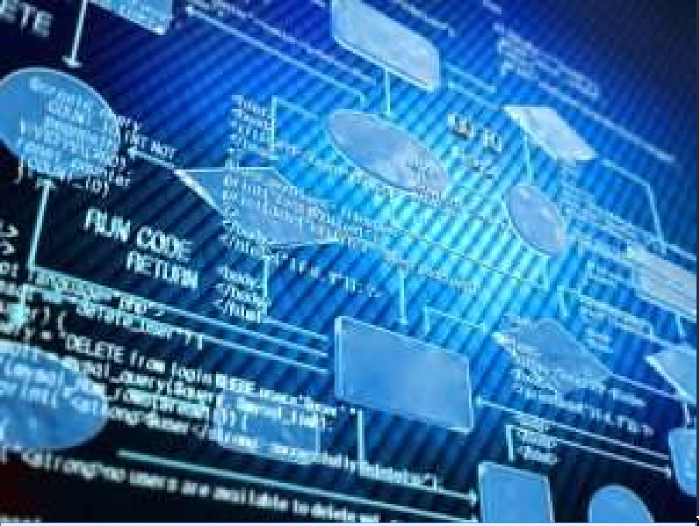
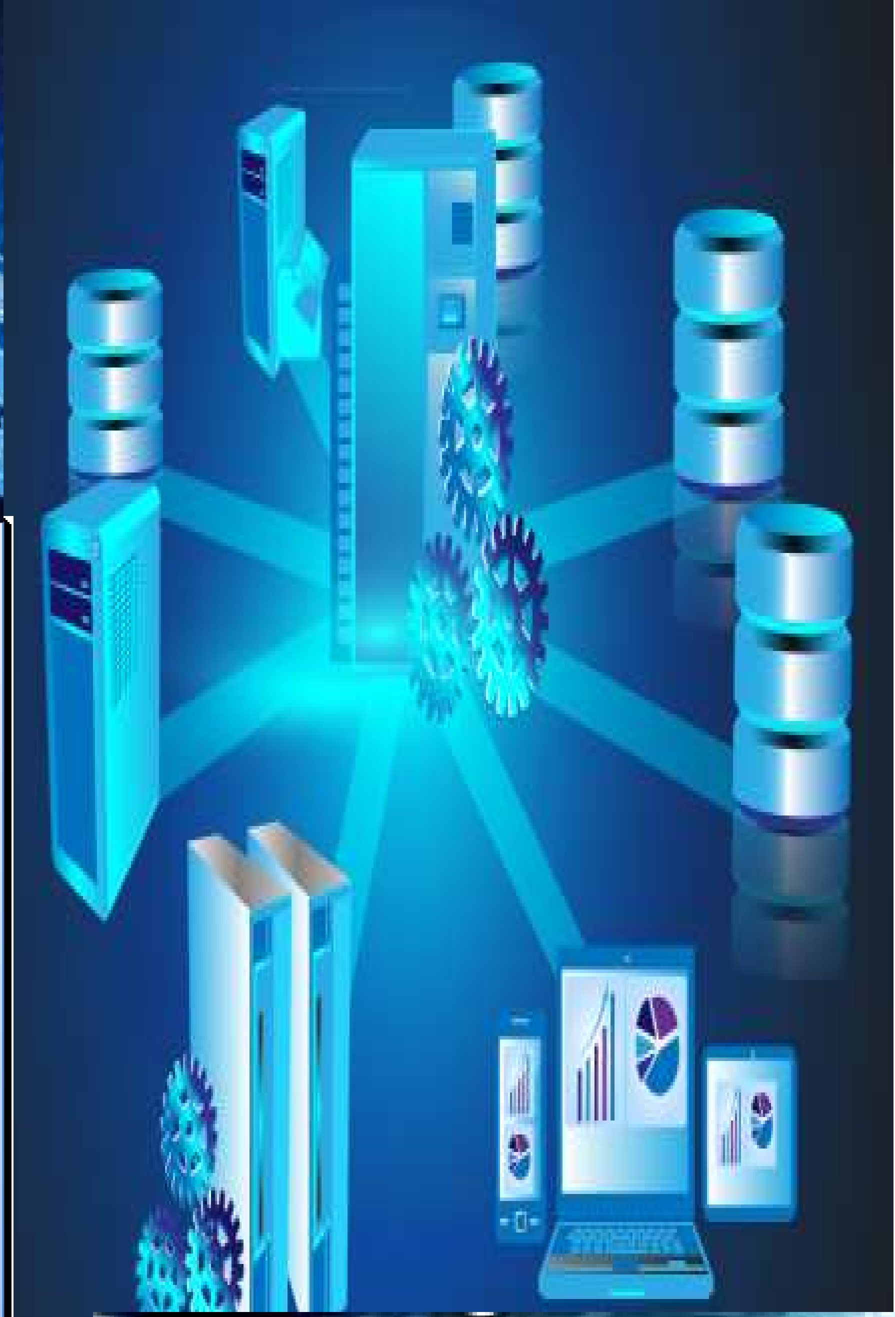
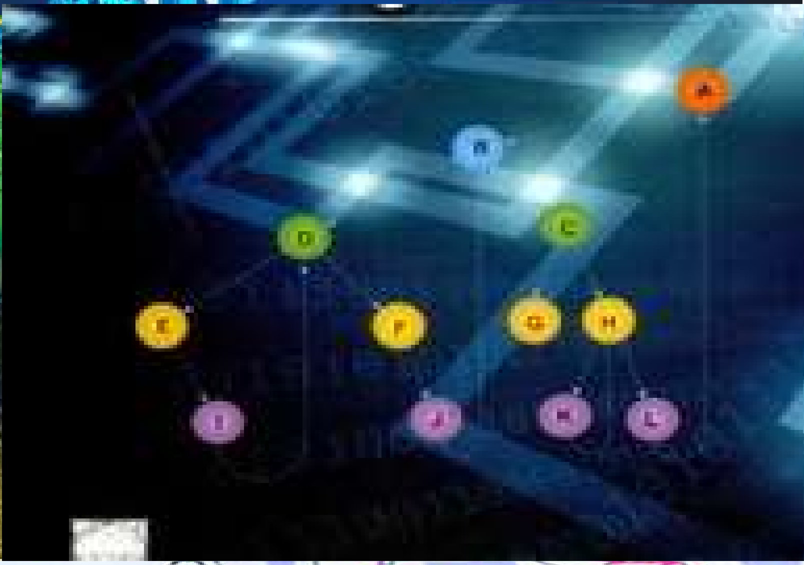


# A Study Outline of Database Management System





ABU SALEH MUSA MIAH(ABID)

A Guidebook of

# Database Management System

**Engr. Abu Saleh Musa Miah (Abid)**
**Lecturer, Bangladesh Army University of Science and Tehcnology(BAUST)**
M.Sc and B.Sc.Engg. in Computer Science and Engineering(**First Class First**)
**Email**:abusalehcse.ru@gmail.com
**Cell:+88-01734264899**

# Database Management System

============================================================================================

| | | |
|---|---|---|
| Writer | : | **Engr. Abu Saleh Musa Miah (Abid).**<br>**M.Sc.Engg** and B.Sc.Engg. in Computer Science and Engineering (**First Class First**)<br>University of Rajshahi.<br>**Cell:+88-01734264899** |
| Email | : | abusalehcse.ru@gmail.com |
| Publisher | : | **Abu Syed Md Mominul Karim Masum.**<br>**Chemical Engr.**<br>Chief Executive Officer (CEO)<br>Swarm Fashion, Bangladesh.<br>Mohakhali,Dhaka. |
| Email | : | swarm.fashion@gmail.com |
| Cover page Design | : | Md. Kaiyum Nuri<br>Chief Executive Officer (CEO)<br>Jia Shah Rich Group(Textile)<br>MBA,Uttara University |
| First Publication | : | Octobor-2016 |
| Copyright | : | Writer |
| Computer Compose | : | Writer |
| Print | : | Royal Engineering Press & Publications.<br>Meherchandi, Padma Residential Area, Boalia, Rajshahi. |
| Reviewer Team | : | Md. Kaiyum Nuri<br>MBA,Uttara University<br><br>Engr. Syed Mir Talha Zobaed.<br>B.Sc. Engg. (First Class First)<br>M.Sc. Engineering (CSE)<br>University of Rajshahi |
| PRICE | : | 400.00 TAKA (Fixed Price).<br>US 05 Dollars (**Fixed Price**). |

| | | |
|---|---|---|
| OOP CPP | : | Engr. Abu Saleh Musa Miah (Abid). |
| Published by | : | Bangladesh Army University of Science and Technology<br>Saidpur cantonment, Bangladesh. |

# Dedicated

# To

[ This Page is Left Blank Intentionally ]

# লেখকের কথা

**Database Management System** কোর্সটি বাংলাদেশের প্রায় সকল বিশ্ববিদ্যালয়ের সিলেবাসে স্থান লাভ করেছে,বাজারে টেক্সটবই থাকলেও গভীর ভাবে অনুধাবন ও পরীক্ষায় ভাল মার্কস উঠানোর মত সাজানো টেক্সটবই সহজলভ্য নয় ।পরীক্ষার প্রস্তুতির স্বার্থে তাই, এই গাইডবইটি আপনাকে সহযোগিতা করবে বলে আশা করা যায় ।

এই বইয়ের পাঠক হিসেবে, আপনিই হচ্ছেন সবচেয়ে গুরুত্বপূর্ণ সমালোচক বা মন্তব্যকারী । আর আপনাদের মন্তব্য আমার কাছে মূল্যবান, কারন আপনিই বলতে পারবেন আপনার উপযোগী করে বইটি লেখা হলো কিনা অর্থাৎ বইটি কিভাবে প্রকাশিত হলে আরও ভাল হতো । সামগ্রিক ব্যাপারে আপনাদের যে কোন পরামর্শ আমাকে উৎসাহিত করবে ।

**Engr. Abu Saleh Musa Miah (Abid)**
**Writer**

# ACKNOWLEDGEMENTS

I wish to express my profound gratitude to all those who helped in making this book a reality; especially to my families, the classmates, and authority of Royal Engineering Publications for their constant motivation and selfless support. Much needed moral support and encouragement was provided on numerous occasions by my families and relatives. I will always be grateful, to the numerous web resources, anonymous tutorials hand-outs and books I used for the resources and concepts, which helped me build the foundation.

I would also like to express my profound gratitude to Engr. Syed Mir Talha Zobaed for his outstanding contribution in inspiring, editing and proofreading of this guidebook. I am also grateful to Omar Faruque Khan (Sabbir) for his relentless support and guideline in making this book a reality.

I am thankful to the following readers those have invested their valuable times to read this book carefully and have given suggestion to improve this book:
**Abu Hurayra (Principle,RESTC)**
**Engr. Mainuddun Maruf (B.Sc. Engg in Electrical and Electronics Engineering. IUT) .**
**Md.Moyeed Hossain (B.Sc.Engr. Computer Science and Engineering ,RUET, Lecturere, RESTC).**
**Md.Shamim Akhtar (B.A Honours,English,M.A. Lecturere, RESTC).**
**MD.Sharafat Hossain(B.Sc. Engg in Electrical and Electronics Engineering. RUET).**
……………………………….. And numerous anonymous readers.
I am also thankful to the different hand notes from where I have used lots of solutions, such as Dynamic Memory Allocation,Operator overloading etc

I wish to express my profound gratitude to the following writers whose books I have used in my Guidebooks:

| | | | |
|---|---|---|---|
| 1. | **A. Silberschatz** | : | **Database System Concepts,** *Mcgraw-Hill.* |
| 2. | **Raghu Ramakrishnan, Johannes Gehrke** | : | **Database Management System,** *McGraw-Hill Higher Education* |
| 3. | **James Martin** | : | **Principles of Database Management**, *Prentice-hall Of India Pvt Ltd* |
| 4. | **Ullman** | : | **Database Management systems**, *Prentice-Hall Publication.* |
| 5. | **Abey** | : | **Oracle 8i a Beginners Guide**, *McGraw Hill.* |

…………………….. and Numerous anonymous Power Point Slides and PDF chapters from different North American Universities.

# Database Management System

**CSE3121: Database Management System**
**75 Marks [70% Exam, 20% Quizzes/Class Tests, 10% Attendance]**
**3 Credits, 33 Contact hours, Exam. Time: 4 hours**

**Introduction:** Database-System Applications, Purpose of Database Systems, View of Data, Database Languages, Relational Databases, Database Design, Data Storage and Querying, Transaction Management, Database Architecture, Data Mining and Information, Retrieval, Specialty Databases, Database Users and Administrators.

**Introduction to the Relational Model:** Structure of Relational Databases, Database Schema, Keys, Schema, Diagrams, Relational Query Languages, Relational Operations.

**Introduction to SQL:** Overview of the SQL Query, Language, SQL Data Definition, Basic Structure of SQL, Queries, Additional Basic Operations, Set Operations, Null Values, Aggregate Functions, Nested Sub-queries, Modification of the Database.

**Intermediate SQL:** Join Expressions, Views, Transactions, Integrity Constraints, SQL Data Types and Schemas, Authorization.

**Advanced SQL:** Accessing SQL From a Programming, Language, Functions and Procedures, Triggers, Recursive Queries, Advanced Aggregation Features, OLAP.

**Formal Relational Query Languages:** The Relational Algebra, The Tuple Relational Calculus, The Domain Relational Calculus.

**Database Design and the E-R Model:** Overview of the Design Process, Entity-Relationship Model, Constraints, Removing Redundant Attributes in Entity Sets, Entity-Relationship Diagrams, Reduction to Relational Schemas, Entity-Relationship Design Issues, Extended E-R Features, Alternative Notations for Modeling, Data, Other Aspects of Database Design.

**Relational Database Design:** Features of Good Relational Designs, Atomic Domains and First Normal Form, Decomposition Using Functional Dependencies, Functional-Dependency Theory, Algorithms for Decomposition, Decomposition Using Multivalued Dependencies, More Normal Forms, Database-Design Process, Modeling Temporal Data, Multivalued Dependencies, Domain-Key Normal Form.

**Application Design and Development:** Application Programs and User Interfaces, Web Fundamentals, Servlets and JSP, Application Architectures, Rapid Application Development, Application Performance, Application Security, Encryption and Its Applications.

**Data Warehousing and Mining:** Decision-Support Systems, DataWarehousing, Data Mining, Classification , Association Rules, Other Types of Associations, Clustering, Other Forms of Data Mining,

**Information Retrieval:** Relevance Ranking Using Terms, Relevance Using Hyperlinks, Synonyms, Homonyms, and Ontologies, Indexing of Documents, Measuring Retrieval Effectiveness, Crawling and Indexing the Web, Information Retrieval: Beyond Ranking of Pages, Directories and Categories

**Object-Based Databases:** Complex Data Types, Structured Types and Inheritance in SQL, Table Inheritance, Array and Multiset Types in SQL, Object-Identity and Reference Types in SQL, Implementing O-R Features, Persistent Programming Languages, Object-Relational Mapping, Object-Oriented versus Object-Relational.

**XML:** Structure of XML Data, XML Document Schema, Querying and Transformation, Application Program Interfaces to XML, Storage of XML Data, XML Applications.

## Books Recommended:

| | | | |
|---|---|---|---|
| 1. | **A. Silberschatz** | : | **Database System Concepts,** *Mcgraw-Hill.* |
| 2. | **Raghu Ramakrishnan, Johannes Gehrke** | : | **Database Management System,** *McGraw-Hill Higher Education* |
| 3. | **James Martin** | : | **Principles of Database Management**, *Prentice-hall Of India Pvt Ltd* |
| 4. | **Ullman** | : | **Database Management systems**, *Prentice-Hall Publication.* |
| 5. | **Abey** | : | **Oracle 8i a Beginners Guide**, *McGraw Hill.* |

# Chapter 1

Introduction to dbms

## Important Question

1. Define Database with example. **3 Marks CSE-2011**
2. What are the main differences between file processing system and database management systems? **7 Marks CSE-2011 CSE-2007 CSE-2004 CSE-2001**
3. Define the following terms with example: (i) DDL (ii) DML (iii) Data Dictionary. **6 Marks CSE-2011 CSE-2007 CSE-2006 CSE-2003**
4. Define the following terms: (i) Data Abstraction (ii) Database Instance (ii) Metadata (iv) Data inconsistency (v) Consistency constraints. **10 Marks CSE-2010 CSE – 2008 CSE-2005 CSE-2002**
5. Mention the advantages and disadvantages of a DBMS. **5 Marks CSE-2010 CSE-2005 CSE-2003 (OLD)**
6. Define DBA. What are the functions of a database administrator? **5 Marks CSE-2010 CSE – 2008 CSE-2007 CSE-2005 CSE-2003 (OLD) CSE-2002 CSE-2000 CSE-2000(OLD)**
7. Define DBMS. **2 Marks CSE-2009 CSE-2003 CSE-2003 (OLD) CSE-2000**
8. What are the disadvantages of file processing system. **6 Marks CSE-2009**
9. What are Instance and Schema? **4 Marks CSE-2000**
10. Describe the ways to access database from application program. **4 Marks CSE-2009**
11. What are the components of Query processor? **4 Marks CSE-2009**
12. Define and explain Database Management system. Write some applications of Database system. **4 Marks CSE – 2008 CSE 2006**
13. What is data abstraction? Why data abstraction is needed? Describe different levels of data abstraction. **7 Marks CSE – 2011 CSE – 2008 CSE-2003 CSE-2003 (OLD) CSE-2001 CSE-2007**
14. Differentiate between Centralized and distributed database system. **2 Marks CSE-2006**
15. **Briefly discuss database users and user interfaces. 4 Marks CSE-2006**
16. What are the purposes of DBMS? **4 Marks CSE-2003 CSE-2000 CSE-2000(OLD)**
17. What is Data Model? Briefly describe different data models. **2+7 Marks CSE-2002**
18. What problems are caused by data redundancies? Can data redundancies be completely eliminated when the database approach is used? Why or why not? **5 Marks CSE-2001**
19. Define and explain different types of operation used in DBMS. **5 Marks CSE-2000 CSE-2000(OLD)**
20. Explain the terms Data Redundancy and Inconsistency with example. **4 Marks CSE-2000(OLD)**

**Question-01: Define and explain the following terms with examples: (a) Database and DBMS (b) DDL (c) DML (d) Data Dictionary (e) Data Abstraction (f) Database Instance (g) Metadata (h) Data inconsistency (i) Consistency constraints (j) Data Redundancy.**

**Database and DBMS:**

A database is a shared, integrated computer structure that stores:

(i) End user Data (Raw facts)
(ii) Metadata (Data About Data)

A database is a collection of data, typically describing the activities of one or more related organizations. For example, a university database might contain information about the following:

- Entities such as students, faculty, courses, and classrooms.
- Relationships between entities, such as students' enrolment in courses, faculty teaching courses, and the use of rooms for courses.

**A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data. Database management systems (DBMSs) are specially designed software applications that interact with the user, other applications, and the database itself to capture and analyse data.** A general-purpose DBMS is a software system designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include MySQL, MariaDB, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, SAP HANA, dBASE, FoxPro, IBM DB2, LibreOffice Base and FileMaker Pro

The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient. Database systems are ubiquitous today, and most people interact, either directly or indirectly, with databases many times every day.

Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information.

A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained.

**Data Definition Language:**

**A data Definition language (DDL) is a language that enables users to** specify a database schema as well as other properties of data by a set of definitions**.** A database system provides a data definition language to specify the database schema. In general, A **data definition language** or **data description language** (**DDL**) is a syntax similar to a computer programming language for defining data structures, especially database schemas.

For instance, the following statement in the SQL language defines the account table:

```
create table account
(account-number char(10),
balance integer)
```

Execution of the above DDL statement creates the account table. In addition, it updates a special set of tables called the data dictionary or data directory.

We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a **data storage and definition language**. These statements define the implementation details of the database schemas, which are usually hidden from the users.

**Data Manipulation Language:**

Data manipulation is

- The retrieval of information stored in the database

- The insertion of new information into the database
- The deletion of information from the database
- The modification of information stored in the database

**A data-manipulation language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model.** A database system provides a data manipulation language to express database queries and updates.
There are basically two types:

- **Procedural DMLs** require a user to specify what data are needed and how to get those data.
- **Declarative DMLs** (also referred to as nonprocedural DMLs) require a user to specify what data are needed without specifying how to get those data.

Declarative DMLs are usually easier to learn and use than are procedural DMLs. However, since a user does not have to specify how to get the data, the database system has to figure out an efficient means of accessing data. The DML component of the SQL language is nonprocedural.

## Data Dictionary:

**A data dictionary, or metadata repository is a "centralized repository of information about data such as meaning, relationships to other data, origin, usage, and format."** The term may have one of several closely related meanings pertaining to databases and database management systems (DBMS):

- a document describing a database or collection of databases
- an integral component of a DBMS that is required to determine its structure
- a piece of middleware that extends or supplants the native data dictionary of a DBMS

**Data dictionary or data directory is a special set of tables or a data structure that contains metadata — that is, data about data.** A database system consults the data dictionary before reading or modifying actual data.

If a data dictionary system is used only by the designers, users, and administrators and not by the DBMS Software, it is called a **passive data dictionary.** Otherwise, it is called an **active data dictionary** or **data dictionary.** When a passive data dictionary is updated, it is done so manually and independently from any changes to a DBMS (database) structure. With an active data dictionary, the dictionary is updated first and changes occur in the DBMS automatically as a result.

## Data Abstraction:

**For the database system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-systems users are not computer trained, developers hide the complexity from users through a process to simplify users' inter-actions with the system. This process of separating complexity of data structures is called data abstraction.**

Data abstraction enforces a clear separation between the *abstract* properties of a data type and the *concrete* details of its implementation. The abstract properties are those that are visible to client code that makes use of the data type—the *interface* to the data type—while the concrete implementation is kept entirely private, and indeed can change, for example to incorporate efficiency improvements over time. The idea is that such changes are not supposed to have any impact on client code, since they involve no difference in the abstract behaviour.

**Example:** For example, one could define an abstract data type called *lookup table* which uniquely associates *keys* with *values*, and in which values may be retrieved by specifying their corresponding keys. Such a lookup table may be implemented in various ways: as a hash table, a binary search tree, or even a simple linear list of (key:value) pairs. As far as client code is concerned, the abstract properties of the type are the same in each case.

## Database Instance and schema:

Databases change over time as information is inserted and deleted. **The collection of information stored in the database at a particular moment is called an instance of the database. The overall design of the database is called the database schema.** Schemas are changed infrequently, if at all.

The concept of database schemas and instances can be understood by analogy to a program written in a programming language. A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an instance of a database schema.

Database systems have several schemas, partitioned according to the levels of abstraction.

- **The physical schema** describes the database design at the physical level. The physical schema is hidden beneath the logical schema, and can usually be changed easily without affecting application programs.
- **The logical schema** describes the database design at the logical level. The logical schema is by far the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema.
- A database may also have several schemas at the view level, sometimes called **subschemas,** that describe different views of the database.

**Metadata:**

**Metadata** is "data about data". Metadata (meta-content) are defined as the data providing information about one or more aspects of the data, such as:

- Means of creation of the data
- Purpose of the data Time and date of creation
- Creator or author of the data
- Location on a computer network where the data were created
- Standards used

For example, a digital image may include metadata that describe how large the picture is, the color depth, the image resolution, when the image was created, and other data. A text document's metadata may contain information about how long the document is, who the author is, when the document was written, and a short summary of the document.

**Data Inconsistency:**

**Data inconsistency is one of the problem that occurs in the database when various copies of the same data no longer agree.** Data inconsistency exists when different and conflicting versions of the same data appear in different places. Data inconsistency creates unreliable information, because it will be difficult to determine which version of the information is correct. (It's difficult to make correct – and timely - decisions if those decisions are based on conflicting information.)

Data inconsistency is likely to occur when there is data redundancy. Data redundancy occurs when the data file/database file contains redundant - unnecessarily duplicated - data. That's why one major goal of good database design is to eliminate data redundancy.

**Example:** For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.

**Data Redundancy:**

**When the same information may be duplicated in several places (files) in files systems or several tables in database systems, then this phenomenon is called data redundancy .**

**Data redundancy** occurs in database systems which have a field that is repeated in two or more tables. **For example, the address and telephone number of a par-ticular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost.** Data redundancy leads to data anomalies and corruption and generally should be avoided by design. Database normalization prevents redundancy and makes the best possible usage of storage. Proper use of foreign keys can minimize data redundancy and chance of destructive anomalies. However, concerns of efficiency and convenience can sometimes result in redundant data design despite the risk of corrupting the data.

**For example, the address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records.**
**Question-02: Describe the ways to access database from application program. 4 Marks CSE-2009**

**Database Access from Application Programs:**
Application programs are programs that are used to interact with the database. Application programs are usually written in a host language, such as Cobol, C, C++, or Java. To access the database, DML statements need to be executed from the host language. There are two ways to do this:

- **By providing an application program interface (set of procedures) that can be used to send DML and DDL statements to the database, and retrieve the results.**
  The Open Database Connectivity (ODBC) standard defined by Microsoft for use with the C language is a commonly used application program inter-face standard. The Java Database Connectivity (JDBC)standardprovidescor-responding features to the Java language.

- **By extending the host language syntax to embed DML calls within the host language program.** Usually, a special character prefaces DML calls, and a pre-processor, called the DML precompiler, converts the DML statements to nor-mal procedure calls in the host language.

**Question-03: Define DBA. What are the functions of a database administrator? 5 Marks CSE-2010 CSE – 2008 CSE-2007 CSE-2005 CSE-2003 (OLD) CSE-2002 CSE-2000 CSE-2000(OLD)**

**Database Administrator:**
One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. **A person who has such central control over the system is called a database administrator (DBA)**. The functions of a DBA include:

- **Schema definition:** The DBA creates the original database schema by executing a set of data definition statements in the DDL.
- **Storage structure and access-method definition**: The DBA creates appropriate storage structures and access methods by writing a set of definitions, which is translated by the data-storage and data-definition –language compiler.
- **Schema and physical-organization modification:** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.
- **Granting of authorization for data access**: By granting different types of authorization, the database administrator can regulate which parts of the data-base various users can access. The authorization information is kept in a special system structure that the database system consults whenever some-one attempts to access the data in the system.
- **Integrity-Constraint Specification:** The data values stored in the database must satisfy certain consistency constraints. Such a constraint mus t be specified explicitly by the database administrator. The integrity constraints are ket in a special system structure that is consulted by the database system whenever an update takes place in the system.
- **Routine maintenance:** Examples of the database administrator's routine maintenance activities are:
  (i) Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.

(ii) Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.
(iii) Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

**Question-04: What are the components of Query processor? 4 Marks CSE-2009**

**<u>The Query Processor:</u>**
The query processor components include

- **DDL interpreter**, which interprets DDL statements and records the definitions in the data dictionary.
- **DML compiler**, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands.
  A query can usually be translated into any of a number of alternative eval-uation plans that all give the same result. The DML compiler also performs query optimization, that is, it picks the lowest cost evaluation plan from amo-ng the alternatives.
- **Query evaluation engine**, which executes low-level instructions generated by the DML compiler.

**Figure 1.4** shows these components and the connections among them.

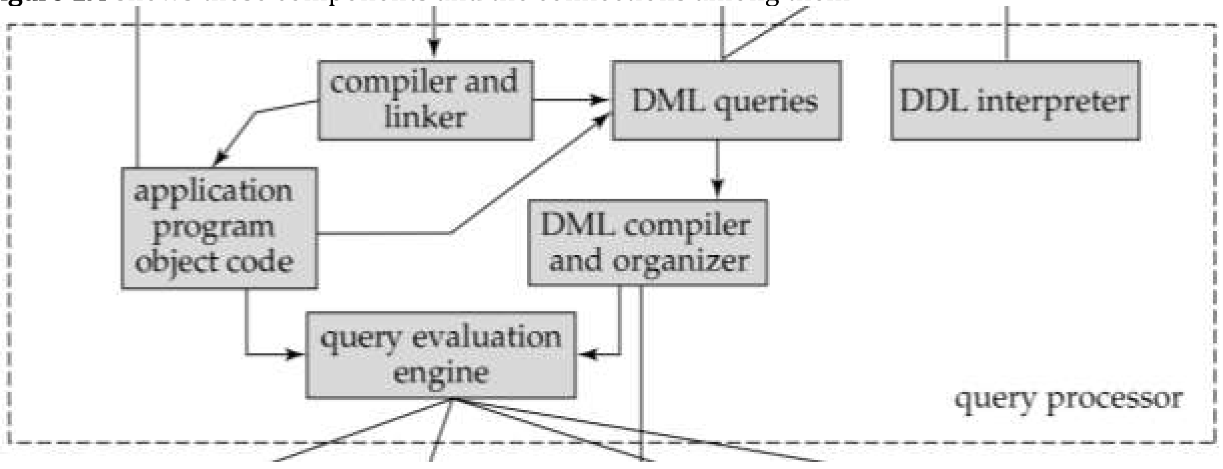


**Figure 1.4 Query Processor**

**Question-05: Briefly discuss database users and user interfaces. 4 Marks CSE-2006**

**<u>Different types of Database Users and user interfaces:</u>**

A primary goal of a database system is to retrieve information from and store new information in the database. People who work with a database can be categorized as database users or database administrators. There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

• **Naive users: Naive users** are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. The typical user interface for naive users is a forms interface, where the user can fill in appropriate fields of the form. Naive users may also simply read reports generated from the database.

• **Application programmers: Application programmers** are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. Rapid application development (RAD) tools are tools that enable an application programmer to construct forms and reports without writing a program.

• **Sophisticated users**: Sophisticated users interact with the system without writing programs. In-stead, they form their requests in a database query language. They submit each such query to a query processor, whose function is to break down DML statements into instructions that the storage manager understands. Analysts who submit queries to explore data in the database fall in this category.

• **Specialized users:** Specialized users are sophisticated users who write specialized database Applications that do not fit into the traditional data-processing framework. Among these applications are computer-aided design systems, knowledgebase and expert systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modelling systems.

**Question-06: Explain the terms Data Redundancy and Inconsistency with example. 4 Marks CSE-2000(OLD)**

<u>Data Redundancy and Inconsistency:</u>
<u>Data Redundancy:</u>
**When the same information may be duplicated in several places (files) in files systems or several tables in database systems, then this phenomenon is called data redundancy .**

**Data redundancy** occurs in database systems which have a field that is repeated in two or more tables. **For example, the address and telephone number of a par-ticular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records. This redundancy leads to higher storage and access cost.** Data redundancy leads to data anomalies and corruption and generally should be avoided by design. Database normalization prevents redundancy and makes the best possible usage of storage. Proper use of foreign keys can minimize data redundancy and chance of destructive anomalies. However, concerns of efficiency and convenience can sometimes result in redundant data design despite the risk of corrupting the data.

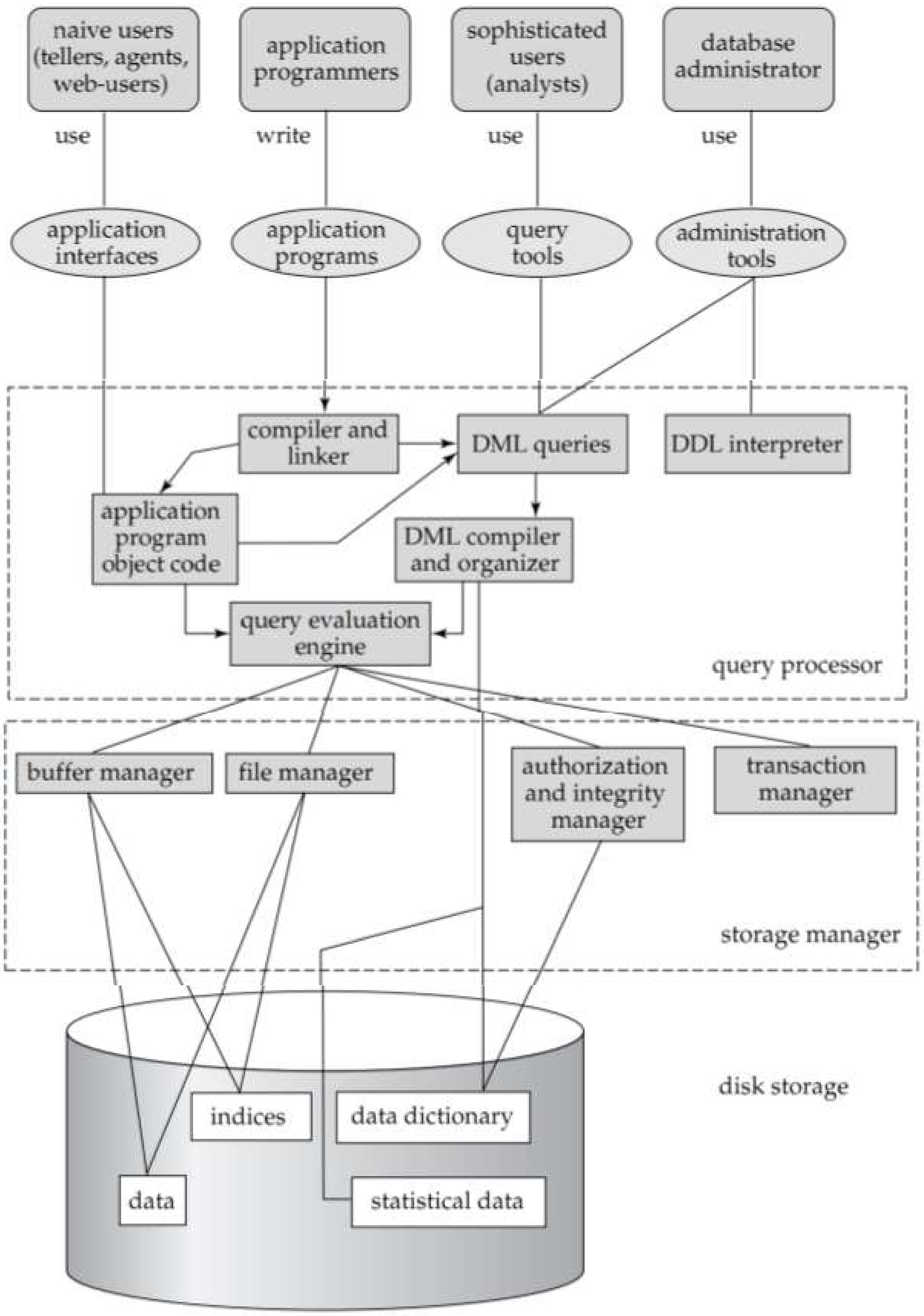
**For example, the address and telephone number of a particular customer may appear in a file that consists of savings-account records and in a file that consists of checking-account records.**

<u>Data Inconsistency:</u>
**Data inconsistency is one of the problem that occurs in the database when various copies of the same data no longer agree.** Data inconsistency exists when different and conflicting versions of the same data appear in different places. Data inconsistency creates unreliable information, because it will be difficult to determine which version of the information is correct. (It's difficult to make correct – and timely - decisions if those decisions are based on conflicting information.)

Data inconsistency is likely to occur when there is data redundancy. Data redundancy occurs when the data file/database file contains redundant - unnecessarily duplicated - data. That's why one major goal of good database design is to eliminate data redundancy.

**Example:** For example, a changed customer address may be reflected in savings-account records but not elsewhere in the system.

**Question-07: Draw the block diagram of Database Management System structure. Class Test- 2013**



**Question-08: Write some applications of Database system. 4 Marks CSE – 2008 CSE 2006**

<u>Database System Applications</u>
Databases are widely used. Here are some representative applications:

- **Banking:** For customer information, accounts, and loans, and banking transactions.

- **Airlines:** For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner — terminals situated around the world accessed the central database system through phone lines and other data networks.
- **Universities:** For student information, course registrations, and grades.
- **Credit card transactions:** For purchases on credit cards and generation of monthly statements.
- **Telecommunication:** For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.
- **Finance:** For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.
- **Sales:** For customer, product, and purchase information.
- **Manufacturing:** For management of supply chain and for tracking production of items in factories, inventories of items in warehouses/stores, and orders for items.
- **Human resources:** For information about employees, salaries, payroll taxes and benefits, and for generation of pay checks.

As the list illustrates, databases form an essential part of almost all enterprises today.

**Question-09: What is data abstraction? Why data abstraction is needed? Describe different levels of data abstraction. 7 Marks CSE – 2008 CSE-2003 CSE-2003 (OLD) CSE-2001 CSE-2007**

**Data Abstraction:**
For the database system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database-systems users are not computer trained, developers hide the complexity from users through a process to simplify users' interactions with the system. This process of separating complexity of data structures is called data abstraction.

**Different levels of data abstraction:**
Since many database-systems users are not computer trained, **to simplify users' interactions with the system**, developers hide the complexity from users through several levels of abstraction:

- **Physical level:** The lowest level of abstraction describes how the data are actually stored. The physical level describes complex low-level data structures in detail.

- **Logical level:** The next-higher level of abstraction describes what data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Although implementation of the simple structures at the logical level may involve complex physical-level structures, the user of the logical level does not need to be aware of this complexity. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.

- **View level:** The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

**Figure 1.1 shows the relationship among the three levels of abstraction.**

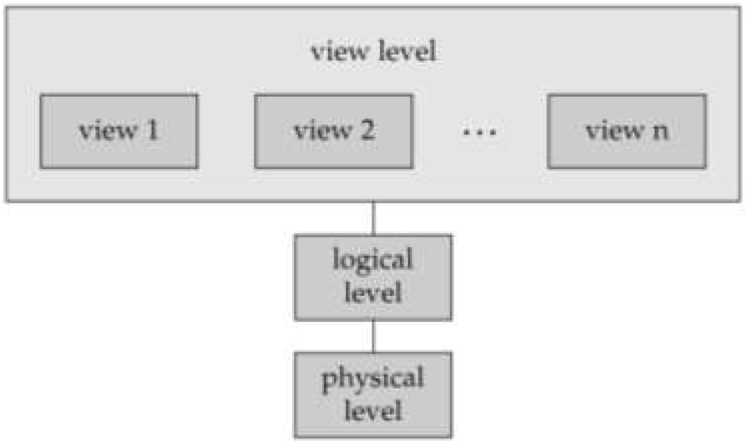


**Figure 1.1** The three levels of data abstraction.

**Question-10: What are the disadvantages of file processing system. 6 Marks CSE-2009**

**Disadvantages of File Processing Systems:**
Keeping organizational information in a file-processing system has a number of major disadvantages:
**Data redundancy and inconsistency:**
Since different programmers create the files and application programs over a long period, the various files are likely to have different structures and the programs may be written in several programming languages. Moreover, the same information may be duplicated in several places (files). This redundancy leads to higher storage and access cost. In addition, it may lead to data inconsistency; that is, the various copies of the same data may no longer agree.

**Difficulty in accessing data:**
Conventional file-processing environments do not allow needed data to be retrieved in a convenient and efficient manner. More responsive data-retrieval systems are required for general use.

**Data isolation:**
Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.

**Integrity problems:**
The data values stored in the database must satisfy certain types of consistency constraints. Developers enforce these constraints in the system by adding appropriate code in the various application programs. However, when new constraints are added, it is difficult to change the programs to enforce them. The problem is compounded when constraints involve several data items from different files.

**Atomicity problems:**
A computer system, like any other mechanical or electrical device, is subject to failure. In many applications, it is crucial that, if a failure occurs, the data be restored to the consistent state that existed prior to the failure. In balance transfer, clearly, it is essential to database consistency that either both the credit and debit occur, or that neither occur. That is, the funds transfer must be atomic-it must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file-processing system.

**Concurrent-access anomalies:**
For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. Indeed, today, the largest Internet retailers may have millions of accesses per day to their data by shoppers. In such an environment, interaction of concurrent updates is possible and may result in inconsistent data. To guard against this

possibility, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously.

__Security problems:__
Not every user of the databases system should be able to access all the data. Since application programs are added to the file-processing system in an ad hoc manner, enforcing such security constraints is difficult.

These difficulties, among others prompted the development of database systems.

**Question-11: What are the main differences between file processing system and database management systems? 7 Marks CSE-2011 CSE-2007 CSE-2004 CSE-2001**

**__Main Differences between file Processing System and Database Management Systems:__**
There are many differences between file processing system and database management system. Following are important from the subject point of view:

| Database Management Systems | File Processing System |
|---|---|
| Both systems contain a collection of data and a set of programs which access that data. A database management system coordinates both the physical and the logical access to the data, whereas a file-processing system coordinates only the physical access. | A file-processing system coordinates only the physical access. |
| A database management system reduces the amount of data duplication by ensuring that a physical piece of data is available to all programs authorized to have access to it. | data written by one program in a file-processing system may not be readable by another program. |
| A database management system is designed to allow flexible access to data (i.e., queries.) | A file-processing system is designed to allow pre-determined access to data (i.e., compiled programs). |
| A database management system is designed to coordinate multiple users accessing the same data at the same time. | A file-processing system is usually designed to allow one or more programs to access different data files at the same time. In a file-processing system, a file can be accessed by two programs concurrently only if both programs have read-only access to the file. |
| Data Redundancy is controlled in DBMS | Since in file systems, data is stored in files, the same information may be duplicated in several places (files) and can arises Data Redundancy. |
| Unauthorized access is restricted in DBMS | Unauthorized access is not restricted in file systems. |
| DBMS provide back-up and recovery. | When data is lost in file system then it cannot recover. |
| In DBMS, Data Inconsistency problem is no longer exists since any data has only one copy. | In File systems, Data redundancy may lead to data inconsistency; that is, the various copies of the same data may no longer agree. |
| A database manager (administrator) stores the relationship in form of structural tables. | A "File manager" is used to store all relationships in directories in File Systems. |
| Data in data bases are more secure compared to data in files!! | Data in file systems are more less secure compared to data in files!! |
| **Databases are basically meant to fragment data into relations store then so that conditioned retrieval is fast and easy.** | **A file is just for mass storage for future use. Here the conditioned retrieval is not a significant factor.** |
| **Database is real gift in area's where we need** | **For file operations there are very very tedious** |

| | |
|---|---|
| to have bulk data (including multimedia) and which includes conditional retrieval. | operations which involves large processing time. |
| In DBMS, Data are stored in Database, writing new application programs to retrieve the appropriate data is easy. | Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult. |
| Enforcing security constraints is easy in DBMS. | Enforcing security constraints is difficult in file systems. |

**Question-12: Mention the advantages and disadvantages of a DBMS. 5 Marks CSE-2010 CSE-2005 CSE-2003 (OLD)**

Advantages of Database Management Systems:

There are many advantages of using DBMS for handling large volume of data. Following are some important from the subject point of view:

1. **Improved data sharing:** The DBMS helps create an environment in which end users have better access to more and better-managed data. Such access makes it possible for end users to respond quickly to changes in their environment.

2. **Improved data security:** The more users access the data, the greater the risks of data security breaches. Corporations invest considerable amounts of time, effort, and money to ensure that corporate data are used properly. A DBMS provides a framework for better enforcement of data privacy and security policies.

3. **Better data integration:** Wider access to well-managed data promotes an integrated view of the organization's operations and a clearer view of the big picture. It becomes much easier to see how actions in one segment of the company affect other segments.

4. **Minimized data inconsistency:** Data inconsistency exists when different versions of the same data appear in different places.. The probability of data inconsistency is greatly reduced in a properly designed database.
5. **Improved decision making:** Better-managed data and improved data access make it possible to generate better-quality information, on which better decisions are based. The quality of the information generated depends on the quality of the underlying data. Data quality is a comprehensive approach to promoting the accuracy, validity, and timeliness of the data. While the DBMS does not guarantee data quality, it provides a framework to facilitate data quality initiatives.
6. **Increased end-user productivity:** The availability of data, combined with the tools that transform data into usable information, empowers end users to make quick, informed decisions that can make the difference between success and failure in the global economy.

7. **Data independence:** Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.

8. **Efficient data access:** A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.

