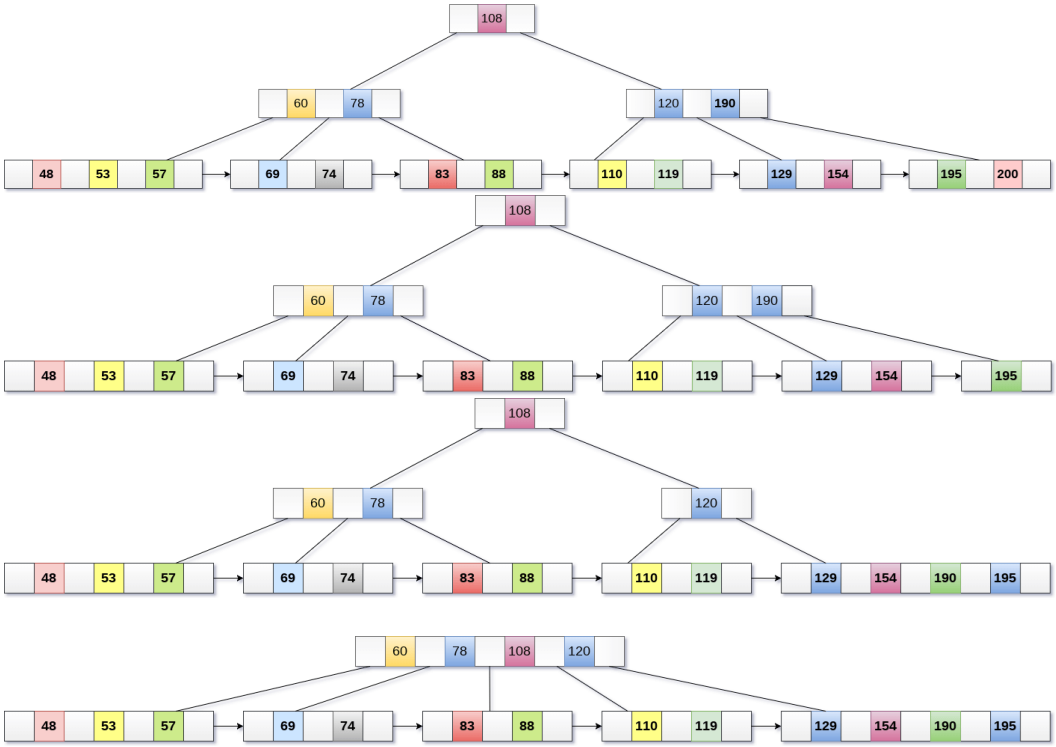
Worst case search time complexity: Θ(logn)

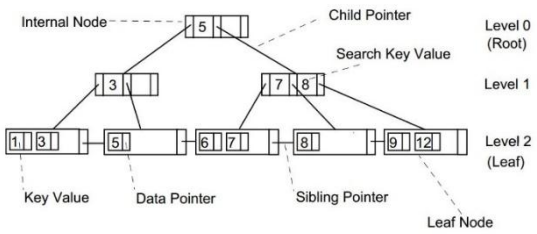
Average case search time complexity: Θ(logn)
Best case search time complexity: Θ(logn)
Worst case insertion time complexity: Θ(logn)
Worst case deletion time complexity: Θ(logn)
Average case Space complexity: Θ(n)
Worst case Space complexity: Θ(n)

Insertion in B Tree :
Step 1: Insert the new node as a leaf node.
Step 2: If the leaf doesn't have required space, split the node and copy the middle node to the next index node.
Step 3: If the index node doesn't have required space, split the node and copy the middle element to the next index page.



Deletion in B Tree :
Step 1: Delete the key and data from the leaves.
Step 2: if the leaf node contains less than minimum number of elements, merge down the node with its sibling and delete the key in between them.
Step 3: if the index node contains less than minimum number of elements, merge the node with the sibling and move down the key in between them.





B Tree For **Any** node → (no-of-keys)*[key-size+record-ptr-size]+(no-of-keys+1)*[block-ptr-size] ≤ block size

B+ Tree For **Leaf** node → (no-of-keys)*[key-size+record-ptr-size]+1*[block-ptr-size] ≤ block size

B+ Tree for **Internal** node → (no-of-keys)*[key-size]+ (no-of-keys+1)*[block-ptr-size] ≤ block size

Questions :

1) An index is clustered, if

it is on a set of fields that form a candidate key.
it is on a set of fields that include the primary key.
the data records of the file are organized in the same order as the data entries of the index.
the data records of the file are organized not in the same order as the data entries of the index.

Answer : C

2) Consider the following query :

SELECT E.eno, COUNT(*)
FROM Employees E
GROUP BY E.eno

If an index on eno is available, the query can be answered by scanning only the index if

the index is only hash and clustered
the index is only B+tree and clustered
index can be hash or B+ tree and clustered or non-clustered
index can be hash or B+ tree and clustered

Answer : C

3) Consider a table that describes the customers :

Customers(custid, name, gender, rating)

The rating value is an integer in the range 1 to 5 and only two values (male and female) are recorded for gender. Consider the query "how many male customers have a rating of 5"? The best indexing mechanism appropriate for the query is

Linear hashing
Extendible hashing
B+ Tree
Bit-mapped hashing

Answer : D, Since we require only 2 bits 0 and 1 to record gender of customers we easily implement it using bit-mapped index.

4) A B+ tree of order d is a tree in which each internal node has between d and 2d key values. An internal node with M key values has M+1 children. The root (if it is an internal node) has between 1 and 2d key values. The distance of a node from the root is the length of the path from the root to the node. All leaves are at the same distance from the root. The height of the tree is the distance of a leaf from the root. a). What is the total number of key values in the internal nodes of a B+ tree with l leaves (l≥2)? b). What is the maximum number of internal nodes in a B+ tree of order 4 with 52 leaves? c). What is the minimum number of leaves in a B+ tree of order d and height h(h≥1)?

Answer :

Let us understand specification of B+ tree first

For a non-root node

Minimum number of keys =d⟹ minimum number of children =d+1

Maximum number of keys =2d⟹ maximum number of children =2d+1

For a root node

Minimum number of keys =1 so, minimum number of children =2

Maximum number of keys =2d so, maximum number of children =2d+1

Part (A). For a given no of leaf node (L≥2) what will be the total no of keys in internal nodes?

Will solve this in three ways:

1. Assuming maximum nodes at each level

Height	#nodes	#keys
0	1	2d
1	2d + 1	2d(2d + 1)
⋮	⋮	⋮
h	$(2d+1)^h$	$2d[(2d+1)^h]$

No. of leaf nodes $= (2d+1)^h = L$
Total no. of keys in internal nodes $= 2d + 2d(2d+1)$
$+2d(2d+1)^2 + \ldots + 2d(2d+1)^{h-1}$
$= (2d+1)^h - 1 = L - 1$

2. Assuming minimum nodes at each level

Height	#nodes	#keys
0	1	1
1	2	2d
⋮	⋮	⋮
h	$2(d+1)^{h-1}$	$2d[(d+1)^{h-1}]$

So, no. of leaf nodes $= 2(d+1)^{h-1} = L$
Total no of keys in internal nodes $= 1 + 2d + 2d(d+1) + \ldots + 2d(d+1)^{h-2}$
$= 2(d+1)^{h-1} - 1 = L - 1$

3. Whenever there is an overflow in a leaf node (or whenever no of leaf node increases by one), then we move a key in the internal node (or we can say, no of internal keys increases by one).

Now, let's start with the base case. Only 2 leaf nodes (as given L≥2). So, no. of keys in root node =1 or L−1.
Once there is an overflow in a single leaf node then no of leaf nodes now would become 3 and at the same the time we will have one more key in our root node.