9. **Data integrity and security:** If data is always accessed through the DBMS, the DBMS can enforce integrity constraints on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded. Also, the DBMS can enforce access controls that govern what data is visible to different classes of users.

10. **Data administration:** When several users share the data, centralizing the ad- ministration of data can offer significant improvements. Experienced professionals who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and for fine-tuning the storage of the data to make retrieval efficient.

11. **Concurrent access and crash recovery:** A DBMS schedules concurrent ac- cesses to the data in such a manner that users can think of the data as being accessed by only one user at a time. Further, the DBMS protects users from the effects of system failures.

12. **Reduced application development time:** Clearly, the DBMS supports many important functions that are common to many applications accessing data stored in the DBMS. This, in conjunction with the high-level interface to the data, facilitates quick development of applications. Such applications are also likely to be more robust than applications developed from scratch because many important tasks are handled by the DBMS instead of being implemented by the application.

**Disadvantages of DBMS:**
Two disadvantages associated with database systems are listed below:

- Setup of the database system requires more knowledge, money, skills, and time.
- When a computer file-based system is replaced with database system, the data stored into data file must be converted to database file. It is very difficult and costly method to convert data of data file into database.
- The complexity of the database may result in poor performance.
- In most of the organization, all data is integrated into a single database. If database is damaged due to electric failure or database is corrupted on the storage media, the valuable data may be lost forever. So, Backup is necessary which increases cost of hardware
- Maintenance of DBMS requires Training at all levels, including programming, application development, and database administration. The organization has to be paid a lot of amount for the training of staff to run the DBMS.
- To maximize the efficiency of the database system, the system must keep current. Therefore, we must perform frequent updates and apply the latest patches and security measures to all components. Because database technology advances rapidly, personnel training costs tend to be significant.
- DBMS vendors frequently upgrade their products by adding new functionality. Such new features often come bundled in new upgrade versions of the software. Some of these versions require hardware upgrades. Not only do the upgrades themselves cost money, but it also costs money to train database users and administrators to properly use and manage the new features

**Question-13: What is Data Model? Briefly describe different data models. 2+7 Marks CSE-2002**

**Data Model:**
**Data model is a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints**. A data model provides a way to describe the design of a database at the physical, Logical, and view level.

Following are some data models used in the database systems:

1. Entity – Relationship (E-R) Data Model
2. Relational Data Model
3. Object-Oriented Data Model
4. Object Relational Data Model
5. Semi structured data model

**Entity- Relationship Data Model:**

The entity-relationship (E-R) data model is based on a perception of a real world that consists of a collection of basic objects, called entities, and of relationships among these objects. **An entity is a "thing" or "object" in the real world that is distinguishable from other objects. A relationship is an association among several entities. The set of all entities of the same type and the set of all relationships of the same type are termed an entity set and relationship set, respectively**. The overall logical structure (schema) of a database can be expressed graphically by an E-R diagram, which is built up from the following components:

- **Rectangles**, which represent entity sets
- **Ellipses,** which represent attributes
- **Diamonds,** which represent relationships among entity sets
- **Lines**, which link attributes to entity sets and entity sets to relationships

Each component is labelled with the entity or relationship that it represents. In addition to entities and relationships, the E-R model represents certain constraints, such as mapping cardinalities, to which the contents of a database must conform. The entity-relationship model is widely used in database design.

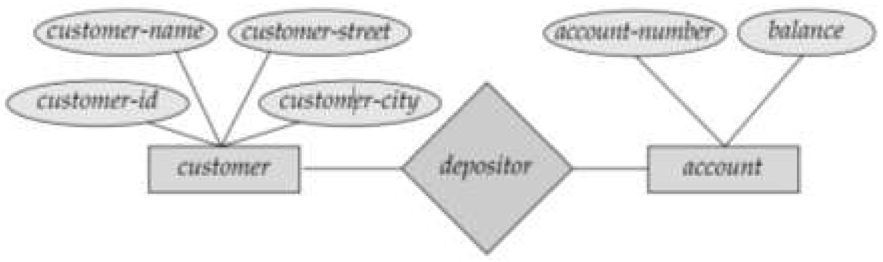


**Figure 1.2** A sample E-R diagram.

**Relational Data Model:**

The relational model is a lower-level model that uses a collection of tables to represent both data and the relationships among those data. Its conceptual simplicity has led to its widespread adoption; today a vast majority of database products are based on the relational model. Designers often formulate database schema design by first modelling data at a high level, using the E-R model, and then translating it into the relational model.

The relational model is an example of a record-based model. Record-based models are so named because the database is structured in fixed-format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record type.

**Object-Oriented Data Model:** The object-oriented data model is based on the object-oriented programming language paradigm, which is now in wide use. Inheritance, object-identity, and encapsulation (information hiding), with methods to provide an interface to objects, are among the key concepts of object-oriented programming that have found applications in data modeling. The object-oriented data model also supports a rich type system, including structured and collection types. The object-oriented model can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity.

**Object-Relational Data Model:** the object-relational data model combines features of the object-oriented data model and relational data model. The object-relational data model extends the traditional relational model with a variety of features such as structured and collection types, as well as object orientation.

**Semistructured data models:** Semistructured data models permit the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast with the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The extensible markup language (XML) is widely used to represent semistructured data.

**কিছু গুরুত্বপূর্ণ প্রশ্ন যা বিগত বছরে এসেছে সেগুলো এখানে দেওয়া হলো । এই প্রশ্নসমূহ না পড়লেও চলবে । তবে ১০০% প্রস্তুতি নিতে চাইলে নিজ দায়িত্বে পড়ে নিতে পারেন ।**

**Differentiate between Centralized and distributed database system. 2 Marks CSE-2006**

**What are the purposes of DBMS? 4 Marks CSE-2003 CSE-2000 CSE-2000(OLD)**

**What problems are caused by data redundancies? Can data redundancies be completely eliminated when the database approach is used? Why or why not? 5 Marks CSE-2001**

**Define and explain different types of operation used in DBMS. 5 Marks CSE-2000 CSE-2000(OLD)**

# Chapter 2 Entity Relationship Model

## Important Question

1. Define the following terms with example: (i) Composite Attribute (ii) Multivalued Attribute (iii) Derive Attribute. **6 Marks CSE-2011 CSE-2006 CSE-2002 CSE-2001**
2. Discuss specialization and generalization with examples. Explain how you can transform the generalization to tabular representation. **7 Marks CSE-2011 CSE-2009 CSE-2000 CSE-2000 (OLD)**
3. Discuss the conventions for displaying an E-R schema as an E-R diagram. **4 Marks CSE-2011**
4. Briefly explain E-R database design issues. **6 Marks CSE-2011 CSE-2007**
5. Define the following terms: (i) Domain (ii) Derived Attribute (iii) Aggregation. **9 Marks CSE-2010 CSE-2005**
6. **Write down the design phases to construct an E-R database schema for "CSE department of RU". Draw the corresponding E-R diagram.** (You should apply your own idea to choose entity set, attributes, keys and mapping constraints for the corresponding entity sets). **6 Marks CSE-2010 CSE-2005 CSE-2003**
7. Explain the differences between a weak and a strong entity set. **5 Marks CSE-2010 CSE-2005 CSE-2002 CSE-2000 (OLD)**
8. Define and explain E-R data model with examples. **5 Marks CSE-2009**
9. Explain Relationship set with example. **4 Marks CSE-2009**
10. What is Mapping Cardinalities? Describe the mapping cardinalities used in a binary relationship set for designing E-R diagram. **6 Marks CSE-2009 CSE-2008 CSE-2007 CSE-2004 CSE-2003**
11. Define the terms entity and attribute. What is null attribute? **3 Marks CSE-2008 CSE-2003**
12. Define Entity set and weak entity set. **3 Marks CSE-2008**
13. Describe extended E-R features with figure. **3 Marks CSE-2008**
14. Define Entity. **2 Marks CSE-2007**
15. Explain the distinctions among the terms: Primary Key, Candidate Key, Super Key with appropriate examples. **6 Marks CSE-2007 CSE-2003 CSE-2001**
16. Describe different types of attributes with examples. **3 Marks CSE-2007 CSE-2000 (OLD)**
17. What is E-R diagram? Explain it with example. Why we use it? **6 Marks CSE-2001 CSE-200O**
18. What is Primary Key? How is Primary key formed for weak entity set? **3 Marks CSE-2004**

19. What decisions should we make in designing an entity relationship database schema? **6 Marks CSE-2004 CSE-2001**
20. Design a generalization-specialization hierarchy using E-R diagram for a motor-vehicle sales company. The company sells motorcycles, car and buses. Justify your placement of attributes at each level of the hierarchy. **5 Marks CSE-2002**
21. What is Entity Relationship (E-R) diagram and why we use it? The people's bank offer five types of accounts: loan, checking, premium savings, daily interest saving and money market. It operates a number of branches and a client of the bank can have any number of accounts. Accounts can be joint. What relationships exist among these entities? Draw the corresponding E-R diagram**. 7 Marks CSE-2001**
22. What do you mean by degree of relationship and degree of relationship set? What is the degree of relationship between manufacturer and model of cars? Justify your answer. **4 Marks CSE-2001**

**Question-01: Define and explain E-R data model with examples. 5 Marks CSE-2009**

**Entity- Relationship Data Model:**

The entity-relationship (E-R) data model is based on a perception of a real world that consists of a collection of basic objects, called entities, and of relationships among these objects. **An entity is a "thing" or "object" in the real world that is distinguishable from other objects. A relationship is an association among several entities. The set of all entities of the same type and the set of all relationships of the same type are termed an entity set and relationship set, respectively**. The overall logical structure (schema) of a database can be expressed graphically by an E-R diagram, which is built up from the following components:

- **Rectangles**, which represent entity sets
- **Ellipses,** which represent attributes
- **Diamonds,** which represent relationships among entity sets
- **Lines**, which link attributes to entity sets and entity sets to relationships

Each component is labelled with the entity or relationship that it represents. In addition to entities and relationships, the E-R model represents certain constraints, such as mapping cardinalities, to which the contents of a database must conform. The entity-relationship model is widely used in database design.

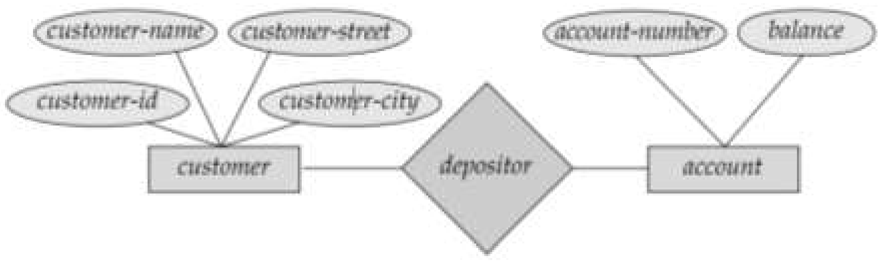


**Figure 1.2** A sample E-R diagram.

**Question-02: Define the following terms with example: (i) entity and attribute (ii) null attribute (iii) Entity set (iv) weak entity set and strong entity set (v) Composite Attribute (vi) Multivalued Attribute (vii) Derived Attribute. (viii) Domain (ix) Aggregation. 9 Marks CSE-2010 CSE-2005 CSE-2011 CSE-2008 CSE-2007 CSE-2006 CSE-2003 CSE-2002 CSE-2001**

**Entity:**

An entity is a "thing" or "object" in the real world that is distinguishable from all other objects. An entity has a set of properties, and the values for some set of properties may uniquely identify an Entity. An entity may be concrete, such as a person or a book, or it may be abstract, such as a loan, or a holiday, or a concept.

**Example: Each person** in an enterprise is an entity. A person may have a **person-id** property whose value uniquely identifies that person. Thus, the value **677-89-9011** for **person-id** would uniquely identify one particular person in the enterprise. Similarly, loans can be thought of as entities, and loan number L-15 at the Rajshahi branch of DBBL uniquely identifies a loan entity.

**Entity set:**

An entity set is a set of entities of the same type that share the same properties, or attributes. The individual entities that constitute a set are said to be the extension of the entity set. Entity sets do not need to be disjoint.

**Example:**

The set of all persons who are customers at a given bank, for example, can be defined as the entity set customer. Similarly, the entity set loan might represent the set of all loans awarded by a particular bank. Thus, all the individual bank customers are the extension of the entity set customer. It is possible to define the entity set of all employees of a bank (employee) and the entity set of all customers of the bank (customer). A person entity may be an employee entity, a customer entity, both, or neither.

**Attributes:**

Attributes are descriptive properties possessed by each member of an entity set. An entity is represented by a set of attributes. The designation of an attribute for an entity set expresses that the database stores similar information concerning each entity in the entity set; however, each entity may have its own value for each attribute. Formally, an attribute of an entity set is a function that maps from the entity set into a domain. Since an entity set may have several attributes, each entity can be described by a set of (attribute, data value) pairs, one pair for each attribute of the entity set.

**Example:**

Possible attributes of the customer entity set are customer-id, customer-name, customer-street, and customer-city. In real life, there would be further attributes, such as street number, apartment number, state, postal code, and country. Possible attributes of the loan entity set are loan-number and amount.

**Domain:**

Each entity has a **value** for each of its attributes. For each attribute, there is a set of permitted values, called the **domain**, or **value set**, of that attribute.

**Example:**

The domain of attribute customer-name might be the set of all text strings of a certain length. Similarly, the domain of attribute loan-number might be the set of all strings of the form "L-n" where n is a positive integer.

**Simple and Composite Attributes:**

**Simple attributes are those attributes of an entity that are not divided into subparts. Composite attributes are those attributes of an entity that can be divided into subparts (that is, other attributes).** Using composite attributes in a design schema is a good choice if a user will wish to refer to an entire attribute on some occasions, and to only a component of the attribute on other occasions. Composite attributes help us to group together related attributes, making the modelling cleaner. A composite attribute may appear as a hierarchy.

**For example**, an attribute **customer_id cannot be divided into subparts.** Hence customer_id is a simple attributes. **customer_name** could be structured as a composite attribute consisting of first-name, middle-initial, and last-name. The composite attribute **customer_address can be divided into subparts** with the attributes street, city, state, and zip-code. In the composite attribute customer**_address**, its component attribute **street** can be further divided into street-number, street-name, and apartment-number. Figure 2.2 depicts these examples of composite attributes for the customer entity set.

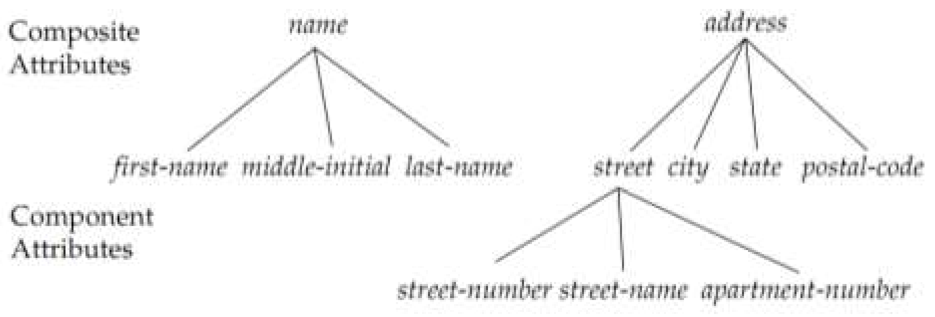

Composite Attributes
name
address
first-name middle-initial last-name
street city state postal-code
Component Attributes
street-number street-name apartment-number

**Figure 2.2** Composite attributes *customer-name* and *customer-address*.

__Single-valued and multivalued attributes__:

The attributes that have a single value for a particular entity are called **single-valued attributes**. The attributes that have a set of values for a specific entity are called **multiple-valued attributes**. Where appropriate, upper and lower bounds may be placed on the number of values in a multivalued attribute.

**Example:**

The loan-number attribute for a specific loan entity refers to only one loan number. Such attributes are said to be **single valued**. Consider an employee entity set with the attribute phone-number. An employee may have zero, one, or several phone numbers, and different employees may have different numbers of phones. This type of attribute is said to be **multivalued.** an attribute **dependent-name** of the employee entity set would be multivalued, since any particular employee may have zero, one, or more dependent(s).

**__Derived attribute:__**

Derived attributes are those attributes whose values can be derived from the values of other related attributes or entities. The value for this type of attribute can be derived from the values of other related attributes or entities. The value of a derived attribute is not stored, but is computed when required.

**Example:**

For instance, let us say that the customer entity set has an attribute **loans-held**, which represents how many loans a customer has from the bank. We can derive the value for this attribute by counting the number of loan entities associated with that customer.

As another example, suppose that the customer entity set has an attribute age, which indicates the customer's age. If the customer entity set also has an attribute date-of-birth, we can calculate age from date-of-birth and the current date. Thus, age is a derived attribute. In this case, date-of-birth may be referred to as a **base attribute**, or a stored attribute.

**__Null attribute:__**

An attribute takes a null value when an entity does not have a value for it. The null value may indicate "not applicable" — that is, that the value does not exist for the entity. **The attribute that takes a null value is called null attribute.** Null can also designate that an attribute value is unknown. An unknown value may be either missing (the value does exist, but we do not have that information) or not known (we do not know whether or not the value actually exists).

**For example**, one may have no middle name. Then **middle_name** is the null attribute for the *person* entity. A null value for the **apartment-number** attribute could mean that the address does not include an apartment number (not applicable), that an apartment number exists but we do not know what it is (missing), or that we do not know whether or not an apartment number is part of the customer's address (unknown). Then, **apartment-number** attribute is a null attribute.

**__Aggregation:__**

Aggregation is an abstraction in which relationship sets (along with their associated entity sets) are treated as higher-level entity sets, and can participate in relationships. One limitation of the E-R model is that it cannot express relationships among relationships. The best way to model a situation such as the one just described is to use aggregation.

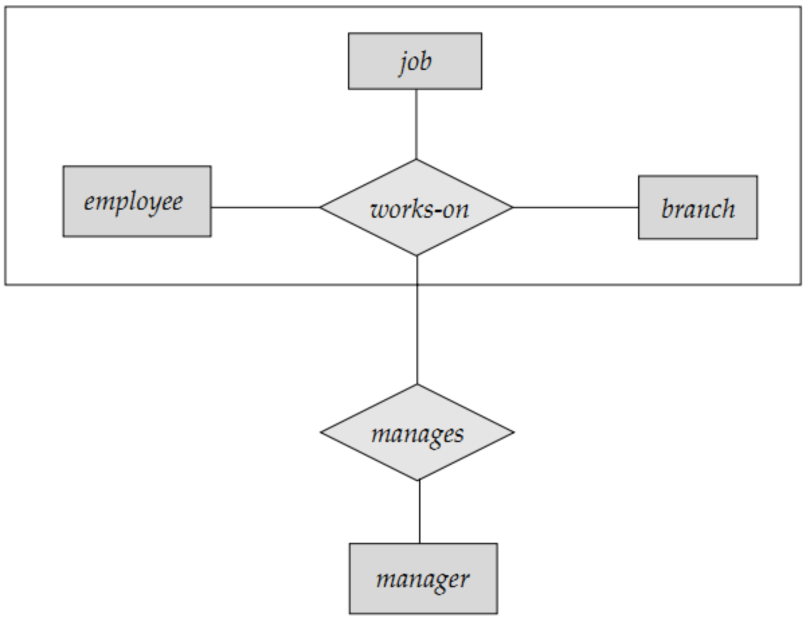


**Figure 2.19** E-R diagram with aggregation.

<u>**Example:**</u>

We regard the relationship set works-on (relating the entity sets employee, branch, and job) as a higher-level entity set called works-on. Such an entity set is treated in the same manner as is any other entity set. We can then create a binary relationship manages between works-on and manager to represent who manages what tasks.

**Question-03: Explain Relationship set with example. 4 Marks CSE-2009**

<u>**Relationship set:**</u>

A relationship is an association among several entities. A relationship set is a set of relationships of the same type. Formally, it is a mathematical relation on n ≥ 2 (possibly nondistinct) entity sets. If $E_1$, $E_2$,..., $E_n$ are entity sets, then a relationship set R is a subset of

$$\{(e_1 , e_2 ,..., e_n) \mid e_1 \in E_1 , e_2 \in E_2 ,..., e_n \in E_n\}$$

where $(e_1 , e_2 ,..., e_n)$is a relationship.

There can be more than one relationship set involving the same entity sets. Most of the relationship sets in a database system are binary that is, one that involves two entity sets.. Occasionally, however, relationship sets involve more than two entity sets. The number of entity sets that participate in a relationship set is also the degree of the relationship set. A binary relationship set is of degree 2; a ternary relationship set is of degree 3.

<u>**Example:**</u>

Consider the two entity sets **customer** and **loan** in Figure 2.3. We define the relationship set **borrower** to denote the association between customers and the bank loans that the customers have. Figure 2.3 depicts this association.

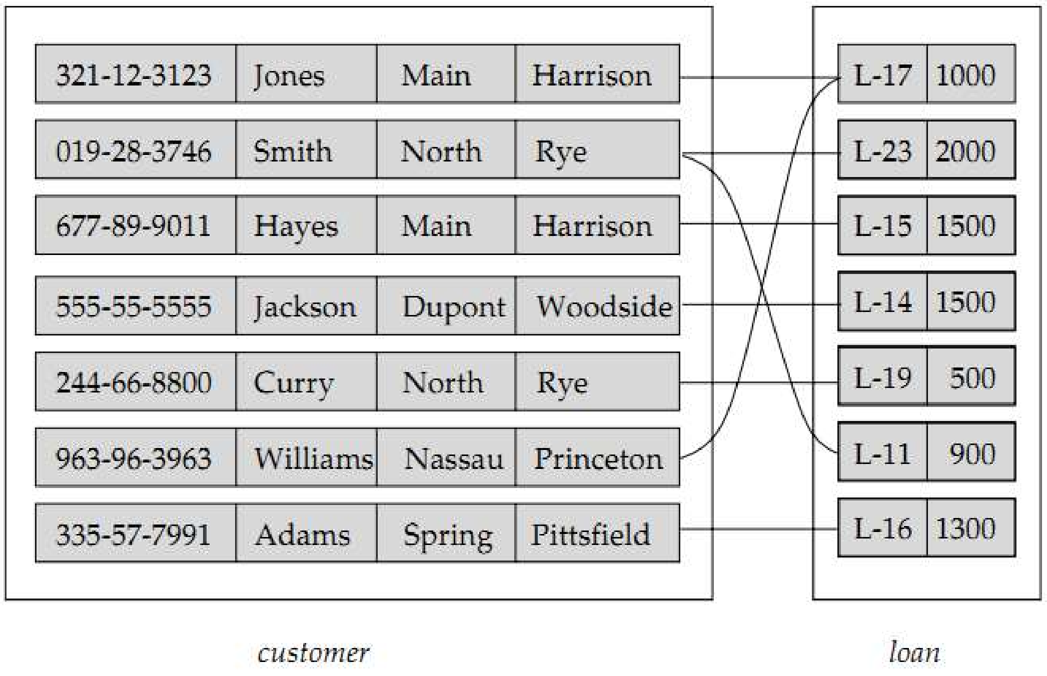

321-12-3123 Jones Main Harrison
019-28-3746 Smith North Rye
677-89-9011 Hayes Main Harrison
555-55-5555 Jackson Dupont Woodside
244-66-8800 Curry North Rye
963-96-3963 Williams Nassau Princeton
335-57-7991 Adams Spring Pittsfield
L-17 1000
L-23 2000
L-15 1500
L-14 1500
L-19 500
L-11 900
L-16 1300
customer
loan

**Figure 2.3** Relationship set *borrower*.

**Question-04: What is Mapping Cardinalities? Describe the mapping cardinalities used in a binary relationship set for designing E-R diagram. 6 Marks CSE-2009 CSE-2008 CSE-2007 CSE-2004 CSE-2003**

<u>**Mapping Cardinalities:**</u>

**Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set.** Mapping cardinalities are most useful in describing binary relationship sets, al-though they can contribute to the description of relationship sets that involve more than two entity sets.

<u>**Mapping cardinalities used in a binary relationship set:**</u>

For a binary relationship set R between entity sets A and B, the mapping cardinal-ity must be one of the following:

- **One to one:** An entity in A is associated with at most one entity in B,andan entity in B is associated with at most one entity in A. (Figure 2.4a.)
- **One to many:** An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A. (Figure 2.4b.)
- **Many to one**: An entity in A is associated with at most one entity in B.An entity in B, however, can be associated with any number (zero or more) of entities in A. (Figure 2.5a.)
- **Many to many**: An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A. (Figure 2.5b.)

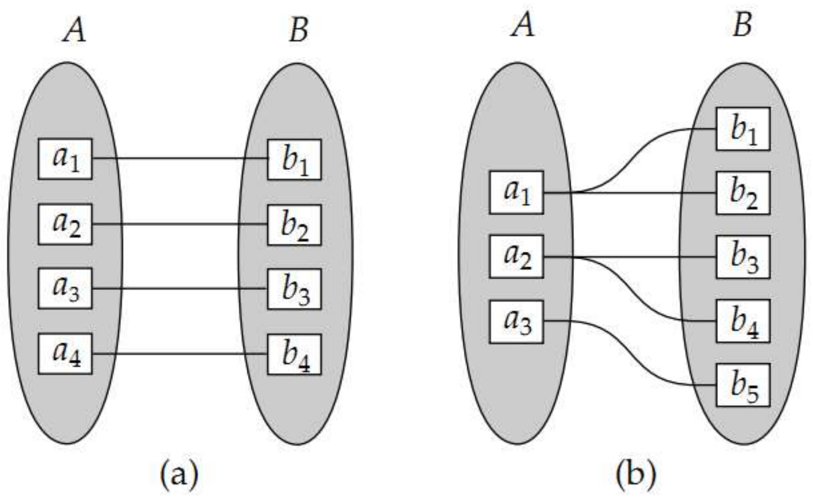


**Figure 2.4** Mapping cardinalities. (a) One to one. (b) One to many.

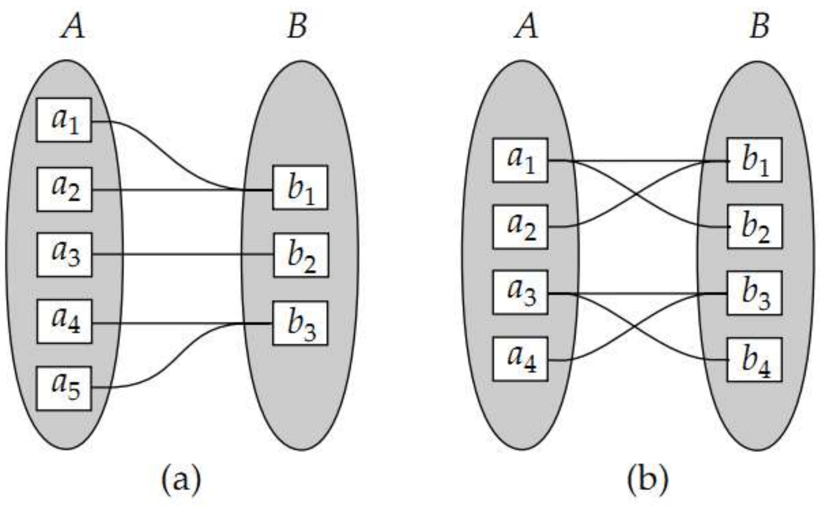


**Figure 2.5** Mapping cardinalities. (a) Many to one. (b) Many to many.

**Question-05: Explain the distinctions among the terms: Primary Key, Candidate Key, and Super Key with appropriate examples. 6 Marks CSE-2007 CSE-2003 CSE-2001**

__Super-Key:__

**A super key is a set of one or more attributes that, taken collectively, allow us to identify uniquely an entity in the entity set.** For example, the customer-id attribute of the entity set customer is sufficient to distinguish one customer entity from another. Thus, customer-id is a super key. Similarly, the combination of customer-name and customer-id is a super key for the entity set customer. The customer-name attribute of customer is not a super key, because several people might have the same name.

__Candidate Key:__

superkey may contain extraneous attributes. If K is a superkey, then so is any superset of K. We are often interested in superkeys for which no proper subset is a superkey. Such minimal superkeys are called candidate keys. It is possible that several distinct sets of attributes could serve as a **candidate key**. A candidate is a subset of a super key. A candidate key is a single field or the least combination of fields that uniquely identifies each record in the table. The least combination of fields distinguishes a candidate key from a super key. Every table must have at least one candidate key but at the same time can have several. Candidate keys must be chosen with care.

**Example:**

Suppose that a combination of customer-name and customer-street is sufficient to distinguish among members of the customer entity set. Then, both {customer-id} and {customer-name, customer-street} are candidate keys. Although the attributes customer-id and customer-name together can distinguish customer entities, their combination does not form a candidate key, since the attribute customer-id alone is a candidate key. In Bangladesh, National_ID_Card_number attribute of a person would be a candidate key.

**Primary Key:**

**Primary key** is a candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. A key (primary, candidate, and super) is a property of the entity set, rather than of the individual entities. Any two individual entities in the set are prohibited from having the same value on the key attributes at the same time. The designation of a key represents a constraint in the real-world enterprise being modelled. A primary key is a candidate key that is most appropriate to be the main reference key for the table. As its name suggests, it is the primary key of reference for the table and is used throughout the database to help establish relationships with other tables. As with any candidate key the primary key must contain unique values, must never be null and uniquely identify each record in the table.
The primary key should be chosen such that its attributes are never, or very rarely, changed.

**Example:**

For instance, the address field of a person should not be part of the primary key, since it is likely to change. Social-security numbers or **national_ID_card_Numbers**, on the other hand, are guaranteed to never change.

**Question-06: Define Entity set, weak entity set and strong entity set. 6 Marks CSE-2008**

**Entity set:**

An entity set is a set of entities of the same type that share the same properties, or attributes. The individual entities that constitute a set are said to be the extension of the entity set. Entity sets do not need to be disjoint.

**Example:**

The set of all persons who are customers at a given bank, for example, can be defined as the entity set customer. Similarly, the entity set loan might represent the set of all loans awarded by a particular bank. Thus, all the individual bank customers are the extension of the entity set customer. It is possible to define the entity set of all employees of a bank (employee) and the entity set of all customers of the bank (customer). A person entity may be an employee entity, a customer entity, both, or neither.

**Weak entity set and Strong Entity Set:**

An entity set that does not have sufficient attributes to form a primary key is termed a **weak entity set**. An entity set that has a primary key is termed a **strong entity set**.

For a weak entity set to be meaningful, it must be associated with another entity set, called the **identifying or owner entity set**. Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be **existence dependent** on the identifying entity set. The identifying entity set is said to own the weak entity set that it identifies. The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**. The identifying relationship is many to one from the weak entity set to the identifying entity set, and the participation of the weak entity set in the relationship is total.

Although a weak entity set does not have a primary key, we nevertheless need a means of distinguishing among all those entities in the weak entity set that depend on one particular strong entity. The discriminator of a weak entity set is a set of attributes that allows this distinction to be made.

The discriminator of a weak entity set is also called the **partial key** of the entity set. The primary key of a weak entity set is formed by the primary key of the identifying entity set, plus the weak entity set's discriminator. The identifying relationship set should have no descriptive attributes, since any required attributes can be associated with the weak entity set. A weak entity set can participate in relationships other than the identifying relationship. A weak entity set may participate as owner in an identifying relationship with another weak entity set. It is also possible to have a weak entity set with more than one identifying entity set. A particular weak entity would then be identified by a combination of entities, one from each identifying entity set. The primary key of the weak entity set would consist of the union of the primary keys of the identifying entity sets, plus the discriminator of the weak entity set.

In E-R diagrams, a doubly outlined box indicates a weak entity set, and a doubly outlined diamond indicates the corresponding identifying relationship. In some cases, the database designer may choose to express a weak entity set as a multivalued composite attribute of the owner entity set. A weak entity set may be more appropriately modelled as an attribute if it participates in only the identifying relationship, and if it has few attributes. Conversely, a weak-entity-set representation will more aptly model a situation where the set participates in relationships other than the identifying relationship, and where the weak entity set has several attributes.

As an illustration, consider the entity set payment, which has the three attributes: payment-number, payment-date, and payment-amount. Payment numbers are typically sequential numbers, starting from 1, generated separately for each loan. Thus, although each payment entity is distinct, payments for different loans may share the same payment number. Thus, this entity set does not have a primary key; it is a weak entity set.

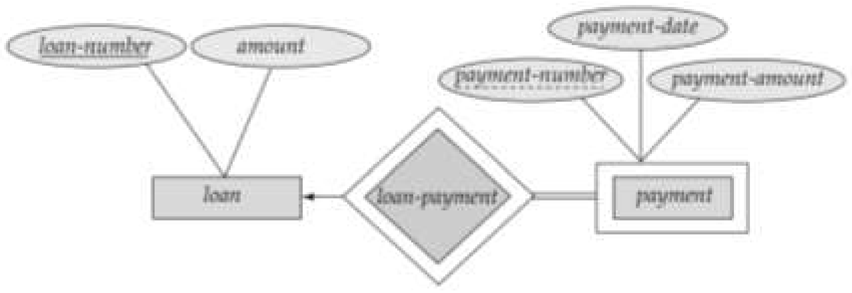


**Figure 2.16** E-R diagram with a weak entity set.

এখানে Week Entity Set সম্পর্কে বিস্তারিত দেওয়া হয়েছে এই উদ্দেশ্যে যে, টীকা আকারে আসলেও যাতে একজন শিক্ষার্থী সম্পূর্ণ উত্তর করতে পারে। এখানে ৬-৭ মার্কসের টীকা আসলে যাতে উত্তর করা যায়, সেই ভাবেই উত্তর তৈরি করা হয়েছে। ৩-৪ মার্কসের সংক্ষিপ্ত প্রশ্ন আসলে কিছু অংশ বাদ দিয়ে সংক্ষিপ্ত আকারে লেখা যেতে পারে।

**Question-07: Explain the differences between a weak and a strong entity set. 5 Marks CSE-2010 CSE-2005 CSE-2002 CSE-2000 (OLD)**

**<u>Differences between a weak and a strong entity set:</u>**
Following are some important differences between weak and strong entity set:

| Strong Entity Set | Week Entity Set |
|---|---|
| An entity set that have sufficient attributes to form a primary key is termed a **strong entity** | An entity set that does not have sufficient attributes to form a primary key is termed a |

| **set.** | **weak entity set.** |
|---|---|
| It is represented by a rectangle. | It is represented by a double rectangle. |
| It contains a primary key represented by an underline. | It contains a partial key or discriminator represented by a dashed underline. |
| The member of strong entity set is called as dominant entity set. | The primary key of week entity set is a combination of partial key and primary key of the strong entity set. |
| The relationship between two strong entity set is represented by a diamond symbol. | The relationship between one strong and a weak entity set is represented by a double diamond sign. It is known as identifying relationship. |
| The line connecting strong entity set with the relationship is single. | The line connecting weak entity set with the identifying relationship is double. |
| Total participation in the relationship may or may not exist. | Total participation in the underlying relationship always exists. |

**Question-08: Describe extended E-R features with figure. 3 Marks CSE-2008**

**<u>Extended E-R Features:</u>**
The extended E-R features are **specialization**, **generalization**, **higher and lower-level entity sets**, **attribute inheritance**, and **aggregation**.

**<u>Specialization and generalization:</u>**
An entity set may include subgroupings of entities that are distinct in some way from other entities in the set. For instance, a subset of entities within an entity set may have attributes that are not shared by all the entities in the entity set. The E-R model provides a means for representing these distinctive entity groupings. The process of designating subgroupings within an entity set is called **specialization.** Specialization is the result of taking a subset of a higher-level entity set to form a lower-level entity set.

**<u>Generalization</u>**
The refinement from an initial entity set into successive levels of entity subgroupings represents a top-down design process in which distinctions are made explicit. The design process may also proceed in a bottom-up manner, in which multiple entity sets are synthesized into a higher-level entity set on the basis of common features. **Generalization is a containment relationship that exists between a higher-level entity set and one or more lower-level entity sets.** Higher- and lower-level entity sets also may be designated by the terms superclass and subclass, respectively. For all practical purposes, generalization is a simple inversion of specialization. Generalization is the result of taking the union of two or more disjoint (lower-level) entity sets to produce a higher-level entity set

**Attribute Inheritance:**

A crucial property of the higher- and lower-level entities created by specialization and generalization is attribute inheritance. The attributes of the higher-level entity sets are said to be inherited by the lower-level entity sets. A lower-level entity set (or subclass) also inherits participation in the relationship sets in which its higher-level entity (or superclass) participates. Attribute inheritance applies through all tiers of lower-level entity sets.

**Aggregation:**

Aggregation is an abstraction in which relationship sets (along with their associated entity sets) are treated as higher-level entity sets, and can participate in relationships.

**Question-09: Discuss specialization and generalization with examples. Explain how you can transform the generalization to tabular representation. 7 Marks CSE-2011 CSE-2009 CSE-2000 CSE-2000 (OLD)**

**Specialization:**

An entity set may include subgroupings of entities that are distinct in some way from other entities in the set. For instance, a subset of entities within an entity set may have attributes that are not shared by all the entities in the entity set. The E-R model provides a means for representing these distinctive entity groupings. The process of designating subgroupings within an entity set is called **specialization.** We can apply specialization repeatedly to refine a design scheme. An entity set may be specialized by more than one distinguishing feature. When more than one specialization is formed on an entity set, a particular entity may belong to multiple specializations.

In terms of an E-R diagram, specialization is depicted by a triangle component labelled ISA, as **Figure 2.17** shows. The label ISA stands for "is a" and represents, for example, that a customer "is a" person. The ISA relationship may also be referred to as a superclass-subclass relationship. Higher- and lower-level entity sets are depicted as regular entity sets — that is, as rectangles containing the name of the entity set.

**Example:**

Consider an entity set **person**, with attributes *name*, *street*, and *city*. A person may be further classified as one of the following:

- *Customer*
- *employee*

Each of these person types is described by a set of attributes that includes all the attributes of entity set person plus possibly additional attributes. For example, customer entities may be described further by the attribute *customer-id*, whereas employee entities may be described further by the attributes *employee-id* and *salary*. The process of designating subgroupings within an entity set is called **specialization**. The specialization of person allows us to distinguish among persons according to whether they are employees or customers.

**Generalization**

The refinement from an initial entity set into successive levels of entity subgroupings represents a top-down design process in which distinctions are made explicit. The design process may also proceed in a bottom-up manner, in which multiple entity sets are synthesized into a higher-level entity set on the basis of common features.

Generalization is a containment relationship that exists between a higher-level entity set and one or more lower-level entity sets. Higher- and lower-level entity sets also may be designated by the terms superclass and subclass, respectively.

For all practical purposes, generalization is a simple inversion of specialization. We will apply both processes, in combination, in the course of designing the E-R schema for an enterprise. In terms of the E-R diagram itself, we do not distinguish between specialization and generalization. New levels of entity representation will be distinguished (specialization) or synthesized (generalization) as the design schema comes to express fully the database application and the user requirements of the database. Differences in the two approaches may be characterized by their starting point and overall goal.

Generalization proceeds from the recognition that a number of entity sets share some common features (namely, they are described by the same attributes and participate in the same relationship sets). On the basis of their commonalities, generalization synthesizes these entity sets into a single, higher-level entity set. Generalization is used to emphasize the similarities among lower-level entity sets and to hide the differences; it also permits an economy of representation in that shared attributes are not repeated.

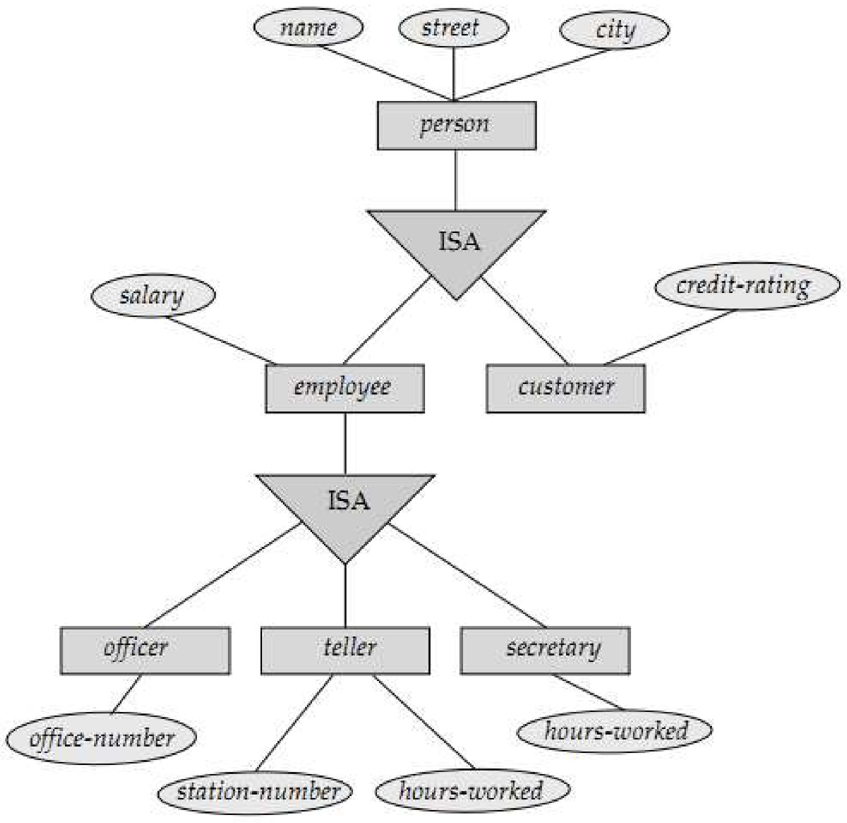


**Figure 2.17** Specialization and generalization.

<u>Tabular Representation of Generalization</u>

There are two different methods for transforming to a tabular form an E-R diagram that includes generalization.

1. **Create a table for the higher-level entity set.** For each lower-level entity set, create a table that includes a column for each of the attributes of that entity set plus a column for each attribute of the primary key of the higher-level entity set.
2. **An alternative representation is possible, if the generalization is disjoint and complete — that is, if no entity is a member of two lower-level entity sets directly below a higher-level entity set, and if every entity in the higher level entity set is also a member of one of the lower-level entity sets.** Here, we do not create a table for the higher-level entity set. Instead, for each lower-level entity set, we create a table that includes a column for each of the attributes of that entity set plus a column for each attribute of the higher-level entity set.

**Question-10: What decisions should we make in designing an entity relationship database schema? 6 Marks CSE-2004 CSE-2001**

<u>Design of an E-R Database Schema:</u>

The E-R data model gives us much flexibility in designing a database schema to model a given enterprise. In this section, we consider how a database designer may select from the wide range of alternatives. Among the designer's decisions are:

- Whether to use an attribute or an entity set to represent an object.
- Whether a real-world concept is expressed more accurately by an entity set or by a relationship set.
- Whether to use a ternary relationship or a pair of binary relationships.
- Whether to use a strong or a weak entity set; a strong entity set and its dependent weak entity sets may be regarded as a single “object” in the database, since weak entities are existence dependent on a strong entity
- Whether using generalization is appropriate; generalization, or a hierarchy of ISA relationships, contributes to modularity by allowing common attributes of similar entity sets to be represented in one place in an E-R diagram
- Whether using aggregation is appropriate; aggregation groups a part of an E-R diagram into a single entity set, allowing us to treat the aggregate entity set as a single unit without concern for the details of its internal structure.

**Question-11: Discuss the conventions for displaying an E-R schema as an E-R diagram. 4 Marks CSE-2011**

**Conventions for displaying an E-R schema as an E-R diagram:**

Entity types are shown in rectangular boxes. Relationship types are shown in diamond-shaped boxes attached to the participating entity types with straight lines. Attributes are shown in ovals, and each attribute is attached by a straight line to its entity type or relationship type. Component attributes of a composite attribute are attached to the oval representing the composite attribute. Multivalued attributes are shown in double ovals and key attributes have their names underlined. Derived attributes are shown in dotted ovals.