Part (B) Maximum number of internal nodes in a B+ tree of order 4 with 52 leaves?
Using Bulk loading approach, here we will use minimum fill factor (d=4 hence, min keys =d=4 and min children/block pointer =d+1=5)
So, we have 52 leaves so and need total 52 block pointers and one node should have minimum 5 block pointers.
So, for 52 leaves we require ⌊52/5⌋=10 nodes
For 11 block pointers we require ⌊10/5⌋=2 nodes
For 2 block pointers we require 1 node "it is root node"
So, max no of internal nodes= 10+2+1=13 nodes
Part (C) Minimum number of leaves in a B+ tree of order d and height h(h≥1)?
By part (A) "assuming minimum nodes at each level" case
Minimum no. of leaves $=2(d+1)^{h-1}$

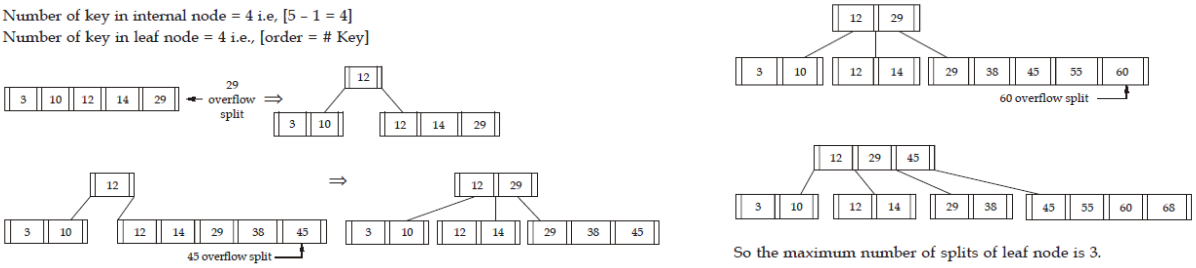
5) Given a block can hold either 3 records or 10 key pointers. A database contains n records, then how many blocks do we need to hold the data file and the dense index

13n/30 *n/10*
n/3 *n/30*

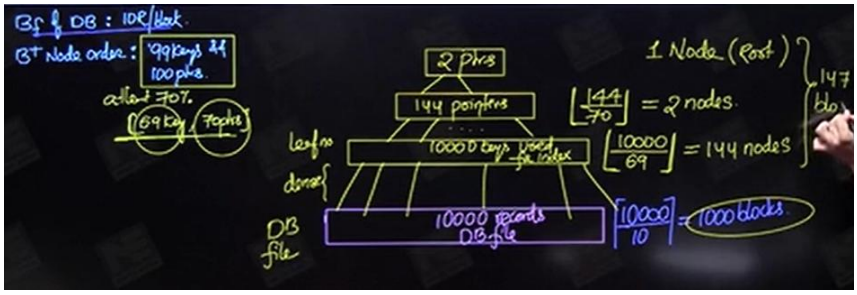
Answer :
Number of blocks needed to store data file with n records = n/3 Number of blocks needed to store dense file index = n/10 Total blocks required = n/3 + n/10 = 13n/30 blocks. Option (A) is correct.

6) The following key values are inserted into a B+-tree in which order of the internal nodes is 5 and that of the leaf nodes is 4, in the sequence given below. The order of internal nodes is the maximum number of tree pointers in each node, and the order of leaf nodes is the maximum number of data items that can be stored in it. The B+-tree is initially empty. 3, 10, 12, 14, 29, 38, 45, 55, 60, 68. The maximum number of times leaf nodes would get split up as a result of these insertions is _____.

Answer :



7) Consider that blocks can hold either ten records or 99 keys and 100 pointers. Assume that the average B+ tree node is atleast 70% full, i.e. except root it will have 69 keys and 70 pointers. The total number of blocks needed for a 10000, record file, if memory initially is empty, the search key is the primary key for the records and the data file is a sequential file, sorted on the search key, with 10 records per block with dense index are _____.

Answer : 

1147 blocks

8. Introduction to Transaction Processing Concepts and Theory

1) Introduction to Transaction Processing :

A) Single-User versus Multiuser Systems : One criterion for classifying a database system is according to the number of users who can use the system concurrently. 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.

Multiple users can access databases—and use computer systems—simultaneously because of the concept of *multiprogramming*, which allows the operating system of the computer to execute multiple programs—or *processes*—at the same time.

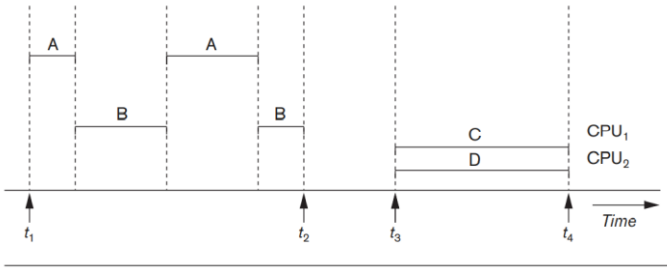


Figure 20.1
Interleaved processing versus parallel processing of concurrent transactions.

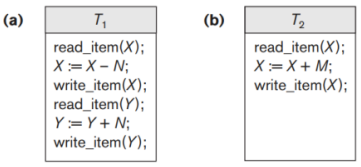


Figure 20.2
Two sample transactions.
(a) Transaction T_1.
(b) Transaction T_2.

B) Transactions, Database Items, Read and Write Operations, and DBMS Buffers : A transaction is an executing program that forms a logical unit of database processing. A transaction includes one or more database access operations—these can include insertion, deletion, modification (update), or retrieval operations.

One way of specifying the transaction boundaries is by specifying explicit begin transaction and end transaction statements in an application program.

A *database* is basically represented as a collection of named *data items*. The size of a data item is called its *granularity*. A data item can be a database record, but it can also be a larger unit such as a whole disk block, or even a smaller unit such as an individual field (attribute) value of some record in the database. The transaction processing concepts we discuss are independent of the data item granularity (size) and apply to data items in general.

■ read_item(X). Reads a database item named X into a program variable. To simplify our notation, we assume that the program variable is also named X.

■ write_item(X). Writes the value of program variable X into the database item named X.

The DBMS will maintain in the *database cache* a number of *data buffers* in main memory. Each buffer typically holds the contents of one database disk block, which contains some of the database items being processed which uses *buffer replacement policy*.

C) Why Concurrency Control Is Needed ?

Figure 20.2(a) shows a transaction T1 that transfers N reservations from one flight whose number of reserved seats is stored in the database item named X to another flight whose number of reserved seats is stored in the database item named Y. Figure 20.2(b) shows a simpler transaction T2 that just reserves M seats on the first flight (X) referenced in transaction T1. To simplify our example, we do not show additional portions of the transactions, such as checking whether a flight has enough seats available before reserving additional seats.

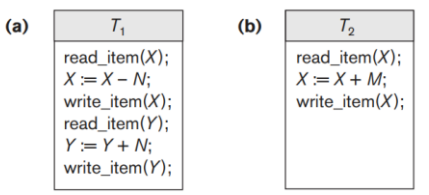
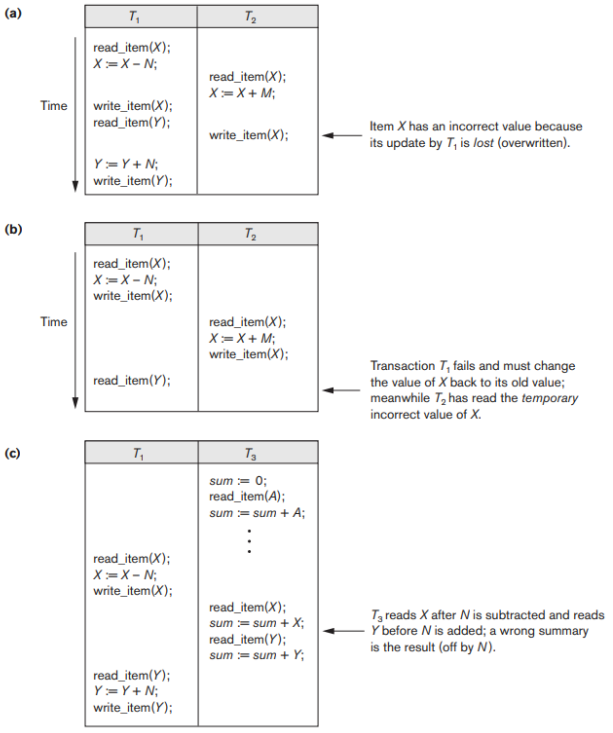


Figure 20.2
Two sample transactions.
(a) Transaction T_1.
(b) Transaction T_2.

Figure 20.3
Some problems that occur when concurrent execution is uncontrolled. (a) The lost update problem. (b) The temporary update problem. (c) The incorrect summary problem.



- *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 items incorrect.
- *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 (see Section 20.1.4). Meanwhile, the updated item is accessed (read) by another transaction before it is changed back (or rolled back) to its original value.
- *The Incorrect Summary Problem*. If one transaction is calculating an aggregate summary function on a number of database items while other transactions are updating some of these items, the aggregate function may calculate some values before they are updated and others after they are updated.
- *The Unrepeatable Read Problem*. Another problem that may occur is called unrepeatable read, where a transaction T reads the same 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.

D) Why Recovery Is Needed ?

The transaction is said to be *committed* -> the system is responsible for making sure that either all the operations in the transaction are completed successfully and their effect is recorded permanently in the database.
the transaction is *aborted* -> that the transaction does not have any effect on the database or any other transactions

Types of Failures.

a) *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.
b) *A transaction or system error*. Some operation in the transaction may cause it to fail, such as integer overflow or division by zero.
c) *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.
d) *Concurrency control enforcement*. <<< Later in next chapter
e) *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.
f) *Physical problems and catastrophes*. This refers to an endless list of problems.

2) Transaction and System Concepts :

A) Transaction States and Additional Operations :

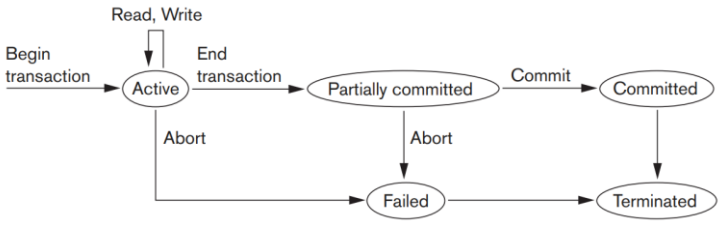


Figure 20.4
State transition diagram illustrating the states for transaction execution.

The recovery manager of the DBMS needs to keep 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 20.5) 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.

B) The System Log :

The log is a sequential, append-only file that is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure. Typically, one (or more) main memory buffers, called the log buffers, hold the last part of the log file, so that log entries are first added to the log main memory buffer.

The following are the types of entries—called log records—that are written to the log file and the corresponding action for each log record. In these entries, T refers to a unique transaction-id that is generated automatically by the system for each transaction and that is used to identify each transaction:
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.

C) Commit Point of a Transaction : A transaction T reaches its commit point when all its operations that access the database have been executed successfully and the effect of all the transaction operations on the database have been recorded in the log. Beyond the commit point, the transaction is said to be committed, and its effect must be permanently recorded in the database. The transaction then writes a commit record [commit, T] into the log.

Hence, before a transaction reaches its commit point, any portion of the log that has not been written to the disk yet must now be written to the disk. This process is called *force-writing* the log buffer to disk before committing a transaction.

D) DBMS-Specific Buffer Replacement Policies : The DBMS cache will hold the disk pages that contain information currently being processed in main memory buffers.

a) *Domain Separation (DS) Method.* In a DBMS, various types of disk pages exist: index pages, data file pages, log file pages, and so on. In this method, the DBMS cache is divided into separate domains (sets of buffers).

b) *Hot Set Method.* This page replacement algorithm is useful in queries that have to scan a set of pages repeatedly, such as when a join operation is performed using the nested-loop method.

There are basically two algorithms used for this purpose :

Note : Always remember that collection of record forms block.

Suppose table T1 contains n1 records and T2 contains n2 records and contains b1 and b2 no. of block respectively.

- Nested loop join algorithm (bnb) : which works as record by record

```
for( int i=0; i<n1; i++)
        for( int j=0; j<n2; j++)
                Join operation.
```

OR

```
for( int i=0; i<n2; i++)
        for( int j=0; j<n1; j++)
                Join operation.
```

*Total block access required = b1 + n1 * b2* ***or*** *b2 + n2 * b1*

- Block loop join algorithm (bbb): this works as block by block.

```
for( int i=0; i<b1; i++)
        for( int j=0; j<b2; j++)
                Join operation.
```

```
for( int i=0; i<b2; i++)
        for( int j=0; j<b1; j++)
                Join operation
```

*Total block access required = b1 + b1 * b2* ***or*** *b2 + b2 * b1*

c) *The DBMIN Method.* This page replacement policy uses a model known as QLSM (query locality set model), which predetermines the pattern of page references for each algorithm for a particular type of database operation. The DBMIN page replacement policy will calculate a locality set using QLSM for each file instance involved in the query, then allocates the appropriate number of buffers to each file instance involved in the query based on the locality set for that file instance.

3) Desirable Properties of Transactions :

The following are the ACID properties:

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

■ **Consistency preservation**. A transaction should be consistency preserving, meaning that if it is completely executed from beginning to end without interference from other transactions, it should take the database from one consistent state to another.

■ **Isolation**. A transaction should appear as though it is being executed in isolation from other transactions, even though many transactions are executing concurrently. That is, the execution of a transaction should not be interfered with by any other transactions executing concurrently.

■ **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.

Levels of Isolation. There have been attempts to define the level of isolation of a transaction.

- A transaction is said to have level 0 (zero) isolation if it does not overwrite the dirty reads of higher-level transactions.
- Level 1 (one) isolation has no lost updates, and
- level 2 isolation has no lost updates and no dirty reads. Finally,
- level 3 isolation (also called true isolation) has, in addition to level 2 properties, repeatable reads.
- Another type of isolation is called snapshot isolation, and several practical concurrency control methods are based on this. (Next chapter).

4) Characterizing Schedules Based on Recoverability :

When transactions are executing concurrently in an interleaved fashion, then the order of execution of operations from all the various transactions is known as a *schedule (or history)*.

A) Schedules (Histories) of Transactions : A schedule (or history) S of n transactions T1, T2, … , Tn is an ordering of the operations of the transactions. The order of operations in S is considered to be a total ordering, meaning that for any two operations in the schedule, one must occur before the other.

- b, r, w, e, c, and a for the operations begin_transaction, read_item, write_item, end_transaction, commit, and abort, respectively,
- The schedule in Figure 20.3(a), which we shall call Sa, can be written as follows in this notation:
  Sa: r1(X); r2(X); w1(X); r1(Y); w2(X); w1(Y);
  Similarly, the schedule for Figure 20.3(b), which we call Sb, can be written as follows, if we assume that transaction T1 aborted after its read_item(Y) operation:
  Sb: r1(X); w1(X); r2(X); w2(X); r1(Y); a1;

Conflicting Operations in a Schedule. 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 Sa, the operations r1(X) and w2(X) conflict.

read-write conflict : if we change the order of the two operations r1(X); w2(X) to w2(X); r1(X), then the value of X that is read by transaction T1 changes, because in the second ordering the value of X is read by r1(X) after it is changed by w2(X), whereas in the first ordering the value is read before it is changed.
For a **write-write conflict**, the last value of X will differ because in one case it is written by T2 and in the other case by T1.

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 relative 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.

committed projection C(S) of a schedule S, which includes only the operations in S that belong to committed transactions—that is, transactions Ti whose commit operation ci is in S.

B) Characterizing Schedules Based on Recoverability : First, we would like to ensure that, once a transaction T is committed, it should never be necessary to roll back T. This ensures that the durability property of transactions.

- A schedule where a committed transaction may have to be rolled back during recovery is called *nonrecoverable* and hence should not be permitted by the DBMS.
- The condition for a recoverable schedule is as follows: A schedule S is *recoverable* if
  - ⇨ No transaction T in S commits until all transactions T' that have written some item X that T reads have committed.
  - ⇨ A transaction T reads from transaction T' in a schedule S if some item X is first written by T' and later read by T.
  - ⇨ In addition, T' should not have been aborted before T reads item X, and
  - ⇨ there should be no transactions that write X after T' writes it and before T reads it (unless those transactions, if any, have aborted before T reads X).

Example :
Sc: r1(X); w1(X); r2(X); r1(Y); w2(X); c2; a1;
Sd: r1(X); w1(X); r2(X); r1(Y); w2(X); w1(Y); c1; c2;
Se: r1(X); w1(X); r2(X); r1(Y); w2(X); w1(Y); a1; a2;

Sc is not recoverable because T2 reads item X from T1, but T2 commits before T1 commits.
For the schedule to be recoverable, the c2 operation in Sc must be postponed until after T1 commits, as shown in Sd.

⇨ A **recoverable schedule** may be any one of these kinds-

- ***Cascading Schedule or cascading recoverable schedule*** : If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a Cascading Schedule or Cascading Rollback or Cascading Abort. It simply leads to the wastage of CPU time.

This is illustrated in schedule Se, where transaction T2 has to be rolled back because it read item X from T1, and T1 then aborted.

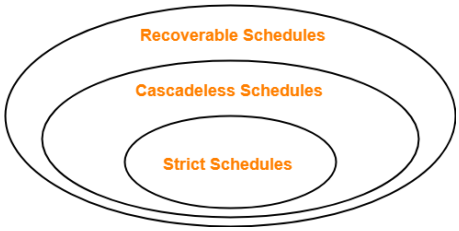
- ***Cascadeless Schedule or Cascadeless recoverable Schedule*** : If in a schedule, a transaction is not allowed to read a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a Cascadeless Schedule.

In other words, Cascadeless schedule allows only committed read operations. Therefore, it avoids cascading roll back and thus saves CPU time.

- ***Strict Schedule or strict recoverable schedule*** : If in a schedule, a transaction is neither allowed to read nor write a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as a Strict Schedule.