Weak entity types are distinguished by being placed in double rectangles and by having their identifying relationship placed in double diamonds. The partial key of the weak entity type is underlined with a dotted line.

The cardinality ratio of each binary relationship type is specified by attaching a 1, M, or N on each participating edge. The participation constraint is specified by a single line for partial participation and by double lines for total participation. We show the role names when one entity type plays both roles in a relationship.

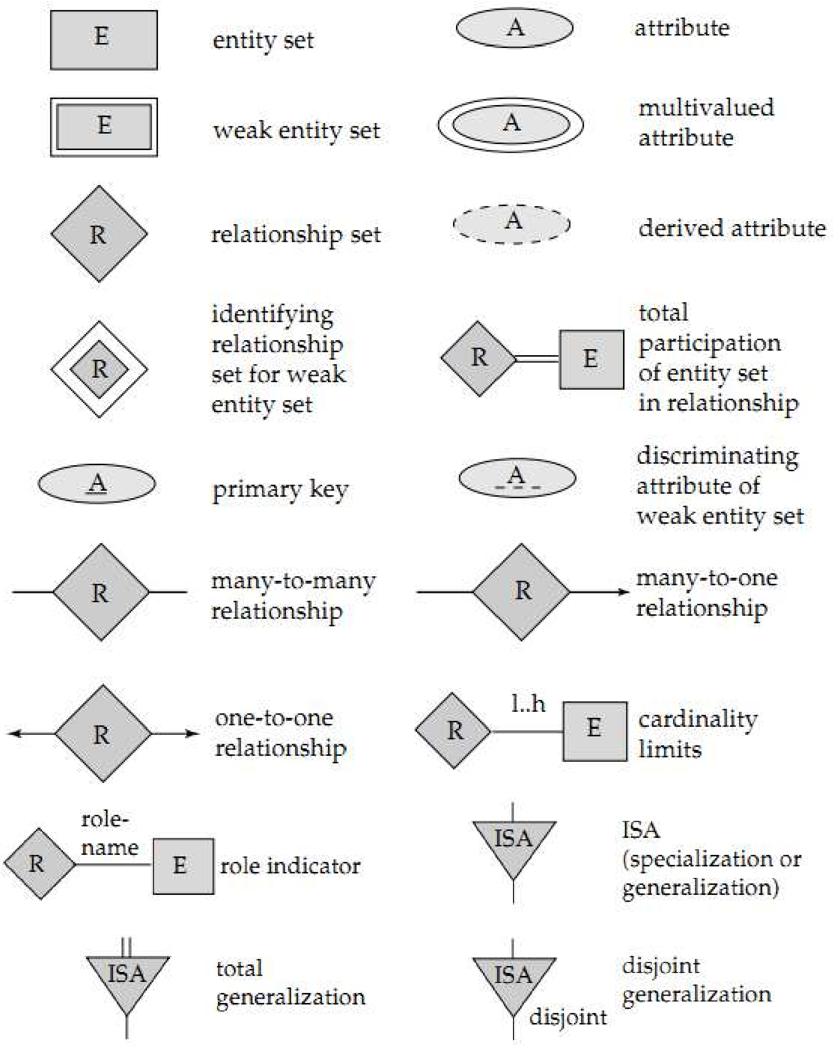


**Figure 2.20** Symbols used in the E-R notation.

**Question-12: What is Primary Key? How is Primary key formed for weak entity set? 3 Marks CSE-2004**

**Primary Key:**

**Primary key** is a candidate key that is chosen by the database designer as the principal means of identifying entities within an entity set. A primary key is a candidate key that is most appropriate to be the main reference key for the table. As its name suggests, it is the primary key of reference for the table and is used throughout the database to help establish relationships with other tables. As with any candidate key the primary key must contain unique values, must never be null and uniquely identify each record in the table. The primary key should be chosen such that its attributes are never, or very rarely, changed.

**How Primary key is formed for weak entity set:**

An entity set that does not have sufficient attributes to form a primary key is termed a **weak entity set**. For a weak entity set to be meaningful, it must be associated with another entity set, called the **identifying or owner entity set**. Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be **existence dependent** on the identifying entity set. The identifying entity set is said to own the weak entity set that it identifies. The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**. The identifying relationship is many to one from the weak entity set to the identifying entity set, and the participation of the weak entity set in the relationship is total.

Although a weak entity set does not have a primary key, we nevertheless need a means of distinguishing among all those entities in the weak entity set that depend on one particular strong entity. The discriminator of a weak entity set is a set of attributes that allows this distinction to be made.

The discriminator of a weak entity set is also called the **partial key** of the entity set. **The primary key of a weak entity set is formed by the primary key of the identifying entity set, plus the weak entity set's discriminator.** The identifying relationship set should have no descriptive attributes, since any required attributes can be associated with the weak entity set. A weak entity set can participate in relationships other than the identifying relationship. A weak entity set may participate as owner in an identifying relationship with another weak entity set. It is also possible to have a weak entity set with more than one identifying entity set. A particular weak entity would then be identified by a combination of entities, one from each identifying entity set. The primary key of the weak entity set would consist of the union of the primary keys of the identifying entity sets, plus the discriminator of the weak entity set.

**Question-13: What is E-R diagram? Explain it with example. Why we use it? 6 Marks CSE-2001 CSE-200O**

**E-R Diagram:**

**An E-R diagram is a diagram that can express the overall logical structure of a database graphically. E-R diagram is a graphical representation of entities and their relationships to each other, typically used in computing in regard to the organization of data within databases or information systems.** E-R diagrams are simple and clear — qualities that may well account in large part for the widespread use of the E-R model. Such a diagram consists of the following major components:

- **Rectangles,** which represent entity sets
- **Ellipses**, which represent attributes
- **Diamonds**, which represent relationship sets
- **Lines,** which link attributes to entity sets and entity sets to relationship sets
- **Double ellipses**, which represent multivalued attributes
- **Dashed ellipses**, which denote derived attributes
- **Double lines**, which indicate total participation of an entity in a relation-ship set
- **Double rectangles**, which represent weak entity sets

**Example:**

Consider the entity-relationship diagram in Figure 2.8, which consists of two entity sets, customer and loan, related through a binary relationship set borrower. The attributes associated with customer are customer-id, customer-name, customer-street, and customer-city. The attributes associated with loan are loan-number and amount.

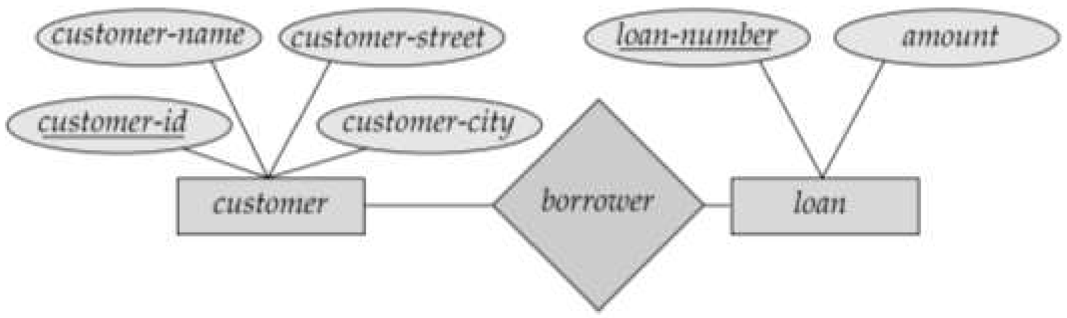


**Figure 2.8** E-R diagram corresponding to customers and loans.

**<u>Why we use it:</u>**

Entity Relationship (E-R) diagrams are drawn when designing a database system. After the systems specification, an E-R diagram is drawn showing the conceptual design of the database, this diagram shows the type of information that is to be stored in the system and how these information associate with each other (e.g. one-to-one, one-to-many, etc). E-R Diagrams are simple and easy to understand, thus, this diagram can be shown and explained to user representatives for confirmation and approval.

**Question-14: What do you mean by degree of relationship and degree of relationship set? What is the degree of relationship between manufacturer and model of cars? Justify your answer. 4 Marks CSE-2001**

**<u>Degree of Relationship:</u>**

**Degree of relationship refers to the number of participating entities in a relationship.** If there are two entities involved in relationship then it is referred to as binary relationship. If there are three entities involved then it is called as ternary relationship and so on. The degree of a relationship is the number of types among which the relationship exists. The most common degree of relationship is *binary*, which relates two types. In a unary relationship one instance of a type is related to another instance of the same type, such as the manager relationship between an employee and another employee. A *ternary* relationship relates three types and an *n-ary* relationship relates any number (n) of types. Ternary and n-ary relationships are mainly theoretical. The EDM supports unary and binary relationships.

**<u>Degree of Relationship set:</u>**

The association between entity sets is referred to as **participation**; that is, the entity sets E1 ,E2 ,...,En participate in relationship set R. **The number of entity sets that participate in a relationship set is also the degree of the relationship set.** A binary relationship set is of degree 2; a ternary relationship set is of degree 3.

**Question-15: Write down the design phases to construct an E-R database schema for "CSE department of RU". Draw the corresponding E-R diagram.** (You should apply your own idea to choose entity set, attributes, keys and mapping constraints for the corresponding entity sets). **6 Marks CSE-2010 CSE-2005 CSE-2003**

**<u>Design phases to construct an E-R database schema for "CSE department of RU":</u>**
A high-level data model serves the database designer by providing a conceptual framework in which to specify, in a systematic fashion, what the data requirements of the database users are, and how the database will be structured to fulfill these requirements.

The initial phase of database design is to characterize fully the data needs of the prospective database users. The database designer needs to interact extensively with domain experts and users to carry out this task. The outcome of this phase is a specification of user requirements.

**Next, the designer chooses a data model, and by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database.** The schema developed at this conceptual-design phase provides a detailed overview of the enterprise. Stated in terms of the E-R model, the schema specifies all **entity sets, relationship sets, attributes, and mapping constraints.** The designer reviews the schema to confirm that all data requirements are indeed satisfied and are not in conflict with one another. She can also examine the design to remove any redundant features. **Her focus at this point is describing the data and their relationships, rather than on specifying physical storage details.**

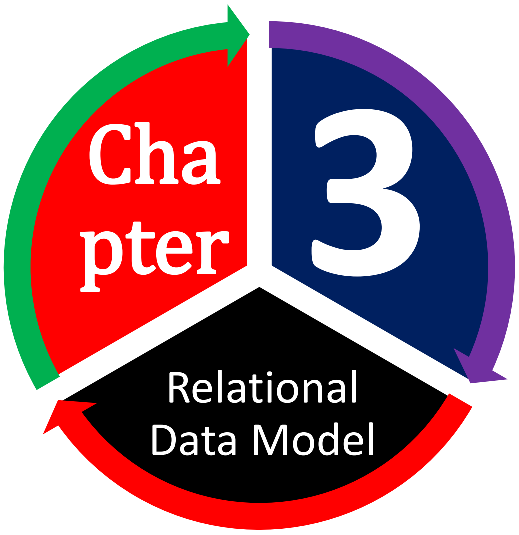
A fully developed conceptual schema will also indicate the functional requirements of the enterprise. In a specification of functional requirements, users describe the kinds of operations (or transactions) (such as modifying or updating data, searching for and retrieving specific data, and deleting data) that will be performed on the data. **At this stage of conceptual design, the designer can review the schema to ensure it meets functional requirements.**
The process of moving from an abstract data model to the implementation of the database proceeds in two final design phases. **In the logical-design phase**, the designer maps the high-level conceptual schema onto the implementation data model of the database system that will be used. The designer uses the resulting system-specific database schema in the subsequent physical-design phase, in which the physical features of the database are specified.

**<u>E-R diagram for "CSE department of RU":</u>**

**ডিজাইন ফেজ হতে পারে নিচের পয়েন্ট আকারেঃ**

1. **Data Requirements**
2. **Entity sets designation**
3. **Relationship sets designation.**
4. **E-R diagram.**

৪র্থ ধাপের ডায়াগ্রাম নিজে তৈরি করু



# Important Question

1. Discuss the following operations in terms of relational algebra: (i) Project **CSE-2011 CSE-2009 CSE-2003 CSE-2002** (ii) Set Difference **CSE-2011 CSE-2002** (iii) Cartesian Product **CSE-2011 CSE-2009 CSE-2003 CSE-2002** (iv) Outer-Join. **CSE-2011 CSE-2009** (V) Select Operation **CSE-2010 CSE-2008** (vi) Union Operation **CSE-2010 CSE-2008** (vii) Rename Operation **CSE-2010 CSE-2008** (viii) Natural Join Operation. **CSE-2010 CSE-2009 CSE-2008 CSE-2003 CSE-2002** (ix) Set intersection. **CSE-2003**
2. Explain relational data model with example. **4 Marks CSE-2009**
3. Define schema diagram. Differentiate between Cartesian product and Natural Join with suitable example. **2+3 Marks CSE-2006**
4. Which operations are used to modify the database? Describe them with examples. **6 Marks CSE-2006**
5. What do you mean by relational algebra? What are the fundamental operations in relational algebra? Briefly explain. **7 Marks CSE-2005 CSE-2003 CSE-2003 (OLD)**
6. List two reasons why null values might be introduced into the database. **2 Marks CSE-2005**
7. What are left outer join, right outer join and full outer join? Explain with example. **5 Marks CSE-2004**
8. Define Domain, Tuple Variable, Query Languages and foreign key. **4 Marks CSE-2007**
9. What is a View? Why do we define a View? **1+2 Marks CSE-2007 CSE-2003 (OLD)**
10. Why do we use a number of relations that are connected in some ways rather than just a relation that contains all the attributes? **2 Marks CSE-2007**
11. What are the reasons for the popularity of relational data model? **3 Marks CSE-2006**

**Question: Define Domain, Tuple Variable, Query Languages and foreign key. 4 Marks CSE-2007**

**Domain:**
**Attributes:**
Attributes are descriptive properties possessed by a relation in a relational data model.

**Example:**
Possible attributes of the account relation are account-number, branch-name, and balance.

**Domain:**
In any relation, there is a **value** for each of its attributes. For each attribute, there is a set of permitted values, called the **domain**, or **value set**, of that attribute.

**Example:**
The domain of attribute customer-name might be the set of all text strings of a certain length. Similarly, the domain of attribute loan-number might be the set of all strings of the form "L-n" where n is a positive integer.

**Tuple Variable:**
Because tables are essentially relations, we shall use the mathematical terms relation and tuple in place of the terms table and row. **A tuple variable is a variable that stands for a tuple; in other words, a tuple variable is a variable whose domain is the set of all tuples.**

In the account relation of Figure 3.1, there are seven tuples. Let the tuple variable t refer to the first tuple of the relation. We use the notation t[account-number] to denote the value of t on the account-number attribute. Thus, t[account-number] = "A-101," and t[branch-name] = "Downtown".

**Query Languages:**
**A query language is a language in which a user requests information from the database.** These languages are usually on a level higher than that of a standard programming language. Query languages can be categorized as either procedural or non-procedural. In a **procedural language**, the user instructs the system to perform a sequence of operations on the database to compute the desired result. In a **nonprocedural language**, the user describes the desired information without giving a specific procedure for obtaining that information. Most commercial relational-database systems offer a query language that includes elements of both the procedural and the nonprocedural approaches.

The relational algebra is procedural, whereas the tuple relational calculus and domain relational calculus are nonprocedural. These query languages are terse and formal, lacking the "syntactic sugar" of commercial languages, but they illustrate the fundamental techniques for extracting data from the database.

**Example:**
1. SQL
2. QBE and
3. Datalog

**Foreign key:**
A relation schema, say $r_1$, derived from an E-R schema may include among its attributes the primary key of another relation schema, say $r_2$. This attribute is called a **foreign key** from $r_1$, referencing $r_2$. The relation $r_1$ is also called the **referencing relation** of the foreign key dependency, and $r_2$ is called the **referenced relation** of the foreign key.

**For example**, the attribute *branch-name* in *Account-schema* is a foreign key from *Account-schema* referencing *Branch-schema*, since *branch-name* is the primary key of *Branch-schema.*

**Question: Define schema diagram. Differentiate between Cartesian product and Natural Join with suitable example. 2+3 Marks CSE-2006**

<u>Schema Diagram:</u>

**Schema diagrams are diagrams by which a database schema, along with primary key and foreign key dependencies, can be depicted pictorially**. Figure 3.9 shows the schema diagram for a banking enterprise. Each **relation** appears as a **box**, with the attributes listed in-side it and the relation name above it. If there are **primary key attributes**, a **horizontal line crosses the box**, with the primary key attributes listed above the line. **Foreign key dependencies** appear as **arrows from the foreign key attributes of the referencing relation** to the primary key of the referenced relation. There is a difference between a schema diagrams with an E-R diagram. In particular, **E-R diagrams do not show foreign key attributes explicitly, whereas schema diagrams show them explicitly.** Many database systems provide design tools with a graphical user interface for creating schema diagrams.

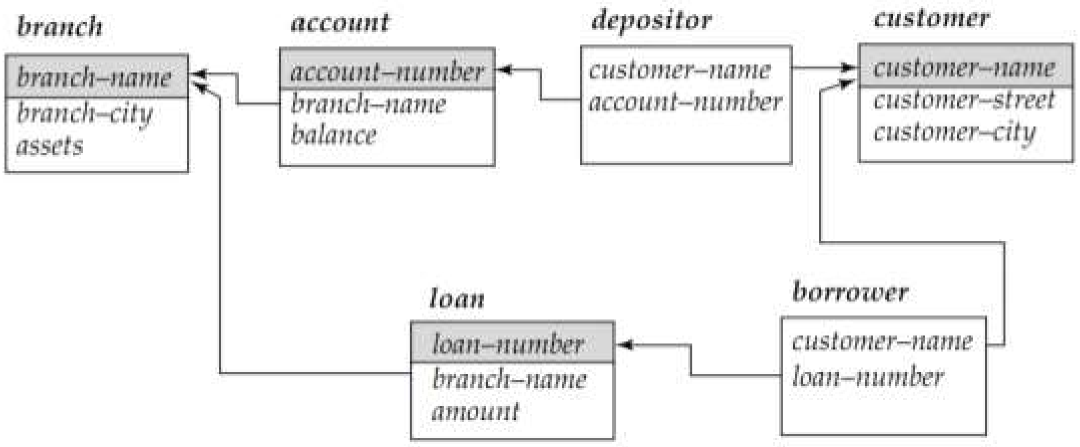


**Figure 3.9** Schema diagram for the banking enterprise.

<u>**Difference between Cartesian product and Natural Join:**</u>

**Question: What do you mean by relational algebra? What are the fundamental operations and extended fundamental operations in relational algebra? Briefly explain. 7 Marks CSE-2005 CSE-2003 CSE-2003 (OLD)**

**Relational Algebra:**

**The relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result.** The fundamental operations in the relational algebra are **select**, **project**, **union, set difference**, **Cartesian product**, and **rename**. In addition to the fundamental operations, there are several other operations — namely, **set intersection**, **natural join**, **division**, and **assignment**.

**Fundamental operations in relational algebra:**

The **select, project, and rename operations** are called **unary** operations, because they operate on one relation. The **other three operations** operate on pairs of relations and are, therefore, called **binary** operations.

**Select Operations:**

**The select operation selects tuples that satisfy a given predicate.** We use the lowercase Greek letter sigma (σ) to denote selection. The predicate appears as a subscript to σ. The argument relation is in parentheses after the σ. Thus, to select those tuples of the loan relation where the branch is "Rajshahi," we write

$$\sigma_{\text{branch}-\text{name}="Rajshahi"}(loan)$$

In general, we allow comparisons using $=, \neq, <, \leq, >, \geq$ in the selection predicate. Furthermore, we can combine several predicates into a larger predicate by using the connectives and ($\wedge$), or ($\vee$), and not ($\neg$). The selection predicate may include comparisons between two attributes.

**Example:**

1. We can find all tuples in which the amount lent is more than $1200 by writing

$$\sigma_{\text{amount}} > 1200 \text{ (loan)}$$

2. To find those tuples pertaining to loans of more than $1200 made by the Rajshahi branch, we write

$$\sigma_{\text{branch}-\text{name} = \text{“Rajshahi”} \wedge \text{amount} > 1200} \text{ (loan)}$$

3. Consider the relation loan-officer that consists of three attributes: customer-name, banker-name, and loan-number, which specifies that a particular banker is the loan officer for a loan that belongs to some customer. To find all customers who have the same name as their loan officer, we can write

$$\sigma_{\text{customer}-\text{name} = \text{banker}-\text{name}} \text{ (loan-officer)}$$

**The Project Operation:**

The project operation allows us to produce this relation. The project operation is a unary operation that returns its argument relation, with certain attributes left out. Since a relation is a set, any duplicate rows are eliminated. Projection is denoted by the uppercase Greek letter pi (Π). We list those attributes that we wish to appear in the result as a subscript to Π. The argument relation follows in parentheses. Suppose we want to list all loan numbers and the amount of the loans, but do not care about the branch name. Then we can write the query as

$$\Pi_{\text{loan}-\text{number},\ \text{amount}} \text{ (loan)}$$

Figure 3.11 shows the relation that results from this query. (If the loan relation is as shown in Figure 3.6)

| *loan-number* | *branch-name* | *amount* |
|---|---|---|
| L-11 | Round Hill | 900 |
| L-14 | Downtown | 1500 |
| L-15 | Perryridge | 1500 |
| L-16 | Perryridge | 1300 |
| L-17 | Downtown | 1000 |
| L-23 | Redwood | 2000 |
| L-93 | Mianus | 500 |

**Figure 3.6** The *loan* relation.

| *loan-number* | *amount* |
|---|---|
| L-11 | 900 |
| L-14 | 1500 |
| L-15 | 1500 |
| L-16 | 1300 |
| L-17 | 1000 |
| L-23 | 2000 |
| L-93 | 500 |

**Figure 3.11** Loan number and the amount of the loan.

**Union Operation:**

**Union operation is a binary operation that take two relations as input and produce a new relation as their result. Union, is denoted, as in set theory, by ∪. We must ensure that unions are taken between compatible relations.**

**Explanation with Example:**

Consider a query to find the names of all bank customers who have either an account or a loan or both. Note that the customer relation does not contain the information, since a customer does not need to have either an account or a loan at the bank. To answer this query, we need the information in the depositor relation (Figure 3.5) and in the borrower relation (Figure 3.7). We know how to find the names of all customers with a loan in the bank:

$$\Pi_{\text{customer}-\text{name}}\ (\text{borrower})$$

We also know how to find the names of all customers with an account in the bank:

$$\Pi_{\text{customer}-\text{name}}\ (\text{depositor})$$

To answer the query, we need the union of these two sets; that is, we need all customer names that appear in either or both of the two relations. So the expression needed is

$$\Pi_{\text{customer}-\text{name}}\ (\text{borrower}) \cup \Pi_{\text{customer}-\text{name}}\ (\text{depositor})$$

The result relation for this query appears in Figure 3.12.

| *customer-name* |
|---|
| Adams |
| Curry |
| Hayes |
| Jackson |
| Jones |
| Smith |
| Williams |
| Lindsay |
| Johnson |
| Turner |

**Figure 3.12** Names of all customers who have either a loan or an account.

**The Cartesian-Product Operation:**

The Cartesian-product operation, denoted by a cross (×), allows us to combine in-formation from any two relations. We write the Cartesian product of relations $r_1$ and $r_2$ as $r_1 \times r_2$ . A relation is by definition a subset of a Cartesian product of a set of domains.

**<u>Explanation with Examples:</u>**
Suppose that we want to find the names of all customers who have a loan at the Perryridge branch. We need the information in both the loan relation and the borrower relation to do so. If we write

$$\sigma_{\text{branch}-\text{name} = \text{"Perryridge"}}(\text{borrower} \times \text{loan})$$

Then the result is the relation in **Figure 3.15.**

| *customer-name* | *borrower. loan-number* | *loan. loan-number* | *branch-name* | *amount* |
|---|---|---|---|---|
| Adams | L-16 | L-15 | Perryridge | 1500 |
| Adams | L-16 | L-16 | Perryridge | 1300 |
| Curry | L-93 | L-15 | Perryridge | 1500 |
| Curry | L-93 | L-16 | Perryridge | 1300 |
| Hayes | L-15 | L-15 | Perryridge | 1500 |
| Hayes | L-15 | L-16 | Perryridge | 1300 |
| Jackson | L-14 | L-15 | Perryridge | 1500 |
| Jackson | L-14 | L-16 | Perryridge | 1300 |
| Jones | L-17 | L-15 | Perryridge | 1500 |
| Jones | L-17 | L-16 | Perryridge | 1300 |
| Smith | L-11 | L-15 | Perryridge | 1500 |
| Smith | L-11 | L-16 | Perryridge | 1300 |
| Smith | L-23 | L-15 | Perryridge | 1500 |
| Smith | L-23 | L-16 | Perryridge | 1300 |
| Williams | L-17 | L-15 | Perryridge | 1500 |
| Williams | L-17 | L-16 | Perryridge | 1300 |

**Figure 3.15** Result of $\sigma_{branch\text{-}name = \text{"Perryridge"}}(borrower \times loan)$.

If we write

$$\sigma_{borrower.loan\text{-}number = loan.loan\text{-}number}$$
$$(\sigma_{branch\text{-}name = \text{"Perryridge"}}(borrower \times loan))$$

We get only those tuples of borrower × loan that pertain to customers who have a loan at the Perryridge branch. Finally, since we want only customer-name, we do a projection:

$$\Pi_{customer\text{-}name}\ (\sigma_{borrower.loan\text{-}number = loan.loan\text{-}number}$$
$$(\sigma_{branch\text{-}name = \text{"Perryridge"}}\ (borrower \times loan)))$$

The result of this expression, shown in Figure 3.16, is the correct answer to our query.

| *customer-name* |
|---|
| Adams |
| Hayes |

**Figure 3.16** Result of $\Pi_{customer\text{-}name}$
$(\sigma_{borrower.loan\text{-}number = loan.loan\text{-}number}$
$(\sigma_{branch\text{-}name = \text{"Perryridge"}}\ (borrower \times loan)))$.

**The Rename Operation:**

Unlike relations in the database, the results of relational-algebra expressions do not have a name that we can use to refer to them. It is useful to be able to give them names; the rename operator, denoted by the lowercase Greek letter rho (ρ), lets us do this. Given a relational-algebra expression E, the expression

$$\rho_x\,(E)$$

returns the result of expression E under the name x. A relation r by itself is considered a (trivial) relational-algebra expression. Thus, we can also apply the rename operation to a relation r to get the same relation under a new name.

A second form of the rename operation is as follows. Assume that a relational-algebra expression E has arity n. Then, the expression

$$\rho_{x(A1, A2, \ldots, An)}\,(E)$$

returns the result of expression E under the name x, and with the attributes renamed to A1 ,A2 ,...,An

**Example:**

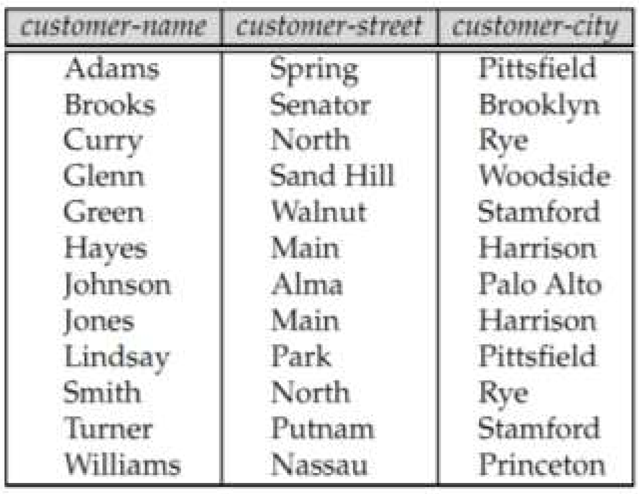
consider the query “Find the names of all customers who live on the same street and in the same city as Smith.” We can obtain Smith’s street and city by writing

$$\Pi_{\text{customer}-\text{street}, \text{customer}-\text{city}}\ (\sigma_{\text{customer}-\text{name} = \text{“Smith”}}\ (\text{customer}))$$

However, in order to find other customers with this street and city, we must reference the customer relation a second time. In the following query, we use the rename operation on the preceding expression to give its result the name smith-addr, and to rename its attributes to street and city, instead of customer-street and customer-city:

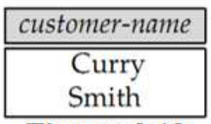
$$\begin{aligned}&\Pi_{customer.customer\text{-}name}\\ &\quad(\sigma_{customer.customer\text{-}street = smith\text{-}addr.street \wedge customer.customer\text{-}city = smith\text{-}addr.city}\\ &\qquad(customer \times \rho_{smith\text{-}addr(street, city)}\\ &\qquad\quad(\Pi_{customer\text{-}street,\ customer\text{-}city}\ (\sigma_{customer\text{-}name\ =\ \text{“Smith”}}(customer)))))\end{aligned}$$

The result of this query, when we apply it to the customer relation of Figure 3.4, appears in Figure 3.19.



| customer-name | customer-street | customer-city |
|---|---|---|
| Adams | Spring | Pittsfield |
| Brooks | Senator | Brooklyn |
| Curry | North | Rye |
| Glenn | Sand Hill | Woodside |
| Green | Walnut | Stamford |
| Hayes | Main | Harrison |
| Johnson | Alma | Palo Alto |
| Jones | Main | Harrison |
| Lindsay | Park | Pittsfield |
| Smith | North | Rye |
| Turner | Putnam | Stamford |
| Williams | Nassau | Princeton |

**Figure 3.4** The *customer* relation.



| customer-name |
|---|
| Curry |
| Smith |

**Figure: 3.19**

__The Set-Intersection Operation:__

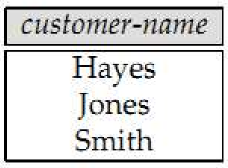
The first additional-relational algebra operation is **set intersection (∩)**. Suppose that we wish to find all customers who have both a loan and an account. Using set intersection, we can write

$$\Pi_{\text{customer}-\text{name}}\ (\text{borrower}) \cap \Pi_{\text{customer}-\text{name}}\ (\text{depositor})$$

The result relation for this query appears in Figure 3.20. we can rewrite any relational algebra expression that uses set intersection by replacing the intersection operation with a pair of set-difference operations as:

$$r \cap s = r - (r - s)$$

Thus, set intersection is not a fundamental operation and does not add any power to the relational algebra. It is simply more convenient to write r ∩ s than to write r − (r − s).



| *customer-name* |
|---|
| Hayes |
| Jones |
| Smith |

**Figure 3.20** Customers with both an account and a loan at the bank.

__Natural Join Operation:__

**The natural join is a binary operation that allows us to combine certain selections and a Cartesian product into one operation.** It is denoted by the "join" symbol ⋈. The natural-join operation forms a Cartesian product of its two arguments, performs a selection forcing equality on those attributes that appear in both relation schemas, and finally removes duplicate attributes. Although the definition of natural join is complicated, the operation is easy to apply.

Consider two relations r(R) and s(S).The natural join of r and s, denoted by r ⋈ s, is a relation on schema R ∪ S formally defined as follows:

$$r \bowtie s = \Pi_{R \cup S}\ (\sigma_{r.A_1 = s.A_1 \wedge r.A_2 = s.A_2 \wedge \ldots \wedge r.A_n = s.A_n}\ r \times s)$$

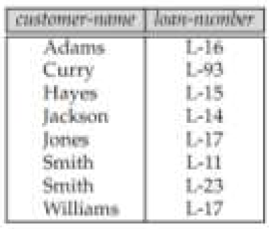
where $R \cap S = \{A_1,\ A_2, \ldots, A_n\}$.

Example:

Consider again the example **"Find the names of all customers who have a loan at the bank, and find the amount of the loan."** We express this query by using the natural join as follows:

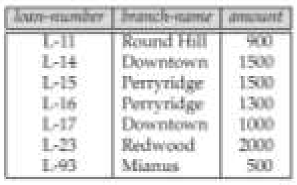
$$\Pi_{\textit{customer-name},\ \textit{loan-number},\ \textit{amount}}\ (\textit{borrower} \bowtie\ \textit{loan})$$

Since the schemas for **borrower** and **loan** (that is, Borrower-schema and Loan-schema) have the attribute loan-number in common, the natural-join operation considers only pairs of tuples that have the same value on loan-number. It combines each such pair of tuples into a single tuple on the union of the two schemas (that is, customer-name, branch-name, loan-number, amount). After performing the projection, we obtain the relation in **Figure 3.21.**



| *customer-name* | *loan-number* |
|---|---|
| Adams | L-16 |
| Curry | L-93 |
| Hayes | L-15 |
| Jackson | L-14 |
| Jones | L-17 |
| Smith | L-11 |
| Smith | L-23 |
| Williams | L-17 |

**Figure 3.7** The *borrower* relation.



| *loan-number* | *branch-name* | *amount* |
|---|---|---|
| L-11 | Round Hill | 900 |
| L-14 | Downtown | 1500 |
| L-15 | Perryridge | 1500 |
| L-16 | Perryridge | 1300 |
| L-17 | Downtown | 1000 |
| L-23 | Redwood | 2000 |
| L-93 | Mianus | 500 |

**Figure 3.6** The *loan* relation.

*Loan-schema* = (*loan-number*, *branch-name*, *amount*)
*Borrower-schema* = (*customer-name*, *loan-number*)

| *customer-name* | *loan-number* | *amount* |
|---|---|---|
| Adams | L-16 | 1300 |
| Curry | L-93 | 500 |
| Hayes | L-15 | 1500 |
| Jackson | L-14 | 1500 |
| Jones | L-17 | 1000 |
| Smith | L-23 | 2000 |
| Smith | L-11 | 900 |
| Williams | L-17 | 1000 |

**Figure 3.21** Result of $\Pi_{customer\text{-}name,\ loan\text{-}number,\ amount}\ (borrower \bowtie\ loan)$.

**The Division Operation**

The division operation, denoted by ÷, is suited to queries that include the phrase "for all." Suppose that we wish to find all customers who have an account at all the branches located in Brooklyn. We can obtain all branches in Brooklyn by the expression

$$r_1 = \Pi_{\text{branch}-\text{name}}\ (\sigma_{\text{branch}-\text{city} = \text{“Brooklyn”}}\ (\text{branch}))$$

The result relation for this expression appears in Figure 3.23.

| *branch-name* |
|---|
| Brighton |
| Downtown |

**Figure 3.23** Result of $\Pi_{branch\text{-}name}(\sigma_{branch\text{-}city\,=\,\text{“Brooklyn”}}\ (branch)$.

We can find all (*customer-name, branch-name*) pairs for which the customer has an account at a branch by writing

$$r_2 = \Pi_{customer\text{-}name,\ branch\text{-}name}\ (depositor \bowtie\ account)$$

Figure 3.24 shows the result relation for this expression.

| *customer-name* | *branch-name* |
|---|---|
| Hayes | Perryridge |
| Johnson | Downtown |
| Johnson | Brighton |
| Jones | Brighton |
| Lindsay | Redwood |
| Smith | Mianus |
| Turner | Round Hill |

**Figure 3.24** Result of $\Pi_{customer\text{-}name,\ branch\text{-}name}\ (depositor \bowtie\ account)$.

**The Assignment Operation:**

It is convenient at times to write a relational-algebra expression by assigning parts of it to temporary relation variables. The assignment operation, denoted by ←, works like assignment in a programming language. We could write r ÷ s as

$$\begin{aligned}
&temp1 \leftarrow \Pi_{R-S}\ (r)\\
&temp2 \leftarrow \Pi_{R-S}\ ((temp1\ \times\ s)\ -\ \Pi_{R-S,S}(r))\\
&result\ =\ temp1\ -\ temp2
\end{aligned}$$

The evaluation of an assignment does not result in any relation being displayed to the user. Rather, the result of the expression to the right of the ← is assigned to the relation variable on the left of the ←. This relation variable may be used in subsequent expressions.

With the assignment operation, a query can be written as a sequential program consisting of a series of assignments followed by an expression whose value is displayed as the result of the query. For relational-algebra queries, assignment must always be made to a temporary relation variable.

Assignments to permanent relations constitute a database modification. The assignment operation does not provide any additional power to the algebra. It is, however, a convenient way to express complex queries.

**Question: Discuss the following operation in terms of relational algebra: Outer-Join. CSE-2011 CSE-2009**

**Outer Join:**

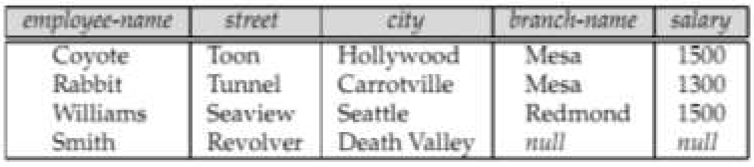
**The outer-join operation is an extension of the join operation to deal with missing information.** Since outer join operations may generate results containing null values, we need to specify how the different relational-algebra operations deal with null values.

There are actually three forms of the operation: **left outer join**, denoted ⟕ ; **right outer join**, denoted ⟖ ;and **full outer join**, denoted ⟗. All three forms of outer join compute the join, and add extra tuples to the result of the join. The outer join operations can be expressed by the basic relational-algebra operations. For instance, the left outer join operation, r ⟕ s, can be written as

$$(r \bowtie s) \cup (r - \Pi_R(r \bowtie s)) \times \{(null, \ldots, null)\}$$

Where the constant relation *{(null,...,null)}* is on the schema S − R.

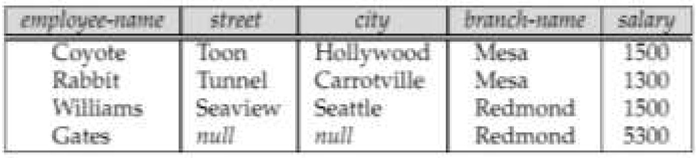
**The left outer join** (⟕) takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation, and adds them to the result of the natural join. In Figure 3.33, tuple (Smith, Revolver, Death Valley, null, null) is such a tuple. All information from the left relation is present in the result of the left outer join. The results of the expression **employee ⟕ ft-works**, appears in Figures 3.33.



| *employee-name* | *street* | *city* | *branch-name* | *salary* |
|---|---|---|---|---|
| Coyote | Toon | Hollywood | Mesa | 1500 |
| Rabbit | Tunnel | Carrotville | Mesa | 1300 |
| Williams | Seaview | Seattle | Redmond | 1500 |
| Smith | Revolver | Death Valley | *null* | *null* |

**Figure 3.33** Result of *employee* ⟕ *ft-works*.

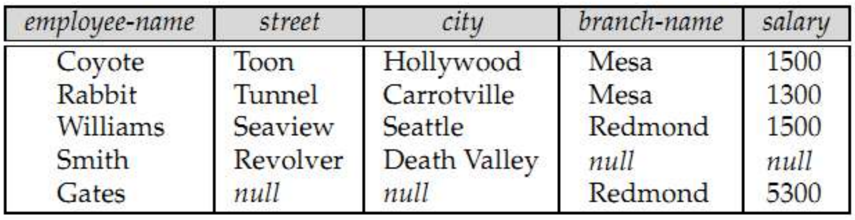
**The right outer join** ( ⟖ ) is symmetric with the left outer join: It pads tuples from the right relation that did not match any from the left relation with nulls and adds them to the result of the natural join. In Figure 3.34, tuple (Gates, null, null, Redmond, 5300) is such a tuple. Thus, all information from the right relation is present in the result of the right outer join. The results of the expression **employee ⟖ ft-works**, appears in Figures 3.34.



| *employee-name* | *street* | *city* | *branch-name* | *salary* |
|---|---|---|---|---|
| Coyote | Toon | Hollywood | Mesa | 1500 |
| Rabbit | Tunnel | Carrotville | Mesa | 1300 |
| Williams | Seaview | Seattle | Redmond | 1500 |
| Gates | *null* | *null* | Redmond | 5300 |

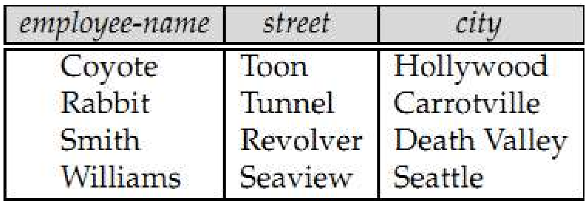
**Figure 3.34** Result of *employee* ⟖ *ft-works*.

**The full outer join**(⟗) does both of those operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join. The results of the expression **employee ⟗ ft-works**, appears in Figures 3.35.

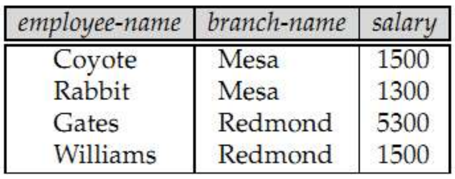


| *employee-name* | *street* | *city* | *branch-name* | *salary* |
|---|---|---|---|---|
| Coyote | Toon | Hollywood | Mesa | 1500 |
| Rabbit | Tunnel | Carrotville | Mesa | 1300 |
| Williams | Seaview | Seattle | Redmond | 1500 |
| Smith | Revolver | Death Valley | *null* | *null* |
| Gates | *null* | *null* | Redmond | 5300 |

**Figure 3.35** Result of *employee* ⟗ *ft-works*.



| *employee-name* | *street* | *city* |
|---|---|---|
| Coyote | Toon | Hollywood |
| Rabbit | Tunnel | Carrotville |
| Smith | Revolver | Death Valley |
| Williams | Seaview | Seattle |



| *employee-name* | *branch-name* | *salary* |
|---|---|---|
| Coyote | Mesa | 1500 |
| Rabbit | Mesa | 1300 |
| Gates | Redmond | 5300 |
| Williams | Redmond | 1500 |

**Figure 3.31** The *employee* and *ft-works* relations.

**Question: What are left outer join, right outer join and full outer join? Explain with example. 5 Marks CSE-2004**

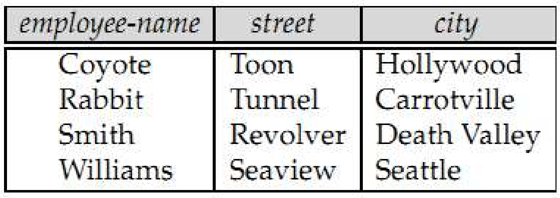
__Left Outer Join:__

**The outer-join operation is an extension of the join operation to deal with missing information. The left outer join** (⟕) takes all tuples in the left relation that did not match with any tuple in the right relation, pads the tuples with null values for all other attributes from the right relation, and adds them to the result of the natural join. All information from the left relation is present in the result of the left outer join. Suppose that we have the relations with the following schemas, which contain data on full-time employees:

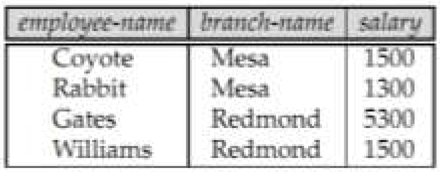
*employee (employee-name, street, city)*

*ft-works (employee-name, branch-name, salary)*

Consider the employee and ft-works relations in Figure 3.31. The results of the expression **employee** ⟕ **ft-works**, appears in **Figures 3.33.**

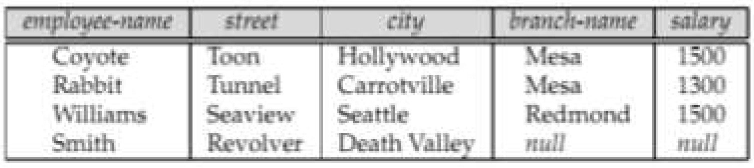


| *employee-name* | *street* | *city* |
|---|---|---|
| Coyote | Toon | Hollywood |
| Rabbit | Tunnel | Carrotville |
| Smith | Revolver | Death Valley |
| Williams | Seaview | Seattle |



| *employee-name* | *branch-name* | *salary* |
|---|---|---|
| Coyote | Mesa | 1500 |
| Rabbit | Mesa | 1300 |
| Gates | Redmond | 5300 |
| Williams | Redmond | 1500 |

**Figure 3.31** The *employee* and *ft-works* relations.



| *employee-name* | *street* | *city* | *branch-name* | *salary* |
|---|---|---|---|---|
| Coyote | Toon | Hollywood | Mesa | 1500 |
| Rabbit | Tunnel | Carrotville | Mesa | 1300 |
| Williams | Seaview | Seattle | Redmond | 1500 |
| Smith | Revolver | Death Valley | *null* | *null* |

**Figure 3.33** Result of *employee* ⟕ *ft-works*.

__Right Outer Join:__

**The right outer join** ( ⟖ ) is symmetric with the left outer join: It pads tuples from the right relation that did not match any from the left relation with nulls and adds them to the result of the natural join. In Figure 3.34, tuple (Gates, null, null, Redmond, 5300) is such a tuple. Thus, all information from the right relation is present in the result of the right outer join. The results of the expression **employee** ⟖ **ft-works**, appears in Figures 3.34.

| employee-name | street | city | branch-name | salary |
|---|---|---|---|---|
| Coyote | Toon | Hollywood | Mesa | 1500 |
| Rabbit | Tunnel | Carrotville | Mesa | 1300 |
| Williams | Seaview | Seattle | Redmond | 1500 |
| Gates | *null* | *null* | Redmond | 5300 |

**Figure 3.34** Result of *employee* ⟕ *ft-works*.

**Full Outer Join:**

**The full outer join(⟗)** does both of those operations, padding tuples from the left relation that did not match any from the right relation, as well as tuples from the right relation that did not match any from the left relation, and adding them to the result of the join. The results of the expression **employee ⟗ ft-works**, appears in Figures 3.35.

| employee-name | street | city | branch-name | salary |
|---|---|---|---|---|
| Coyote | Toon | Hollywood | Mesa | 1500 |
| Rabbit | Tunnel | Carrotville | Mesa | 1300 |
| Williams | Seaview | Seattle | Redmond | 1500 |
| Smith | Revolver | Death Valley | *null* | *null* |
| Gates | *null* | *null* | Redmond | 5300 |

**Figure 3.35** Result of *employee* ⟗ *ft-works*.

**Question: List two reasons why null values might be introduced into the database. 2 Marks CSE-2005**

**Question: Which operations are used to modify the database? Describe them with examples. 6 Marks CSE-2006**

**Modification of the Database:**

We address how to add, remove, or change information in the database. We express database modifications by using the assignment operation.