Example : Sf: w1(X, 5); w2(X, 8); a1;

Suppose that the value of X was originally 9, which is the before image stored in the system log along with the w1(X, 5) operation. If T1 aborts, as in Sf, the recovery procedure that restores the before image of an aborted write operation will restore the value of X to 9, even though it has already been changed to 8 by transaction T2, thus leading to potentially incorrect results.



NOTE : It is important to note that any strict schedule is also cascadeless, and any cascadeless schedule is also recoverable. In short all strict schedules are cascadeless, and all cascadeless schedules are recoverable.

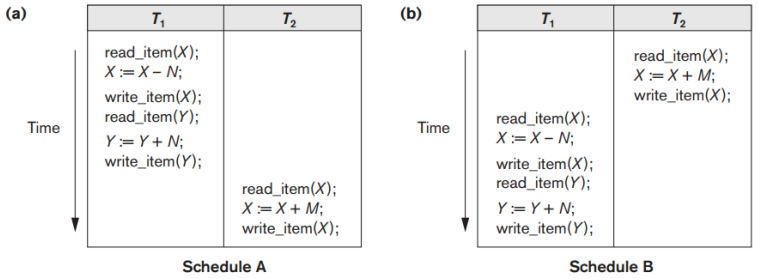
5) **Characterizing Schedules Based on Serializability** : we characterize the types of schedules that are always considered to be correct when concurrent transactions are executing.

If no interleaving of operations is permitted, there are only two possible outcomes:

1. Execute all the operations of transaction T1 (in sequence) followed by all the operations of transaction T2 (in sequence).
2. Execute all the operations of transaction T2 (in sequence) followed by all the operations of transaction T1 (in sequence). These two schedules—called *serial schedules*.

A) Serial, Non-serial and Conflict-Serializable Schedules :

Schedules C and D in Figure 20.5(c) are called non-serial because each sequence interleaves operations from the two transactions. Formally, a schedule S is *serial* if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule; otherwise, the schedule is called *non-serial*.



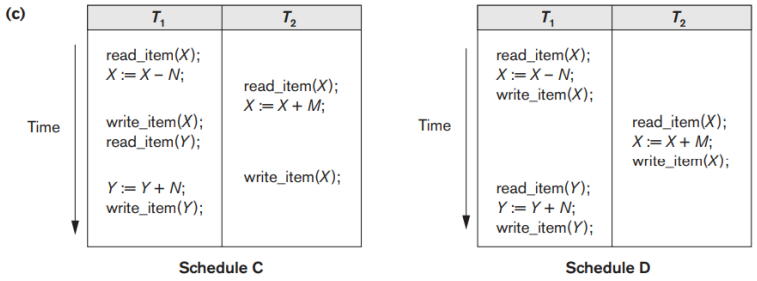


Figure 20.5
Examples of serial and nonserial schedules involving transactions T_1 and T_2. (a) Serial schedule A: T_1 followed by T_2. (b) Serial schedule B: T_2 followed by T_1. (c) Two nonserial schedules C and D with interleaving of operations.

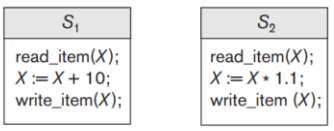


Figure 20.6
Two schedules that are result equivalent for the initial value of $X = 100$ but are not result equivalent in general.

- The definition of *serializable schedule* is as follows: A schedule S of n transactions is serializable if it is equivalent to some serial schedule of the same n transactions. We will define the concept of equivalence of schedules shortly.
- Two schedules are called result *equivalent* if they produce the same final state of the database.
- *Conflict Equivalence* of Two Schedules. Two schedules are said to be conflict equivalent if the relative order of any two conflicting operations is the same in both schedules. Two schedule are conflict if they belong to different transactions, access the same database item, and either both are write_item operations or one is a write_item and the other a read_item.
- Serializable Schedules. Using the notion of conflict equivalence, we define a schedule S to be serializable if it is (conflict) equivalent to some serial schedule S'.

  Example :
  - ⇨ schedule D in Figure 20.5(c) is equivalent to the serial schedule A in Figure 20.5(a). In both schedules, the read_item(X) of T2 reads the value of X written by T1, whereas the other read_item operations read the database values from the initial database state. Additionally, T1 is the last transaction to write Y, and T2 is the last transaction to write X in both schedules. Because A is a serial schedule and schedule D is equivalent to A, D is a serializable schedule. Notice that the operations r1(Y) and w1(Y) of schedule D do not conflict with the operations r2(X) and w2(X), since they access different data items. Therefore, we can move r1(Y), w1(Y) before r2(X), w2(X), leading to the equivalent serial schedule T1, T2.
  - ⇨ Schedule C in Figure 20.5(c) is not equivalent to either of the two possible serial schedules A and B, and hence is not serializable. Trying to reorder the operations of schedule C to find an equivalent serial schedule fails because r2(X) and w1(X)

conflict, which means that we cannot move r2(X) down to get the equivalent serial schedule T1, T2. Similarly, because w1(X) and w2(X) conflict, we cannot move w1(X) down to get the equivalent serial schedule T2, T1.

B) Testing for Serializability of a Schedule :

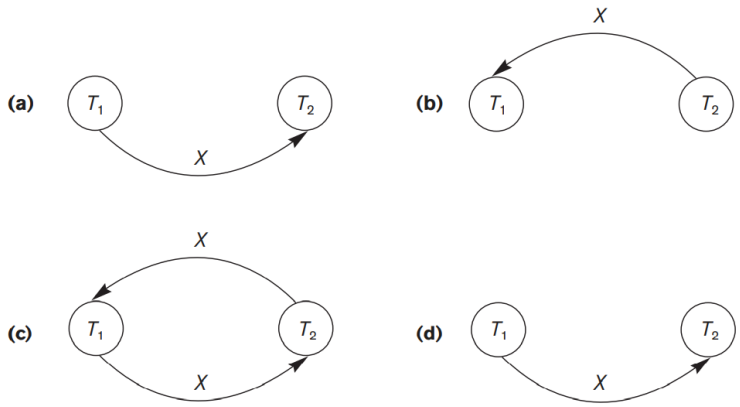


Figure 20.7
Constructing the precedence graphs for schedules A to D from Figure 20.5 to test for conflict serializability. (a) Precedence graph for serial schedule A. (b) Precedence graph for serial schedule B. (c) Precedence graph for schedule C (not serializable). (d) Precedence graph for schedule D (serializable, equivalent to schedule A).

Testing Conflict Serializability of a Schedule S

1. For each transaction Ti participating in schedule S, create a node labeled Ti in the precedence graph.
2. For each case in S where Tj executes a read_item(X) after Ti executes a write_item(X), create an edge (Ti → Tj) in the precedence graph.
3. For each case in S where Tj executes a write_item(X) after Ti executes a read_item(X), create an edge (Ti → Tj) in the precedence graph.
4. For each case in S where Tj executes a write_item(X) after Ti executes a write_item(X), create an edge (Ti → Tj) in the precedence graph.
5. The schedule S is serializable if and only if the precedence graph has no cycles.

- In general, several serial schedules can be equivalent to S if the precedence graph for S has no cycle. However, if the precedence graph has a cycle, it is easy to show that we cannot create any equivalent serial schedule, so S is not serializable.
- The precedence graphs created for schedules A to D, respectively, in Figure 20.5 appear in Figures 20.7(a) to (d). The graph for schedule C has a cycle, so it is not serializable. The graph for schedule D has no cycle, so it is serializable, and the equivalent serial schedule is T1 followed by T2. The graphs for schedules A and B have no cycles, as expected, because the schedules are serial and hence serializable.
- In Figures 20.8(d) and (e). Schedule E is not serializable because the corresponding precedence graph has cycles. Schedule F is serializable, and the serial schedule equivalent to F is shown in Figure 20.8(e).

Figure 20.8
Another example of serializability testing. (a) The read and write operations of three transactions T_1, T_2, and T_3. (b) Schedule E. (c) Schedule F.

(a)

Transaction T_1	Transaction T_2	Transaction T_3
read_item(X);	read_item(Z);	read_item(Y);
write_item(X);	read_item(Y);	read_item(Z);
read_item(Y);	write_item(Y);	write_item(Y);
write_item(Y);	read_item(X);	write_item(Z);
	write_item(X);	

(b)

Time

Transaction T_1	Transaction T_2	Transaction T_3
	read_item(Z);	
	read_item(Y);	
	write_item(Y);	
		read_item(Y);
		read_item(Z);
read_item(X);		
write_item(X);		
		write_item(Y);
		write_item(Z);
	read_item(X);	
read_item(Y);		
write_item(Y);		
	write_item(X);	

Schedule E

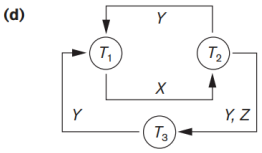
(c)

Time

Transaction T_1	Transaction T_2	Transaction T_3
		read_item(Y);
		read_item(Z);
read_item(X);		
write_item(X);		
		write_item(Y);
		write_item(Z);
	read_item(Z);	
read_item(Y);		
write_item(Y);		
	read_item(Y);	
	write_item(Y);	
	read_item(X);	
	write_item(X);	

Schedule F

Figure 20.8 (continued)
Another example of serializability testing. (d) Precedence graph for schedule E. (e) Precedence graph for schedule F. (f) Precedence graph with two equivalent serial schedules.

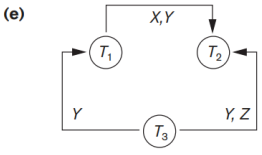


Equivalent serial schedules

None

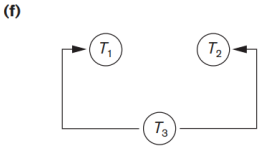
Reason

Cycle $X(T_1 \rightarrow T_2), Y(T_2 \rightarrow T_1)$
Cycle $X(T_1 \rightarrow T_2), YZ(T_2 \rightarrow T_3), Y(T_3 \rightarrow T_1)$



Equivalent serial schedules

$T_3 \rightarrow T_1 \rightarrow T_2$



Equivalent serial schedules

$T_3 \rightarrow T_1 \rightarrow T_2$

$T_3 \rightarrow T_2 \rightarrow T_1$

How Serializability Is Used for Concurrency Control :

⇨ Difference between serial and serilizability :

A serial schedule represents inefficient processing because no interleaving of operations from different transactions is permitted. This can lead to low CPU utilization while a transaction waits for disk I/O, or for a long transaction to delay other transactions, thus slowing down transaction processing considerably. A serializable schedule gives the benefits of concurrent execution without giving up any correctness.

More of this in next and last chapter.

View Equivalence and View Serializability:

First question : we know that If graph precedence graph contains no loop then it is conflict serializable, If it is CS than it serial and if it is serial than it is consistence. But what if precedence graph contains loop then it is not conflict serializable but is it consistent or not ?

To check whether it is consistence or not we use view equivalence and view serializability.

The idea behind *view equivalence* is that, as long as each read operation of a transaction reads the result of the same write operation in both schedules, the write operations of each transaction must produce the same results. A schedule S is said to be *view serializable* if it is view equivalent to a serial schedule. A *blind write* is a write operation in a transaction T on an item X that is not dependent on the old value of X, if read is not present before write. Then it is *blind write*. And if blind write then view equivalance.

Example :

schedule Sg of three transactions T1: r1(X); w1(X); T2: w2(X); and T3: w3(X):

Sg: r1(X); w2(X); w1(X); w3(X); c1; c2; c3;

In Sg the operations w2(X) and w3(X) are blind writes, since T2 and T3 do not read the value of X. The schedule Sg is view serializable, since it is view equivalent to the serial schedule T1, T2, T3. However, Sg is not conflict serializable, since it is not conflict equivalent to any serial schedule (as an exercise, the reader should construct the serializability graph for Sg and check for cycles, if cycle exists then it is not serializable if not than it is serializable). For schedule to be view equal, three things must be considered initial read, updated read, final write. Updated read is combination of write after read. In Sg, r1(X) is initial read, there is no updated read and w3(X) is final write so you cannot change the position of final write and position of write and read involved in updated read.

NOTE : It has been shown that any conflict-serializable schedule is also view serializable but not vice versa. the problem of testing for view serializability has been shown to be NP-hard, meaning that finding an efficient polynomial time algorithm for this problem is highly unlikely.

Concurrency Control Techniques

1) Two-Phase Locking Techniques for Concurrency Control :

A lock is a variable associated with a data item that describes the status of the item with respect to possible operations that can be applied to it. Generally, there is one lock for each data item in the database. Locks are used as a means of synchronizing the access by concurrent transactions to the database items.

A) Types of Locks and System Lock Tables :

a) Binary Locks : A binary lock can have two states or values: locked and unlocked (or 1 and 0, for simplicity). A distinct lock is associated with each database item X. If the value of the lock on X is 1, item X cannot be accessed by a database operation that requests the item. If the value of the lock on X is 0, the item can be accessed when requested, and the lock value is changed to 1. We refer to the current value (or state) of the lock associated with item X as lock(X).

```
lock_item(X):
B:  if LOCK(X) = 0              (*item is unlocked*)
        then LOCK(X) ←1         (*lock the item*)
    else
        begin
        wait (until LOCK(X) = 0
            and the lock manager wakes up the transaction);
        go to B
        end;
unlock_item(X):
    LOCK(X) ← 0;                (* unlock the item *)
    if any transactions are waiting
        then wakeup one of the waiting transactions;
```

Figure 21.1
Lock and unlock operations for binary locks.

In its simplest form, each lock can be a record with three fields: plus a queue for transactions that are waiting to access the item. The system needs to maintain only these records for the items that are currently locked in a *lock table*, which could be organized as a hash file on the item name. Items not in the lock table are considered to be unlocked. The DBMS has a *lock manager subsystem* to keep track of and control access to locks. Between the lock_item(X) and unlock_item(X) operations in transaction T, T is said to *hold the lock* on item X.

If the simple binary locking scheme described here is used, every transaction must obey the following rules:

1. A transaction T must issue the operation lock_item(X) before any read_item(X) or write_item(X) operations are performed in T.
2. A transaction T must issue the operation unlock_item(X) after all read_item(X) and write_item(X) operations are completed in T.

3. A transaction T will not issue a lock_item(X) operation if it already holds the lock on item X.
4. A transaction T will not issue an unlock_item(X) operation unless it already holds the lock on item X.

b) Shared/Exclusive (or Read/Write) Locks. The preceding binary locking scheme is too restrictive for database items because at most one transaction can hold a lock on a given item. We should allow several transactions to access the same item X if they all access X for reading purposes only. This is because read operations on the same item by different transactions are not conflicting (see Section 21.4.1). However, if a transaction is to write an item X, it must have exclusive access to X.

For this purpose, a different type of lock, called a *multiple-mode lock*, is used. In this scheme—called *shared/exclusive or read/write locks*—there are three locking operations: *read_lock(X), write_lock(X), and unlock(X)*. A lock associated with an item X, LOCK(X), now has three possible states: *read-locked, write-locked, or unlocked.* A *read-locked item* is also called *share-locked* because other transactions are allowed to read the item, whereas a *write-locked* item is called *exclusive-locked* because a single transaction exclusively holds the lock on the item. Each record in the lock table will have four fields: <Data_item_name, LOCK, No_of_reads, Locking_transaction(s)>

```
read_lock(X):
B:  if LOCK(X) = "unlocked"
        then begin LOCK(X) ← "read-locked";
            no_of_reads(X) ← 1
            end
    else if LOCK(X) = "read-locked"
        then no_of_reads(X) ← no_of_reads(X) + 1
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
write_lock(X):
B:  if LOCK(X) = "unlocked"
        then LOCK(X) ← "write-locked"
    else begin
            wait (until LOCK(X) = "unlocked"
                and the lock manager wakes up the transaction);
            go to B
            end;
unlock (X):
    if LOCK(X) = "write-locked"
        then begin LOCK(X) ← "unlocked";
                wakeup one of the waiting transactions, if any
                end
    else it LOCK(X) = "read-locked"
        then begin
                no_of_reads(X) ← no_of_reads(X) −1;
                if no_of_reads(X) = 0
                    then begin LOCK(X) = "unlocked";
                            wakeup one of the waiting transactions, if any
                            end
                end;
```

Figure 21.2
Locking and unlocking operations for two-mode (read/write, or shared/exclusive) locks.