**Deletion**

We express a delete request in much the same way as a query. However, instead of displaying tuples to the user, we remove the selected tuples from the database. We can delete only whole tuples; we cannot delete values on only particular attributes. In relational algebra a deletion is expressed by

$$r \leftarrow r - E$$

where r is a relation and E is a relational-algebra query.

**Several examples of relational-algebra delete requests:**

• Delete all of Smith's account records.

$$\text{depositor} \leftarrow \text{depositor} - \sigma_{cu\,stomer-name = \text{"Smith"}}(\text{depositor})$$

• Delete all loans with amount in the range 0 to 50.

$$\text{loan} \leftarrow \text{loan} - \sigma_{amount \geq 0 \text{and } amount \leq 50}(\text{loan})$$

• Delete all accounts at branches located in Needham.

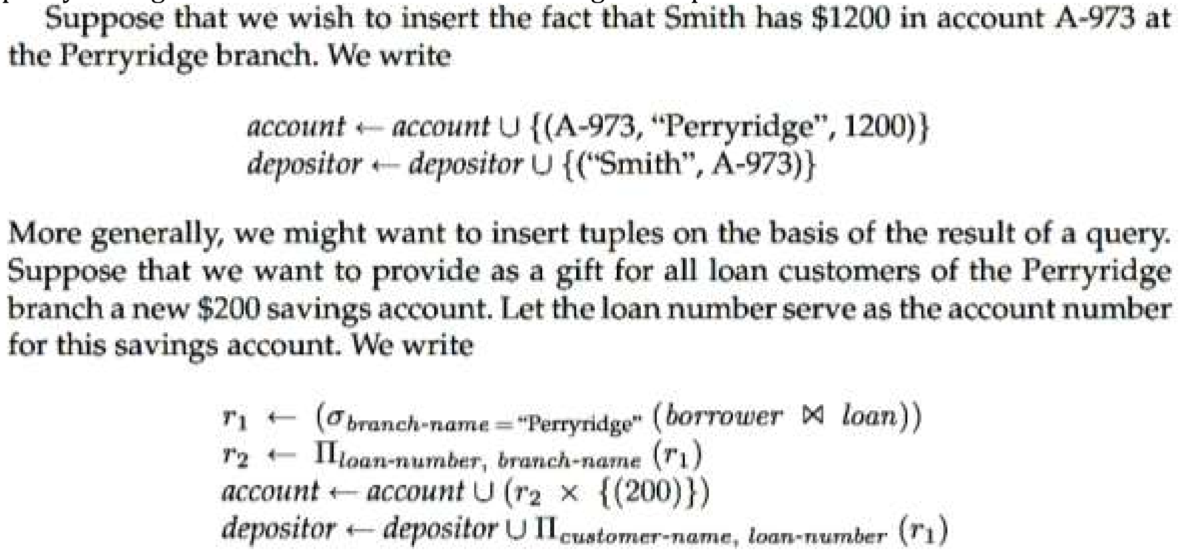
$$r_1 \leftarrow \sigma_{branch-city = \text{"Needham"}}(\text{account}\ \ \text{branch})$$
$$r_2 \leftarrow \Pi_{branch-name, account-number, balance}(r1)$$
$$\text{account} \leftarrow \text{account} - r_2$$

**Insertion**

To insert data into a relation, we either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted. Obviously, the attribute values for inserted tuples must be members of the attribute's domain. Similarly, tuples inserted must be of the correct arity. The relational algebra expresses an insertion by

$$\mathbf{r \leftarrow r \cup E}$$

where r is a relation and E is a relational-algebra expression. We express the insertion of a single tuple by letting E be a constant relation containing one tuple.


Suppose that we wish to insert the fact that Smith has $1200 in account A-973 at the Perryridge branch. We write

$$account \leftarrow account \cup \{(\text{A-973}, \text{"Perryridge"}, 1200)\}$$
$$depositor \leftarrow depositor \cup \{(\text{"Smith"}, \text{A-973})\}$$

More generally, we might want to insert tuples on the basis of the result of a query. Suppose that we want to provide as a gift for all loan customers of the Perryridge branch a new $200 savings account. Let the loan number serve as the account number for this savings account. We write

$$r_1 \leftarrow (\sigma_{branch\text{-}name = \text{"Perryridge"}}(borrower \bowtie loan))$$
$$r_2 \leftarrow \Pi_{loan\text{-}number,\ branch\text{-}name}(r_1)$$
$$account \leftarrow account \cup (r_2 \times \{(200)\})$$
$$depositor \leftarrow depositor \cup \Pi_{customer\text{-}name,\ loan\text{-}number}(r_1)$$

**Updating**

In certain situations, we may wish to change a value in a tuple without changing all values in the tuple. We can use the generalized-projection operator to do this task:

$$\mathbf{r \leftarrow \Pi_{F1,F2,\ldots,Fn}\ (r)}$$

where each Fi is either the ith attribute of r, if the ith attribute is not updated, or, if the attribute is to be updated, Fi is an expression, involving only constants and the attributes of r, that gives the new value for the attribute.

If we want to select some tuples from r and to update only them, we can use the following expression; here, P denotes the selection condition that chooses which tuples to update:

$$\mathbf{r \leftarrow \Pi_{F1,F2,\ldots,Fn}\ (\sigma_P\ (r)) \cup (r - \sigma_P\ (r))}$$

**Example:**

Suppose that interest payments are being made, and that all balances are to be increased by 5 percent. We write

$$account \leftarrow \Pi_{account-number, branch-name, balance * 1.05}\ (account)$$

Now suppose that accounts with balances over $10,000 receive 6 percent interest, whereas all others receive 5 percent. We write

$$\mathbf{account \leftarrow \Pi_{AN,BN,balance * 1.06}\ (\sigma_{balance > 10000}\ (account))}$$
$$\mathbf{\cup\ \Pi_{AN,BN\ balance * 1.05}\ (\sigma balance \leq 10000\ (account))}$$

where the abbreviations AN and BN stand for account-number and branch-name, respectively.

**Question: What is a View? Why do we define a View? 1+2 Marks CSE-2007**

**View:**

**Any relation that is not part of the logical model, but is made visible to a user as a virtual relation, is called a view**. It is possible to support a large number of views on top of any given set of actual relations.

We define a view by using the create view statement. To define a view, we must give the view a name, and must state the query that computes the view. The form of the create view statement is

**create view v as <query expression>**

where **<query expression>** is any legal relational-algebra query expression. The view name is represented by v.

**Why we define a view:**

Following are three reasons why we define a view:

1. **Views are useful mechanisms for simplifying database queries.**
2. **It is not desirable for all users to see the entire logical model. Security considerations may require that certain data be hidden from users.**
   **Example:**
   Consider a person who needs to know a customer's loan number and branch name, but has no need to see the loan amount. This person should see a relation described, in the relational algebra, by
   $$\Pi_{\text{customer-name, loan-number, branch-name}} (\text{borrower} \times \text{loan})$$
3. Aside from security concerns, **we may wish to create a personalized collection of relations that is better matched to a certain user's intuition than is the logical model.**
   **Example:**
   An employee in the advertising department, for example, might like to see a relation consisting of the customers who have either an account or a loan at the bank, and the branches with which they do business. The relation that we would create for that employee is
   $$\Pi_{branch\text{-}name,\ customer\text{-}name} (depositor \bowtie account) \cup \Pi_{branch\text{-}name,\ customer\text{-}name} (borrower \bowtie loan)$$

**Similar Questions: List two reasons why we define a view. 2 Marks CSE- 2003 (OLD)**

**Question: Why do we use a number of relations that are connected in some ways rather than just a relation that contains all the attributes? 2 Marks CSE-2007**

**Question: What are the reasons for the popularity of relational data model? 5 Marks CSE-2006**

**Reasons for the popularity of Relational Data Model:**

There are several reasons why the relational data model has become the most common one for use in databases, some of them are:

1. The relational model is today the primary data model for commercial data-processing applications. It has attained its primary position because of its simplicity, which eases the job of the programmer, as compared to earlier data models such as the network model or the hierarchical model.
2. It is demonstrably superior in many respects to the previous data models that were employed in database construction. It is also widely acknowledged and understood to be so.

3. The structure of a relational database also allowed sorting based on any field and the generation of reports that contain only certain fields from each record. A relational database uses the relationship of similar data to increase the speed and versatility of the database. This data access methodology made the relational model a lot different from and better than the earlier database models because it was a much simpler model to understand and work with. This is another widely-cited reason for the popularity of relational database systems in the world today.

4. In addition to being relatively easy to create and access, a relational database also had the important advantage of being easy to extend. After the original database creation, a new

data category could be added without requiring that all existing applications be modified (such as would normally be required with flat file databases, for example).

5. Another benefit of the relational system is that it provided extremely useful tools for database administration. Essentially, tables can not only store actual data but they can also be used as the means for generating meta-data (data about the table and field names which form the database structure, access rights to the database, integrity and data validation rules etc).

6. Lastly, along with the development of the relational data model came the development of SQL (Structured Query Language, a database query language). SQL is relatively easy to learn, has been adopted as an industry-wide standard since 1986, and allows people to quickly develop the capability to query and modify a relational database. The straightforwardness and simplicity of manipulation that SQL permits is another part of the reason that relational databases now form the majority of databases to be found.

# Chapter 4 Structured Query Language

## Important Question

1. Describe the basic structure of an SQL expression with example. **4 Marks CSE-2011 CSE-2009 CSE-2008 CSE-2006 CSE-2003 CSE-2003 (OLD) CSE-2001 CSE-2000 CSE-2000 (OLD)**
2. What is view? Why do we define view? Create a view consisting of branch name and customer name, who have either a loan with amount greater than 10 lacs or an account with balance less than 5 lacs. 3 + 2 Marks **CSE-2011 CSE-2006 CSE -2001**
3. What do you mean by aggregate functions? Briefly explain. **5 Marks CSE-2010 CSE-2005**
4. What is the function of string operation in SQL? **5 Marks CSE-2010 CSE-2008 CSE-2005 CSE-2003 CSE-2003 (OLD)**
5. Explain the string operation of SQL statement with examples. **4 Marks CSE-2009 CSE-2007 CSE-2001**
6. **What** is aggregate function? Describe different types of aggregation function with example. **6 Marks CSE-2009 CSE-2007**
7. Discuss the several parts of SQL language. **5 Marks CSE-2007**
8. **Discuss** different types of domain that SQL-92 standard supports. **5 Marks CSE-2007**
9. Explain the use of **group by** and **order by** clauses. **4 Marks CSE-2006**
10. Explain how Schema is defined in SQL. **3 Marks CSE-2004 CSE-2000**
11. Show that in SQL, **<> all** is identical to **'not in'**. **3 Marks CSE-2004**
12. What is SQL? Is there any query language like SQL? Define DDL and DML with examples **2 Marks CSE-2003 (OLD) CSE-2001 CSE-2013**
13. What do you mean by referential integrity and how it is implemented in SQL? 3 Marks CSE-2001
14. Illustrate the use of **'from'** clause in SQL. **3 Marks CSE-2001**
15. Illustrate the use of **'where'** clause in SQL. **3 Marks CSE-2001 CSE-2000**
16. Describe following topics of SQL: **CSE-2000**
    (a) Modification of the database. **3 Marks**
    (b) Domain type in Database. **2 Marks**

(c) Schema Definition in SQL. **2 Marks**

**Question -01: What is SQL? Is there any query language like SQL? 2 Marks CSE-2003 (OLD) CSE-2001**

**SQL:**

**SQL, Structured Query Language,** is a special-purpose programming language designed for managing data held in a relational database management system (RDBMS). Originally based upon relational algebra and tuple relational calculus, SQL consists of a data definition language and a data manipulation language. The scope of SQL includes data insert, query, update and delete, schema creation and modification, and data access control. Although SQL is often described as, and to a great extent is, a declarative language (4GL), it also includes procedural elements. SQL became the most widely used database language.

**The SQL language has several parts:**
- **Data-definition language (DDL).**
- **Interactive data-manipulation language (DML).**
- **View definition.**
- **Transaction control.**
- **Embedded SQL and dynamic SQL**
- **Integrity.**
- **Authorization.**

**Query Languages Like SQL:**

1. **QUEL** is a relational database query language, based on tuple relational calculus, with some similarities to SQL.
2. **QL** is a proprietary object-oriented query language for querying relational databases; successor of Datalog;
3. **Tutorial D** is a query language for truly relational database management systems (TRDBMS)
4. **YQL** is an SQL-like query language created by Yahoo!

**Question-02: Describe following topics of SQL: (a) Modification of the database. 3 Marks (b) Domain type in Database. 2 Marks (c) Schema Definition in SQL. 2 Marks CSE-2000**

**Modification of the database:**

Modification of database can be done through add, remove or change information of the database. Now we show how to add, remove, or change information with SQL.

**Deletion**

A delete request is expressed in much the same way as a query. We can delete only whole tuples; we cannot delete values on only particular attributes. SQL expresses a deletion by

**delete from** r
**where** P

where P represents a predicate and r represents a relation. The delete statement first finds all tuples t in r for which P (t) is true, and then deletes them **from** r. The where clause can be omitted, in which case all tuples in r are deleted. A delete command operates on only one relation. If we want to delete tuples **from** several relations, we must use one delete command for each relation.

**Example:**

Examples of SQL delete requests:

- Delete all account tuples in the Perryridge branch.

  **delete from** account
  **where** branch-name = 'Perryridge'

- Delete all loans with loan amounts between $1300 and $1500.

  **delete from** loan
  **where** amount between 1300 and 1500

## Insertion

To insert data into a relation, we either specify a tuple to be inserted or write a query whose result is a set of tuples to be inserted. Obviously, the attribute values for inserted tuples must be members of the attribute's domain. Similarly, tuples inserted must be of the correct arity. The simplest insert statement is a request to insert one tuple. Suppose that we wish to insert the fact that there is an account A-9732 at the Perryridge branch and that is has a balance of $1200. We write:

**insert** into account
**values** ('A-9732', 'Perryridge', 1200)

## Updates

In certain situations, we may wish to change a value in a tuple without changing all values in the tuple. For this purpose, the update statement can be used. As we could for insert and delete, we can choose the tuples to be updated by using a query. Suppose that annual interest payments are being made, and all balances are to be increased by 5 percent. We write

**update** account
**set** balance = balance *1.05

## Domain type in Database:

The SQL standard supports a variety of built-in domain types, including:

• **char(n):** A fixed-length character string with user-specified length n. The full form, character, can be used instead.

• **varchar(n):** A variable-length character string with user-specified maximum length n. The full form, character varying, is equivalent.

• **int:** An integer (a finite subset of the integers that is machine dependent). The full form, integer, is equivalent.

• **smallint:** A small integer (a machine-dependent subset of the integer domain type).

• **numeric(p, d):** A fixed-point number with user-specified precision. The number consists of p digits (plus a sign), and d of the p digits are to the right of the decimal point. Thus, numeric(3,1) allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.

• **real, double precision:** Floating-point and double-precision floating-point numbers with machine-dependent precision.

• **float(n):** A floating-point number, with precision of at least n digits.

• **date:** A calendar date containing a (four-digit) year, month, and day of the month.

• **time:** The time of day, in hours, minutes, and seconds. A variant, *time(p),* can be used to specify the number of fractional digits for seconds (the default being 0). It is also possible to store time zone information along with the time.

• **timestamp:** A combination of date and time. A variant, *timestamp(p),* can be used to specify the number of fractional digits for seconds (the default here being 6).

**Schema Definition in SQL:**
We define an SQL relation by using the create table command:
create table $r(A_1D_1, A_2D_2, \dots\dots\dots., A_nD_n,$

$$\{integrity - constraint_1\},$$
$$\{integrity - constraint_2\},$$
$$\dots\dots\dots\dots\dots\dots\dots\dots\,,$$
$$\{integrity - constraint_k\})$$

where r is the name of the relation, each $A_i$ is the name of an attribute in the schema of relation r, and $D_i$ is the domain type of values in the domain of attribute $A_i$. The allowed integrity constraints include

- **primary key** $(\boldsymbol{A_{j1}, A_{j2}, \dots.., A_{jm}})$**:** The primary key specification says that attributes $A_{j1}, A_{j2}, \dots.., A_{jm}$): form the primary key for the relation. The primary key attributes are required to be non-null and unique; that is, no tuple can have a null value for a primary key attribute, and no two tuples in the relation can be equal on all the primary-key attributes.1 Although the primary key specification is optional, it is generally a good idea to specify a primary key for each relation.

- **check(P):** The check clause specifies a predicate P that must be satisfied by every tuple in the relation.

**Question-03: Discuss the several parts of SQL language. 5 Marks CSE-2007**

**Parts of SQL language:**
The SQL language has several parts:

• **Data-definition language (DDL):** The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.

• **Interactive data-manipulation language (DML):** The SQL DML includes a query language based on both the relational algebra and the tuple relational calculus. It includes also commands to insert tuples into, delete tuples **from**, and modify tuples in the database.

• **View definition:** The SQL DDL includes commands for defining views.

• **Transaction control:** SQL includes commands for specifying the beginning and ending of transactions.
• **Embedded SQL and dynamic SQL:** Embedded and dynamic SQL define how SQL statements can be embedded within general-purpose programming languages, such as C, C++, Java, PL/I, Cobol, Pascal, and Fortran.

• **Integrity:** The SQL DDL includes commands for specifying integrity constraints that the data stored in the database must satisfy. Updates that violate integrity constraints are disallowed.

• **Authorization:** The SQL DDL includes commands for specifying access rights to relations and views.

**Question-04: Describe the basic structure of an SQL expression with example. 4 Marks CSE-2011 CSE-2009 CSE-2008 CSE-2006 CSE-2003 CSE-2003 (OLD) CSE-2001 CSE-2000 CSE-2000 (OLD)**

**Basic structure of an SQL expression:**

The basic structure of an SQL expression consists of three clauses: **select**, **from**, and **where**.

- **Select clause:** The **select** clause corresponds to the projection operation of the relational algebra. It is used to list the attributes desired in the result of a query.
- The **select** clause list the attributes desired in the result of a query
- corresponds to the projection operation of the relational algebra

- **From clause:** The **from** clause corresponds to the Cartesian-product operation of the relational algebra. It lists the relations to be scanned in the evaluation of the expression.
- The **from** clause lists the relations involved in the query
- corresponds to the Cartesian product operation of the relational algebra.

- **Where clause**: The where clause corresponds to the **selec**tion predicate of the relational algebra. It consists of a predicate involving attributes of the relations that appear in the **from** clause.
- The **where** clause specifies conditions that the result must satisfy
- corresponds to the selection predicate of the relational algebra.

A typical SQL query has the form

**select** $A_1, A_2, \ldots, A_n$
**from** $r_1, r_2, \ldots, r_m$
**where** $P$

Each $A_i$ represents an **attribute**, and each $r_i$ a **relation**. **P** is a **predicate**. The query is equivalent to the relational-algebra expression

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \cdots \times r_m))$$

**Question-05: Illustrate the use of 'Select' clause in SQL. 3 Marks CSE-2001**

**The select Clause:**

The result of an SQL query is, of course, a relation. Let us consider a simple query, **"Find the names of all branches in the loan relation":**

**select** branch-name
**from** loan

The result is a relation consisting of a single attribute with the heading branch-name. Formal query languages are based on the mathematical notion of a relation being a set. Thus, duplicate tuples never appear in relations. In practice, duplicate elimination is time-consuming. Therefore, SQL (**like** most other commercial query languages) allows duplicates in relations as well as in the results of SQL expressions. Thus, the preceding query will list each branch-name once for every tuple in which it appears in the loan relation.

In those cases where we want to force the elimination of duplicates, we insert the keyword distinct after **select**. We can rewrite the preceding query as

**select** distinct branch-name
**from** loan

if we want duplicates removed. SQL allows us to use the keyword **all** to specify explicitly that duplicates are not removed:

```
select all branch-name
from loan
```

Since duplicate retention is the default, we will not use all in our examples. To ensure the elimination of duplicates in the results of our example queries, we will use distinct whenever it is necessary. In most queries where distinct is not used, the exact number of duplicate copies of each tuple present in the query result is not important.

The asterisk symbol **" * "** can be used to denote **"all attributes."** Thus, the use of loan.* in the preceding **select** clause would indicate that all attributes of loan are to be **selec**ted. A **select** clause of the form **select** * indicates that all attributes of all relations appearing in the **from** clause are **select**ed.

The **select** clause may also contain arithmetic expressions involving the operators +, −, ∗, and / operating on constants or attributes of tuples. For example, the query

```
select loan-number, branch-name, amount * 100
from loan
```

will return a relation that is the same as the loan relation, except that the attribute amount is multiplied by 100.

**Question-06: Illustrate the use of 'from' clause in SQL. 3 Marks CSE-2001**

**<u>The from Clause</u>**

Finally, let us discuss the use of the **from** clause. The **from** clause by itself defines a Cartesian product of the relations in the clause. Since the natural join is defined in terms of a Cartesian product, a **select**ion, and a projection, it is a relatively simple matter to write an SQL expression for the natural join. We write the relational-algebra expression

$$\Pi_{customer-name,loan-number,amount}\ (\text{borrower} \bowtie \text{loan})$$

for the query "For all customers who have a loan **from** the bank, find their names, loan numbers and loan amount." In SQL, this query can be written as

```
select customer-name, borrower.loan-number, amount
from borrower, loan
where borrower.loan-number = loan.loan-number
```

We can extend the preceding query and consider a more complicated case in which we require also that the loan be **from** the Perryridge branch: "Find the customer names, loan numbers, and loan amounts for all loans at the Perryridge branch." To write this query, we need to state two constraints in the **where** clause, connected by the logical connective **and:**

```
select customer-name, borrower.loan-number, amount
from borrower, loan
where borrower.loan-number = loan.loan-number and
        branch-name = 'Perryridge'
```

SQL includes extensions to perform natural joins and outer joins in the **from** clause.

**Question-07: Illustrate the use of 'where' clause in SQL. 3 Marks CSE-2001 CSE-2000**

**<u>The where Clause:</u>**

Let us illustrate the use of the where clause in SQL. Consider the query "Find all loan numbers for loans made at the Perryridge branch with loan amounts greater that $1200." This query can be written in SQL as:

```
select loan-number
from loan
where branch-name = 'Perryridge' and amount > 1200
```

SQL uses the logical connectives **and**, **or**, and **not** — rather than the mathematical symbols ∧ , ∨ ,and ¬ —in the **where** clause. The operands of the logical connectives can be expressions involving the comparison operators <, <=, >, >=, =,and <>. SQL allows us to use the comparison operators to compare strings and arithmetic expressions, as well as special types, such as date types.
SQL includes a between comparison operator to simplify where clauses that specify that a value be less than or equal to some value and greater than or equal to some other value. If we wish to find the loan number of those loans with loan amounts between $90,000 and $100,000, we can use the between comparison to write

```
select loan-number
from loan
where amount between 90000 and 100000
```

instead of

```
select loan-number
from loan
where amount <= 100000 and amount >= 90000
```

Similarly, we can use the **not between** comparison operator.

**Question-08: Explain the Rename operation of SQL statement with examples. 4 Marks CSE-2009 CSE-2007 CSE-2001**

## The Rename Operation

SQL provides a mechanism for renaming both relations and attributes. It uses the as clause, taking the form:

*old-name* ***as*** *new-name*

The **as** clause can appear in both the **select** and **from** clauses.
Consider the query:

```
select customer-name, borrower.loan-number, amount
from borrower, loan
where borrower.loan-number = loan.loan-number
```

The result of this query is a relation with the following attributes:

*customer-name, loan-number, amount.*

The names of the attributes in the result are derived **from** the names of the attributes in the relations in the **from** clause.
We cannot, however, always derive names in this way, for several reasons: First, two relations in the **from** clause may have attributes with the same name, in which case an attribute name is duplicated in the result. Second, if we used an arithmetic expression in the **select** clause, the resultant attribute does not have a name. Third, even if an attribute name can be derived **from** the base relations as in the preceding example, we may want to change the attribute name in the result. Hence, SQL provides a way of renaming the attributes of a result relation.

**For example**, if we want the attribute name *loan-number* to be replaced with the name *loan-id*, we can rewrite the preceding query as

```
select customer-name, borrower.loan-number as loan-id, amount
from borrower, loan
where borrower.loan-number = loan.loan-number
```

**Question-09: Explain the string operation of SQL statement with examples. 4 Marks CSE-2009 CSE-2007 CSE-2001**

## String Operations:

SQL specifies strings by enclosing them in single quotes, for example, ’Perryridge’. A single quote character that is part of a string can be specified by using two single quote characters; for example the string “It’s right” can be specified by ’It”s right’. The most commonly used operation on strings is pattern matching using the operator **like**. We describe patterns by using two special characters:

• **Percent (%):** The % character matches any substring.
• **Underscore ( _ ):** The _ character matches any character.

Patterns are case sensitive; that is, uppercase characters do not match lowercase characters, or vice versa. To illustrate pattern matching, we consider the following examples:

• ’Perry%’ matches any string beginning with “Perry”.

• ’%idge%’ matches any string containing “idge” as a substring, for example, ’Perryridge’, ’Rock Ridge’, ’Mianus Bridge’, and ’Ridgeway’.

• ’_ _ _ ’ matches any string of exactly three characters.

• ’_ _ _ %’ matches any string of at least three characters.

SQL expresses patterns by using the **like** comparison operator. Consider the query **“Find the names of all customers whose street address includes the substring ‘Main’.”** This query can be written as

```
select customer-name
from customer
where customer-street like ’%Main%’
```

For patterns to include the special pattern characters (that is, % and _ ), SQL allows the specification of an escape character. The escape character is used immediately before a special pattern character to indicate that the special pattern character is to be treated **like** a normal character. We define the escape character for a **like** comparison using the **escape** keyword. To illustrate, consider the following patterns, which use a backslash (\) as the escape character:

• **like** ’ab\%cd%’ **escape** ’\’ matches all strings beginning with “ab%cd”.
• **like** ’ab\\cd%’ **escape** ’\’ matches all strings beginning with “ab\cd”.

SQL allows us to search for mismatches instead of matches by using the **not like** comparison operator. SQL also permits a variety of functions on character strings, such as concatenating (using “||”), extracting substrings, finding the length of strings, converting be-tween uppercase and lowercase, and so on. SQL:1999 also offers a similar to operation, which provides more powerful pattern matching than the **like** operation; the syntax for specifying patterns is similar to that used in Unix regular expressions.

**Question-10: What is aggregate function? Describe different types of aggregation function with example. 6 Marks CSE-2009 CSE-2007**

**Aggregate Functions:**
Aggregate functions are functions that take a collection (a set or multiset) of values as input and return a single value.

**Different types of aggregation function:**
SQL offers five built-in aggregate functions:

- Average: **avg**
- Minimum: **min**
- Maximum: **max**
- Total : **sum**
- Count: **count**

The input to **sum** and **avg** must be a collection of numbers, but the other operators can operate on collections of nonnumeric data types, such as strings, as well.

**Aggregate function – sum and avg:**

As an illustration for the aggregate function **sum** , consider the query "Find the **sum** of account balance at the Perryridge branch." We write this query as follows:

```
select sum(balance)
from account
where branch-name = 'Perryridge'
```

As an illustration for aggregate function **avg** , consider the query "Find the average account balance at the Perryridge branch." We write this query as follows:

```
select avg (balance)
from account
where branch-name = 'Perryridge'
```

If we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuples; we use the **group by** clause. The attribute or attributes given in the group by clause are used to form groups. Tuples with the same value on all attributes in the group by clause are placed in one group.
As an illustration, consider the query "Find the average account balance at each branch." We write this query as follows:

```
select branch-name, avg (balance)
from account
group by branch-name
```

**Aggregate Function - Count:**

We use the aggregate function **count** frequently to count the number of tuples in a relation. The notation for this function in SQL is **count (*).** Thus, to find the number of tuples in the customer relation, we write

```
select count (*)
from customer
```

Again we consider the query **"Find the average balance for each customer who lives in Harrison and has at least three accounts.". we write this query as follows:**

```
select depositor.customer-name, avg (balance)
from depositor, account, customer
where depositor.account-number = account.account-number and
        depositor.customer-name = customer.customer-name and
        customer-city = 'Harrison'
group by depositor.customer-name
having count (distinct depositor.account-number) >= 3
```

**Question-11: Explain the use of group by and order by clauses. 4 Marks CSE-2006**

**Group by Clause:**

There are circumstances where we would like to apply the aggregate function not only to a single set of tuples, but also to a group of sets of tuples; we specify this wish in SQL using the group by clause. The attribute or attributes given in the group by clause are used to form groups. Tuples with the same value on all attributes in the group by clause are placed in one group.

**As an illustration**, consider the query **"Find the average account balance at each branch."** We write this query as follows:

```
select branch-name, avg (balance)
from account
group by branch-name
```

**Order By Clause:**

SQL offers the user some control over the order in which tuples in a relation are displayed. The order by clause causes the tuples in the result of a query to appear in sorted order. To list in alphabetic order all customers who have a loan at the Perryridge branch, we write

```
select distinct customer-name
from borrower, loan
where borrower.loan-number = loan.loan-number and branch-name =
'Perryridge'
order by customer-name
```

By default, the **order by** clause lists items in ascending order. To specify the sort order, we may specify **desc** for descending order or **asc** for ascending order. Furthermore, ordering can be performed on multiple attributes. Suppose that we wish to list the entire loan relation in descending order of amount. If several loans have the same amount, we order them in ascending order by loan number. We express this query in SQL as follows:

```
select *
from loan
order by amount desc, loan-number asc
```

To fulfill an **order by** request, SQL must perform a sort. Since sorting a large number of tuples may be costly, it should be done only when necessary.

**Question-12: Discuss different types of domain that SQL-92 standard supports. 5 Marks CSE-2007**

**Different types of domain that SQL-92 standard supports:**

The SQL standard supports a variety of built-in domain types, including:

- **char(n):** A fixed-length character string with user-specified length n. The full form, character, can be used instead.

- **varchar(n):** A variable-length character string with user-specified maximum length n. The full form, character varying, is equivalent.

- **int:** An integer (a finite subset of the integers that is machine dependent). The full form, integer, is equivalent.

- **smallint:** A small integer (a machine-dependent subset of the integer domain type).

- **numeric(p, d):** A fixed-point number with user-specified precision. The number consists of p digits (plus a sign), and d of the p digits are to the right of the decimal point. Thus, numeric(3,1) allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.

• **real, double precision:** Floating-point and double-precision floating-point numbers with machine-dependent precision.

• **float(n):** A floating-point number, with precision of at least n digits.

• **date:** A calendar date containing a (four-digit) year, month, and day of the month.

• **time:** The time of day, in hours, minutes, and seconds. A variant, *time(p),* can be used to specify the number of fractional digits for seconds (the default being 0). It is also possible to store time zone information along with the time.

• **timestamp:** A combination of date and time. A variant, *timestamp(p),* can be used to specify the number of fractional digits for seconds (the default here being 6).

**Question-13: Show that in SQL, <> all is identical to 'not in'. 3 Marks CSE-2004**

**Answer:** Let the set denote the result of an SQL subquery. We compare ($x <>$ **all** S) with ($x$ **not in** S). If a particular value $x_1$ satisfies ($x_1 <>$ **all** S) then for all elements $y$ of S, $x_1 \neq y$. Thus $x_1$ is not a member of S and must satisfy ($x_1$ **not in** S). Similarly, suppose there is a particular value $x_2$which satisfies ($x_2$ **not in** S ). It cannot be equal to any element $w$ belonging to S, and hence ($x_2 <>$ **all** S) will be satisfied. Therefore the two expressions are equivalent.

**Question-14: Explain how Schema is defined in SQL. 3 Marks CSE-2004 CSE-2000**

<u>**Schema Definition in SQL:**</u>
We define an SQL relation by using the create table command:
create table $r(A_1D_1, A_2D_2, \ldots\ldots\ldots., A_nD_n,$

$$\{integrity - constraint_1\},$$
$$\{integrity - constraint_2\},$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots ,$$
$$\{integrity - constraint_k\})$$

where r is the name of the relation, each $A_i$ is the name of an attribute in the schema of relation r, and $D_i$ is the domain type of values in the domain of attribute $A_i$. The allowed integrity constraints include

- **primary key ($A_{j1}, A_{j2}, \ldots\ldots, A_{jm}$):** The primary key specification says that attributes $A_{j1}, A_{j2}, \ldots\ldots, A_{jm}$): form the primary key for the relation. The primary key attributes are required to be non-null and unique; that is, no tuple can have a null value for a primary key attribute, and no two tuples in the relation can be equal on all the primary-key attributes.1 Although the primary key specification is optional, it is generally a good idea to specify a primary key for each relation.

- **check(P):** The check clause specifies a predicate P that must be satisfied by every tuple in the relation.

**Example:** Figure 4.8 presents a partial SQL DDL definition of a bank database. We use customer-name as a primary key to keep our database schema simple and short.

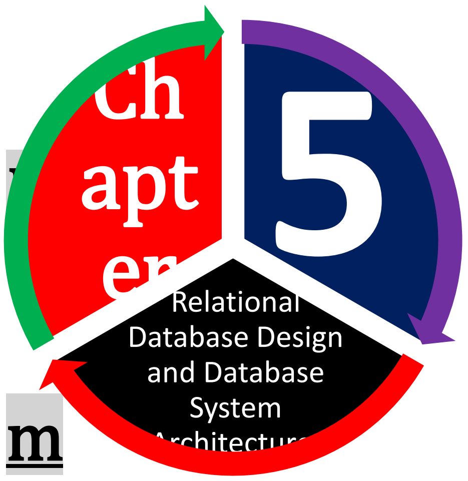
```
create table customer
   (customer-name    char(20),
    customer-street  char(30),
    customer-city    char(30),
    primary key (customer-name))

create table branch
   (branch-name      char(15),
    branch-city      char(30),
    assets           integer,
    primary key (branch-name),
    check (assets >= 0))

create table account
   (account-number   char(10),
    branch-name      char(15),
    balance          integer,
    primary key (account-number),
    check (balance >= 0))

create table depositor
   (customer-name    char(20),
    account-number   char(10),
    primary key (customer-name, account-number))
```

**Figure 4.8** SQL data definition for part of the bank database.



m

## Important Question

1. **What are the problems caused by data redundancy in a database? 4 Marks CSE-2013**
2. What is Normalization? State the goals of Normalization. Why is it necessary? **4 Marks CSE-2012 CSE-2011 CSE-2010 CSE-2005 CSE-2003 CSE-2003 (OLD)**
3. Explain First Normal Form (1NF) and Second Normal Form (2NF) with examples. **6 Marks CSE-2012 CSE-2010 CSE-2005 CSE-2003**
4. Define the following terms with examples:
   (a) Extraneous Attributes. **CSE- 2012 CSE-2011** (b) Lossless Join Decomposition. **CSE- 2012 CSE-2011 CSE-2002** (c) Functional Dependencies. **CSE-2011 CSE-2000** (d) Multivalued Dependencies. **CSE-2011** (e) Dependency Preservation. **CSE-2002**
5. Define Functional Dependency. What are Armstrong's axioms? Define Keys in terms of Functional Dependencies. **5 Marks CSE-2013 (Engg) CSE-2000**
6. What is Boyce-Codd Normal Form? Write the algorithm for decomposing a relation schema R into BCNF. **4 Marks CSE-2012 (Engg) CSE-2002**
7. What is Decomposition? Can you explain the formal definition of functional dependency? **1+2 CSE-2012 (Engg)**
8. How can you define the closure of a set of Functional dependencies? **3.75 Marks CSE-2012 (Engg)**
9. Define closure set. Compute four members of $F^+$ from the relation $R = \{A, B, C, G, H, I\}$ and the functional dependency set $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$. **5 Marks CSE-2011**
10. Define Canonical Cover. Compute the canonical cover of the following of the following set $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$ of functional dependencies for relational schema $R = \{A, B, C\}$. **5 Marks CSE-2011**
11. Compute the closure of the following set F of functional dependencies for the following relational schema, $R = \{A, B, C, D, E\}$ and $F = \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$. List candidate keys for R. **4 Marks CSE-2010 CSE-2009 CSE-2005 CSE-2002**
12. What criterion should be fulfilled for lossless join decomposition? **3 Marks CSE-2003 (OLD)**
13. Briefly describe 2NF and 3 NF with examples. **5 Marks CSE-2003 (OLD) CSE-2002**
14. What are the design goals of a good relational database design? Is it always possible to achieve these goals? Justify your answer. **5 Marks CSE-2003 (OLD) CSE-2000**
15. Discuss the problem of spurious tuples and how we may prevent it. **2 Marks CSE-2002**
16. Why Normalization is needed? What is a non-prime attribute? **2+1 CSE-2002**
17. What do you mean by the term "Decomposition" of a relation? What are the desirable properties of decomposition? **4 Marks CSE-2002 CSE-2000**
18. When can we say a relational schema is in 3NF? **2 Marks CSE-2000**

**Question-01: What is Normalization? State the goals of Normalization. Why is it necessary? What are advantages and disadvantages of Normalization? 4 Marks CSE-2012 CSE-2011 CSE-2010 CSE-2005 CSE-2003 CSE-2003 (OLD) CSE-2002**

**Normalization:**

**Normalization is a design technique that is widely used as a guide in designing relational databases to eliminate the redundancy of data and to make sure that the data dependencies (relationship) make sense that** improves storage efficiency, data integrity and scalability. Normalization is basically a two-step process that puts data into tabular form by removing repeating groups and then removes duplicate data from the relational tables. Normalization is based on the concepts of normal forms. A relational table is a normal form if it's satisfied with certain constraints. There are currently five normal forms for normalization:

1. First Normal Form.
2. Second Normal Form.
3. Third normal form.
4. Fourth Normal Form.
5. Fifth Normal Form.

**Goals of Normalization:**

The main goals of normalization process are:

a. To eliminate the redundancy of data
b. To make sure that the data dependencies (relationship) make sense.

By these two goals we reduce the space occupied by the duplicate data in the database tables and ensure that the data is logically stored there.

**When Is It Necessary:**

Normalization is the aim of well design Relational Database Management System (RDBMS). It is step by step set of rules by which data is put in its simplest forms. We normalize the relational database management system because of the following reasons:

1. Minimize data redundancy i.e. no unnecessarily duplication of data.
2. To make database structure flexible i.e. it should be possible to add new data values and rows without reorganizing the database structure.
3. Data should be consistent throughout the database i.e. it should not suffer from following anomalies.
4. **Insert Anomaly** - Due to lack of data i.e., all the data available for insertion such that null values in keys should be avoided. This kind of anomaly can seriously damage a database
5. **Update Anomaly** - It is due to data redundancy i.e. multiple occurrences of same values in a column. This can lead to inefficiency.
6. Deletion Anomaly - It leads to loss of data for rows that are not stored else where. It could result in loss of vital data.
7. Complex queries required by the user should be easy to handle.
8. On decomposition of a relation into smaller relations with fewer attributes on normalization the resulting relations whenever joined must result in the same relation without any extra rows. The join operations can be performed in any order. This is known as Lossless Join decomposition.
9. The resulting relations (tables) obtained on normalization should possess the properties such as each row must be identified by a unique key, no repeating groups, homogenous columns, each column is assigned a unique name etc.

**Advantages Of Normalization**

The following are the advantages of the normalization.

1. More efficient data structure.

2. Avoid redundant fields or columns.
3. More flexible data structure i.e. we should be able to add new rows and data values easily
4. Better understanding of data.
5. Ensures that distinct tables exist when necessary.
6. Easier to maintain data structure i.e. it is easy to perform operations and complex queries can be easily handled.
7. Minimizes data duplication.
8. Close modelling of real world entities, processes and their relationships.

**Disadvantages Of Normalization**
The following are disadvantages of normalization.
1. You cannot start building the database before you know what the user needs.
2. On Normalizing the relations to higher normal forms i.e. 4NF, 5NF the performance degrades.
3. It is very time consuming and difficult process in normalizing relations of higher degree.
4. Careless decomposition may leads to bad design of database which may leads to serious problems.

**সংক্ষেপে লেখা যেতে পারে, কেন নরমালাইজেশন প্রয়োজনঃ**

**Why is it necessary:**
Data normalization is necessary for the following reasons:
- To eliminate the redundancy of data and ensure data integrity.
- To make sure that the data dependencies (relationship) make sense.
- To make optimized queries on the normalized tables and produce fast, efficient results.
- To make faster index and also make a perfect sorting.
- To increase the performance of the database.

**Question-02: What are the problems caused by data redundancy in a database? 4 Marks CSE-2013**

**Problems Caused by Data Redundancy:**
Storing the same information redundantly, that is, in more than one place within a database, can lead to several problems:

1. **Redundant storage:** Some information is stored repeatedly.

2. **Update anomalies:** If one copy of such repeated data is updated, an inconsistency is created unless all copies are similarly updated.

3. **Insertion anomalies:** It may not be possible to store some information unless some other information is stored as well.

4. **Deletion anomalies:** It may not be possible to delete some information without losing some other information as well.