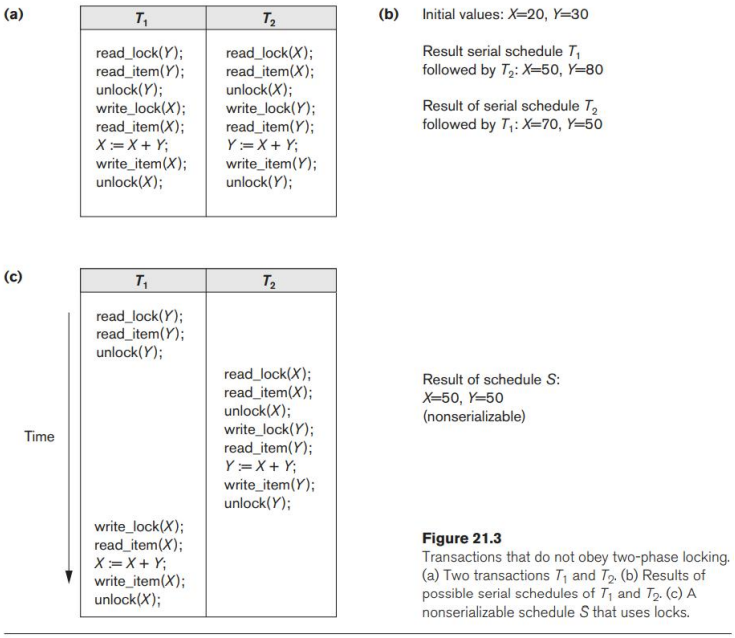


Figure 21.3
Transactions that do not obey two-phase locking. (a) Two transactions T_1 and T_2. (b) Results of possible serial schedules of T_1 and T_2. (c) A nonserializable schedule S that uses locks.

Using binary locks or read/write locks in transactions, as described earlier, does not guarantee serializability of schedules on its own. Figure 21.3 shows an example where the preceding locking rules are followed but a nonserializable schedule may result. This is because in Figure 21.3(a) the items Y in T1 and X in T2 were unlocked too early.

Note : An exclusive lock is released at the end of to avoid cascading rollback problem and to ensure recoverability.

c) Conversion (Upgrading, Downgrading) of Locks. It is desirable to relax conditions 4 and 5 in the preceding list in order to allow *lock conversion*; that is, a transaction that already holds a lock on item X is allowed under certain conditions to **convert** the lock from one locked state to another. For example, it is possible for a transaction T to issue a read_lock(X) and then later to **upgrade** the lock by issuing a write_lock(X) operation. It is also possible for a transaction T to issue a write_lock(X) and then later to **downgrade** the lock by issuing a read_lock(X) operation.

d) Guaranteeing Serializability by Two-Phase Locking :

A transaction is said to follow the *two-phase locking protocol* if all locking operations (read_lock, write_lock) precede the first unlock operation in the transaction. Such a transaction can be divided into two phases: an *expanding or growing (first) phase*, during which new locks on items can be acquired but none can be released; and a *shrinking (second) phase*, during which existing locks can be released but no new locks can be acquired.

T_1'	T_2'
read_lock(Y);	read_lock(X);
read_item(Y);	read_item(X);
write_lock(X);	write_lock(Y);
unlock(Y)	unlock(X)
read_item(X);	read_item(Y);
X := X + Y;	Y := X + Y;
write_item(X);	write_item(Y);
unlock(X);	unlock(Y);

Figure 21.4
Transactions T_1' and T_2', which are the same as T_1 and T_2 in Figure 21.3 but follow the two-phase locking protocol. Note that they can produce a deadlock.

Although the two-phase locking protocol guarantees serializability (that is, every schedule that is permitted is serializable), it does not permit all possible serializable schedules (that is, some serializable schedules will be prohibited by the protocol).

- Basic, Conservative, Strict, and Rigorous Two-Phase Locking. :
  There are a number of variations of two-phase locking (2PL). The technique just described is known as basic 2PL.
  a) Conservative 2PL : requires a transaction to lock all the items it accesses before the transaction begins execution, by predeclaring its read-set and write-set. read-set of a transaction is the set of all items that the transaction reads, and the write-set is the set of all items that it writes. (advantage ensure no dead lock)
  b) Strict 2PL (Recoverability-less protocol) : a transaction T does not release any of its exclusive (write) locks until after it commits or aborts. Hence, no other transaction can read or write an item that is written by T unless T has committed, leading to a strict schedule for recoverability. (Disadvantages recoverability)
  c) Rigorous 2PL : A transaction T does not release any of its locks (exclusive or shared) until after it commits or aborts, and so it is easier to implement than strict 2PL.
- Difference between three of this :

strict and rigorous 2PL : the former holds write-locks until it commits, whereas the latter holds all locks (read and write).
conservative and rigorous 2PL : the former must lock all its items before it starts, so once the transaction starts it is in its shrinking phase; the latter does not unlock any of its items until after it terminates (by committing or aborting), so the transaction is in its expanding phase until it ends.

B) Dealing with Deadlock and Starvation : Deadlock occurs when each transaction T in a set of two or more transactions is waiting for some item that is locked by some other transaction T' in the set.

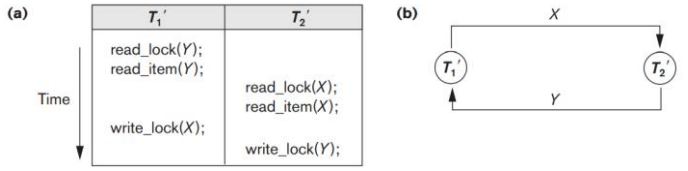


Figure 21.5
Illustrating the deadlock problem. (a) A partial schedule of T_1' and T_2' that is in a state of deadlock. (b) A wait-for graph for the partial schedule in (a).

a) **Deadlock Prevention Protocols** : The timestamps are typically based on the order in which transactions are started; hence, if transaction T1 starts before transaction T2, then TS(T1) < TS(T2).

■ **Wait-die.** If TS(Ti) < TS(Tj), then (Ti older than Tj) Ti is allowed to wait; otherwise (Ti younger than Tj) abort Ti (Ti dies) and restart it later with the same timestamp. If an older transaction requests for a resource held by a younger transaction then older transaction is allowed to wait. And if an younger transaction requests for a resource held by a older transaction then younger transaction should be aborted and should be restarted later with the same timestamp.

■ **Wound-wait**. If TS(Ti) < TS(Tj), then (Ti older than Tj) abort Tj (Ti wounds Tj) and restart it later with the same timestamp; otherwise (Ti younger than Tj) Ti is allowed to wait. If an older transaction requests for a resource held by a younger transaction, then an older transaction forces a younger transaction to kill the transaction and release the resource.

This both avoids *Starvation.*

■ **no waiting algorithm :** If a transaction is unable to obtain a lock, it is immediately aborted and then restarted after a certain time delay without checking whether a deadlock will actually occur or not. In this case, no transaction ever waits, so no deadlock will occur. However, this scheme can cause transactions to abort and restart needlessly.

■ **Cautious waiting.** If Tj is not blocked (not waiting for some other locked item), then Ti is blocked and allowed to wait; otherwise abort Ti.

Deadlock Detection :

A simple way to detect a state of deadlock is for the system to construct and maintain a wait-for graph. One node is created in the wait-for graph for each transaction that is currently executing. Whenever a transaction Ti is waiting to lock an item X that is currently locked by a transaction Tj , a directed edge (Ti → Tj) is created in the wait-for graph.

If the system is in a state of deadlock, some of the transactions causing the deadlock must be aborted. Choosing which transactions to abort is known as victim selection.

Timeouts : if a transaction waits for a period longer than a system-defined timeout period, the system assumes that the transaction may be deadlocked and aborts it—regardless of whether a deadlock actually exists.

Starvation. Another problem that may occur when we use locking is starvation, which occurs when a transaction cannot proceed for an indefinite period of time while other transactions in the system continue normally. One solution for starvation is to have a fair waiting scheme, such as using a *first-come-first-served queue*; transactions are enabled to lock an item in the order in which they originally requested the lock.

2) **Concurrency Control Based on Timestamp Ordering :**

A) *Timestamp :* Recall that a timestamp is a unique identifier created by the DBMS to identify a transaction. Typically, timestamp values are assigned in the order in which the transactions are submitted to the system, so a timestamp can be thought of as the transaction start time. We will refer to the timestamp of transaction T as TS(T). Concurrency control techniques based on timestamp ordering do not use locks; hence, deadlocks cannot occur.
Oldest Transection has lowest timestamp value. We always assume one thing that oldest transaction will complete first in all timestamp protocol.

1. *read_TS(X).* The read timestamp of item X is the last or latest timestamp among all the timestamps of transactions that have successfully read item X—that is, read_TS(X) = TS(T), where T is the youngest transaction that has read X successfully.
2. *write_TS(X).* The write timestamp of item X is the last or latest of all the timestamps of transactions that have successfully written item X—that is, write_TS(X) = TS(T), where T is the youngest transaction that has written X successfully.

B) *Basic Timestamp Ordering (TO)* : Whenever some transaction T tries to issue a read_item(X) or a write_item(X) operation, the basic TO algorithm compares the timestamp of T with read_TS(X) and write_TS(X) to ensure that the timestamp order of transaction execution is not violated. If this order is violated, then transaction T is aborted and resubmitted to the system as a new transaction with a new timestamp.

If T is aborted and rolled back, any transaction T1 that may have used a value written by T must also be rolled back. Similarly, any transaction T2 that may have used a value written by T1 must also be rolled back, and so on. This effect is known as *cascading rollbac*k and is one of the problems associated with basic TO, since the schedules produced are not guaranteed to be *recoverable.*

1. Whenever a transaction T issues a write_item(X) operation, the following check is performed:
a. If read_TS(X) > TS(T) or if write_TS(X) > TS(T), then abort and roll back T and reject the operation. This should be done because some younger transaction with a timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already read or written the value of item X before T had a chance to write X, thus violating the timestamp ordering.
b. If the condition in part (a) does not occur, then execute the write_item(X) operation of T and set write_TS(X) to TS(T).

2. Whenever a transaction T issues a read_item(X) operation, the following check is performed:
a. If write_TS(X) > TS(T), then abort and roll back T and reject the operation. This should be done because some younger transaction with timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already written the value of item X before T had a chance to read X.
b. If write_TS(X) ≤ TS(T), then execute the read_item(X) operation of T and set read_TS(X) to the larger of TS(T) and the current read_TS(X).

In sort it says that if youngest transaction wants to read/write on X before oldest transaction than current transaction must be rollback. *If you are stuck at this just watch gate smashers video and don't waste time.*

NOTE : The schedules produced by basic TO are hence guaranteed to be conflict serializable. As mentioned earlier, deadlock does not occur with timestamp ordering.

Strict Timestamp Ordering (TO). A variation of basic TO called strict TO ensures that the schedules are both strict (for easy recoverability) and (conflict) serializable. In this variation, a transaction T issues a read_item(X) or write_item(X) such that TS(T) > write_TS(X) has its read or write operation delayed until the transaction T' that wrote the value of X (hence TS(T') = write_TS(X)) has committed or aborted.

Thomas's Write Rule. A modification of the basic TO algorithm, known as Thomas's write rule, does not enforce conflict serializability, but it rejects fewer write operations by modifying the checks for the write_item(X) operation as follows:
1. If read_TS(X) > TS(T), then abort and roll back T and reject the operation.
2. If write_TS(X) > TS(T), then do not execute the write operation but continue processing. This is because some transaction with timestamp greater than TS(T)—and hence after T in the timestamp ordering—has already written the value of X. Thus, we must ignore the write_item(X) operation of T because it is already outdated and obsolete. Notice that any conflict arising from this situation would be detected by case (1).
3. If neither the condition in part (1) nor the condition in part (2) occurs, then execute the write_item(X) operation of T and set write_TS(X) to TS(T).

Note :

1) In computing, a blind write occurs when a transaction writes a value without reading it. Any view serializable schedule that is not conflict serializable must contain a blind write.

Questions on Transactions :

1) Consider the following schedule S of transactions T1, T2, T3, T4:

T1	T2	T3	T4
	Reads(X)		
		Writes(X)	
		Commit	
Writes(X)			
Commit			
	Writes(Y)		
	Reads(Z)		
	Commit		
			Reads(X)
			Reads(Y)
			Commit

Which one of the following statements is CORRECT?

S is conflict-serializable but not recoverable
S is both conflict-serializable and recoverable
S is not conflict-serializable but is recoverable
S is neither conflict-serializable not is it recoverable

Answer : S is both conflict serializable and recoverable.

Recoverable? Look if there are any dirty reads? Since there are no dirty read, it simply implies schedule is recoverable(if there were dirty read, then we would have taken into consideration the order in which transactions commit)

Conflict serializable? Draw the precedence graph(make edges if there is a conflict instruction among Ti and Tj. But for the given schedule, no cycle exists in precedence graph, thus it's conflict serializable.

2) Consider the following log sequence of two transactions on a bank account, with initial balance 12000, that transfer 2000 to a mortgage payment and then apply a 5% interest.

1. T1 start
2. T1 B old=12000 new=10000
3. T1 M old=0 new=2000
4. T1 commit
5. T2 start
6. T2 B old=10000 new=10500
7. T2 commit

Suppose the database system crashes just before log record 7 is written. When the system is restarted, which one statement is true of the recovery procedure?

We must redo log record 6 to set B to 10500
We must undo log record 6 to set B to 10000 and then redo log records 2 and 3.
We need not redo log records 2 and 3 because transaction T1 has committed.
We can apply redo and undo operations in arbitrary order because they are idempotent

Answer : Answer should be B. Here we are not using checkpoints. Checkpoints are inserted to minimize the task of undo-redo in recoverability. Checkpoints are inserted not to insure recoverability. So, redo log records 2 and 3 and undo log record 6.

Consider the following steps taken from the book 'Navathe':

PROCEDURE RIU_M :

- Use two lists of transactions maintained by the system: the committed transactions since the last checkpoint and the active transactions.
- Undo all the write_item operations of the active (uncommitted) transaction, using the UNDO procedure. The operations should be undone in the reverse order in which they were written into the log.
- Redo all the write_item operations of the committed transactions from the log, in the order in which they were written into the log.

3) Which of the following scenarios may lead to an irrecoverable error in a database system ?

A transaction writes a data item after it is read by an uncommitted transaction
A transaction reads a data item after it is read by an uncommitted transaction
A transaction reads a data item after it is written by a committed transaction
A transaction reads a data item after it is written by an uncommitted transaction

Answer : D, Dirty read and understand each word of each sentence.

4) Consider the following transaction involving two bank accounts x and y.

read(x); x := x – 50; write(x); read(y); y := y + 50; write(y)

The constraint that the sum of the accounts x and y should remain constant is that of

Atomicity
Consistency
Isolation
Durability

Answer : Consistency in database systems refers to the requirement that any given database transaction must only change affected data in allowed ways, that is sum of x and y must not change. Again read ACID properties.

5) Consider a simple checkpointing protocol and the following set of operations in the log.
(start, T4); (write, T4, y, 2, 3); (start, T1); (commit, T4); (write, T1, z, 5, 7);
(checkpoint);
(start, T2); (write, T2, x, 1, 9); (commit, T2); (start, T3); (write, T3, z, 7, 2);
If a crash happens now and the system tries to recover using both undo and redo operations, what are the contents of the undo list and the redo list

Undo: T3, T1; Redo: T2
Undo: T3, T1; Redo: T2, T4
Undo: none; Redo: T2, T4, T3; T1
Undo: T3, T1, T4; Redo: T2

Answer : Since T1 and T3 are not committed yet, they must be undone. The transaction T2 must be redone because it is after the latest checkpoint. T4 should not be included in redo because it is done and committed before checkpoint.

6) Which level of locking provides the highest degree of concurrency in a relational data base?

Page *Row*
Table *Page, table and row level locking allow the same degree of concurrency*

Answer : Row level locking provides more concurrency, because different transactions can access different rows in a table / page at same time.

7) Suppose a database schedule S involves transactions T1, ...Tn. Construct the precedence graph of S with vertices representing the transactions and edges representing the conflicts. If S is serializable, which one of the following orderings of the vertices of the precedence graph is guaranteed to yield a serial schedule?

Topological order *Breadth-first order*
Depth-first order *Ascending order of transaction indices*

Answer : A
DFS and BFS traversal of graph are possible even if graph contains cycle. And hence DFS and BFS are also possible for non serializable graphs. But Topological sort of any cyclic graph is not possible. Thus topological sort guarantees graph to be serializable.

8) Consider the following database schedule with two transactions, T1 and T2.
S = r2(X); r1(X); r2(Y); w1(X); r1(Y); w2(X); a1; a2;
where ri(Z) denotes a read operation by transaction Ti on a variable Z, wi(Z) denotes a write operation by Ti on a variable Z and ai denotes an abort by transaction Ti . Which one of the following statements about the above schedule is TRUE?

S is non-recoverable *S does not have a cascading abort*
S is recoverable, but has a cascading abort *S is strict*

Answer : Dirty read means process T1 writes the item and then T2 reads its then T1 abort for some reason. Means t2 has read the wrong value of variable. This is called dirty read.
(A): This is not possible, because we have no dirty read ! No dirty read ⟹ Recoverable
(B): This is not possible, because of no Dirty read ! No dirty read ⟹ No cascading aborts !
(D): This is not true, because we can see clearly in image that after W1(X) before T1 commit or aborts T2 does W2(x) !
C is only option remaining !

9) Consider the following three schedules of transactions T1, T2 and T3. [Notation: In the following NYO represents the action Y (R for read, W for write) performed by transaction N on object O.]