**Question-03: Compute the closure of the following set F of functional dependencies for the following relational schema, $R = \{A, B, C, D, E\}$ and $F = \{A \rightarrow BC, CD \rightarrow E, B \rightarrow D, E \rightarrow A\}$. List candidate keys for R. 4 Marks CSE-2010 CSE-2009 CSE-2005 CSE-2002**

Starting with A → BC, we can conclude: A → B and A → C.

| | |
|---|---|
| Since A → B and B → D, A → D | (decomposition, transitive) |
| Since A → CD and CD → E, A → E | (union, decomposition, transitive) |
| Since A → A, we have | (reflexive) |
| A → ABCDE from the above steps | (union) |

Since E → A, E → ABCDE (transitive)
Since CD → E, CD → ABCDE (transitive)
Since B → D and BC → CD, BC → ABCDE (augmentative, transitive)
Also, C → C, D → D, BD → D,etc.

Therefore, any functional dependency with A, E, BC, or CD on the left h and side of the arrow is in F + , no matter which other attributes appear in the FD. Allow * to represent any set of attributes in R, then F + is BD → B, BD → D, C → C, D → D, BD → BD, B → D, B → B, B → BD, and all FDs of the form A ∗→ α, BC ∗→ α, CD ∗→ α, E ∗→ α where α is any subset of {A, B, C, D, E}.

The candidate keys are **A**, **BC**, **CD**, and **E.**

**Question-04: Define closure set. Compute four members of $F^+$ from the relation $R = \{A, B, C, G, H, I\}$ and the functional dependency set $F = \{A \rightarrow B, A \rightarrow C, CG \rightarrow H, CG \rightarrow I, B \rightarrow H\}$. 5 Marks CSE-2011**

**<u>Closure Set:</u>**

Given a relational schema R, a functional dependency f on R is logically implied by a set of functional dependencies F on R if every relation instance r(R) that satisfies F also satisfies f. Let F be a set of functional dependencies. The closure of F, denoted by $F^+$, is the set of all functional dependencies logically implied by F.

Let us consider a relation schema R =(A, B, C, G, H, I) and the set F of functional dependencies {A → B, A → C, CG → H, CG → I, B → H}. We list several members of $F^+$ here:

- **A → H.**
- **CG → HI .**
- **AG → I.**

Then the closure set is the set of above functional dependencies logically implied by F.

**<u>Second Part:</u>**
Given relation R =(A, B, C, G, H, I) and the set F of functional dependencies {A → B, A → C, CG → H, CG → I, B → H}. We list four members of F + here:

• **A → H.** Since A → B and B → H hold, we apply the transitivity rule. It was much easier to use Armstrong's axioms to show that A → H holds than it was to argue directly from the definitions, as we did earlier in this section.

• **CG → HI.** Since CG → H and CG → I, the union rule implies that CG → HI .

• **AG → I.** Since A → C and CG → I, the pseudo transitivity rule implies that AG → I holds.

**Question-05: Define Canonical Cover. Compute the canonical cover of the following of the following set $F = \{A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C\}$ of functional dependencies for relational schema $R = \{A, B, C\}$. 5 Marks CSE-2011**

**<u>Canonical Cover:</u>**

Suppose that a set of functional dependencies on a relation schema R is F. A canonical cover $F_C$ for F is a set of dependencies such that F logically implies all dependencies in $F_C$, and $F_C$ logically implies all dependencies in F. Furthermore, $F_C$ must have the following properties:

- No functional dependency in Fc contains an extraneous attribute.
- Each left side of a functional dependency in Fc is unique. That is, there are no two dependencies α1 → β1 and α2 → β2 in Fc such that α1 = α2 .

**<u>Second Part:</u>**
Consider the following set F of functional dependencies on schema (A, B, C):

A → BC

$$B \rightarrow C$$
$$A \rightarrow B$$
$$AB \rightarrow C$$

Let us compute the canonical cover for F.

- There are two functional dependencies with the same set of attributes on the left side of the arrow:

$$A \rightarrow BC$$
$$A \rightarrow B$$

We combine these functional dependencies into A → BC.

- A is extraneous in AB → C because F logically implies (F −{AB → C}) ∪ {B → C}. This assertion is true because B → C is already in our set of functional dependencies.
- C is extraneous in A → BC, since A → BC is logically implied by A → B and B → C.

Thus, our canonical cover is

$$\mathbf{A \rightarrow B}$$
$$\mathbf{B \rightarrow C}$$

**Question-06: What is Boyce-Codd Normal Form? Write the algorithm for decomposing a relation schema R into BCNF. 4 Marks CSE-2012 (Engg) CSE-2002**

**<u>Boyce-Codd Normal Form:</u>**

A relation schema R is in BCNF with respect to a set F of functional dependencies if, for all functional dependencies in F + of the form α → β,where α ⊆ R and β ⊆ R, at least one of the following holds:

- α → β is a trivial functional dependency (that is, β ⊆ α).
- α is a super key for schema R.

A database design is in BCNF if each member of the set of relation schemas that constitutes the design is in BCNF.

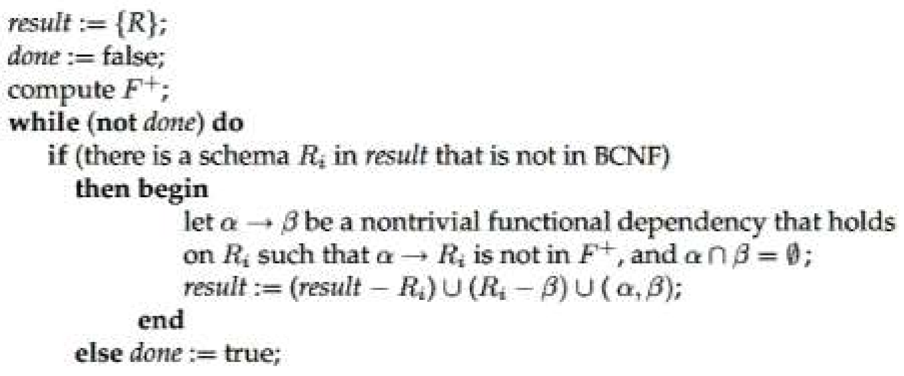
**<u>Example:</u>**

Consider the following relation schemas and their respective functional dependencies:

- *Customer-schema = (customer-name, customer-street, customer-city)*
  *customer-name → customer-street customer-city*
- *Branch-schema = (branch-name, assets, branch-city)*
  *branch-name → assets branch-city*
- *Loan-info-schema = (branch-name, customer-name, loan-number, amount)*
  *loan-number → amount branch-name*

Here, a candidate key for the schema is **customer-name**. The only nontrivial functional dependencies that hold on Customer-schema have customer-name on the left side of the arrow. Since customer-name is a candidate key, functional dependencies with customer-name on the left side do not violate the definition of BCNF. Hence **Customer-schema** is in BCNF. The schema Loan-info-schema, however, is not in BCNF.

**<u>Algorithm for decomposing a relation schema R into BCNF:</u>**



```
result := {R};
done := false;
compute $F^+$;
while (not done) do
    if (there is a schema $R_i$ in result that is not in BCNF)
      then begin
            let $\alpha \rightarrow \beta$ be a nontrivial functional dependency that holds
            on $R_i$ such that $\alpha \rightarrow R_i$ is not in $F^+$, and $\alpha \cap \beta = \emptyset$;
            result := (result − $R_i$) ∪ ($R_i - \beta$) ∪ ($\alpha, \beta$);
         end
      else done := true;
```

**Figure 7.13** BCNF decomposition algorithm.

**Question-07: Define Functional Dependency. What are Armstrong's axioms? Define Keys in terms of Functional Dependencies. 5 Marks CSE-2013 (Engg) CSE-2000**

**<u>Functional Dependency:</u>**

**A functional dependency (FD) is a type of Integrity Constraint that generalizes the concept of a key** and plays a key role in differentiating good database designs from bad database designs. Functional dependencies are constraints on the set of legal relations. They allow us to express facts about the enterprise that we are modelling with our database.

When we design a relational database, we first list those functional dependencies that must always hold.

**Example:** Let us consider following schema for a banking enterprise:

*Branch-schema =(branch-name, branch-city, assets)*
*Customer-schema =(customer-name, customer-street, customer-city)*
*Loan-schema =(loan-number, branch-name, amount)*
*Borrower-schema =(customer-name, loan-number)*
*Account-schema = (account-number, branch-name, balance)*
*Depositor-schema = (customer-name, account-number)*

For the schema mentioned above, our list of dependencies includes the following:

- **On Branch-schema:**
  branch-name → branch-city
  branch-name → assets
- **On Customer-schema:**
  customer-name → customer-city
  customer-name → customer-street
- **On Loan-schema:**
  loan-number → amount
  loan-number → branch-name
- **On Borrower-schema:**
  No functional dependencies
- **On Account-schema:**
  account-number → branch-name
  account-number → balance
- **On Depositor-schema:**
  No functional dependencies

**<u>Armstrong's axioms:</u>**

We can use the following three rules to find **logically implied** functional dependencies. By applying these rules repeatedly, we can find all of $F^+$ for a given set of functional dependencies, F. This collection of rules is called **Armstrong's axioms**:

- **Reflexivity rule:** If α is a set of attributes and β ⊆ α,then α → β holds.
- **Augmentation rule**: If α → β holds and γ is a set of attributes, then γα → γβ holds.
- **Transitivity rule:** If α → β holds and β → γ holds, then α → γ holds.

Armstrong's axioms are **sound**, because they do not generate any incorrect functional dependencies. They are **complete**, because, for a given set F of functional dependencies, they allow us to generate all $F^+$.

<u>**Super key in terms of Functional Dependencies:**</u>

Let R be a relation schema. A subset K of R is a **superkey of R** if, in any legal relation r(R), for all pairs t1 and t2 of tuples in r such that t1 ≠t2 ,then t1 [K] ≠ t2 [K]. That is, no two tuples in any legal relation r(R) may have the same value on attribute set K.

The notion of functional dependency generalizes the notion of superkey. Consider a relation schema R, and let α ⊆ R and β ⊆ R. The functional dependency α → β holds on schema R if, in any legal relation r(R), for all pairs of tuples t1 and t2 in r such that t1 [α]= t2 [α],it is also the case that t1 [β]= t2 [β]. Using the functional-dependency notation, we say that K is a superkey of R if K → R. Thatis, K is a superkey if, whenever t1 [K]= t2 [K], it is also the case that t1 [R]= t2 [R] (that is, t1 = t2 ). Functional dependencies allow us to express constraints that we cannot express with superkeys.

**Question-08: How can you define the closure of a set of Functional dependencies? 3.75 Marks CSE-2012 (Engg)**

<u>**Closure of a Set of Functional Dependencies:**</u>

Given a relational schema R, a functional dependency f on R is logically implied by a set of functional dependencies F on R if every relation instance r(R) that satisfies F also satisfies f. Let F be a set of functional dependencies. The closure of a set of functional dependencies F, denoted by $F^+$, is the set of all functional dependencies logically implied by F.

Let us consider a relation schema R =(A, B, C, G, H, I) and the set F of functional dependencies {A → B, A → C, CG → H, CG → I, B → H}. We list several members of $F^+$ here:

- **A → H.**
- **CG → HI .**
- **AG → I.**

Then the closure set is the set of such functional dependencies logically implied by F.

We can compute the closure of a set of functional dependencies, $F^+$as follows:

```
$F^+ = F$
repeat
    for each functional dependency $f$ in $F^+$
        apply reflexivity and augmentation rules on $f$
        add the resulting functional dependencies to $F^+$
    for each pair of functional dependencies $f_1$ and $f_2$ in $F^+$
        if $f_1$ and $f_2$ can be combined using transitivity
            add the resulting functional dependency to $F^+$
until $F^+$ does not change any further
```

**Figure 7.6** A procedure to compute $F^+$.

**Question-09: Explain First Normal Form (1NF), Second Normal Form (2NF) and Third Normal Form (3NF) with examples. 6 Marks CSE-2012 CSE-2010 CSE-2005 CSE-2003 CSE-2002 CSE-2000**

**First Normal Form:**

A domain is **atomic** if elements of the domain are considered to be indivisible units. We say that a relation schema R is in **first normal form** (1NF) if the domains of all attributes of R are **atomic.**
In other words, in First Normal Form, table must have at least one candidate key and make sure that the table don't have any duplicate record. In First Normal Form repeating groups are not allowed, that is no attributes which occur a different number of times on different records.

First Normal Form (1NF) sets the very basic rules for an organized database:

- Eliminate duplicative columns from the same table.
- Create separate tables for each group of related data and identify each row with a unique column (the primary key).

**Example:**

**Second Normal Form:**

**"A relation schema R is in 2NF if no non-prime attribute A is partially dependent on any candidate key for R."** A functional dependency α → β is called a **partial dependency** if there is a proper subset γ of α such that γ → β. We say that β is **partially dependent** on α. A relation schema R is in **second normal form** (2NF) if each attribute A in R meets one of the following criteria:

- It appears in a candidate key.
- It is not partially dependent on a candidate key.

The Second Normal Form can be achieved only when a table is in the 1NF. We can make the 2NF by eliminating the partial dependencies. As the First Normal Form deals with only the atomicity of data, but Second Normal Form deals with the relationships of tables like composite keys and non-key columns. In Second Normal Form subset of data is removed and is organized in separate tables. This process is applied to multiple rows of a table till the duplicity get reduced.

The general requirements of 2NF:

- Remove subsets of data that apply to multiple rows of a table and place them in separate tables.
- Create relationships between these new tables and their predecessors through the use of foreign keys.

**Example:**

**Third Normal Form:**

A relation schema R is in **third normal form** (3NF) with respect to a set F of functional dependencies if, for all functional dependencies in $F^+$ of the form α → β, where α ⊆ R and β ⊆ R, at least one of the following holds:

- α → β is a trivial functional dependency.
- α is a superkey for R.
- Each attribute A in β − α is contained in a candidate key for R.

There are two basic requirements for a database to be in third normal form:

- Already meet the requirements of both 1NF and 2NF
- Remove columns that are not fully dependent upon the primary key.

**Example:**

**Question-10: Define the following terms with examples: (a) Extraneous Attributes. CSE- 2012 CSE-2011 (b) Lossless Join Decomposition. CSE- 2012 CSE-2011 CSE-2003 (OLD) CSE-2002 (c) Functional Dependencies. CSE-2011 CSE-2000 (d) Multivalued Dependencies. CSE-2011**

**Extraneous Attribute:**

An attribute of a functional dependency is said to be **extraneous** if we can remove it without changing the closure of the set of functional dependencies. The formal definition of extraneous attributes is as follows.

Consider a set F of functional dependencies and the functional dependency $\alpha \rightarrow \beta$ in F.

1. Attribute A is extraneous in $\alpha$ if $A \in \alpha$,and F logically implies $(F - \{\alpha \rightarrow \beta\}) \cup \{(\alpha - A) \rightarrow \beta\}$.
2. Attribute A is extraneous in $\beta$ if $A \in \beta$, and the set of functional dependencies $(F - \{\alpha \rightarrow \beta\}) \cup \{\alpha \rightarrow (\beta - A)\}$ logically implies F.

**Example:**

For example, suppose we have the functional dependencies $AB \rightarrow C$ and $A \rightarrow C$ in F . Then, B is extraneous in $AB \rightarrow C$. As another example, suppose we have the functional dependencies $AB \rightarrow CD$ and $A \rightarrow C$ in F . Then C would be extraneous in the right-hand side of $AB \rightarrow CD$.

**Lossless Join Decomposition:**

Let R be a relation schema, and let F be a set of functional dependencies on R. Let R1 and R2 form a decomposition of R. This decomposition is a lossless-join decomposition of R if at least one of the following functional dependencies is in $F^+$ :

- $R1 \cap R2 \rightarrow R1$
- $R1 \cap R2 \rightarrow R2$

In other words, if $R1 \cap R2$ forms a superkey of either R1 or R2, the decomposition of R is a lossless-join decomposition.

**Example:**

Let is consider **Lending-schema** as follows:

***Lending-schema = (branch-name, branch-city, assets, customer-name, loan-number, amount)***

We decomposing Lending-schema into two schemas:

***Branch-schema = (branch-name, branch-city, assets)***

***Loan-info-schema = (branch-name, customer-name, loan-number, amount)***

Since branch-name → branch-city assets, the augmentation rule for functional dependencies implies that

***branch-name → branch-name branch-city assets***

Since Branch-schema ∩ Loan-info-schema = {branch-name}, it follows that our initial decomposition is a lossless-join decomposition.

**Functional Dependency:**

**A functional dependency (FD) is a type of Integrity Constraint that generalizes the concept of a key** and plays a key role in differentiating good database designs from bad database designs. Functional dependencies are constraints on the set of legal relations. They allow us to express facts about the enterprise that we are modelling with our database.

Formally, consider a relation schema R, and let $\alpha \subseteq R$ and $\beta \subseteq R$. The functional dependency

$$\alpha \rightarrow \beta$$

holds on schema R if, in any legal relation r(R), for all pairs of tuples t1 and t2 in **r** such that t1 [α]= t2 [α],it is also the case that t1 [β]= t2 [β].

When we design a relational database, we first list those functional dependencies that must always hold.

**Example:** Let us consider following schema for a banking enterprise:

***Branch-schema =(branch-name, branch-city, assets)***
***Customer-schema =(customer-name, customer-street, customer-city)***
***Loan-schema =(loan-number, branch-name, amount)***
***Borrower-schema =(customer-name, loan-number)***
***Account-schema = (account-number, branch-name, balance)***
***Depositor-schema = (customer-name, account-number)***

For the schema mentioned above, our list of dependencies includes the following:

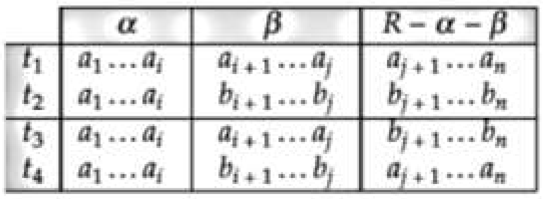
- **On Branch-schema:**
  branch-name → branch-city
  branch-name → assets
- **On Customer-schema:**
  customer-name → customer-city
  customer-name → customer-street
- **On Loan-schema:**
  loan-number → amount
  loan-number → branch-name
- **On Borrower-schema:**
  No functional dependencies
- **On Account-schema:**
  account-number → branch-name
  account-number → balance
- **On Depositor-schema:**
  No functional dependencies

**<u>Multivalued Dependencies:</u>**
Let R be a relation schema and let α ⊆ R and β ⊆ R.The multivalued dependency

$$\alpha \rightarrow\rightarrow \beta$$

holds on R if, in any legal relation r(R), for all pairs of tuples t1 and t2 in r such that t1 [α]= t2 [α], there exist tuples t3 and t4 in r such that

$$\begin{aligned}
&t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]\\
&t_3[\beta] = t_1[\beta]\\
&t_3[R-\beta] = t_2[R-\beta]\\
&t_4[\beta] = t_2[\beta]\\
&t_4[R-\beta] = t_1[R-\beta]
\end{aligned}$$



| | $\alpha$ | $\beta$ | $R-\alpha-\beta$ |
|---|---|---|---|
| $t_1$ | $a_1 \ldots a_i$ | $a_{i+1} \ldots a_j$ | $a_{j+1} \ldots a_n$ |
| $t_2$ | $a_1 \ldots a_i$ | $b_{i+1} \ldots b_j$ | $b_{j+1} \ldots b_n$ |
| $t_3$ | $a_1 \ldots a_i$ | $a_{i+1} \ldots a_j$ | $b_{j+1} \ldots b_n$ |
| $t_4$ | $a_1 \ldots a_i$ | $b_{i+1} \ldots b_j$ | $a_{j+1} \ldots a_n$ |

**Figure 7.16** Tabular representation of $\alpha \rightarrow\rightarrow \beta$.

From the definition of multivalued dependency, we can derive the following rule:

- **If α → β,then α →→ β.**

In other words, every functional dependency is also a multivalued dependency. Functional dependencies rule out certain tuples from being in a relation. If A → B, then we cannot have two tuples with the same A value but different B values. Multivalued dependencies, on the other hand, do not rule out the existence of certain tuples. Instead, they require that other tuples of a certain form be present in the relation. For this reason, functional dependencies sometimes are referred to as **equality-generating dependencies**, and multivalued dependencies are referred to as **tuple-generating dependencies**.

**Question-11: What are the design goals of a good relational database design? Is it always possible to achieve these goals? Justify your answer. 5 Marks CSE-2003 (OLD) CSE-2000**

**Design Goals of a Good Relational Database Design:**
There are three design goals of a good relational database design:

1. Lossless join
2. BCNF
3. Dependency preservation

It is not possible to achieve all three design goals. Consider the relation schema

**Banker-schema = (branch-name, customer-name, banker-name)**

which indicates that a customer has a "personal banker" in a particular branch. The set F of functional dependencies that we require to hold on the Banker-schema is

**banker-name → branch-name**
**branch-name customer-name → banker-name**

Clearly, Banker-schema is not in BCNF since banker-name is not a superkey. If we apply the algorithm of BCNF decomposition algorithm, we obtain the following BCNF decomposition:

**Banker-branch-schema = (banker-name, branch-name)**
**Customer-banker-schema = (customer-name, banker-name)**

The decomposed schemas preserve only banker-name → branch-name (and trivial dependencies), but the closure of {banker-name → branch-name} does not include customer-name branch-name → banker-name. The violation of this dependency cannot be detected unless a join is computed.
To see why the decomposition of Banker-schema into the schemas Banker-branch-schema and Customer-banker-schema is not dependency preserving, we apply the algorithm of TESTING Dependency preservation. We find that the restrictions F1 and F2 of F to each schema are:

**F1 = {banker-name → branch-name}**
**F2 = ∅ (only trivial dependencies hold on Customer-banker-schema)**

(For brevity, we do not show trivial functional dependencies.) It is easy to see that the dependency customer-name branch-name → banker-name is not in $(F1 \cup F2)^+$ even though it is in $F^+$. Therefore, $(F1 \cup F2)^+ \neq F^+$, and the decomposition is not depen-dency preserving.
This example demonstrates that not every BCNF decomposition is dependency preserving. Moreover, it is easy to see that any BCNF decomposition of Banker-schema must fail to preserve customer-name branch-name → banker-name. Thus, the example shows that we cannot always satisfy all three design goals:

**Question-12: What do you mean by the term "Decomposition" of a relation? What are the desirable properties of decomposition? Explain With Examples. 8 Marks CSE-2002 CSE-2000 CSE-2013 (Engg)**

**Decomposition of a Relation:**
Decomposition of a relation is process to decompose a relation schema that has many attributes into several schemas with fewer attributes.
Let R be a relation schema. A set of relation schemas $\{R_1, R_2, \cdots, R_n\}$ is a decomposition of R if

$$R = R_1 \cup R_2 \cup \cdots \cup R_n$$

That is, $\{R_1, R_2, \cdots, R_n\}$ is a decomposition of R if, for i =1, 2,...,n, each $R_i$ is a subset of R, and every attribute in R appears in at least one $R_i$.

We illustrate our concepts with the Lending-schema schema:

**Lending-schema = (branch-name, branch-city, assets, customer-name, loan-number, amount)**

We decompose it to the following three relations:

**Branch-schema = (branch-name, branch-city, assets)**
**Loan-schema = (loan-number, branch-name, amount)**
**Borrower-schema = (customer-name, loan-number)**

**Desirable properties of decomposition:**
Decomposition has several desirable properties:
1. Lossless-Join Decomposition
2. Dependency Preservation

3. Repetition of Information

**Lossless-Join Decomposition:**

Let R be a relation schema, and let F be a set of functional dependencies on R. Let R1 and R2 form a decomposition of R. This decomposition is a lossless-join decomposition of R if at least one of the following functional dependencies is in $F^+$ :

- R1 ∩ R2 → R1
- R1 ∩ R2 → R2

In other words, if R1 ∩ R2 forms a superkey of either R1 or R2, the decomposition of R is a lossless-join decomposition.

**Example:**

Let is consider **Lending-schema** as follows:

***Lending-schema = (branch-name, branch-city, assets, customer-name, loan-number, amount)***

We decomposing Lending-schema into two schemas:

***Branch-schema = (branch-name, branch-city, assets)***

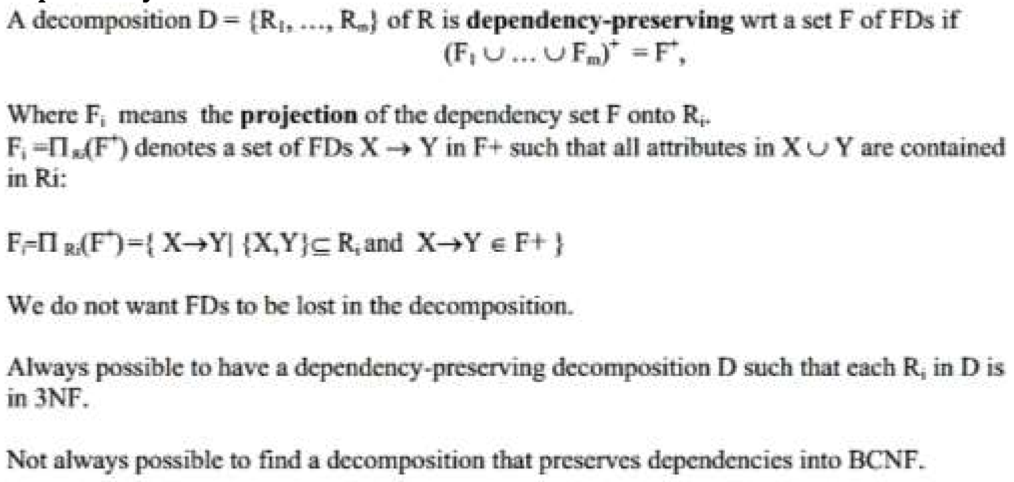
***Loan-info-schema = (branch-name, customer-name, loan-number, amount)***

Since branch-name → branch-city assets, the augmentation rule for functional dependencies implies that

***branch-name → branch-name branch-city assets***

Since Branch-schema ∩ Loan-info-schema = {branch-name}, it follows that our initial decomposition is a lossless-join decomposition.

**Dependency Preservation:**



A decomposition D = {$R_1$, …, $R_n$} of R is **dependency-preserving** wrt a set F of FDs if

$$(F_1 \cup \ldots \cup F_m)^+ = F^+,$$

Where $F_i$ means the **projection** of the dependency set F onto $R_i$.

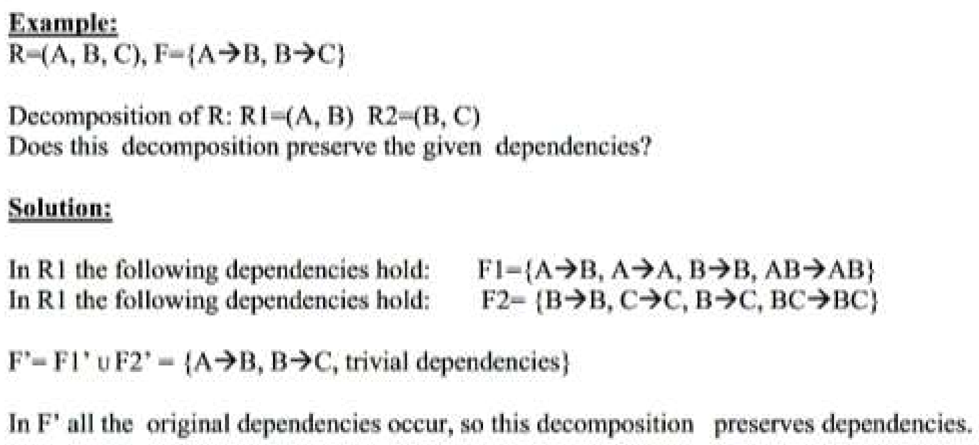
$F_i = \Pi_{Ri}(F^+)$ denotes a set of FDs X → Y in F+ such that all attributes in X ∪ Y are contained in Ri:

$F_i = \Pi_{Ri}(F^+) = \{ X \to Y \mid \{X,Y\} \subseteq R_i \text{ and } X \to Y \in F+ \}$

We do not want FDs to be lost in the decomposition.

Always possible to have a dependency-preserving decomposition D such that each $R_i$ in D is in 3NF.

Not always possible to find a decomposition that preserves dependencies into BCNF.



**Example:**

R=(A, B, C), F={A→B, B→C}

Decomposition of R: R1=(A, B) R2=(B, C)

Does this decomposition preserve the given dependencies?

**Solution:**

In R1 the following dependencies hold: F1={A→B, A→A, B→B, AB→AB}

In R1 the following dependencies hold: F2= {B→B, C→C, B→C, BC→BC}

F'= F1' ∪ F2' = {A→B, B→C, trivial dependencies}

In F' all the original dependencies occur, so this decomposition preserves dependencies.

**Repetition of Information:**

The decomposition of Lending-schema does not suffer from the problem of repetition of information. In Lending-schema, it was necessary to repeat the city and assets of a

branch for each loan. The decomposition separates branch and loan data into distinct relations, thereby eliminating this redundancy. Similarly, observe that, if a single loan is made to several customers, we must repeat the amount of the loan once for each customer (as well as the city and assets of the branch) in lending-schema. In the decomposition, the relation on schema Borrower-schema contains the loan-number, customer-name relationship, and no other schema does. Therefore, we have one tuple for each customer for a loan in only the relation on Borrower-schema. In the other relations involving loan-number (those on schemas Loan-schema and Borrower-schema), only one tuple per loan needs to appear.
Clearly, the lack of redundancy in our decomposition is desirable. The degree to which we can achieve this lack of redundancy is represented by several normal forms.

## Others important Topics:

**Non-prime attribute:** A non-prime attribute is an attribute that does not occur in any candidate key. Employee Address would be a non-prime attribute in the "Employees' Skills" table.

**Prime attribute:** A prime attribute, conversely, is an attribute that does occur in some candidate key.

**Objective/Goals of Normalization:**
The objectives of normalization beyond 1NF (First Normal Form) were stated as follows by Codd:

1. To free the collection of relations from undesirable insertion, update and deletion dependencies;
2. To reduce the need for restructuring the collection of relations, as new types of data are introduced, and thus increase the life span of application programs;
3. To make the relational model more informative to users;
4. To make the collection of relations neutral to the query statistics, where these statistics are liable to change as time goes by.

পরবর্তী অংশ থেকে শুরু হলো ডাটাবেজ সিস্টেম আর্কিটেকচার এর প্রশ্নোত্তর । ডাটাবেজ সিস্টেম আর্কিটেকচার থেকে যদিও ২০১৩ সালে অনুষ্ঠিতব্য বিএসসি (অনার্স) পার্ট-৩ এর পরীক্ষায় আসার মতো প্রশ্ন নেই বললেই চলে, তবুও কিছু প্রশ্ন যা সাধারনত এসে থাকে, সেগুলো এখানে সংযুক্ত করা হয়েছে । এই প্রশ্নসমূহ পড়া থাকলে, প্রশ্ন নিয়ে আর দুশ্চিন্তা করতে হবে না ।

**Question-01: What is Replication? Write some advantages and disadvantages of Data Replication. 2+4 Marks CSE-2011**

**Replication:**

Consider a relation r that is to be stored in the database. Replication is an approach to storing this relation in the distributed database in which the system maintains several identical replicas (copies) of the relation, and stores each replica at a different site. The alternative to replication is to store only one copy of relation r.

**Advantages and disadvantages of Data Replication:**

There are a number of advantages and disadvantages to replication:

- **Availability**:

If one of the sites containing relation r fail, then the relation r can be found in another site. Thus, the system can continue to process queries involving r, despite the failure of one site.

- **Increased parallelism:**

In the case where the majority of accesses to the relation r result in only the reading of the relation, then several sites can process queries involving r in parallel. The more replicas of r there are, the greater the chance that the needed data will be found in the site where the transaction is executing. Hence, data replication minimizes movement of data between sites.

- **Increased overhead on update**:

The system must ensure that all replicas of a relation **r** are consistent! Otherwise, erroneous computations may result. Thus, whenever r is updated, the update must be propagated to all sites containing replicas. The result is increased overhead. For example, in a banking system, where account information is replicated in various sites, it is necessary to ensure that the balance in a particular account agrees in all sites.

In general, replication enhances the performance of read operations and increases the availability of data to read-only transactions. However, update transactions incur greater overhead. Controlling concurrent updates by several transactions to replicated data is more complex than in centralized systems.

**Question-02: What are the main reasons of failure of a Distributed Transaction? 4 Marks CSE-2011**

**Main reasons of failure of a Distributed Transaction:**

A distributed system may suffer from the same types of failure that a centralized system does (for example, **software errors**, **hardware errors**, or **disk crashes**). There are, however, additional types of failure with which we need to deal in a distributed environment. The basic failure types are

- **Failure of a site**
- **Loss of messages**
- **Failure of a communication link**
- **Network partition**

The loss or corruption of messages is always a possibility in a distributed system. The system uses transmission-control protocols, such as TCP /IP, to handle such errors.

However, if two sites A and B are not directly connected, messages from one to the other must be routed through a sequence of communication links. If a communication link fails, messages that would have been transmitted across the link must be rerouted. In some cases, it is possible to find another route through the network, so that the messages are able to reach their destination. In other cases, a failure may result in there being no connection between some pairs of sites.

A system is partitioned if it has been split into two (or more) subsystems, called partitions, that lack any connection between them.

**Question-03: Distinguish between homogenous and heterogeneous Distributed Database. 5 Marks CSE-2011**

<u>Homogeneous Distributed Database:</u>

**In a homogeneous distributed database system,** all sites have identical database-management system software, are aware of one another, and agree to cooperate in processing users' requests. In such a system, local sites surrender a portion of their autonomy in terms of their right to change schemas or database-management system software. That software must also cooperate with other sites in exchanging information about transactions, to make transaction processing possible across multiple sites.

<u>Heterogeneous Distributed Database:</u>

**In contrast, in a heterogeneous distributed database,** different sites may use different schemas, and different database-management system software. The sites may not be aware of one another, and they may provide only limited facilities for cooperation in transaction processing. The differences in schemas are often a major problem for query processing, while the divergence in software becomes a hindrance for processing transactions that access multiple sites.

**Question-04: Write the role of transaction manager and transaction coordinator for a distributed database. 5 Marks CSE-2011**

<u>Role of transaction manager and transaction coordinator:</u>

Each site has its own local transaction manager, whose function is to ensure the ACID properties of those transactions that execute at that site. The various transaction managers cooperate to execute global transactions. In a transaction system, each site contains two subsystems:

- The **transaction manager** manages the execution of those transactions (or sub- transactions) that access data stored in a local site. Each such transaction may be either a local transaction (that is, a transaction that executes at only that site) or part of a global transaction (that is, a transaction that executes at several sites).

- The **transaction coordinator** coordinates the execution of the various transactions (both local and global) initiated at that site.

The overall system architecture appears in Figure 22.1.

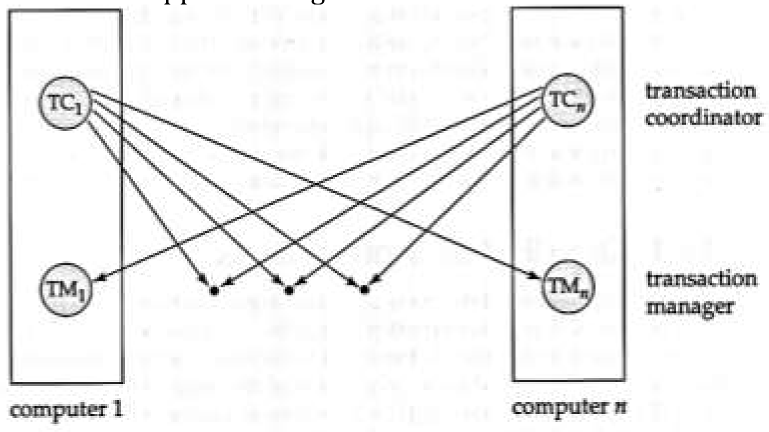


**Figure 22.1** System architecture.

The structure of a transaction manager is similar in many respects to the structure of a centralized system. Each transaction manager is responsible for:

- **Maintaining a log for recovery purposes**
- **Participating in an appropriate concurrency-control scheme to coordinate the concurrent execution of the transactions executing at that site**

We need to modify both the recovery and concurrency schemes to accommodate the distribution of transactions. The transaction coordinator subsystem is not needed in the centralized environment, since a transaction accesses data at only a single site. A transaction coordinator, as its name implies, is responsible for coordinating the execution of all the transactions initiated at that site. For each such transaction, the coordinator is responsible for:

- **Starting the execution of the transaction**
- **Breaking the transaction into a number of subtransactions and distributing these subtransactions to the appropriate sites for execution**
- **Coordinating the termination of the transaction, which may result in the trans- action being committed at all sites or aborted at all sites**

**Question-03: Explain how the following differ: Fragmentation Transparency, Replication Transparency and Location Transparency. 5 Marks CSE-2010 CSE-2003**

**<u>Transparency:</u>**

The user of a distributed databases system should not be required to know either where the data are physically located or how the data can be accessed at the specific local site. This characteristic is called **data transparency** and transparency can take several forms:

**<u>Fragmentation Transparency:</u>**

Users are not required to know how a relation has been fragmented.

**<u>Replication Transparency:</u>**

Users view each data object as logically unique. The distributed system may replicate an object to increase either system performance or data availability. Users do not have to be concerned with what data objects have been replicated, or where replicas have been placed.

**<u>Location Transparency:</u>**

Users are not required to know the physical location of the data. The distributed database system should be able to find any data as long as the data identifier is supplied by the user transaction.

**<u>Question:</u> Discuss the relative advantages of Centralized and Distributed Databases. 5 Marks CSE-2010**

**<u>Advantages of Centralized Distributed Database:</u>**

Here are some real advantages in a centralized database system:

1. **<u>Consistency:</u>** It's easier to avoid multiple listing of individual pieces of data-and the possible inconsistences so introduced-if a central programming group has control of all data entry.

2. **<u>Retrievability:</u>** <u>A central controlling group can make certain that all data entry follows a standard format and that the accessing of information also follows a standard and compatible format-all of which makes it very unlikely that any data will simply disappear.</u>

3. **<u>Data independence.</u>** User applications should be immune to possible changes in data storage and access methods. This immunity is easier to achieve if a central programming group is in control of all internal-level and conceptual-level storage and all user interfaces with the conceptual level of the system.

4. **<u>Security and Privacy:</u>** Under the centralized strategy, a large amount of valuable information is stored in one place. Initially it may seem that the centralized system poses a security liability by presenting a clear target, which once compromised, exposes everything to a would-be cyber intruder. However, with the proper investment in security technology and procedures, this central cache of information can be adequately protected against compromises to confidentiality, integrity, availability and privacy.

In the distributed strategy, it is a more complex matter to ensure that each information sharing participant provides sufficient protection. In general, the technology for protecting information and identifying authorized participants in a highly distributed network is not as mature as the technology used for centralized information organization and sharing.

**Advantages of Distributed Database:**
There are several reasons for building distributed database systems, including sharing of data, autonomy, and availability.

1. **Sharing data:** The major advantage in building a distributed database system is the provision of an environment where users at one site may be able to access the data residing at other sites.

2. **Autonomy:** The primary advantage of sharing data by means of data distribution is that each site is able to retain a degree of control over data that are stored locally. In a centralized system, the database administrator of the central site controls the database. In a distributed system, there is a global database administrator responsible for the entire system. A part of these responsibilities is delegated to the local database administrator for each site. Depending on the design of the distributed database system, each administrator may have a different degree of local autonomy. The possibility of local autonomy is often a major advantage of distributed databases.

3. **Availability:** If one site fails in a distributed system, the remaining sites may be able to continue operating. In particular, if data items are replicated in several sites, a transaction needing a particular data item may find that item in any of several sites. Thus, the failure of a site does not necessarily imply the shutdown of the system.

4. **Scalability:** Although some very large centralized systems are scalable, the distributed strategy has a clear advantages in the area of scalability. No centralized systems rivals the size of the ultimate example of distributed information sharing: the Internet. The growth pattern and rate of the Internet is also a testimony to the scalability of distributed information systems.

**Question: Mention at least three reasons for what we trend to distributed database system. 3 Marks CSE-CSE-2005**

**Three reasons for building distributed database system:**
There are several reasons for building distributed database systems, including sharing of data, autonomy, and availability.

There are several reasons for building distributed database systems, including sharing of data, autonomy, and availability.

- **Sharing data:** The major advantage in building a distributed database system is the provision of an environment where users at one site may be able to access the data residing at other sites.

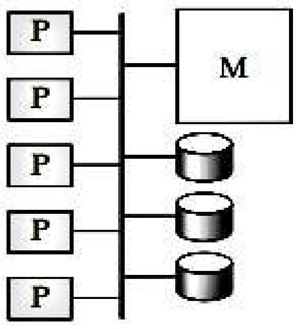
- **Autonomy:** The primary advantage of sharing data by means of data distribution is that each site is able to retain a degree of control over data that are stored locally. In a centralized system, the database administrator of the central site controls the database. In a distributed system, there is a global database administrator responsible for the entire system. A part of these responsibilities is delegated to the local database administrator for each site. Depending on the design of the distributed database system, each administrator may have a different degree of local autonomy. The possibility of local autonomy is often a major advantage of distributed databases.

- **Availability:** If one site fails in a distributed system, the remaining sites may be able to continue operating. In particular, if data items are replicated in several sites, a transaction needing a particular data item may find that item in any of several sites. Thus, the failure of a site does not necessarily imply the shutdown of the system.

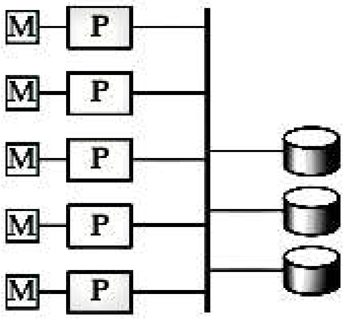
**Question: Discuss Parallel System architecture. 8 Marks CSE-2005**

**<u>Parallel Database Architectures</u>**
There are several architectural models for parallel machines. Among the most prominent ones are those in Figure 18.8 (in the figure, M denotes memory, P denotes a processor, and disks are shown as cylinders):

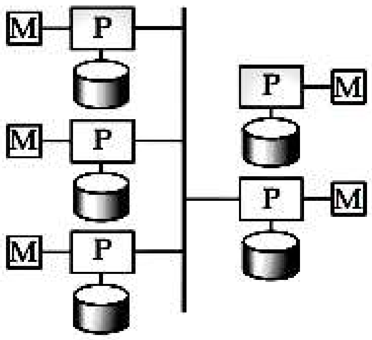
- **Shared memory.** All the processors share a common memory (Figure 18.8a).
- **Shared disk.** All the processors share a common set of disks (Figure 18.8b). Shared-disk systems are sometimes called clusters.
- **Shared nothing.** The processors share neither a common memory nor com-mon disk (Figure 18.8c).
- **Hierarchical.** This model is a hybrid of the preceding three architectures (Fig-ure 18.8d).



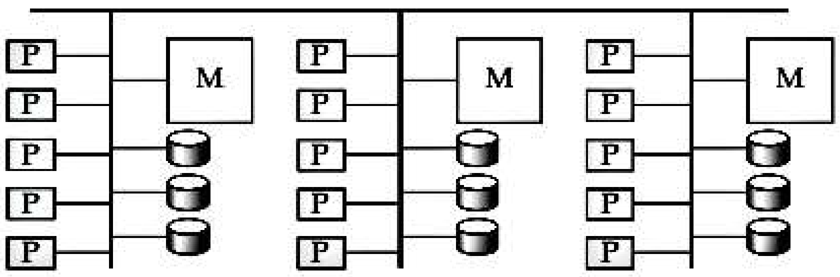
(a) shared memory



(b) shared disk



(c) shared nothing



(d) hierarchical

**Question: What is Data Fragmentation? Discuss Horizontal Fragmentation and Vertical Fragmentation. 4 Marks CSE-2003 CSE-2000 CSE-2000 (OLD)**

**<u>Data Fragmentation</u>**
If relation r is fragmented, r is divided into a number of fragments r1 , r2 ,...,rn. These fragments contain sufficient information to allow reconstruction of the original re-lation r. There are two different schemes for fragmenting a relation: horizontal frag-mentation and vertical fragmentation. Horizontal fragmentation splits the relation by assigning each tuple of r to one or more fragments. Vertical fragmentation splits the relation by decomposing the scheme R of relation r.

We shall illustrate these approaches by fragmenting the relation account, with the schema

**Account-schema =(account-number, branch-name, balance)**

## Horizontal fragmentation:

In **horizontal fragmentation**,a relation r is partitioned into a number of subsets, r1 , r2 ,...,rn. Each tuple of relation r must belong to at least one of the fragments, so that the original relation can be reconstructed, if needed.

As an illustration, the account relation can be divided into several different fragments, each of which consists of tuples of accounts belonging to a particular branch. If the banking system has only two branches—Hillside and Valleyview—then there are two different fragments:

$$\text{account1} = \sigma_{\text{branch-name} = \text{“Hillside”}}\ (\text{account})$$
$$\text{account2} = \sigma_{\text{branch-name} = \text{“Valleyview”}}\ (\text{account})$$

Horizontal fragmentation is usually used to keep tuples at the sites where they are used the most, to minimize data transfer. In general, a horizontal fragment can be defined as a selection on the global relation r. That is, we use a predicate Pi to construct fragment ri:

$$\mathbf{r}_i = \sigma_{\mathrm{Pi}}\ (\mathbf{r})$$

We reconstruct the relation r by taking the union of all fragments; that is,

$$\mathbf{r = r1 \cup r2 \cup \cdots \cup rn}$$

In our example, the fragments are disjoint. By changing the selection predicates used to construct the fragments, we can have a particular tuple of r appear in more than one of the ri.

## Vertical Fragmentation:

Vertical fragmentation is the same as decomposition. Vertical fragmentation of r(R) involves the definition of several subsets of attributes R1 ,R2 ,...,Rn of the schema R so that

$$\boldsymbol{R = R1 \cup R2 \cup \cdots \cup Rn}$$

Each fragment ri of r is defined by

$$\mathbf{r}_i = \boldsymbol{\Pi}_{\mathrm{Ri}}\ (\mathbf{r})$$

The fragmentation should be done in such a way that we can reconstruct relation r from the fragments by taking the natural join

$$\mathbf{r = r1 \bowtie r2 \bowtie r3 \cdots \bowtie rn}$$

One way of ensuring that the relation r can be reconstructed is to include the primary-key attributes of R in each of the Ri. More generally, any superkey can be used. It is often convenient to add a special attribute, called a tuple-id, to the schema R. The tuple-id value of a tuple is a unique value that distinguishes the tuple from all other tuples. The tuple-id attribute thus serves as a candidate key for the augmented schema, and is included in each of the Ris. The physical or logical address for a tuple can be used as a tuple-id, since each tuple has a unique address.

To illustrate vertical fragmentation, consider a university database with a relation employee-info that stores, for each employee, employee-id, name, designation, and salary. For privacy reasons, this relation may be fragmented into a relation employee-private-info containing employee-id and salary, and another relation employee-public-info containing attributes employee-id, name, and designation. These may be stored at different sites, again for security reasons.

The two types of fragmentation can be applied to a single schema; for instance, the fragments obtained by horizontally fragmenting a relation can be further partitioned vertically. Fragments can also be replicated. In general, a fragment can be replicated, replicas of fragments can be fragmented further, and so on.

# Important Question

23. What are the different states of transaction? Give some example of non-ACID transaction. **3+2 Marks CSE-2012 CSE-2002**
24. What are software RAID and Hardware RAID? How can you measure the performance of storage disk? **2+4 Marks CSE-2012**
25. Discuss shared mode lock, Exclusive Mode Lock and Compatible Lock. **6 Marks CSE-2012 CSE-2011 CSE-2007 CSE-2002**
26. What are main reasons of Starvation? How can you avoid starvation? **4 Marks CSE-2012 CSE-2007**
27. What are the different Techniques for Crash Recovery? Discuss the shadow paging technique of crash recovery. **7 Marks CSE-2012 CSE-2008 CSE-2005 CSE-2003 CSE-2000 CSE-2000 (OLD)**
28. Discuss Deadlock Detection Technique in database system. **3 Marks CSE-2012 CSE-2008 CSE-2003 CSE-2000 CSE-2000 (OLD)**
29. What do you mean by recovery system? Mention different types of log records. **4 Marks CSE-2012 CSE-2007**
30. Explain the use of checkpoint in log based recovery schema. **6 Marks CSE-2012 CSE-2007**
31. Briefly explain Time Stamp Based Locking Protocol. **10 Marks CSE-2011 CSE-2007**
32. Why concurrency control is necessary? **4 Marks CSE-2011 CSE-2010 CSE-2009 CSE-2004 CSE-2003 CSE-2003 (OLD) CSE-2002 CSE-2000**
33. Discuss different types of failure in database system. **6 Marks CSE-2010 CSE-2009 CSE-2004 CSE-2003 (OLD) CSE-2002 CSE-2000 CSE-2000 (OLD)**
34. Compare the Shadow paging Recovery scheme with log-based recovery schemes in terms of case of implementation and overhead cost. **10 Marks CSE-2010 CSE-2009 CSE-2004**
35. What is RAID? Discuss Different Levels of RAID. **8 Marks CSE-2010 CSE-2009 CSE-2008 CSE-2005 CSE-2013**
36. Discuss the factors that should be considered while choosing RAID level. **4 Marks CSE-2010 CSE-2009 CSE-2008**
37. What are the ways of optimizing records in files? **3 Marks CSE-2010**

38. Explain the ACID properties for transaction. **4 Marks CSE-2009 CSE-2008 CSE-2002 CSE-2012**
39. Describe Lock-Based Protocols for Transaction. **8 Marks CSE-2008**
40. Define Data Dictionary. What are the information that the system stores, about relations, users and others if any? **1+4 Marks CSE-2005**
41. What are the advantages and disadvantages of clustering file organization? **2 Marks CSE-2005**
42. Discuss the actions need to be taken to recover a database system from deadlock. **7 Marks CSE-2003 CSE-2003 (OLD) CSE-2002 CSE-2000**
43. What is transaction? What properties should maintain a database system to ensure data integrity transaction? Describe transaction states in brief. **7 Marks CSE-2002 CSE-2012**
44. Explain the terms seek time, rotational delay and transfer time. **1.75 Marks CSE-2013**
45. Discuss the functionalities of disk space manager of a DBMS. **3 Marks CSE-2013**
46. **Discuss the steps involved in processing a query. 3 Marks CSE-2007**

**Question-01: Explain Memory Hierarchy. 4 Marks CSE-2013**

**Memory Hierarchy:**

Memory in a computer system is arranged in a hierarchy, as shown in Figure 7.1. At the top, we have primary storage, which consists of cache and main memory, and provides very fast access to data. Then comes secondary storage, which consists of slower devices such as magnetic disks. Tertiary storage is the slowest class of storage devices; for example, optical disks and tapes. Currently, the cost of a given amount of main memory is about 100 times the cost of the same amount of disk space, and tapes are even less expensive than disks. Slower storage devices such as tapes and disks play an important role in database systems because the amount of data is typically very large. Since buying enough main memory to store all data is prohibitively expensive, we must store data on tapes and disks and build database systems that can retrieve data from lower levels of the memory hierarchy into main memory as needed for processing.

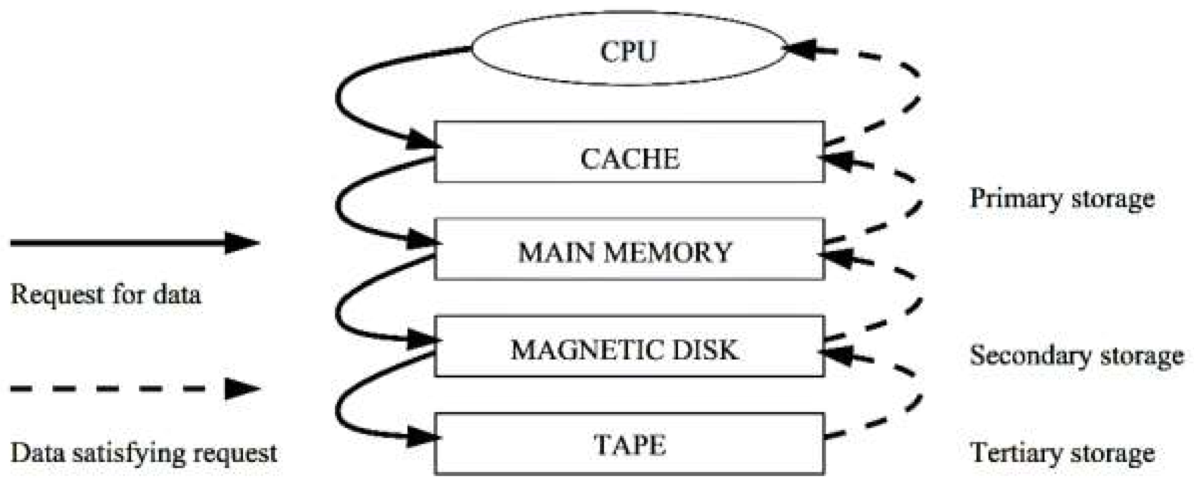


**Figure 7.1** The Memory Hierarchy

**Question-02: What are RAID, software RAID and Hardware RAID? How can you measure the performance of storage disk? 2+4 Marks CSE-2012**

**RAID:**

**RAID (originally** redundant array of inexpensive disks**; now commonly** redundant array of independent disks**) is a data** storage virtualization **technology that combines multiple** disk drive **components into a logical unit for the purposes of data redundancy and performance improvement.** Data is distributed across the drives in one of several ways, referred to as RAID levels, depending on the specific level of redundancy and performance required. The different schemes or architectures are named by the word RAID followed by a number (e.g. RAID 0, RAID 1). Each scheme provides a different balance between the key goals: reliability and availability, performance and capacity. RAID levels greater than RAID 0 provide protection against unrecoverable (sector) read errors, as well as whole disk failure.

অথবা

**RAID:**

RAID is stands for **redundant arrays of independent disks which is nothing but** disk organization techniques that manage a large numbers of disks, providing a view of a single disk of

- high capacity and high speed by using multiple disks in parallel, and
- high reliability by storing data redundantly, so that data can be recovered even if a disk fails.

A **disk array** is an arrangement of several disks, organized so as to increase performance and improve reliability of the resulting storage system. Performance is increased through data striping. Data striping distributes data over several disks to give the impression of having a single large, very fast disk. Reliability is improved through redundancy. Instead of having a single copy of the data, redundant information is maintained. The redundant information is carefully organized so that in case of a disk failure, it can be used to reconstruct the contents of the failed disk. Disk arrays that implement a combination of data striping and redundancy are called **redundant arrays of independent disks**, or in short, RAID.

**Software RAID:**

**RAID (originally** redundant array of inexpensive disks**; now commonly** redundant array of independent disks**) is a data** storage virtualization **technology that combines multiple** disk drive **components into a logical unit for the purposes of data redundancy and performance improvement. RAID can be implemented with no change at the hardware level, using only software modification. Such RAID implementations are called software RAID.** Software RAID implements the various RAID levels in the kernel disk (block device) code. It offers the cheapest possible solution, as expensive disk controller cards or hot-swap chassis are not required. Software RAID also works with cheaper IDE disks as well as SCSI disks. With today's fast CPUs, Software RAID performance can excel against Hardware RAID.
The MD driver in the Linux kernel is an example of a RAID solution that is completely hardware independent. The performance of a software-based array is dependent on the server CPU performance and load. Software RAID can be implemented in a variety of ways: 1) as a pure software solution, or 2) as a hybrid solution that includes some hardware designed to increase performance and reduce system CPU overhead.

**Hardware RAID:**

**RAID (originally** redundant array of inexpensive disks**; now commonly** redundant array of independent disks**) is a data** storage virtualization **technology that combines multiple** disk drive **components into a logical unit for the purposes of data redundancy and performance improvement. A RAID deployed using special purpose hardware is called Hardware RAID.**
A hardware RAID solution has its own processor and memory to run the RAID application. In this implementation, the RAID system is an independent small computer system dedicated to the RAID application, offloading this task from the host system. Hardware RAID can be found as an integral part of the solution (e.g. integrated in the motherboard) or as an add-in card. Hardware RAID can be implemented in a variety of ways: 1) as a discrete RAID Controller Card, or 2) as integrated hardware based on RAID-on-Chip technology.

**Measure the performance of storage disk:**

The main measures of the qualities of a disk are **capacity**, **access time**, **data-transfer rate**, and **reliability.**

1. **Access time: Access Time** is the time from when a read or write request is issued to when data transfer begins. To access (that is, to read or write) data on a given sector of a disk, the arm first must move so that it is positioned over the correct track, and then must wait for the sector to appear under it as the disk rotates. The time for repositioning the arm is called the **seek time**, and it increases with the distance that the arm must move. **Typical seek times range from 2 to 30 milliseconds, depending on how far the track is from the initial arm position. Smaller disks tend to have lower seek times since the head has to travel a smaller distance.**

   The **average seek time** is the average of the seek times, measured over a sequence of (uniformly distributed) random requests. The average seek time is around one-half of the maximum seek time. Average seek times currently range between 4 milliseconds and 10 milliseconds, depending on the disk model.

   Once the seek has started, the time spent waiting for the sector to be accessed to appear under the head is called the **rotational latency time**. Rotational speeds of disks today range from 5400 rotations per minute (90 rotations per second) up to 15,000 rotations per minute (250 rotations per second), or, equivalently, 4 milliseconds to 11.1 milliseconds per rotation. The average latency time of the disk is one-half the time for a full rotation of the disk.

The access time is then the sum of the seek time and the latency, and ranges from 8 to 20 milliseconds.

2. <u>Data Transfer Rate:</u> **The data-transfer rate is the rate at which data can be retrieved from or stored to the disk.** Current disk systems claim to support maximum transfer rates of about 25 to 40 megabytes per second, although actual transfer rates may be significantly less, at about 4 to 8 megabytes per second.

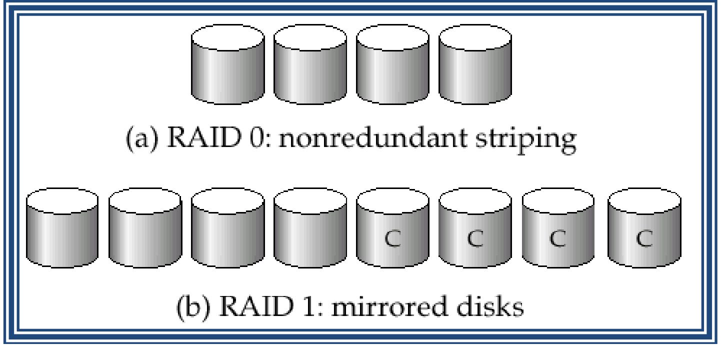
3. <u>Reliability:</u> The final commonly used measure of a disk is the **mean time to failure (MTTF),** which is a measure of the reliability of the disk. **The mean time to failure of a disk (or of any other system) is the amount of time that, on average, we can expect the system to run continuously without any failure.** According to vendors' claims, the mean time to failure of disks today ranges from 30,000 to 1,200,000 hours — about 3.4 to 136 years. Most disks have an expected life span of about 5 years, and have significantly higher rates of failure once they become more than a few years old.

**Question-03: What is RAID? Discuss Different Levels of RAID. 8 Marks CSE-2010 CSE-2009 CSE-2008 CSE-2005 CSE-2013**

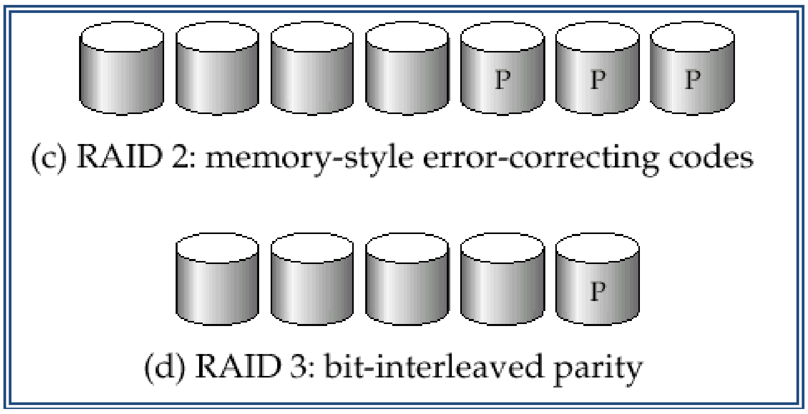
**<u>RAID:</u>**
**RAID (originally** redundant array of inexpensive disks**; now commonly** redundant array of independent disks**) is a data storage virtualization technology that combines multiple disk drive components into a logical unit for the purposes of data redundancy and performance improvement.** There are two possible RAID approaches: Hardware RAID and Software RAID.

**<u>Different Levels of RAID:</u>**
Depending on the specific level of redundancy and performance required, data is distributed across the drives in several ways. These ways or schemes are referred to as RAID levels. The different schemes or architectures are named by the word RAID followed by a number (e.g. RAID 0, RAID 1). Each scheme provides a different balance between the key goals: reliability and availability, performance and capacity. different RAID levels are described below.

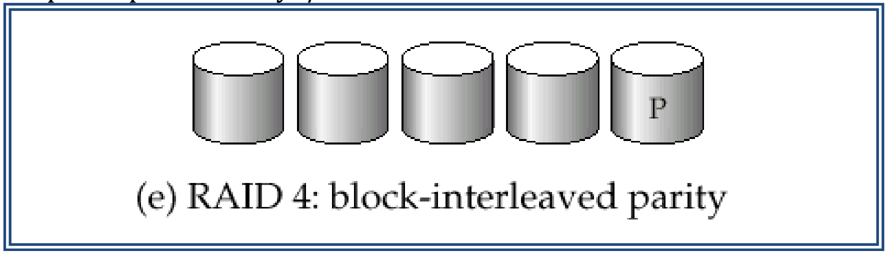
- **RAID Level 0**: Block striping; non-redundant.
  - Used in high-performance applications where data lost is not critical.



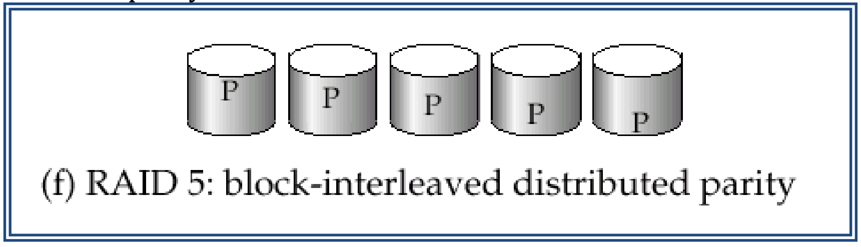
- **RAID Level 1**: Mirrored disks with block striping
  - Offers best write performance.
  - Popular for applications such as storing log files in a database system.
- **RAID Level 2**: Memory-Style Error-Correcting-Codes (ECC) with bit striping.


P
P
P
(c) RAID 2: memory-style error-correcting codes
P
(d) RAID 3: bit-interleaved parity

- **RAID Level 3**: Bit-Interleaved Parity
  - a single parity bit is enough for error correction, not just detection
    - ⊕ When writing data, corresponding parity bits must also be computed and written to a parity bit disk
    - ⊕ To recover data in a damaged disk, compute XOR of bits from other disks (including parity bit disk)
  - Faster data transfer than with a single disk, but fewer I/Os per second since every disk has to participate in every I/O.



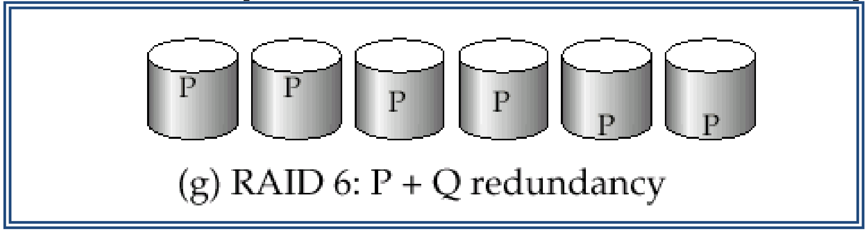
- **RAID Level 4:** Block-Interleaved Parity; uses block-level striping, and keeps a parity block on a separate disk for corresponding blocks from *N* other disks.
  - When writing data block, corresponding block of parity bits must also be computed and written to parity disk
  - To find value of a damaged block, compute XOR of bits from corresponding blocks (including parity block) from other disks.
  - Provides higher I/O rates for independent block reads than Level 3
    - ⊕ block read goes to a single disk, so blocks stored on different disks can be read in parallel
  - Before writing a block, parity data must be computed
    - ⊕ Can be done by using old parity block, old value of current block and new value of current block (2 block reads + 2 block writes)
    - ⊕ Or by recomputing the parity value using the new values of blocks corresponding to the parity block
      - n More efficient for writing large amounts of data sequentially
  - Parity block becomes a bottleneck for independent block writes since every block write also writes to parity disk



- **RAID Level 5:** Block-Interleaved Distributed Parity; partitions data and parity among all $N + 1$ disks, rather than storing data in *N* disks and parity in 1 disk.
  - E.g., with 5 disks, parity block for *n*th set of blocks is stored on disk ($n \bmod 5$) + 1, with the data blocks stored on the other 4 disks.
  - ⊕ Higher I/O rates than Level 4.

- Block writes occur in parallel if the blocks and their parity blocks are on different disks.

⊕ Subsumes Level 4: provides same benefits, but avoids bottleneck of parity disk.



- **RAID Level 6**: P+Q Redundancy scheme; similar to Level 5, but stores extra redundant information to guard against multiple disk failures.
  - Better reliability than Level 5 at a higher cost; not used as widely.

**Question-04: Discuss the factors that should be considered while choosing RAID level. 4 Marks CSE-2010 CSE-2009 CSE-2008**

**Factors that should be considered while choosing RAID level:**
The factors to be taken into account in choosing a RAID level are

- **Monetary cost of extra disk-storage requirements.**
- **Performance requirements in terms of number of I/O operations.**
- **Performance when a disk has failed**
- **Performance during rebuild (that is, while the data in a failed disk are being rebuilt on a new disk)**

- RAID 0 is used only when data safety is not important
  - l E.g. data can be recovered quickly from other sources
- Level 2 and 4 never used since they are subsumed by 3 and 5
- Level 3 is not used since bit-striping forces single block reads to access all disks, wasting disk arm movement
- Level 6 is rarely used since levels 1 and 5 offer adequate safety for most applications
- So competition is mainly between 1 and 5

- Level 1 provides much better write performance than level 5
  - Level 5 requires at least 2 block reads and 2 block writes to write a single block, whereas Level 1 only requires 2 block writes
  - Level 1 preferred for high update environments such as log disks
- Level 1 had higher storage cost than level 5
  - disk drive capacities increasing rapidly (50%/year) whereas disk access times have decreased much less (x 3 in 10 years)
  - I/O requirements have increased greatly, e.g. for Web servers
  - When enough disks have been bought to satisfy required rate of I/O, they often have spare storage capacity
    - So there is often no extra monetary cost for Level 1!
- Level 5 is preferred for applications with low update rate, and large amounts of data
- Level 1 is preferred for all other applications

**Question-05: What are the ways of organizing records in files? 3 Marks CSE-2010**

**Organization of Records in Files**
Several of the possible ways of organizing records in files are:

- **Heap file organization:**

**In Heap File organization,** any record can be placed anywhere in the file where there is space for the record. There is no ordering of records. Typically, there is a single file for each relation.

## Sequential file organization:

**A sequential file is designed for efficient processing of records in sorted order based on some search-key. A search key is any attribute or set of attributes; it need not be the primary key, or even a superkey. In sequential file organization**, Records are stored in sequential order, according to the value of a "search key" of each record.

## Hashing file organization:

**In a hash file organization, we obtain the address of the disk block containing a desired record directly by computing a function on the search-key value of the record. Let K denote the set of all search-key values, and let B denote the set of all bucket addresses. A hash function h is a function from K to B. In hashing file organization,** A hash function is computed on some attribute of each record. The result of the hash function specifies in which block of the file the record should be placed.

## Multitable clustering file organization:

**A clustering file organization is a file organization that stores related records of two or more relations in each block. Such a file organization allows us to read records that would satisfy the join condition by using one block read. Thus, we are able to process this particular query more efficiently. In Multitable clustering file organization,** records of several different relations are stored in the same file; further, related records of the different relations are stored on the same block, so that one I/O operation fetches related records from all the relations. For example, records of the two relations can be considered to be related if they would match in a join of the two relations.

**Question-06: What is Query Processing? Discuss the steps involved in processing a query. 5 Marks CSE-2007**

## Query Processing:

**Query processing refers to the range of activities involved in extracting data from a database.** The activities include translation of queries in high-level database languages into expressions that can be used at the physical level of the file system, a variety of query-optimizing transformations, and actual evaluation of queries.

## Steps involved in Processing a Query:

The steps involved in processing a query appear in Figure 13.1. The basic steps are:

1. Parsing and translation
2. Optimization
3. Evaluation

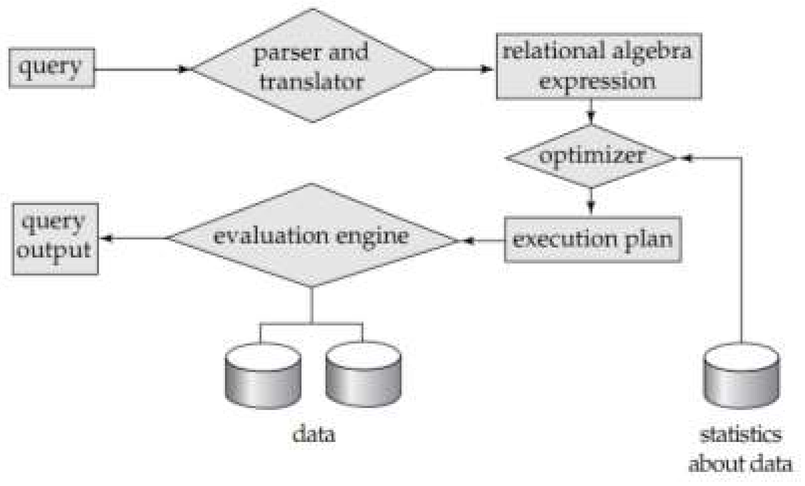


**Figure 13.1** Steps in query processing.

<u>**Parsing and Translation:**</u> Before query processing can begin, the system must translate the query into a usable form. A language such as SQL is suitable for human use, but is ill-suited to be the system's internal representation of a query. A more useful internal representation is one based on the extended relational algebra. Thus, the first action the system must take in query processing is to **translate** a given query into its internal form. This translation process is similar to the work performed by the parser of a compiler. In generating the internal form of the query, the parser checks the syntax of the user's query, verifies that the relation names appearing in the query are names of the relations in the database, and so on. The system constructs a parse-tree representation of the query, which it then translates into a relational-algebra expression. If the query was expressed in terms of a view, the translation phase also replaces all uses of the view by the relational-algebra expression that defines the view.

<u>**Optimization:**</u> **Query optimization is the process of selecting the most efficient query-evaluation plan from among the many strategies usually possible for processing a given query, especially if the query is complex.** We do not expect users to write their queries so that they can be processed efficiently. Rather, we expect the system to construct a query-evaluation plan that minimizes the cost of query evaluation. This is where query optimization comes into play.

<u>**Evaluation:**</u> To specify fully how to evaluate a query, we need not only to provide the relational-algebra expression, but also to annotate it with instructions specifying how to evaluate each operation. Annotations may state the algorithm to be used for a specific operation, or the particular index or indices to use. A relational-algebra operation annotated with instructions on how to evaluate it is called an **evaluation primitive**. A sequence of primitive operations that can be used to evaluate a query is a query-execution plan or **query-evaluation plan**. The query-execution engine takes a query-evaluation plan, executes that plan, and returns the answers to the query. It is the responsibility of the system to construct a query-evaluation plan that minimizes the cost of query evaluation. Once the query plan is chosen, the query is evaluated with that plan, and the result of the query is output.

**Question-07: What is Transaction? Explain the ACID properties for transaction. 5 Marks CSE-2009 CSE-2008 CSE-2002 CSE-2012**

<u>**Transaction:**</u>

**Collections of operations that form a single logical unit of work are called transactions. A transaction is a unit of program execution that accesses and possibly updates various data items.** Usually, a transaction is initiated by a user program written in a high-level data-manipulation language or programming language (for example, SQL, COBOL, C, C++, or Java), where it is delimited by statements (or function calls) of the form begin transaction and end transaction. The transaction consists of all operations executed between the begin transaction and end transaction.
**Example:** For instance, transfer of money from one account to another is a transaction consisting of two updates, one to each account.

<u>**ACID properties for transaction:**</u>
To ensure integrity of the data, we require that the database system maintain the following properties of the transactions:

- **Atomicity:** It is important that either all actions of a transaction be executed completely, or, in case of some failure, partial effects of a transaction be undone. This property is called atomicity i.e. **either all operations of the transaction are reflected properly in the database, or none are.**

- **Consistency:** Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the **consistency** of the database.

- **Isolation**: Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions $T_i$ and $T_j$ , it appears to $T_i$ that either $T_j$

finished execution before $T_i$ started, or $T_j$ started execution after $T_i$ finished. Thus, each transaction is unaware of other transactions executing concurrently in the system. Thus, Database systems must provide mechanisms to isolate transactions from the effects of other concurrently executing transactions. This property is called **isolation**. The isolation property of a transaction ensures that the concur-rent execution of transactions results in a system state that is equivalent to a state that could have been obtained had these transactions executed one at a time in some order.

l **Durability:** After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures. This property is called **durability**.

These properties are often called the **ACID** properties; the acronym is derived from the first letter of each of the four properties.

**Question-08: What are the different states of transaction? Give some example of non-ACID transaction. 3+2 Marks CSE-2012 CSE-2002**

**Different states of transaction:**

**A transaction is a unit of program execution that accesses and possibly updates various data items.** A transaction must be in one of the following states:

- **Active,** the initial state; the transaction stays in this state while it is executing.
- **Partially committed,** after the final statement has been executed.
- **Failed,** after the discovery that normal execution can no longer proceed.
- **Aborted: In the absence of failures, all transactions complete successfully. However, as we noted earlier, a transaction may not always complete its execution successfully. Such a transaction is termed aborted.** If we are to ensure the atomicity property, an aborted transaction must have no effect on the state of the database. Thus, any changes that the aborted transaction made to the database must be undone. Once the changes caused by an aborted transaction have been undone, we say that the transaction has been **rolled back.** After the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

- **Committed:** A transaction that completes its execution successfully is said to be **committed.** A committed transaction that has performed updates transforms the database into a new consistent state, which must persist even if there is a system failure. Once a transaction has committed, we cannot undo its effects by aborting it. The only way to undo the effects of a committed transaction is to execute a compensating transaction. The transaction will be in committed state, after successful completion.

The state diagram corresponding to a transaction appears in Figure 15.1.

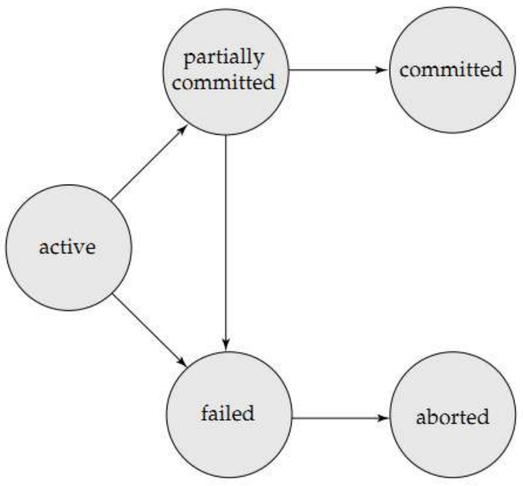


**Figure 15.1** State diagram of a transaction.

A transaction starts in the **active state.** When it finishes its final statement, it enters the **partially committed state.** At this point, the transaction has completed its execution, but it is still possible that it may have to be **aborted,** since the actual output may still be temporarily residing in main memory, and thus a hardware failure may preclude its successful completion. The database system then writes out enough information to disk that, even in the event of a failure, the updates performed by the transaction can be re-created when the system restarts after the failure. When the last of this information is written out, the transaction enters the **committed state.**

**Example of non-ACID transaction:**

Suppose that, just before the execution of transaction Ti the values of accounts A and B are \$1000 and \$2000, respectively. Now suppose that, during the execution of transaction Ti, a failure occurs that prevents Ti from completing its execution successfully. Examples of such failures include power failures, hardware failures, and software errors. Further, suppose that the failure happened after the write(A) operation but before the write(B)operation. In this case, the values of accounts A and B reflected in the database are \$950 and \$2000. The system destroyed \$50 as a result of this failure. In particular, we note that the sum A + B is no longer preserved.

**Question-09: Discuss shared mode lock, Exclusive Mode Lock, Compatible Lock and Binary Lock. 8 Marks CSE-2012 CSE-2011 CSE-2007 CSE-2002**

One way to ensure serializability is to require that data items be accessed in a mutually exclusive manner; that is, while one transaction is accessing a data item, no other transaction can modify that data item. The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a lock on that item. There are various modes in which a data item may be locked:

1. Shared Mode Lock
2. Exclusive Mode Lock
3. Compatible Lock.
4. Binary Lock

**Shared Mode Lock:**

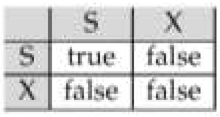
The shared lock is also called as a Read Lock. The intention of this mode of locking is to ensure that the data item does not undergo any modifications while it is locked in this mode. This mode allows several transactions to access the same item Q if they all access Q for reading purpose only. Thus, any number of transaction can concurrently lock and access a data item in the shared mode, but none of these transaction can modify the data item. A data item locked in the shared mode cannot be locked in the exclusive mode until the shared lock is released by all the transaction holding the lock. A data item locked in the exclusive mode cannot be locked in the shared mode until the exclusive lock on the data item is released. If a transaction Ti has obtained a shared-mode lock (denoted by S) on item Q, then Ti can read, but cannot write, Q. A transaction requests a shared lock on data item Q by executing the $lock\ - S\ (Q)$ instruction.

**Exclusive Mode Lock:**

This mode of locking provides an exclusive use of data item to one particular transaction. The exclusive mode of locking is also called an UPDATE or a WRITE lock. If a transaction Ti has obtained an exclusive-mode lock (denoted by X) on item Q, then Ti can both read and write Q, no other transaction can access Q, not even to Read Q, until the lock is released by transaction Ti. A transaction requests an exclusive lock through the $lock\ - X\ (Q)$ instruction.

**Compatible Lock:**

Given a set of lock modes, we can define a compatibility function on them as follows. Let A and B represent arbitrary lock modes. Suppose that a transaction Ti requests a lock of mode A on item Q on which transaction Tj (Ti ≠Tj ) currently holds a lock of mode B. If transaction Ti can be granted a lock on Q immediately, in spite of the presence of the mode B lock, then we say mode A is **compatible** with mode B. Such a function can be represented conveniently by a matrix. The compatibility relation between the two modes of locking discussed in this section appears in the matrix comp of Figure 16.1. An element comp(A, B) of the matrix has the value true if and only if mode A is compatible with mode B. shared mode is compatible with shared mode, but not with exclusive mode. At any time, several shared-mode locks can be held simultaneously (by different transactions) on a particular data item. A subsequent exclusive-mode lock request has to wait until the currently held shared-mode locks are released.



| | S | X |
|---|---|---|
| S | true | false |
| X | false | false |

**Figure 16.1** Lock-compatibility matrix comp.

**Binary Lock:**

A Binary Lock can have two states or values locked and unlocked (1 or 0 for simplicity). Two operations lock item and unlock item are used with binary locking. A transaction requests a lock by issuing a lock item (X) operation. If Lock(X) =1 the transaction is forced to wait. If Lock(X) = 0, it is set to 1 (the transaction locks the item) and the transaction is allowed to access item X. When the transaction finishes using item X, it issues an Unlock_Item (X) operation, which sets Lock (X) to 0, so that X may be accessed by other transactions. Hence, a binary lock enforces mutual exclusion on the data item. When a binary locking scheme is used, every transaction must obey the following rules:

1. The transaction 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) 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.

**Question-10: What do you mean by starvation? What are main reasons of Starvation? How can you avoid starvation? 4 Marks CSE-2012 CSE-2007**

**Starvation:**

Suppose a transaction T2 has a shared-mode lock on a data item, and another transaction $T_1$ requests an exclusive-mode lock on the data item. Clearly, $T_1$ has to wait for T2 to release the shared-mode lock. Meanwhile, a transaction T3 may request a shared-mode lock on the same data item. The lock request is compatible with the lock granted to T2, so T3 may be granted the shared-mode lock. At this point T2 may release the lock, but still T1 has to wait for T3 to finish. But again, there may be a new transaction T4 that requests a shared-mode lock on the same data item, and is granted the lock before T3 releases it. In fact, it is possible that there is a sequence of transactions that each requests a shared-mode lock on the data item, and each transaction releases the lock a short while after it is granted, but T1 never gets the exclusive-mode lock on the data item. The transaction $T_1$ may never make progress, and $T_1$ is said to be **starved**. **This problem encountered in concurrent transaction, where a transaction is perpetually denied necessary data item or a transaction has to wait for a long time and may not get the lock on a desired item is called starvation**. Without the requested data item, the transaction can never finish its task.

**Avoidance of starvation:**

We can avoid starvation of transactions by granting locks in the following manner:
When a transaction Ti requests a lock on a data item Q in a particular mode M, the concurrency-control manager grants the lock provided that

1. There is no other transaction holding a lock on Q in a mode that conflicts with M.

2. There is no other transaction that is waiting for a lock on Q, and that made its lock request before Ti.

Thus, a lock request will never get blocked by a lock request that is made later.

**Question-11: Why concurrency control is necessary? 4 Marks CSE-2011 CSE-2010 CSE-2009 CSE-2004 CSE-2003 CSE-2003 (OLD) CSE-2002 CSE-2000**

**Why is concurrency control needed:**

If transactions are executed *serially*, i.e., sequentially with no overlap in time, no transaction concurrency exists. However, if concurrent transactions with interleaving operations are allowed in an uncontrolled manner, some unexpected, undesirable result may occur. Here are some typical examples:

1. **The lost update problem:** A second transaction writes a second value of a data-item (datum) on top of a first value written by a first concurrent transaction, and the first value is lost to other transactions running concurrently which need, by their precedence, to read the first value. The transactions that have read the wrong value end with incorrect results.

2. **The dirty read problem:** Transactions read a value written by a transaction that has been later aborted. This value disappears from the database upon abort, and should not have

been read by any transaction ("dirty read"). The reading transactions end with incorrect results.

3. **The incorrect summary problem:** While one transaction takes a summary over the values of all the instances of a repeated data-item, a second transaction updates some instances of that data-item. The resulting summary does not reflect a correct result for any (usually needed for correctness) precedence order between the two transactions (if one is executed before the other), but rather some random result, depending on the timing of the updates, and whether certain update results have been included in the summary or not.

Most high-performance transactional systems need to run transactions concurrently to meet their performance requirements. Thus, without concurrency control such systems can neither provide correct results nor maintain their databases consistent. That's why Concurrency control is necessary.

**Question-12: Describe Lock-Based Protocols for Transaction. 8 Marks CSE-2008**

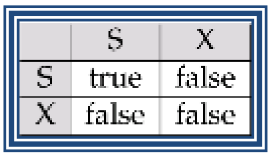
**Lock Based Protocols for Transaction:**
One way to ensure serializability is to require that data items be accessed in a mutually exclusive manner; that is, while one transaction is accessing a data item, no other transaction can modify that data item. The most common method used to implement this requirement is to allow a transaction to access a data item only if it is currently holding a lock on that item.

**Locks:**
There are various modes in which a data item may be locked. In this section, we restrict our attention to two modes:

1. **Shared:** If a transaction Ti has obtained a shared-mode lock (denoted by S) on item Q, then Ti can read, but cannot write, Q.
2. **Exclusive:** If a transaction Ti has obtained an exclusive-mode lock (denoted by X) on item Q, then Ti can both read and write Q.

n Lock requests are made to concurrency-control manager. Transaction can proceed only after request is granted.

n **Lock-compatibility matrix**



| | S | X |
|---|---|---|
| S | true | false |
| X | false | false |

n A transaction may be granted a lock on an item if the requested lock is compatible with locks already held on the item by other transactions

n Any number of transactions can hold shared locks on an item,

  l but if any transaction holds an exclusive on the item no other transaction may hold any lock on the item.

n If a lock cannot be granted, the requesting transaction is made to wait till all incompatible locks held by other transactions have been released. The lock is then granted.

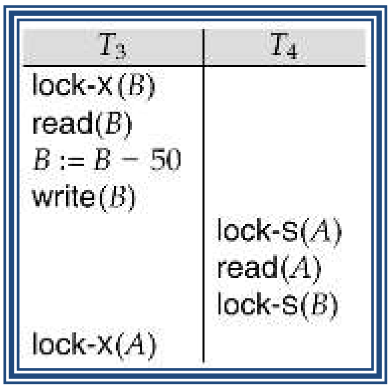
n Example of a transaction performing locking:

```
T2:     lock-S(A);
        read (A);
        unlock(A);
        lock-S(B);
        read (B);
        unlock(B);
        display(A+B)
```

n Locking as above is not sufficient to guarantee serializability — if *A* and *B* get updated in-between the read of *A* and *B*, the displayed sum would be wrong.

n A **locking protocol** is a set of rules followed by all transactions while requesting and releasing locks. Locking protocols restrict the set of possible schedules.

n Consider the partial schedule



| $T_3$ | $T_4$ |
|---|---|
| lock-X($B$) | |
| read($B$) | |
| $B := B - 50$ | |
| write($B$) | |
| | lock-S($A$) |
| | read($A$) |
| | lock-S($B$) |
| lock-X($A$) | |

n Neither $T_3$ nor $T_4$ can make progress — executing **lock-S**$(B)$ causes $T_4$ to wait for $T_3$ to release its lock on $B$, while executing **lock-X**$(A)$ causes $T_3$ to wait for $T_4$ to release its lock on $A$.

n Such a situation is called a **deadlock**.

l To handle a deadlock one of $T_3$ or $T_4$ must be rolled back and its locks released.

n The potential for deadlock exists in most locking protocols. Deadlocks are a necessary evil.

**<u>Granting of Locks:</u>**

When a transaction requests a lock on a data item in a particular mode, and no other transaction has a lock on the same data item in a conflicting mode, the lock can be granted.

n **Starvation** is also possible if concurrency control manager is badly designed. For example:

l A transaction may be waiting for an X-lock on an item, while a sequence of other transactions request and are granted an S-lock on the same item.

l The same transaction is repeatedly rolled back due to deadlocks.

We can avoid starvation of transactions by granting locks in the following manner: When a transaction Ti requests a lock on a data item Q in a particular mode M, the concurrency-control manager grants the lock provided that

1. There is no other transaction holding a lock on Q in a mode that conflicts with M.
2. There is no other transaction that is waiting for a lock on Q, and that made its lock request before Ti.

Thus, a lock request will never get blocked by a lock request that is made later

**Question-13: Briefly explain Time Stamp Based Locking Protocol. 10 Marks CSE-2011 CSE-2007**

**<u>Timestamp-Based Protocols:</u>**

A method for determining the serializability order is to select an ordering among transactions in advance. The most common method for doing so is to use a timestamp-ordering scheme. A timestamp-ordering scheme ensures serializability by selecting an ordering in advance between every pair of transactions. A unique fixed timestamp is associated with each transaction in the system. The timestamps of the transactions determine the serializability order.