(S_1)	(S_2)	(S_3)
2RA	3RC	2RA
2WA	2RA	3RC
3RC	2WA	3WA
2WB	2WB	2WA
3WA	3WA	2WB
3WC	1RA	3WC
1RA	1RB	1RA
1RB	1WA	1RB
1WA	1WB	1WA
1WB	3WC	1WB

Which of the following statements is TRUE?

S1, S2 and S3 are all conflict equivalent to each other *S2 is conflict equivalent to S3, but not to S1*
No two of S1, S2 and S3 are conflict equivalent to each other *S1 is conflict equivalent to S2, but not to S3*

Answer : Two schedules are conflict equivalent when the precedence graphs are isomorphic.
For S1, edges in precedence graph are: 2->3, 3->1, 2->1.
For S2, edges in precedence graph are: 2->1, 3->1, 2->3.

For S3, edges in precedence graph are: 3->1, 3->2, 2->1
Hence, S1 is conflict equivalent to S2, but not to S3.

10) For the schedule given below, which of the following is Correct?

1	Read A	
2		Read B
3	Write A	
4		Read A
5		Write A
6		Write B
7	Read B	
8	Write B	

This schedule is serializable and can occur in a scheme using 2PL protocol.
This schedule is serializable but cannot occur in a scheme using 2PL protocol.
This schedule is not serializable but can occur in a scheme using 2PL protocol.
This schedule is not serializable and cannot occur in a scheme using 2PL protocol.
Answer : Precedence graph contains cycle so this is not conflict serializable. Now, if 2PL them conflict serializable, but it is not CS so it is not 2PL ... by using truth table.

11) Consider the given schedule and choose the suitable option.
S = T1:R(x), T1:R(y), T1:W(x), T2:R(y), T3:W(y), T1:W(x), T2:R(y)

Schedule is view serializable
Schedule is view serializable but not conflict serializable
Schedule is conflict serializable but not view serializable
Neither view serializable nor conflict serializable

Answer : Since there is a cycle T2→T3→T2 produced due to R(y)W(y) and W(y)R(y) conflicts.
Since the graph contains a blind write at T3 so view serializable schedule is not equal to conflict serializable schedule.
T1:R(y) and T2:R(y) are initial reads and T2:R(y) is at last and also we have only 1 Write operation by T3:W(y) so we can't make a serial schedule by executing in the any order.
So the sequence is not view serializable.
∴ Option D. is the correct answer.

12) Which of the following is the highest isolation level in transaction management?

Serializable
Committed Read
Repeated Read
Uncommitted Read

Answer : Serializable

13) Let us assume that transaction T1 has arrived before transaction T2. Consider the schedule
S=r1(A); r2(B) ; w2(A); w1(B)
Which of the following is true?

Allowed under basic timestamp protocol.
Not allowed under basic timestamp protocols because T2 is rolled back
Not allowed under basic timestamp protocols because T1 is rolled back
None of these

Answer : If a schedule is conflict serializable schedule then it is allowed in basic timestamp protocol but the given schedule is not conflict serializable so it is not allowed in basic timestamp.
Here the order is T1 followed by T2, this must be maintained but again there is a conflict operation r2(B) -> w1(B), which is showing T2 is followed by T1 which is incorrect hence T1 must be rolled back.
Or you can use basic timestamp protocol which says that timestamp of B was higher since it is youngest and T1 is going to write so TS(B)<read_TS(B) T1 is rolled back.

14) Suppose a database system crashes again while recovering from a previous crash. Assume checkpointing is not done by the database either during the transactions or during recovery.
Which of the following statements is/are correct?

The same undo and redo list will be used while recovering again
All the transactions that are already undone and redone will not be recovered again
The system cannot recover any further
The database will become inconsistent

Answer :

Support for option A and against option C: Since check-pointing is not used we have to depend on the system logs. Let's suppose we have three transactions A, B and C. Also assume that transaction A and C commits before failure and B was started but the system crashed before it can commit. So, in the first recovery process database will redo A and C as per the system logs. Now consider that while redoing A successfully commits, the system crashed for the second time before the B can commit. So, while recovering for the second time the same system logs will be used. However, it is should be noted that the system logs will also have

entry to redo transaction A since it was committed after the first failure. However, the undo/redo operations are idempotent (they are the same no matter how many time they are executed).

Against option B: If the system crashes again same logic as above can be used for recovery.

Against option D: Inconsistency refers to situations (generally) when the value of a shared variable varies in two or more transactions, but that doesn't seem to happen here as no uncommitted transaction's data is being read/written during the entire recovery process.

Conclusion: So, the only option of selecting the same list for undo/redo seems to be correct.

Consider the relation $R(P, Q, S, T, X, Y, Z, W)$ with the following functional dependencies.

$$PQ \rightarrow X; \quad P \rightarrow YX; \quad Q \rightarrow Y; \quad Y \rightarrow ZW$$

Consider the decomposition of the relation R into the constituent relations according to the following two decomposition schemes.

- $D_1: \quad R = [(P, QS, T); \ (P, T, X); \ (Q, Y); \ (Y, Z, W)]$
- $D_2: \quad R = [(P, Q, S); \ (T, X); \ (Q, Y); \ (Y, Z, W)]$

Which one of the following options is correct?

A. D_1 is a lossless decomposition, but D_2 is a lossy decomposition
B. D_1 is a lossy decomposition, but D_2 is a lossless decomposition
C. Both D_1 and D_2 are lossless decompositions
D. Both D_1 and D_2 are lossy decompositions

15)

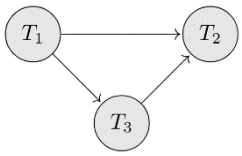
Answer : A, https://youtu.be/HrggFKi-awQ <- Chess Method, best method for non-binary decomposition.
Table contain all 'a' entries in first row so D1 is lossless and if a is not present in any row than it is lossy.

16) Which of the following concurrency control protocols ensure both conflict serializability and freedom from deadlock? I. 2-phase locking II. Time-stamp ordering

I only *Both I and II*
II only *Neither I nor II*

Answer : Read 2 phase locking again plz for heaven sake. answer is II only.

17) Consider the following schedule for transactions T1,T2 and T3:
Which one of the schedules below is the correct serialization of the above?



T1→T3→T2 *T2→T3→T1*
T2→T1→T3 *T3→T1→T2*

Answer : From precedence graph we remove each node containing minimum amount of incident arrow T1→T3→T2.

18) Consider the following two phase locking protocol. Suppose a transaction T accesses (for read or write operations), a certain set of objects {O1,...,Ok}. This is done in the following manner: Step 1. T acquires exclusive locks to O1, . . . , Ok in increasing order of their addresses. Step 2. The required operations are performed. Step 3. All locks are released. This protocol will

guarantee serializability and deadlock-freedom *guarantee serializability but not deadlock-freedom*
guarantee neither serializability nor deadlock-freedom *guarantee deadlock-freedom but not serializability*

Answer : The above scenario is Conservative 2PL(or Static 2PL). In Conservative 2PL protocol, a transaction has to lock all the items it accesses before the transaction begins execution. It is used to avoid deadlocks. Also, 2PL is conflict serializable, therefore it guarantees serializability. Therefore, option A

Advantages of Conservative 2PL : No possibility of deadlock, Ensure serializability.

Drawbacks of Conservative 2PL :

Less throughput and resource utilisation because it hold the resources before the transaction begins execution, Starvation is possible since no restriction on unlock operation, 2pl is a deadlock free protocol but it is difficult to use in practice.

19) The P and V operations on counting semaphore 'a' are defined as follows :

P(a) : a = a – 1; if a<0 then wait.
V(a) : a = a + 1; if a<= then wake-up a process waiting on a.

Assume that X and Y be the wait and signal operations on binary semaphore are provided. Two binary semaphore b = 1 and c = 0 are used implement the semaphore operations P and V.

```
P(a) :  X(b)                    V(a) :  X(b)
        a = a - 1                       a = a + 1
        if (a < 0)                      if (a ≤ 0)
{       E1 ;                            E3 ;
        E2 ;                            E4 ;
}
else Y(b)
```

In order to get correct operations which option is correct.

$E_1 : Y(\mathrm{b})$ $E_3 : Y(c)$ $E_2 : X(c)$ $E_4 : X(\mathrm{b})$

$E_1 : X(\mathrm{c})$ $E_3 : Y(c)$ $E_2 : Y(b)$ $E_4 : Y(\mathrm{b})$

$E_1 : Y(b)$ $E_3 : Y(c)$ $E_2 : X(c)$ $E_4 : Y(\mathrm{b})$

Answer : Third option is true. As we can not suspend the values of b and c. suspend means if b = 0 we cannot apply wait on it. And similarly for c.