**<u>Timestamps:</u>**

With each transaction Ti in the system, we associate a unique fixed timestamp, de-noted by TS(Ti). This timestamp is assigned by the database system before the transaction Ti starts execution. If a transaction Ti has been assigned timestamp TS(Ti), and a new transaction Tj enters the system, then TS(Ti) < TS(Tj ). There are two simple methods for implementing this scheme:

1. Use the value of the **system clock** as the timestamp; that is, a transaction's time-stamp is equal to the value of the clock when the transaction enters the system.
2. Use a **logical counter** that is incremented after a new timestamp has been assigned; that is, a transaction's timestamp is equal to the value of the counter when the transaction enters the system.

The timestamps of the transactions determine the serializability order. Thus, if TS(Ti) < TS(Tj ), then the system must ensure that the produced schedule is equivalent to a serial schedule in which transaction Ti appears before transaction Tj. To implement this scheme, we associate with each data item Q two timestamp values:

- **W-timestamp(Q)** denotes the largest timestamp of any transaction that exe-cuted wr ite(Q) successfully.
- **R-timestamp(Q)** denotes the largest timestamp of any transaction that exe-cuted read(Q) successfully.

These timestamps are updated whenever a new read(Q)or write(Q) instruction is executed.

**The Timestamp-Ordering Protocol:**
The timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order. This protocol operates as follows:

1. **Suppose that transaction Ti issues read(Q):**
   a. If **TS(Ti) < W-timestamp(Q),** then Ti needs to read a value of Q that was already overwritten. Hence, the read operation is rejected, and Ti is rolled back.
   b. If **TS(Ti) ≥ W-timestamp(Q)**, then the read operation is executed, and R-timestamp(Q) is set to the maximum of R-timestamp(Q)and TS(Ti).

2. **Suppose that transaction Ti issues write(Q):**
   a. If TS(Ti) < R-timestamp(Q), then the value of Q that Ti is producing was needed previously, and the system assumed that that value would never be produced. Hence, the system rejects the wr ite operation and rolls Ti back.
   b. If TS(Ti) < W-timestamp(Q), then Ti is attempting to write an obsolete value of Q. Hence, the system rejects this wr ite operation and rolls Ti back.
   c. Otherwise, the system executes the wr ite operation and sets W-time-stamp(Q)toTS(Ti).

If a transaction Ti is rolled back by the concurrency-control scheme as result of issuance of either a read or write operation, the system assigns it a new timestamp and restarts it.

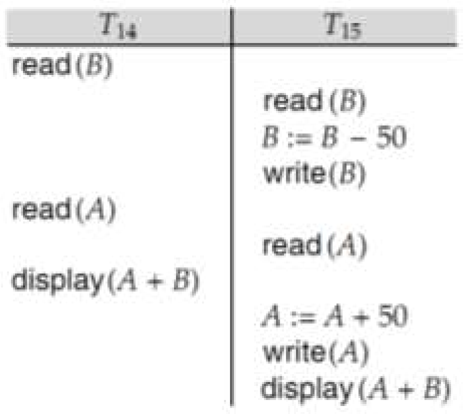
To illustrate this protocol, we consider transactions T14 and T15 .Transaction T14 displays the contents of accounts A and B:

```
T14 :        read(B);
             read(A);
             display(A + B).
```

Transaction T15 transfers $50 from account A to account B, and then displays the contents of both:

```
T15 :        read(B);
             B := B − 50;
             write(B);
             read(A);
             A := A + 50;
             write(A);
             display(A + B).
```

In presenting schedules under the timestamp protocol, we shall assume that a transaction is assigned a timestamp immediately before its first instruction. Thus, in schedule 3 of Figure 16.13, TS(T14 ) < TS(T15 ), and the schedule is possible under the time-stamp protocol.
The timestamp-ordering protocol ensures conflict serializability. This is because conflicting operations are processed in timestamp order. The protocol ensures freedom from deadlock, since no transaction ever waits. However, there is a possibility of starvation of long transactions if a sequence of conflicting short transactions causes repeated restarting of the long transaction. If a transaction is found to be getting restarted repeatedly, conflicting transactions need to be temporarily blocked to enable the transaction to finish.



| $T_{14}$ | $T_{15}$ |
|---|---|
| read($B$) | |
| | read($B$) |
| | $B := B - 50$ |
| | write($B$) |
| read($A$) | |
| | read($A$) |
| display($A + B$) | |
| | $A := A + 50$ |
| | write($A$) |
| | display($A + B$) |

**Figure 16.13** Schedule 3.

**Question-14: Discuss the functionalities of disk space manager of a DBMS. 5 Marks CSE-2013**

<u>Functionalities of a Disk Space Manager:</u>

**The disk space manager (DSM) (implemented as part of the DB class) is the component of Minibase that takes care of the allocation and deallocation of pages within a database.** It also performs reads and writes of pages to and from disk, and provides a logical *file* layer within the context of a database management system. The lowest level of software in the DBMS architecture, called the disk space manager, manages space on disk. Abstractly,

1. The disk space manager **supports the concept of a page as a unit of data, and provides commands to allocate or deallocate a page and read or write a page**. The size of a page is chosen to be the size of a disk block and pages are stored as disk blocks so that reading or writing a page can be done in one disk I/O.

2. It is often useful **to allocate a sequence of pages as a contiguous sequence of blocks to hold data that is frequently accessed in sequential order**. This capability is essential for exploiting the advantages of sequentially accessing disk blocks. Such a capability, if desired, must be provided by the disk space manager to higher-level layers of the DBMS.

3. **Keeping Track of Free Blocks:** A database grows and shrinks as records are inserted and deleted over time. The disk space manager **keeps track of which disk blocks are in use, in addition to keeping track of which pages are on which disk blocks**. Although it is likely that blocks are initially allocated sequentially on disk, subsequent allocations and deallocations could in general create `holes.' One way to keep track of block usage is to maintain a list of free blocks. The disk space manager **keeps track of free blocks and used disk blocks**.

4. Operating systems also manage space on disk. Typically, a database disk space manager could be built using OS files. The disk space manager is then **responsible for managing the space in these OS files**.

5. Disk space manager also provides **the abstraction of the data being a collection of disk pages**.

6. The disk space manager **hides details of the underlying hardware (and possibly the operating system) and allows higher levels of the software to think of the data as a collection of pages**.

**Question-15: Explain the terms Access Time, seek time, rotational delay and transfer time. 5 Marks CSE-2013**

**Access Time:**

**Access time is the time from when a read or write request is issued to when data transfer begins.** The access time has three components: the time to move the disk arm to the desired track (**seek time**), the time to wait for the desired block to rotate under the disk head (**rotational delay**), and the time to transfer the data (**transfer time)** i.e. **access time is the sum of seek time, rotational delay and transfer time and ranges from 8 to 20 milliseconds.**

$$\text{Access time} = \textit{Seek Time} + \textit{Rotational Delay} + \textit{Transfer Time}$$

**Seek Time:**

**Seek time is the time taken to move the disk heads to the track on which a desired block is located. To access (that is, to read or write) data on a given sector of a disk, the arm first must move so that it is positioned over the correct track, and then must wait for the sector to appear under it as the disk rotates. The time for repositioning the arm is called the seek time**, and it increases with the distance that the arm must move. Typical seek times range from **2 to 30 milliseconds**, depending on how far the track is from the initial arm position. Smaller disks tend to have lower seek times since the head has to travel a smaller distance.

The average seek time is the average of the seek times, measured over a sequence of (uniformly distributed) random requests. Generally, the average seek time is one-third the worst case seek time. The average seek time is around one-half of the maximum seek time. Average seek times currently range between **4 milliseconds and 10 milliseconds**, depending on the disk model.

**Rotational Delay:**

**Rotational delay or rotational latency is the time required for the addressed area of a computer's disk drive to rotate into a position where it is accessible by the read/write head. Rotational delay is the waiting time for the desired block to rotate under the disk head; it is the time required for half a rotation on average and is usually less than seek time.** Rotational delay is one of the three delays associated with reading or writing data on a disk, and somewhat similar for CD or DVD drives. The others are seek time and transfer time.

**Transfer Time:**

**Transfer time is the time to actually read or write the data in the block once the head is positioned, that is, the time for the disk to rotate over the block. It is also called data transfer rate.** It is the rate at which data can be retrieved from or stored to the disk. Current disk systems claim to support maximum transfer rates of about 25 to 40 megabytes per second, although actual transfer rates may be significantly less, at about 4 to 8 megabytes per second.

**An example of a current disk:**

The IBM Deskstar 14GPX is a 3.5 inch, 14.4 GB hard disk with an **average seek time of 9.1 milliseconds** (msec) and an **average rotational delay of 4.17 msec.** However, the time to seek from one track to the next is just 2.2 msec, the maximum seek time is 15.5 msec. The disk has five double-sided platters that spin at 7,200 rotations per minute. Each platter holds 3.35 GB of data, with a density of 2.6 gigabit per square inch. The data transfer rate is about 13 MB per second.

**Question-16: Discuss Deadlock Detection Technique in database system. 3 Marks CSE-2012 CSE-2008 CSE-2003 CSE-2000 CSE-2000 (OLD)**

**Deadlock Detection:**

Deadlocks can be described precisely in terms of a directed graph called a wait-for graph. This graph consists of a pair G =(V, E), where V is a set of vertices and E is a set of edges. The set of vertices consists of all the transactions in the system. Each element in the set E of edges is an ordered pair Ti → Tj .If Ti → Tj is in E, then there is a directed edge from transaction Ti to Tj , implying that transaction Ti is waiting for transaction Tj to release a data item that it needs.

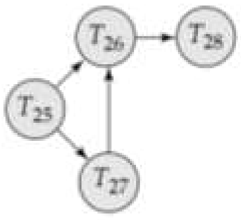


**Figure 16.18** Wait-for graph with no cycle.

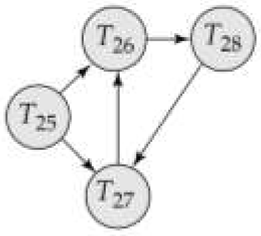


**Figure 16.19** Wait-for graph with a cycle.

When transaction Ti requests a data item currently being held by transaction Tj , then the edge Ti → Tj is inserted in the wait-for graph. This edge is removed only when transaction Tj is no longer holding a data item needed by transaction Ti. A deadlock exists in the system if and only if the wait-for graph contains a cycle. Each transaction involved in the cycle is said to be deadlocked. To detect deadlocks, the system needs to maintain the wait-for graph, and periodically to invoke an algorithm that searches for a cycle in the graph.

To illustrate these concepts, consider the wait-for graph in Figure 16.18, which depicts the following situation:

- Transaction T25 is waiting for transactions T26 and T27.
- Transaction T27 is waiting for transaction T26.
- Transaction T26 is waiting for transaction T28.

Since the graph has no cycle, the system is not in a deadlock state.
Suppose now that transaction T28 is requesting an item held by T27 .The edge T28 → T27 is added to the wait-for graph, resulting in the new system state in Figure 16.19. This time, the graph contains the cycle

T26 → T28 → T27 → T26

implying that transactions T26 , T27 ,and T28 are all deadlocked.
If deadlocks occur frequently, then the detection algorithm should be invoked more frequently than usual. Data items allocated to deadlocked transactions will be unavailable to other transactions until the deadlock can be broken. In addition, the number of cycles in the graph may also grow. In the worst case, we would invoke the detection algorithm every time a request for allocation could not be granted immediately.

**Question-17: Discuss the actions need to be taken to recover a database system from deadlock. 7 Marks CSE-2003 CSE-2003 (OLD) CSE-2002 CSE-2000**

**Recovery from Deadlock:**
When a detection algorithm determines that a deadlock exists, the system must re-cover from the deadlock. The most common solution is to roll back one or more trans-actions to break the deadlock. Three actions need to be taken:

1. **Selection of a victim**: Given a set of deadlocked transactions, we must deter-mine which transaction (or transactions) to roll back to break the deadlock. We should roll back those transactions that will incur the minimum cost. Unfortunately, the term minimum cost is not a precise one. Many factors may determine the cost of a rollback, including
    a. How long the transaction has computed, and how much longer the trans-action will compute before it completes its designated task.
    b. How many data items the transaction has used.
    c. How many more data items the transaction needs for it to complete.
    d. How many transactions will be involved in the rollback.

2. **Rollback**: Once we have decided that a particular transaction must be rolled back, we must determine how far this transaction should be rolled back. The simplest solution is a total rollback: Abort the transaction and then restart it. However, it is more effective to roll back the transaction only as far as necessary to break the deadlock. Such partial rollback

requires the system to maintain additional information about the state of all the running trans-actions. Specifically, the sequence of lock requests/grants and updates per-formed by the transaction needs to be recorded. The deadlock detection mechanism should decide which locks the selected transaction needs to release in order to break the deadlock. The selected transaction must be rolled back to the point where it obtained the first of these locks, undoing all actions it took after that point. The recovery mechanism must be capable of performing such partial rollbacks. Furthermore, the transactions must be capable of resuming execution after a partial rollback. See the bibliographical notes for relevant references.

3. **Starvation:** In a system where the selection of victims is based primarily on cost factors, it may happen that the same transaction is always picked as a victim. As a result, this transaction never completes its designated task, thus there is starvation. We must ensure that transaction can be picked as a victim only a (small) finite number of times. The most common solution is to include the number of rollbacks in the cost factor.

**Question-18: Discuss different types of failure in database system. 6 Marks CSE-2010 CSE-2009 CSE-2004 CSE-2003 (OLD) CSE-2002 CSE-2000 CSE-2000 (OLD)**

**<u>Failure Classification:</u>**
There are various types of failure that may occur in a system, each of which needs to be dealt with in a different manner. The simplest type of failure is one that does not result in the loss of information in the system. The failures that are more difficult to deal with are those that result in loss of information. In this chapter, we shall consider only the following types of failure:

- **Transaction failure:** There are two types of errors that may cause a transaction to fail:

  **Logical error**. The transaction can no longer continue with its normal execution because of some internal condition, such as bad input, data not found, overflow, or resource limit exceeded.

  **System error.** The system has entered an undesirable state (for example, deadlock), as a result of which a transaction cannot continue with its nor-mal execution. The transaction, however, can be re-executed at a later time.

- **System crash**: There is a hardware malfunction, or a bug in the database software or the operating system, that causes the loss of the content of volatile storage, and brings transaction processing to a halt. The content of non-volatile storage remains intact, and is not corrupted.

- **Disk failure:** A disk block loses its content as a result of either a head crash or failure during a data transfer operation. Copies of the data on other disks, or archival backups on tertiary media, such as tapes, are used to recover from the failure.

**Question-19: What are the different Techniques for Crash Recovery? Discuss the shadow paging technique of crash recovery. 7 Marks CSE-2012 CSE-2008 CSE-2005 CSE-2003 CSE-2000 CSE-2000 (OLD)**

**<u>Different Techniques for Crash Recovery:</u>**
A computer system, like any other device, is subject to failure from a variety of causes: disk crash, power outage, software error, a fire in the machine room, even sabotage. In any failure, information may be lost. Therefore, the database system must take actions in advance to ensure that the atomicity and durability properties of transactions are preserved. An integral part of a database system is a recovery scheme that can restore the database to the consistent state that existed before the failure. There are several techniques for crash recovery of a database systems:

1. Log-Based Recovery

2. Shadow Paging
3. Recovery with Concurrent Transactions

Shadow paging technique of crash recovery:

**Shadow paging** is an alternative technique to log-based crash-recovery techniques for providing atomicity and durability (two of the ACID properties) in database systems. Shadow paging is a copy-on-write technique for avoiding in-place updates of pages. Instead, when a page is to be modified, a **shadow page** is allocated. Since the shadow page has no references (from other pages on disk), it can be modified liberally, without concern for consistency constraints, etc. When the page is ready to become durable, all pages that referred to the original are updated to refer to the new replacement page instead. Because the page is "activated" only when it is ready, it is atomic.

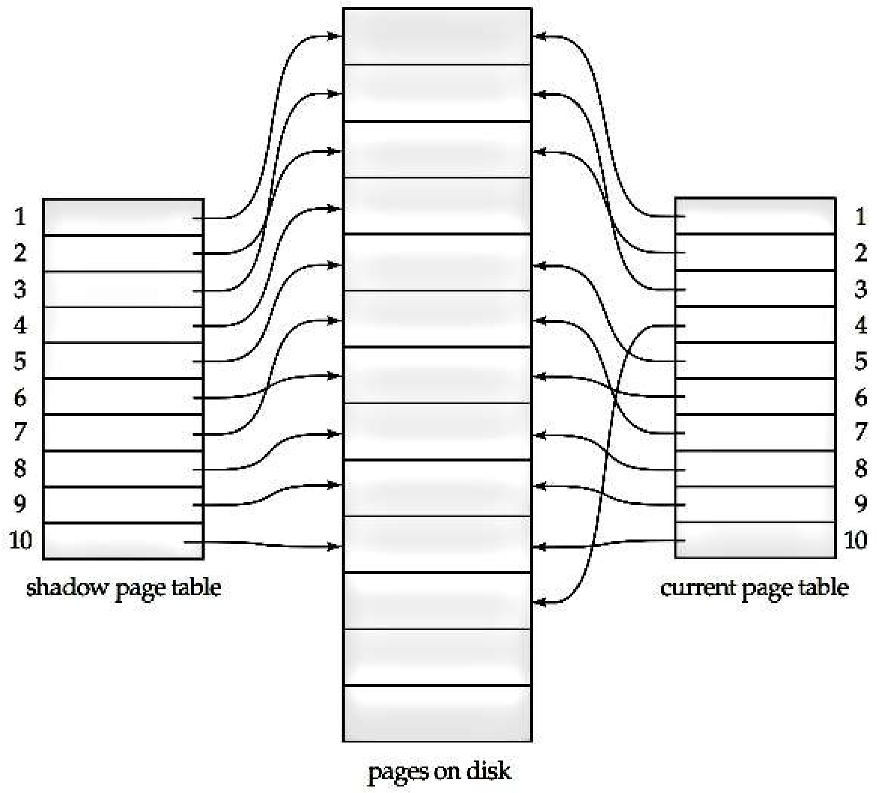


**Figure 17.9** Shadow and current page tables.

**In shadow paging,** two page tables are maintained during the life of a trans-action: the current page table and the shadow page table. When the transaction starts, both page tables are identical. The shadow page table and pages it points to are never changed during the duration of the transaction. When the transaction partially commits, the shadow page table is discarded, and the current table becomes the new page table. If the transaction aborts, the current page table is simply discarded.

The shadow-page approach to recovery is to store the shadow page table in non-volatile storage, so that the state of the database prior to the execution of the transaction can be recovered in the event of a crash, or transaction abort. When the transaction commits, the system writes the current page table to non-volatile storage. The current page table then becomes the new shadow page table, and the next transaction is allowed to begin execution. It is important that the shadow page table be stored in non-volatile storage, since it provides the only means of locating database pages. The current page table may be kept in main memory (volatile storage). We do not care whether the current page table is lost in a crash, since the system recovers by using the shadow page table.

**Question-20: Compare the Shadow paging Recovery scheme with log-based recovery schemes in terms of case of implementation and overhead cost. 10 Marks CSE-2010 CSE-2009 CSE-2004**

Implementation of Log Based Recovery Scheme:

In log-based schemes, all updates are recorded on a log, which must be kept in stable storage. In the deferred-modifications scheme, during the execution of a transaction, all the write operations are deferred until the transaction partially commits, at which time the system uses the information on the log associated with the transaction in executing the deferred writes. In the immediate-modifications scheme, the system applies all updates directly to the database. If a crash occurs, the system uses the information in the log in restoring the state of the system to a previous consistent state.

To reduce the overhead of searching the log and redoing transactions, we can use the checkpointing technique.

<u>**Overhead:**</u>

**1.** For log records to be useful for recovery from system and disk failures, the log must reside in stable storage. For now, we assume that every log record is written to the end of the log on stable storage as soon as it is created. The log contains a complete record of all database activity. **As a result, the volume of data stored in the log may become unreasonably large.**

**2.** The search process is time consuming.

**3.** Most of the transactions that, according to our algorithm, need to be redone have already written their updates into the database. Although redoing them will cause no harm, it will nevertheless cause recovery to take longer.

<u>**Implementation of Shadow Paging Recovery Scheme:**</u>

In shadow paging, two page tables are maintained during the life of a transaction: the current page table and the shadow page table. When the transaction starts, both page tables are identical. The shadow page table and pages it points to are never changed during the duration of the transaction. When the transaction partially commits, the shadow page table is discarded, and the current table becomes the new page table. If the transaction aborts, the current page table is simply discarded.

<u>**Overhead:**</u>

**Commit overhead.** The commit of a single transaction using shadow paging requires multiple blocks to be output—the actual data blocks, the current page table, and the disk address of the current page table.

**Question-21: What do you mean by recovery system? Mention different types of log records. 4 Marks CSE-2012 CSE-2007**

<u>**Recovery system:**</u>

A computer system, like any other device, is subject to failure from a variety of causes: disk crash, power outage, software error, a fire in the machine room, even sabotage. In any failure, information may be lost. Therefore, the database system must take actions in advance to ensure that the atomicity and durability properties of transactions are preserved. An integral part of a database system is a recovery scheme that can restore the database to the consistent state that existed before the failure. The recovery scheme must also provide high availability; that is, it must minimize the time for which the database is not usable after a crash. Recovery of a database system from different failures is called Recovery System.

<u>**Different Types of Log Records:**</u>

The most widely used structure for recording database modifications is the **log. The log is a sequence of log records, recording all the update activities in the database.** There are several types of log records. An **update log record** describes a single data-base write. Other special log records exist to record significant events during transaction pro-cessing, such as the start of a transaction and the commit or abort of a transaction.
We denote the various types of log records as:

- **<Ti start>.**Transaction Ti has started.
- **<Ti,Xj ,V1 ,V2 >.**Transaction Ti has performed a write on data item Xj . Xj had value V1 before the write, and will have value V2 after the write.
- **<Ti commit>.**Transaction Ti has committed.

- **<Ti abort>.**Transaction Ti has aborted.

**Question-22: Explain the use of checkpoint in log based recovery schema. 6 Marks CSE-2012 CSE-2007**

**Checkpoint in log based recovery schema:**
During execution, the system maintains the log, using Deferred Database Modification or Immediate Database Modification. In addition, the system periodically performs checkpoints, which require the following sequence of actions to take place:

1. Output onto stable storage all log records currently residing in main memory.
2. Output to the disk all modified buffer blocks.
3. Output onto stable storage a log record <checkpoint>.

Transactions are not allowed to perform any update actions, such as writing to a buffer block or writing a log record, while a checkpoint is in progress. The presence of a <checkpoint> record in the log allows the system to streamline its recovery procedure. Consider a transaction Ti that committed prior to the check-point. For such a transaction, the <Ti commit> record appears in the log before the <checkpoint> record. Any database modifications made by Ti must have been written to the database either prior to the checkpoint or as part of the checkpoint itself. Thus, at recovery time, there is no need to perform a redo operation on Ti.

After a failure has occurred, the recovery scheme examines the log to determine the most recent transaction Ti that started executing before the most recent checkpoint took place. It can find such a transaction by searching the log backward, from the end of the log, until it finds the first <checkpoint> record (since we are searching backward, the record found is the final <checkpoint> record in the log); then it continues the search backward until it finds the next <Ti start> record. This record identifies a transaction Ti. Once the system has identified transaction Ti, the redo and undo operations need to be applied to only transaction Ti and all transactions Tj that started executing after transaction Ti. Let us denote these transactions by the set T. The remainder (earlier part) of the log can be ignored, and can be erased whenever desired. The exact recovery operations to be performed depend on the modification technique being used. For the immediate-modification technique, the recovery operations are:

- For all transactions Tk in T that have no <Tk commit> record in the log, execute undo(Tk).
- For all transactions Tk in T such that the record <Tk commit> appears in the log, execute redo(Tk).

Obviously, the undo operation does not need to be applied when the deferred-modification technique is being employed.

As an illustration, consider the set of transactions {T0 , T1 ,...,T100 } executed in the order of the subscripts. Suppose that the most recent checkpoint took place during the execution of transaction T67 . Thus, only transactions T67 , T68 ,...,T100 need to be considered during the recovery scheme. Each of them needs to be redone if it has committed; otherwise, it needs to be undone.

**Question-23: Define Data Dictionary. What are the information that the system stores, about relations, users and others if any? 1+4 Marks CSE-2005**

**Data Dictionary:**
A relational-database system needs to maintain data about the relations, such as the schema of the relations. This information is called the **data dictionary**, or **system catalog**. Among the types of information that the system must store are these:

- Names of the relations
- Names of the attributes of each relation
- Domains and lengths of attributes

- Names of views defined on the database, and definitions of those views
- Integrity constraints (for example, key constraints)

In addition, many systems keep the following data on users of the system:

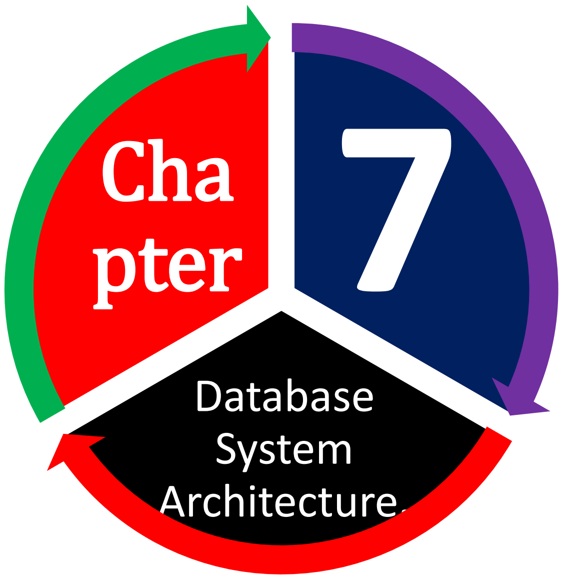
- Names of authorized users
- Accounting information about users
- Passwords or other information used to authenticate users

Further, the database may store statistical and descriptive data about the relations, such as:

- Number of tuples in each relation
- Method of storage for each relation (for example, clustered or nonclustered)

The data dictionary may also note the storage organization (sequential, hash or heap) of relations, and the location where each relation is stored:

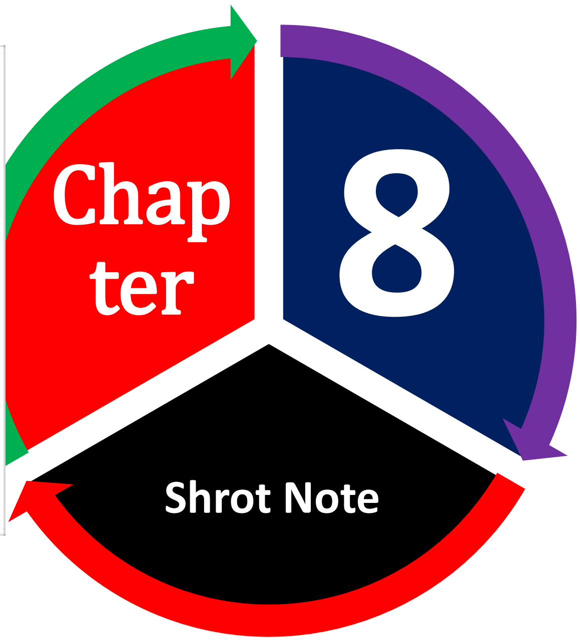
- If relations are stored in operating system files, the dictionary would note the names of the file (or files) containing each relation.
- If the database stores all relations in a single file, the dictionary may note the blocks containing records of each relation in a data structure such as a linked list.



# Chapter 7 Database System Architecture

## Important Question

1. Discuss the following operations in terms of relational algebra: (i) Project **CSE-2011 CSE-2009 CSE-2003 CSE-2002** (ii) Set Difference **CSE-2011** (iii) Cartesian Product **CSE-2011 CSE-2009 CSE-2003 CSE-2002** (iv) Outer-Join. **CSE-2011 CSE-2009** (V) Select Operation **CSE-2010 CSE-2008** (vi) Union Operation **CSE-2010 CSE-2008** (vii) Rename Operation **CSE-2010 CSE-2008** (viii) Natural Join Operation. **CSE-2010 CSE-2009 CSE-2008 CSE-2003 CSE-2002** (ix) Set intersection. **CSE-2003** (x) Set Difference. **CSE-2002**
2. Explain relational data model with example. **4 Marks CSE-2009**
3. Define schema diagram. Differentiate between Cartesian product and Natural Join with suitable example. **2+3 Marks CSE-2006**
4. Which operations are used to modify the database? Describe them with examples. **6 Marks CSE-2006**
5. What do you mean by relational algebra? What are the fundamental operations in relational algebra? Briefly explain. **7 Marks CSE-2005 CSE-2003 CSE-2003 (OLD)**
6. List two reasons why null values might be introduced into the database. **2 Marks CSE-2005**
7. What are left outer join, right outer join and full outer join? Explain with example. **5 Marks CSE-2004**
8. List two reasons why we may choose to define a view. **2 Marks CSE-2003 (OLD)**
9. Define Domain, Tuple Variable, Query Languages and foreign key. 4 Marks CSE-2007
10. What is a View? Why do we define a View? **1+2 Marks CSE-2007**
11. Why do we use a number of relations that are connected in some ways rather than just a relation that contains all the attributes? **2 Marks CSE-2007**
12. What are the reasons for the popularity of relational data model? **3 Marks CSE-2006**



# Important Question

1. **Write Short Notes on the Following Topics:**
   (a) Shadow Paging **CSE-2011**
   (b) Transaction States. **CSE-2011**
   (c) ACID Properties **CSE-2011**
   (d) False Cycle **CSE-2011**
   (e) Database Administrator (DBA) **CSE-2011 CSE-2008**
   (f) Object-Oriented Data Model. **CSE-2011 CSE-2009**
   (g) Data Server System **CSE-2010 CSE-2004**
   (h) Deadlock Handling in Distributed System. **CSE-2010 CSE-2004**
   (i) Decomposition of Database Schema. **CSE-2010 CSE-2004**
   (j) PL/SQL. **CSE-2010 CSE-2008**
   (k) Distributed Data Storage. **CSE-2010**
   (l) Homogeneous and Heterogeneous Databases. **CSE-2010**
   (m) File Organization. **CSE-2009**
   (n) Transaction Server Process Architecture. **CSE-2009**
   (o) Parallel Database Architectures. **CSE-2009**
   (p) Triggers in SQL. **CSE-2009**
   (q) Modification of the Database. **CSE-2009**
   (r) Triggers and Encryptions. **CSE-2008**
   (s) Data Definition Language. **CSE-2008**
   (t) Log Based Recovery System. **CSE-2008**

## Shadow Paging

**Shadow paging** is an alternative technique to log-based crash-recovery techniques for providing atomicity and durability (two of the ACID properties) in database systems. Shadow paging is a copy-on-write technique for avoiding in-place updates of pages. Instead, when a page is to be modified, a **shadow page** is allocated. Since the shadow page has no references (from other pages on disk), it can be modified liberally, without concern for consistency constraints, etc. When the page is ready to become durable, all pages that referred to the original are updated to refer to the new replacement page instead. Because the page is "activated" only when it is ready, it is atomic.

**In shadow paging,** two page tables are maintained during the life of a trans-action: the current page table and the shadow page table. When the transaction starts, both page tables are identical. The shadow page table and pages it points to are never changed during the duration of the transaction. When the transaction partially commits, the shadow page table is discarded, and the current table becomes the new page table. If the transaction aborts, the current page table is simply discarded.

The shadow-page approach to recovery is to store the shadow page table in non-volatile storage, so that the state of the database prior to the execution of the transaction can be recovered in the event of a crash, or transaction abort. When the transaction commits, the system writes the current page table to non-volatile storage. The current page table then becomes the new shadow page table, and the next transaction is allowed to begin execution. It is important that the shadow page table be stored in non-volatile storage, since it provides the only means of locating database pages. The current page table may be kept in main memory (volatile storage). We do not care whether the current page table is lost in a crash, since the system recovers by using the shadow page table.

### Advantages of shadow-paging over log-based schemes

1. Under certain circumstances, shadow paging may require fewer disk accesses than do the log-based methods discussed previously
2. No Overhead Of Writing Log Records
3. Recovery Is Trivial

### Disadvantages:

1. Copying the entire page table is very expensive
2. **Commit overhead.** The commit of a single transaction using shadow paging requires multiple blocks to be output -- the current page table, the actual data and the disk address of the current page table. Log-based schemes need to output only the log records. Data gets fragmented (related pages get separated on disk)
3. **Garbage collection.** Each time that a transaction commits, the database pages containing the old version of data changed by the transactions must become inaccessible. Such pages are considered to be *garbage* since they are not part of the free space and do not contain any usable information. Periodically it is necessary to find all of the garbage pages and add them to the list of free pages. This process is called *garbage collection* and imposes additional overhead and complexity on the system.
4. **Data fragmentation.** Shadow paging causes database pages to change locations (therefore, no longer contiguous.
5. Shadow paging is more difficult than logging to adapt to systems that allow several transactions to exe-cute concurrently.

<u>Transaction States</u>

**A transaction is a unit of program execution that accesses and possibly updates various data items. Usually, a transaction is initiated by a user program written in a high-level data-manipulation language or programming language (for example, SQL, COBOL, C, C++, or Java), where it is delimited by statements (or function calls) of the form begin transaction and end transaction. The transaction consists of all operations executed between the begin transaction and end transaction.** A transaction must be in one of the following states:

- **Active,** the initial state; the transaction stays in this state while it is executing.
- **Partially committed,** after the final statement has been executed.
- **Failed,** after the discovery that normal execution can no longer proceed.
- **Aborted: In the absence of failures, all transactions complete successfully. However, as we noted earlier, a transaction may not always complete its execution successfully. Such a transaction is termed aborted.** If we are to ensure the atomicity property, an aborted transaction must have no effect on the state of the database. Thus, any changes that the aborted transaction made to the database must be undone. Once the changes caused by an aborted transaction have been undone, we say that the transaction has been **rolled back.** After the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

- **Committed:** A transaction that completes its execution successfully is said to be **committed.** A committed transaction that has performed updates transforms the database into a new consistent state, which must persist even if there is a system failure. Once a transaction has committed, we cannot undo its effects by aborting it. The only way to undo the effects of a committed transaction is to execute a compensating transaction. The transaction will be in committed state, after successful completion.

The state diagram corresponding to a transaction appears in Figure 15.1.

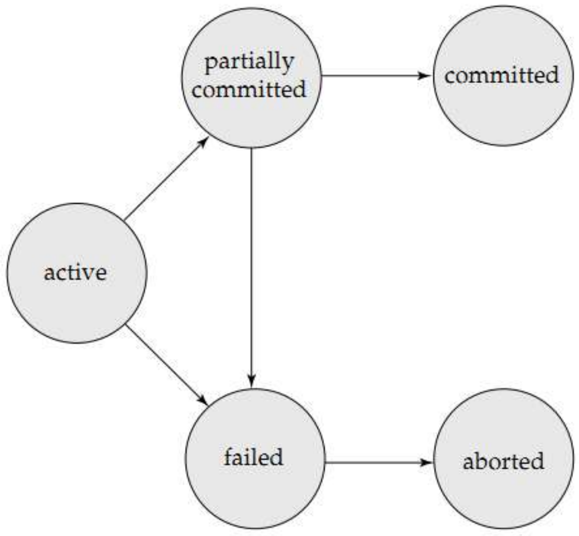


**Figure 15.1** State diagram of a transaction.

A transaction starts in the **active state.** When it finishes its final statement, it enters the **partially committed state.** At this point, the transaction has completed its execution, but it is still possible that it may have to be **aborted,** since the actual output may still be temporarily residing in main memory, and thus a hardware failure may preclude its successful completion. The database system then writes out enough information to disk that, even in the event of a failure, the updates performed by the transaction can be re-created when the system restarts after the failure. When the last of this information is written out, the transaction enters the **committed state.**

<u>ACID Properties</u>

A transaction is a unit of program execution that accesses and possibly updates various data items. To ensure integrity of the data, we require that the database system maintain the following properties of the transactions:

- **Atomicity**. Either all operations of the transaction are reflected properly in the database, or none are.

- **Consistency**. Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database.

- **Isolation.** Even though multiple transactions may execute concurrently, the system guarantees that, for every pair of transactions Ti and Tj , it appears to Ti that either Tj finished execution before Ti started, or Tj started execu-tion after Ti finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.

- **Durability**. After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

These properties are often called the **ACID properties.**

<u>**Database Administrator (DBA)**</u>
One of the main reasons for using DBMSs is to have central control of both the data and the programs that access those data. **A person who has such central control over the system is called a database administrator (DBA)**. The functions of a DBA include:

- **Schema definition:** The DBA creates the original database schema by executing a set of data definition statements in the DDL.
- **Storage structure and access-method definition**: The DBA creates appropriate storage structures and access methods by writing a set of definitions, which is translated by the data-storage and data-definition –language compiler.
- **Schema and physical-organization modification:** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.
- **Granting of authorization for data access**: By granting different types of authorization, the database administrator can regulate which parts of the data-base various users can access. The authorization information is kept in a special system structure that the database system consults whenever some-one attempts to access the data in the system.
- **Integrity-Constraint Specification:** The data values stored in the database must satisfy certain consistency constraints. Such a constraint mus t be specified explicitly by the database administrator. The integrity constraints are ket in a special system structure that is consulted by the database system whenever an update takes place in the system.
- **Routine maintenance:** Examples of the database administrator’s routine maintenance activities are:
  (iv) Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
  (v) Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.
  (vi) Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

<u>**Object-Oriented Data Model**</u>
The object-oriented data model is based on the object-oriented programming language paradigm, which is now in wide use. The object-oriented data model extends the representation of entities by adding notions of encapsulation, methods (functions), and object identity. Inheritance, object-identity, and encapsulation (information hiding), with

methods to provide an interface to objects, are among the key concepts of object-oriented programming that have found applications in data modelling. The object-oriented data model also supports a rich type system, including structured and collection types. The object-oriented model can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity.

## Data Server System

Server systems can be broadly categorized as transaction servers and data servers. **Transaction-server systems,** also called query-server systems, provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client. **Data-server systems** allow clients to interact with the servers by making re-quests to read or update data, in units such as files or pages.

**For example**, file servers provide a file-system interface where clients can create, update, read, and delete files. Data servers for database systems offer much more functionality; they support units of data — such as pages, tuples, or objects — that are smaller than a file. They provide indexing facilities for data, and provide transaction facilities so that the data are never left in an inconsistent state if a client machine or process fails.

Data-server systems are used in local-area networks, where there is a high-speed connection between the clients and the server, the client machines are comparable in processing power to the server machine, and the tasks to be executed are computation intensive. In such an environment, it makes sense to ship data to client machines, to perform all processing at the client machine (which may take a while), and then to ship the data back to the server machine. This architecture requires full back-end functionality at the clients. Data-server architectures have been particularly popular in object-oriented database systems.

## Parallel Database Architectures

There are several architectural models for parallel machines. Among the most promi-nent ones are those in Figure 18.8 (in the figure, M denotes memory, P denotes a processor, and disks are shown as cylinders):

- **Shared memory**. All the processors share a common memory (**Figure 18.8a).**
- **Shared disk.** All the processors share a common set of disks (**Figure 18.8b).** Shared-disk systems are sometimes called clusters.
- **Shared nothing**. The processors share neither a common memory nor common disk (**Figure 18.8c).**
- **Hierarchical.** This model is a hybrid of the preceding three architectures (**Figure 18.8d).**

Techniques used to speed up transaction processing on data-server systems, such as data and lock caching and lock de-escalation can also be used in shared-disk parallel databases as well as in shared-nothing parallel databases. In fact, they are very important for efficient transaction processing in such systems.

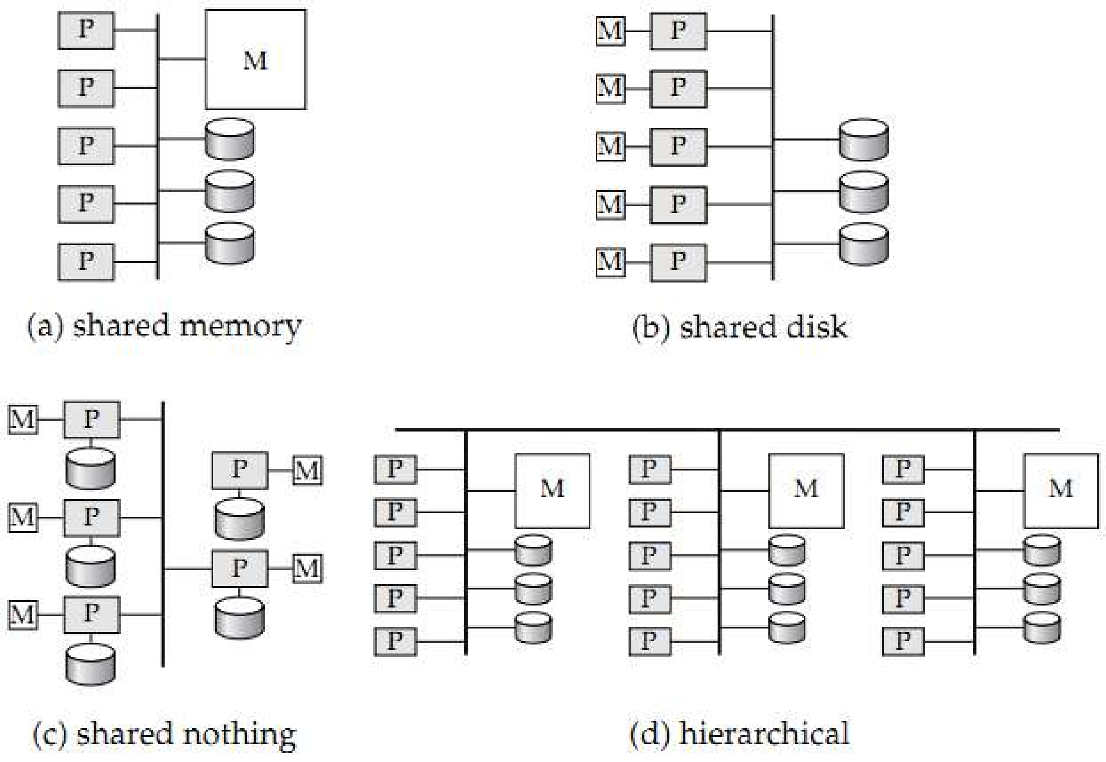


**Figure 18.8** Parallel database architectures.

<u>Triggers in SQL</u>

**A trigger is a statement that the system executes automatically as a side effect of a modification to the database.** To design a trigger mechanism, we must meet two requirements:

1. Specify when a trigger is to be executed. This is broken up into an event that causes the trigger to be checked and a condition that must be satisfied for trigger execution to proceed.
2. Specify the actions to be taken when the trigger executes.

```
create trigger overdraft-trigger after update on account
referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
    insert into borrower
            (select customer-name, account-number
             from depositor
             where nrow.account-number = depositor.account-number);
    insert into loan values
            (nrow.account-number, nrow.branch-name, – nrow.balance);
    update account set balance = 0
            where account.account-number = nrow.account-number
end
```

**Figure 6.3** Example of SQL:1999 syntax for triggers.

SQL-based database systems use triggers widely, although before **SQL:1999** they were not part of the SQL standard. Unfortunately, each database system implemented its own syntax for triggers, leading to incompatibilities. We outline in **Figure 6.3** the SQL:1999 syntax for triggers (which is similar to the syntax in the IBM DB2 and Oracle database systems).

This trigger definition specifies that the trigger is initiated *after* any update of the relation account is executed. An SQL update statement could update multiple tuples of the relation, and the for **each row clause** in the trigger code would then explicitly iterate over each updated row. The **referencing new row** as clause creates a variable *nrow* (called a **transition variable**), which stores the value of an updated row after the update.

The **when** statement specifies a condition, namely *nrow.balance* $< 0$. The system executes the rest of the trigger body only for tuples that satisfy the condition. The **begin atomic ... end** clause serves to collect multiple SQL statements into a single compound statement. The two **insert** statements with the **begin ... end** structure carry out the specific tasks of creating new tuples in the borrower and loan relations to represent the new loan. The **update** statement serves to set the account balance back to 0 from its earlier negative value.

Triggers should be written with great care, since a trigger error detected at run time causes the failure of the insert/delete/update statement that set off the trigger. Furthermore, the action of one trigger can set off another trigger. In the worst case, this could even lead to an infinite chain of triggering.

<u>Log Based Recovery System</u>
The most widely used structure for recording database modifications is the log. The log is a sequence of log records, recording all the update activities in the database. There are several types of log records. An update log record describes a single data-base write. It has these fields:

- **Transaction identifier** is the unique identifier of the transaction that performed the write operation.
- **Data-item identifier** is the unique identifier of the data item written. Typically, it is the location on disk of the data item.
- **Old value** is the value of the data item prior to the write.
- **New value** is the value that the data item will have after the write.

Other special log records exist to record significant events during transaction pro-cessing, such as the start of a transaction and the commit or abort of a transaction. We denote the various types of log records as:

• **<Ti start>.**Transaction Ti has started.
• **<Ti,Xj ,V1 ,V2 >.**Transaction Ti has performed a write on data item Xj . Xj had value V1 before the write, and will have value V2 after the write.
• **<Ti commit>.**Transaction Ti has committed.
• **<Ti abort>.**Transaction Ti has aborted.

Whenever a transaction performs a write, it is essential that the log record for that write be created before the database is modified. Once a log record exists, we can output the modification to the database if that is desirable. Also, we have the ability to undo a modification that has already been output to the database. We undo it by using the old-value field in log records.

For log records to be useful for recovery from system and disk failures, the log must reside in stable storage. For now, we assume that every log record is written to the end of the log on stable storage as soon as it is created. The log contains a complete record of all database activity. As a result, the volume of data stored in the log may become unreasonably large.
**<u>Question:</u> Write Short notes on: Distributed Data Storage. 5 Marks CSE-2010**

**<u>Distributed Data Storage:</u>**
Consider a relation r that is to be stored in the database. There are two approaches to storing this relation in the distributed database:

- **Replication.** The system maintains several identical replicas (copies) of the relation, and stores each replica at a different site. The alternative to replication is to store only one copy of relation r.

- **Fragmentation**. The system partitions the relation into several fragments, and stores each fragment at a different site.

Fragmentation and replication can be combined: A relation can be partitioned into several fragments and there may be several replicas of each fragment. In the following subsections, we elaborate on each of these techniques.

**Data Replication:**

If relation r is replicate d, a copy of relation r is stored in two or more sites. In the most extreme case, we have full replication, in which a copy is stored in every site in the system. There are a number of advantages and disadvantages to replication.

- **Availability.** If one of the sites containing relation r fails, then the relation r can be found in another site. Thus, the system can continue to process queries involving r, despite the failure of one site.

- **Increased parallelism.** In the case where the majority of accesses to the relation r result in only the reading of the relation, then several sites can process queries involving r in parallel. The more replicas of r there are, the greater the chance that the needed data will be found in the site where the transaction is executing. Hence, data replication minimizes movement of data between sites.

- **Increased overhead on update.** The system must ensure that all replicas of a relation r are consistent! Otherwise, erroneous computations may result. Thus, whenever r is updated, the update must be propagated to all sites containing replicas. The result is increased overhead.

In general, replication enhances the performance of read operations and increases the availability of data to read-only transactions. However, update transactions incur greater overhead. Controlling concurrent updates by several transactions to replicated data is more complex than in centralized systems

**Data Fragmentation**

If relation r is fragmented, r is divided into a number of fragments $r_1$ , $r_2$ , $r_3$, ……… , $r_n$. These fragments contain sufficient information to allow reconstruction of the original relation r. There are two different schemes for fragmenting a relation: horizontal fragmentation and vertical fragmentation. Horizontal fragmentation splits the relation by assigning each tuple of r to one or more fragments. Vertical fragmentation splits the relation by decomposing the scheme R of relation r.

In **horizontal fragmentation,** a relation r is partitioned into a number of subsets, rr, r2t . . . , rn. Each tuple of relation r must belong to at least one of the fragments, so that the original relation can be reconstructed, if needed.

**Vertical fragmentation** of r(R) involves the definition of several subsets of attributes $R_1, R_2,, \ldots, R_n$, of the schema R so that

$$R = R_1 \; U \; R_2 \; U \ldots\ldots\ldots U R_n,,$$

Each fragment $r_i$ of r is defined by

$$r_i = \prod_{R_i}(r)$$

The fragmentation should be done in such a way that we can reconstruct relation r from the fragments by taking the natural join

$$r = r_1 \boxtimes r_2 \boxtimes r_3 \boxtimes \;\ldots\ldots \boxtimes r_n$$

**Question: Write Short Notes on: Homogeneous and Heterogeneous Databases. 5 Marks CSE-2010**

## Homogeneous Distributed Database:

**In a homogeneous distributed database system,** all sites have identical database-management system software, are aware of one another, and agree to cooperate in processing users' requests. In such a system, local sites surrender a portion of their autonomy in terms of their right to change schemas or database-management system software. That software must also cooperate with other sites in exchanging information about transactions, to make transaction processing possible across multiple sites.

## Heterogeneous Distributed Database:

**In contrast, in a heterogeneous distributed database,** different sites may use different schemas, and different database-management system software. The sites may not be aware of one another, and they may provide only limited facilities for cooperation in transaction processing. The differences in schemas are often a major problem for query processing, while the divergence in software becomes a hindrance for processing transactions that access multiple sites.

## File Organization

**A file organization is a way of arranging the records in a file when the file is stored on disk.** A file of records is likely to be accessed and modified in a variety of ways, and different ways of arranging the records enable different operations over the file to be carried out efficiently. For example, if we want to retrieve employee records in alphabetical order, sorting the file by name is a good file organization. On the other hand, if we want to retrieve all employees whose salary is in a given range, sorting employee records by name is not a good file organization. A DBMS supports several file organization techniques, and an important task of a DBA is to choose a good organization for each file, based on its expected pattern of use.

There are **three basic file organizations:** files of randomly ordered records (i.e., **heap files**), files sorted on some field, and files that are hashed on some fields.

A heap file has good storage efficiency and supports fast scan, insertion, and deletion of records. However, it is slow for searches.

A **sorted file** also offers good storage efficiency, but insertion and deletion of records is slow. It is quite fast for searches, and it is the best structure for range selections. It is worth noting that in a real DBMS, a file is almost never kept fully sorted. A structure called a B+ tree, offers all the advantages of a sorted file and supports inserts and deletes efficiently. There is a space overhead for these benefits, relative to a sorted file, but the trade-off is well worth it.

Files are sometimes kept `almost sorted' in that they are originally **sorted,** with some free space left on each page to accommodate future insertions, but once this space is used, overflow pages are used to handle insertions. The cost of insertion and deletion is similar to a heap file, but the degree of sorting deteriorates as the file grows.

A **hashed file** does not utilize space quite as well as a sorted file, but insertions and deletions are fast, and equality selections are very fast. However, the structure offers no support for range selections, and full file scans are a little slower; the lower space utilization means that files contain more pages.

**In summary,** no one file organization is uniformly superior in all situations. An unordered file is best if only full file scans are desired. A hashed file is best if the most common operation is an equality selection. A sorted file is best if range selections are desired. The organizations that we have studied here can be improved on the problems of overflow pages in static hashing can be overcome by using dynamic hashing structures, and the high cost of inserts and deletes in a sorted file can be overcome by using tree-structured indexes but the main observation, that the choice of an appropriate file organization depends on how the file is commonly used, remains valid.

নিচের উত্তরবিহীন টীকাসমূহের উত্তর নিজে সন্ধান করুন, যদিও খুব বেশি প্রয়োজন নেই এই টীকাসমূহ পড়ার !!!

1. Transaction Server Process Architecture
2. Deadlock Handling in Distributed System
3. Decomposition of Database Schema
4. PL/SQL
5. False Cycle

# Chapter 8 Database Security and Integrity

## Important Question

01. What are the main objectives to consider while designing a secure database application? Explain.

02. Explain the forms of authorization which exist to access database and modify database schema. **4 Marks CSE-2009**

03. Define integrity constraints. Mention different forms of integrity constraints. When is it specified and enforced? **8 Marks CSE-2013 CSE-2005 CSE-2000 CSE-2000 (OLD)**

04. What do you mean by Key Constraints? How can you specify key constraints in SQL-92?

05. How to translate an Entity Sets into a collection of tables with associated constraints?

06. Define the following terms: relation schema, relational database schema, domain, relation instance, relation cardinality, and relation degree.

07. What is a foreign key constraint? Why are such constraints important? What is referential integrity? 5 Marks

08. What is view? Why do we define view? **3 + 2 Marks CSE-2011 CSE-2006 CSE -2001**

09. "The Database Design process can be divided into six steps" – Please explain the steps in short. **7 Marks CSE-2013**

10. What is Trigger and why they are used? **5 Marks CSE-2012 CSE-2004 CSE-2003 (OLD)**
11. Describe the basic parts of a trigger and show the syntax for creating a trigger. **6 Marks CSE-2003 (OLD) CSE-2002 CSE-2001 CSE-2000 CSE-2000 (OLD)**
12. What Security measures do we need to be taken at several levels to protect database? **4 Marks CSE-2003 CSE-2002**
13. Define Candidate key, Foreign Key and Assertion. **6 Marks CSE-2009**
14. What are the differences between an assertion and a trigger? **5 Marks CSE-2012 CSE-2004 CSE-2003 (OLD) CSE-2001**
15. Discuss the strengths and weaknesses of the trigger mechanism. **1.75 Marks CSE-2013**
16. What is access Control? What are the approaches to access control? Discuss in brief.
17. What are the advantages of encrypting data stored in the database? **4 Marks CSE-2003 CSE-2002 CSE-2012**
18. What is the role of the DBA with respect to security? **4 Marks**

**Question-01: What are the main objectives to consider while designing a secure database application? Explain.**

**Main objectives to consider while designing a secure database application:**
**There are three main objectives to consider while designing a secure database application:**

1. **Secrecy:** Information should not be disclosed to unauthorized users. For example, a student should not be allowed to examine other students' grades.

2. **Integrity:** Only authorized users should be allowed to modify data. For example, students may be allowed to see their grades, yet not allowed (obviously!) to modify them.

3. **Availability:** Authorized users should not be denied access. For example, an instructor who wishes to change a grade should be allowed to do so.

**Question-02: Explain the forms of authorization which exist to access database and modify database schema. 4 Marks CSE-2009**

Forms of authorization to access database:
We may assign a user several forms of authorization on parts of the database. For example,

- **Read authorization** allows reading, but not modification, of data.

- **Insert authorization** allows insertion of new data, but not modification of ex-isting data.

- **Update authorization** allows modification, but not deletion, of data.

- **Delete authorization** allows deletion of data.

We may assign the user all, none, or a combination of these types of authorization.

Forms of authorization to modify database Schema:
In addition to these forms of authorization for access to data, we may grant a user authorization to modify the database schema:

- **Index authorization** allows the creation and deletion of indices.

- **Resource authorization** allows the creation of new relations.

- **Alteration authorization** allows the addition or deletion of attributes in a relation.

- **Drop authorization** allows the deletion of relations.

**Question-03: Define integrity constraints. Mention different forms of integrity constraints. When is it specified and enforced? 8 Marks CSE-2013 CSE-2005 CSE-2000 CSE-2000 (OLD)**

**Integrity Constraints:**

A database is only as good as the information stored in it, and a DBMS must therefore help prevent the entry of incorrect information. **An integrity constraint (IC) is a condition that is specified on a database schema, and restricts the data that can be stored in an instance of the database.** If a database instance satisfies all the integrity constraints specified on the database schema, it is a **legal** instance. A DBMS enforces integrity constraints, in that it permits only legal instances to be stored in the database.

Integrity constraints ensure that changes made to the database by authorized users do not result in a loss of data consistency. Thus, integrity constraints guard against accidental damage to the database.

Some forms of integrity constraints are:

1. Functional Dependency
2. Triggers
3. Domain Constraints
4. Key Constraints
5. General Constrains
6. Form of a relationship

**When Integrity constraints are specified and enforced:**

Integrity constraints are specified and enforced at different times:

1. When the DBA or end user defines a database schema, he or she specifies the ICs that must hold on any instance of this database.

2. When a database application is run, the DBMS checks for violations and disallows changes to the data that violate the specified ICs. (In some situations, rather than disallow the change, the DBMS might instead make some compensating changes to the data to ensure that the database instance satisfies all ICs. In any case, changes to the database are not allowed to create an instance that violates any IC.)

**Question-04: What do you mean by Key Constraints? How can you specify key constraints in SQL-92?**

**Key Constraints:**

**A key constraint is a statement that a certain minimal subset of the fields of a relation is a unique identifier for a tuple.**

**Example:**

Consider the Students relation and the constraint that no two students have the same student id. This IC is an example of a key constraint.

**Specifying Key Constraints in SQL-92**

In SQL we can declare that a subset of the columns of a table constitute a **key** by using the **UNIQUE** constraint. At most one of these `candidate' keys can be declared to be a primary key, using the **PRIMARY KEY** constraint. SQL does not require that such constraints be declared for a table.

Let us consider the following example:

```
CREATE TABLE Students      ( sid CHAR(20),
                             name CHAR(30),
                             login CHAR(20),
                             age INTEGER,
```

*gpa* REAL,
**UNIQUE** (name, age),
**CONSTRAINT** StudentsKey **PRIMARY KEY** (sid) )

This definition says that *sid* is the primary key and that the combination of name and age is also a key. The definition of the primary key also illustrates how we can name a constraint by preceding it with **CONSTRAINT** *constraint-name*. If the constraint is violated, the constraint name is returned and can be used to identify the error.

**Question-05: How to translate an Entity Sets into a collection of tables with associated constraints?**

__Entity Sets to Tables__

An entity set is mapped to a relation in a straightforward way: Each attribute of the entity set becomes an attribute of the table. We know both the domain of each attribute and the (primary) key of an entity set. Consider the Employees entity set with attributes **ssn**, **name,** and **lot** shown in Figure 3.8. A possible instance of the Employees entity set, containing three Employees entities, is shown in Figure 3.9 in a tabular format.

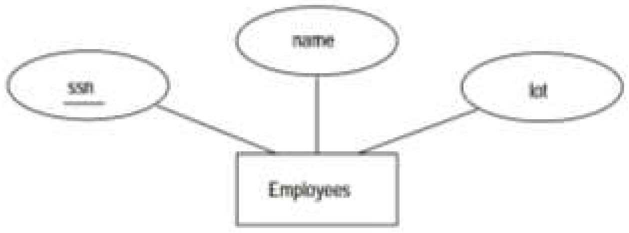
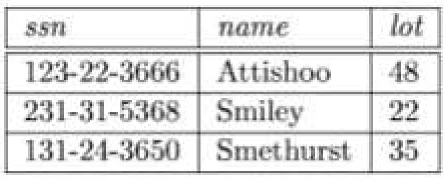


**Figure 3.8** The Employees Entity Set



| *ssn* | *name* | *lot* |
|---|---|---|
| 123-22-3666 | Attishoo | 48 |
| 231-31-5368 | Smiley | 22 |
| 131-24-3650 | Smethurst | 35 |

**Figure 3.9** An Instance of the Employees Entity Set

The following SQL statement captures the preceding information, including the domain constraints and key information:

**CREATE TABLE** Employees ( ssn CHAR(11),
*name* CHAR(30),
*lot* INTEGER,
**PRIMARY KEY** (ssn) )

**Question-06: Define the following terms: relation schema, relational database schema, domain, relation instance, relation cardinality, and relation degree.**

__Relation Schema:__

**A relation schema describes the structure of a relation by specifying the relation name and the names of each field. A relation schema consists of a list of attributes and their corresponding domains. In addition, the relation schema includes domain constraints, which are type restrictions on the fields of the relation.**

The schema specifies the relation's name, the name of each field (or column, or attribute), and the domain of each field. A domain is referred to in a relation schema by the domain name and has a set of associated values.

We use **Account-schema** to denote the relation schema for relation **account**. Thus, *Account-schema = (account-number, branch-name, balance)*

<u>Relational Database Schema:</u>
**A relational database is a collection of relations with distinct relation names. The relational database schema is the collection of schemas for the relations in the database.**

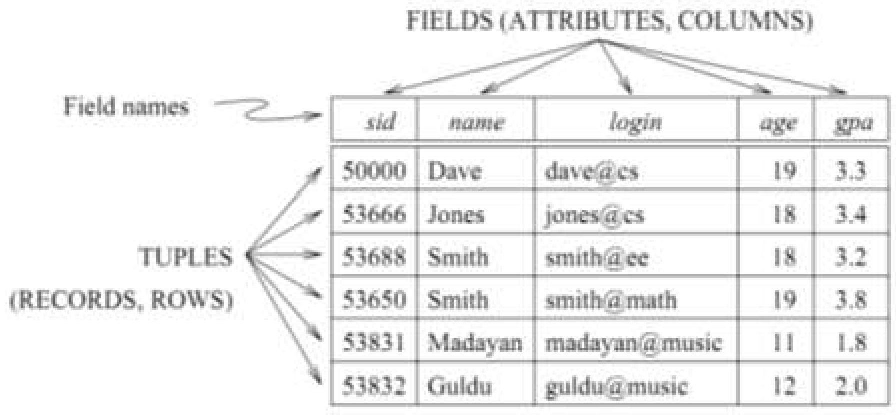
For example, following is a university database schema with relations called Students, Faculty, Courses, Rooms, Enrolled, Teaches, and Meets In.

*Students(sid: string, name: string, login: string, age: integer, gpa: real)*
*Faculty(fid: string, fname: string, sal: real)*
*Courses(cid: string, cname: string, credits: integer)*
*Rooms(rno: integer, address: string, capacity: integer)*
*Enrolled(sid: string, cid: string, grade: string)*
*Teaches(fid: string, cid: string)*
*Meets In(cid: string, rno: integer, time: string)*

<u>Degree and Cardinality of the Relation:</u>
**The degree, also called arity, of a relation is the number of fields. The cardinality of a relation instance is the number of tuples in it.** In Figure 3.1, the degree of the relation (the number of columns) is five, and the cardinality of this instance is six.

<u>Relation Instance:</u>
The **relation instance** is a table, and the relation schema describes the column heads for the table. We first describe the relation schema and then the relation instance.
**An instance of a relation is a set of tuples, also called records, in which each tuple has the same number of fields as the relation schema. A relation instance can be thought of as a table in which each tuple is a row, and all rows have the same number of fields. The term relation instance is often abbreviated to just relation, when there is no confusion with other aspects of a relation such as its schema.**

An instance of the Students relation appears in Figure 3.1. The instance S1 contains six tuples and has, as we expect from the schema, five fields.



| *sid* | *name* | *login* | *age* | *gpa* |
|---|---|---|---|---|
| 50000 | Dave | dave@cs | 19 | 3.3 |
| 53666 | Jones | jones@cs | 18 | 3.4 |
| 53688 | Smith | smith@ee | 18 | 3.2 |
| 53650 | Smith | smith@math | 19 | 3.8 |
| 53831 | Madayan | madayan@music | 11 | 1.8 |
| 53832 | Guldu | guldu@music | 12 | 2.0 |

**Figure 3.1** An Instance *S*1 of the Students Relation

অথবা

**Exercise 3.1 Define the following terms: relation schema, relational database schema, domain, attribute, attribute domain, relation instance, relation cardinality,and relation degree.**

Answer 3.1 A relation schema can be thought of as the basic information describing a table or relation. This includes a set of column names, the data types associated with each column, and the name associated with the entire table. For example, a relation schema for the relation called Students could be expressed using the following representation:

*Students(sid: string , name: string , login: string , age: integer , gpa: real )*

There are five fields or columns, with names and types as shown above.
A relational database schema is a collection of relation schemas, describing one or more relations.

Domain is synonymous with data type. Attributes can be thought of as columns in a table. Therefore, an attribute domain refers to the data type associated with a column.

A relation instance is a set of tuples (also known as rows or records) that each conform to the schema of the relation.

The relation cardinality is the number of tuples in the relation.

The relation degree is the number of fields (or columns) in the relation.

**Question-07: What is a foreign key constraint? Why are such constraints important? What is referential integrity? 5 Marks**

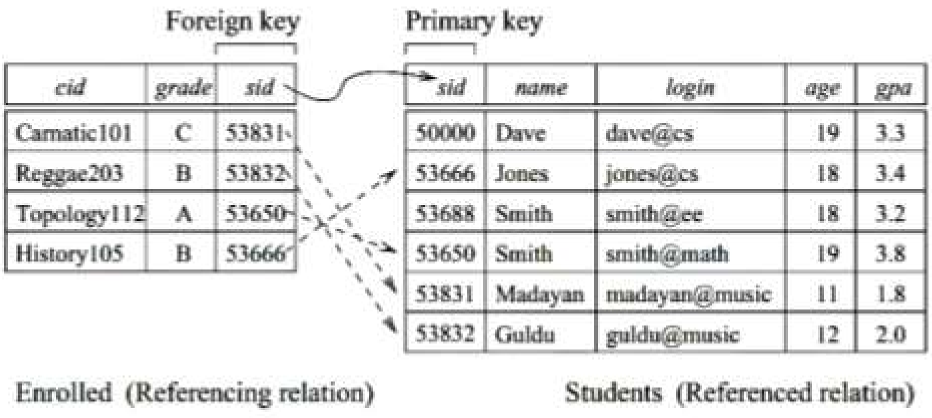
**<u>Foreign Key Constraints:</u>**

Sometimes the information stored in a relation is linked to the information stored in another relation. If one of the relations is modified, the other must be checked, and perhaps modified, to keep the data consistent. An Integrity Constraints involving both relations must be specified if a DBMS is to make such checks. The most common Integrity Constraint involving two relations is a **foreign key constraint**.

Suppose that in addition to Students, we have a second relation:

***Students(sid: string, name: string, login: string, age: integer, gpa: real)***
***Enrolled(sid: string, cid: string, grade: string)***

To ensure that only bona fide students can enroll in courses, any value that appears in the ***sid*** field of an instance of the Enrolled relation should also appear in the ***sid*** field of some tuple in the Students relation. The ***sid*** field of Enrolled is called a **foreign key** and refers to Students. **The foreign key in the referencing relation (Enrolled, in our example) must match the primary key of the referenced relation** (Students), i.e., it must have the same number of columns and compatible data types, although the column names can be different.
This constraint is illustrated in Figure 3.4.


Foreign key
Primary key
Enrolled (Referencing relation)
Students (Referenced relation)

| *cid* | *grade* | *sid* |
|---|---|---|
| Carnatic101 | C | 53831 |
| Reggae203 | B | 53832 |
| Topology112 | A | 53650 |
| History105 | B | 53666 |

| *sid* | *name* | *login* | *age* | *gpa* |
|---|---|---|---|---|
| 50000 | Dave | dave@cs | 19 | 3.3 |
| 53666 | Jones | jones@cs | 18 | 3.4 |
| 53688 | Smith | smith@ee | 18 | 3.2 |
| 53650 | Smith | smith@math | 19 | 3.8 |
| 53831 | Madayan | madayan@music | 11 | 1.8 |
| 53832 | Guldu | guldu@music | 12 | 2.0 |

**Figure 3.4** Referential Integrity

**<u>Referential Integrity:</u>**

**Often, we wish to ensure that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation. This condition is called referential integrity. Referential-integrity constraints ensure that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.**
Referential-integrity constraints arise frequently. If we derive our relational-database schema by constructing tables from E-R diagrams, then every relation arising from a relationship set has referential-integrity constraints. Figure 6.1 shows an n-ary relationship set R, relating entity sets

E1 ,E2 ,...,En.Let Ki denote the primary key of Ei. The attributes of the relation schema for relationship set R include K1 ∪ K2 ∪ ··· ∪ Kn. The following referential integrity constraints are then present: For each i, Ki in the schema for R is a foreign key referencing Ki in the relation schema generated from entity set Ei.

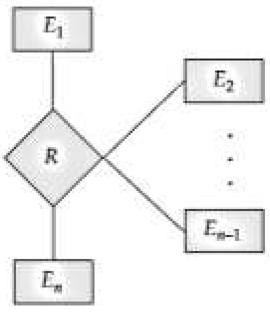


**Figure 6.1** An n-ary relationship set.

Another source of referential-integrity constraints is weak entity sets. The relation schema for a weak entity set must include the primary key of the entity set on which the weak entity set depends. Thus, the relation schema for each weak entity set includes a foreign key that leads to a referential-integrity constraint.

**Question-08: What is view? Why do we define view? 3 + 2 Marks CSE-2011 CSE-2006 CSE -2001**

**View:**

**A view is a relation/table whose instance/rows are not explicitly stored in the database but are computed as needed from a view definition.** In addition to enabling logical data independence by defining the external schema through views, views play an important role in restricting access to data for security reasons.

Consider the Students*(sid,,name, login, age, gpa*) and Enrolled(*sid, cid, grade*) relations. Suppose that we are often interested in finding the names and student identifiers of students who got a grade of B in some course, together with the *cid f*or the course. We can define a view for this purpose. Using SQL-92 notation:

```
CREATE VIEW B-Students (name, sid, course)
        AS SELECT S.sname, S.sid, E.cid
        FROM Students S, Enrolled E
        WHERE S.sid = E.sid AND E.grade = `B'
```

**Why we define View:**

The **physical schema** for a relational database describes how the relations in the conceptual schema are stored, in terms of the file organizations and indexes used. The **conceptual schema** is the collection of schemas of the relations stored in the database. While some relations in the conceptual schema can also be exposed to applications, i.e., be part of the external schema of the database, additional relations in the external schema can be defined using the view mechanism.

1. **The view mechanism thus provides the support for logical data independence in the relational model. That is, it can be used to define relations in the external schema that mask changes in the conceptual schema of the database from applications.**
   **For example**, if the schema of a stored relation is changed, we can define a view with the old schema, and applications that expect to see the old schema can now use this view.

2. **Views are also valuable in the context of security: We can define views that give a group of users access to just the information they are allowed to see.**
   **For example**, we can define a view that allows students to see other students' name and age but not their gpa, and allow all students to access this view, but not the underlying Students table.

**Question-09: "The Database Design process can be divided into six steps" – Please explain the steps in short. 7 Marks CSE-2013**

<u>**Steps in Database Design Process**</u>

The database design process can be divided into six steps. The ER model is most relevant to the first three steps:

**(1) Requirements Analysis:** The very first step in designing a database application is to understand what data is to be stored in the database, what applications must be built on top of it, and what operations are most frequent and subject to performance requirements. In other words, we must find out what the users want from the database. This is usually an informal process that involves discussions with user groups, a study of the current operating environment and how it is expected to change, analysis of any available documentation on existing applications that are expected to be replaced or complemented by the database, and so on. Several methodologies have been proposed for organizing and presenting the information gathered in this step, and some automated tools have been developed to support this process.

**(2) Conceptual Database Design:** The information gathered in the requirements analysis step is used to develop a high-level description of the data to be stored in the database, along with the constraints that are known to hold over this data. This step is often carried out using the ER model, or a similar high-level data model.

**(3) Logical Database Design:** We must choose a DBMS to implement our database design, and convert the conceptual database design into a database schema in the data model of the chosen DBMS. We will only consider relational DBMSs, and therefore, the task in the logical design step is to convert an ER schema into a relational database schema. The result is a conceptual schema, sometimes called the logical schema, in the relational data model.

**(4) Schema Refinement:** The fourth step in database design is to analyse the collection of relations in our relational database schema to identify potential problems, and to refine it. In contrast to the requirements analysis and conceptual design steps, which are essentially subjective, schema refinement can be guided by some elegant and powerful theory.

**(5) Physical Database Design:** In this step we must consider typical expected workloads that our database must support and further refine the database design to ensure that it meets desired performance criteria. This step may simply involve building indexes on some tables and clustering some tables, or it may involve a substantial redesign of parts of the database schema obtained from the earlier design steps.

**(6) Security Design:** In this step, we identify different user groups and different roles played by various users (e.g., the development team for a product, the customer support representatives, the product manager). For each role and user group, we must identify the parts of the database that they must be able to access and the parts of the database that they should not be allowed to access, and take steps to ensure that they can access only the necessary parts. A DBMS provides several mechanisms to assist in this step.

**Question-10: What is Trigger and why they are used? 5 Marks CSE-2012 CSE-2004 CSE-2003 (OLD)**

<u>**Trigger:**</u>

A trigger is a statement that the system executes automatically as a side effect of a modification to the database. To design a trigger mechanism, we must meet two requirements:

1. **Specify when a trigger is to be executed.** This is broken up into an event that causes the trigger to be checked and a condition that must be satisfied for trigger execution to proceed.
2. **Specify the actions to be taken when the trigger executes.**

The database stores triggers just as if they were regular data, so that they are persistent and are accessible to all database operations. Once we enter a trigger into the database, the database system takes on the responsibility of executing it whenever the specified event occurs and the corresponding condition is satisfied.

**Need For Triggers:**

1. **Triggers are useful mechanisms for alerting humans or for starting certain tasks automatically when certain conditions are met.**
2. A common use of triggers is to maintain database consistency. Triggers allow us to maintain database integrity in more flexible ways
3. Many potential uses of triggers go beyond integrity maintenance. Triggers can alert users to unusual events (as reflected in updates to the database). For example, we may want to check whether a customer placing an order has made enough purchases in the past month to qualify for an additional discount; if so, the sales clerk must be informed so that he can tell the customer, and possibly generate additional sales! We can relay this information by using a trigger that checks recent purchases and prints a message if the customer qualifies for the discount.
4. Triggers can generate a log of events to support auditing and security checks.
5. We can use triggers to gather statistics on table accesses and modifications. Some database systems even use triggers internally as the basis for managing replicas of relations.
6. Triggers have also been considered for workflow management and enforcing business rules.

**Question-11: Describe the basic parts of a trigger and show the syntax for creating a trigger. 6 Marks CSE-2003 (OLD) CSE-2002 CSE-2001 CSE-2000 CSE-2000 (OLD)**

**Basic Parts of Trigger:**

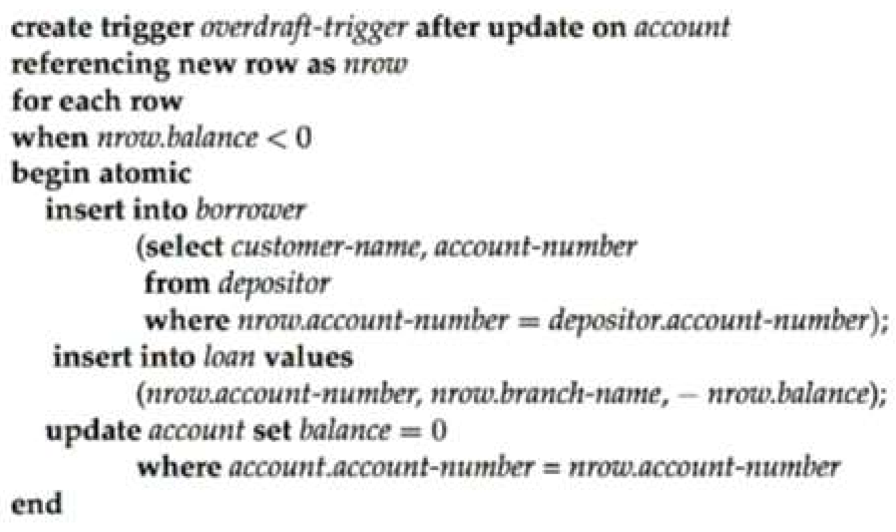
**A trigger is a procedure that is automatically invoked by the DBMS in response to specified changes to the database, and is typically specified by the DBA. A database that has a set of associated triggers is called an active database. A trigger description contains three parts:**

- **Event:** A change to the database that activates the trigger. A trigger can be thought of as a `daemon' that monitors a database, and is executed when the database is modified in a way that matches the event specification. An insert, delete or update statement could activate a trigger, regardless of which user or application invoked the activating statement; users may not even be aware that a trigger was executed as a side effect of their program.

- **Condition:** A query or test that is run when the trigger is activated. A condition in a trigger can be a true/false statement (e.g., all employee salaries are less than $100,000) or a query. A query is interpreted as true if the answer set is nonempty, and false if the query has no answers. If the condition part evaluates to true, the action associated with the trigger is executed.

- **Action:** A procedure that is executed when the trigger is activated and its condition is true. A trigger action can examine the answers to the query in the condition part of the trigger, refer to old and new values of tuples modified by the statement activating the trigger, execute new queries, and make changes to the database. In fact, an action can even execute a series of data-definition commands (e.g., create new tables, change authorizations) and transaction-oriented commands (e.g., commit), or call host-language procedures.

**Syntax for creating a trigger:**



```
create trigger overdraft-trigger after update on account
referencing new row as nrow
for each row
when nrow.balance < 0
begin atomic
   insert into borrower
          (select customer-name, account-number
           from depositor
           where nrow.account-number = depositor.account-number);
    insert into loan values
          (nrow.account-number, nrow.branch-name, – nrow.balance);
   update account set balance = 0
          where account.account-number = nrow.account-number
end
```

**Figure 6.3** Example of SQL:1999 syntax for triggers.

**Question-12: What Security measures do we need to be taken at several levels to protect database? 4 Marks CSE-2003 CSE-2002**

<u>**Database Security:**</u>

Database security refers to protection from malicious access. Absolute protection of the database from malicious abuse is not possible, but the cost to the perpetrator can be made high enough to deter most if not all attempts to access the database without proper authority.

<u>**Security measures to be taken at several levels to protect database:**</u>

To protect the database, we must take security measures at several levels:

- **Database system:** Some database-system users may be authorized to access only a limited portion of the database. Other users may be allowed to issue queries, but may be forbidden to modify the data. It is the responsibility of the database system to ensure that these authorization restrictions are not violated.
- **Operating system:** No matter how secure the database system is, weakness in operating-system security may serve as a means of unauthorized access to the database.
- **Network:** Since almost all database systems allow remote access through terminals or networks, software-level security within the network software is as important as physical security, both on the Internet and in private networks.
- **Physical**: Sites with computer systems must be physically secured against armed or surreptitious entry by intruders.
- **Human**: Users must be authorized carefully to reduce the chance of any user giving access to an intruder in exchange for a bribe or other favors.

Security at all these levels must be maintained if database security is to be ensured.

**Question-13: Define Candidate key, Foreign Key and Assertion. 6 Marks CSE-2009**

<u>**Candidate Key:**</u>

A set of fields that uniquely identifies a tuple according to a key constraint is called a **candidate key** for the relation; we often abbreviate this to just **key**. There are two parts to the definition:

1. Two distinct tuples in a legal instance (an instance that satisfies all ICs, including the key constraint) cannot have identical values in all the fields of a key.
2. No subset of the set of fields in a key is a unique identifier for a tuple.

**Example:** In the case of the Students relation, the (set of fields containing just the) sid field is a candidate key.

**Foreign Key:**

**A relation schema, say $r_1$ , derived from an E-R schema may include among its attributes the primary key of another relation schema, say $r_2$ . This attribute is called a foreign key from $r_1$ , referencing $r_2$ .The relation $r_1$ is also called the referencing relation of the foreign key dependency, and $r_2$ is called the referenced relation of the foreign key.**

**For example,**
Let us consider **Account-schema and Branch-schema**:

*Account-schema =(account-number, branch-name, balance)*
*Branch-schema = (branch-name, branch-city, assets)*

the attribute **branch-name** in **Account-schema** is a foreign key from **Account-schema** referencing **Branch-schema**, since **branch-name** is the primary key of **Branch-schema**.

**Assertion:**

**An assertion is a predicate expressing a condition that we wish the database always to satisfy**. Domain constraints and referential-integrity constraints are special forms of assertions.

**An assertion in SQL takes the form:**

***create assertion*** *<assertion-name>* ***check*** *<predicate>*

When an assertion is created, the system tests it for validity. If the assertion is valid, then any future modification to the database is allowed only if it does not cause that assertion to be violated. This testing may introduce a significant amount of overhead if complex assertions have been made. Hence, assertions should be used with great care. The high overhead of testing and maintaining assertions has led some system developers to omit support for general assertions, or to provide specialized forms of assertions that are easier to test.

**Question-14: What are the differences between an assertion and a trigger? 5 Marks CSE-2012 CSE-2004 CSE-2003 (OLD) CSE-2001**

**Differences between an Assertion and a Trigger:**
Following are some differences between an assertion and a trigger:

| Assertion | Trigger |
|---|---|
| An **assertion** is a predicate expressing a condition that we wish the database always to satisfy. | A trigger is a statement that the system executes automatically as a side effect of a modification to the database. |
| An assertion in SQL takes the form:<br>**create assertion** <assertion-name> **check** <predicate> | **A Trigger in SQL takes the form:**<br>**CREATE TRIGGER** triggerName<br>**AFTER UPDATE**<br>**INSERT INTO** CustomerLog (blah, blah, blah)<br>**SELECT** blah, blah, blah **FROM** deleted |
| When an assertion is created, the system tests it for validity. If the assertion is valid, then any future modification to the database is allowed only if it does not cause that assertion to be violated. | Once we enter a trigger into the database, the database system takes on the responsibility of executing it whenever the specified event occurs and the corresponding condition is satisfied. |
|  | The database stores triggers just as if they were regular data, so that they are persistent and are accessible to all database operations. |
|  | Triggers are useful mechanisms for alerting humans or for starting certain tasks automatically when certain conditions are met. |

| | |
|---|---|
| Assertions do not modify the data, they only check certain conditions. | Triggers are more powerful because the can check conditions and also modify the data. |
| Assertions are not linked to specific tables in the database and not linked to specific events. | Triggers are linked to specific tables and specific events. |
| Domain constraints and referential-integrity constraints are special forms of assertions. | |

**Question-15: Discuss the strengths and weaknesses of the trigger mechanism. 1.75 Marks CSE-2013**

**Strength of Trigger Mechanism:**
Following are some strength of trigger mechanism:

1. **Triggers are useful mechanisms for alerting humans or for starting certain tasks automatically when certain conditions are met**.
2. A common use of triggers is to maintain database consistency. Triggers allow us to maintain database integrity in more flexible ways
3. Many potential uses of triggers go beyond integrity maintenance. Triggers can alert users to unusual events (as reflected in updates to the database). For example, we may want to check whether a customer placing an order has made enough purchases in the past month to qualify for an additional discount; if so, the sales clerk must be informed so that he can tell the customer, and possibly generate additional sales! We can relay this information by using a trigger that checks recent purchases and prints a message if the customer qualifies for the discount.
4. Triggers can generate a log of events to support auditing and security checks.
5. We can use triggers to gather statistics on table accesses and modifications. Some database systems even use triggers internally as the basis for managing replicas of relations.
6. Triggers have also been considered for workflow management and enforcing business rules.

**Weaknesses of Trigger Mechanism:**

Following are some weaknesses of trigger mechanism:

1. Triggers should be written with great care, since a trigger error detected at run time causes the failure of the insert/delete/update statement that set off the trigger.
2. Furthermore, the action of one trigger can set off another trigger. In the worst case, this could even lead to an infinite chain of triggering. For example, suppose an insert trigger on a relation has an action that causes another (new) insert on the same relation. The insert action then triggers yet another insert action, and so on ad infinitum.
3. Database systems typically limit the length of such chains of triggers.

**Question-16: What is access Control? What are the approaches to access control? Discuss in brief.**

**Access Control:**

A database for an enterprise contains a great deal of information and usually has several groups of users. Most users need to access only a small part of the database to carry out their tasks. Allowing users unrestricted access to all the data can be undesirable, and a DBMS should provide mechanisms to control access to data. An access control mechanism is a way to control the data that is accessible to a given user.

**Approaches to access control:**

A DBMS offers two main approaches to access control:

1. Discretionary Access Control
2. Mandatory Access Control.

**Discretionary access control:**

Discretionary access control is based on the concept of access rights, or privileges, and mechanisms for giving users such privileges. A privilege allows a user to access some data object in a certain manner (e.g., to read or to modify). A user who creates a database object such as a table or a view automatically gets all applicable privileges on that object. The DBMS subsequently keeps track of how these privileges are granted to other users, and possibly revoked, and ensures that at all times only users with the necessary privileges can access an object. SQL-92 supports discretionary access control through the GRANT and REVOKE commands. The **GRANT** command gives privileges to users, and the **REVOKE** command takes away privileges.

The syntax of **GRANT** and **REVOKE command** is as follows:

```
GRANT privileges ON object TO users [ WITH GRANT OPTION ]
REVOKE [ GRANT OPTION FOR ] privileges
          ON object FROM users f RESTRICT j CASCADE g
```

**Several privileges can be specified, including these:**

**SELECT:** The right to access (read) all columns of the table specified as the object, including columns added later through ALTER TABLE commands.

**INSERT(column-name):** The right to insert rows with (non-null or nondefault) values in the named column of the table named as object. If this right is to be granted with respect to all columns, including columns that might be added later, we can simply use INSERT. The privileges UPDATE(column-name)and UPDATE are similar.

**DELETE:** The right to delete rows from the table named as object.

**REFERENCES(column-name):** The right to define foreign keys (in other tables) that refer to the specified column of the table object. REFERENCES without a column name specified denotes this right with respect to all columns, including any that are added later.

Discretionary access control mechanisms, while generally effective, have certain weaknesses. In particular, a devious unauthorized user can trick an authorized user into disclosing sensitive data.

**Mandatory Access Control:**

Mandatory access control is based on system wide policies that cannot be changed by individual users. In this approach each database object is assigned a security class, each user is assigned clearance for a security class, and rules are imposed on reading and writing of database objects by users. The DBMS deter mines whether a given user can read or write a given object based on certain rules that involve the security level of the object and the clearance of the user. These rules seek to ensure that sensitive data can never be `passed on' to a user without the necessary clearance. The SQL-92 standard does not include any support for mandatory access control.

The popular model for mandatory access control, called the Bell-LaPadula model, is described in terms of objects (e.g., tables, views, rows, columns), subjects (e.g., users, programs), security classes, and clearances. Each database object is assigned a security class, and each subject is assigned clearance for a security class; we will denote the class of an object or subject A as class(A).The security classes in a system are organized according to a partial order, with a most secure class and a least secure class. For simplicity, we will assume that there are four classes: top secret (TS), secret (S), confidential (C), and unclassified (U). In this system, TS > S > C > U, where A > B means that class A data is more sensitive than class B data.

**The Bell-LaPadula model imposes two restrictions on all reads and writes of database objects:**

**1. Simple Security Property:** Subject S is allowed to read object O only if class(S) ≥ class(O). For example, a user with TS clearance can read a table with C clearance, but a user with C clearance is not allowed to read a table with TS classification.

**2. *-Property:** Subject S is allowed to write object O only if class(S) ≤ class(O).For example, a user with S clearance can only write objects with S or TS classification.

**Question-17:** Define Encryption. **What are the advantages of encrypting data stored in the database? 4 Marks CSE-2003 CSE-2002 CSE-2012**

**Encryption:**

In cryptography, encryption is the process of encoding messages or information in such a way that only authorized parties can read it. Encryption doesn't prevent hacking but it reduces the likelihood that the hacker will be able to read the data that is encrypted. In an encryption scheme, the message or information, referred to as plaintext, is encrypted using an encryption algorithm, turning it into an unreadable ciphertext. This is usually done with the use of an encryption key, which specifies how the message is to be encoded. Any adversary that can see the ciphertext should not be able to determine anything about the original message. An authorized party, however, is able to decode the ciphertext using a decryption algorithm, that usually requires a secret decryption key, that adversaries do not have access to. For technical reasons, an encryption scheme usually needs a key-generation algorithm to randomly produce keys. Encryption forms the basis of good schemes for authenticating users to a database.

**There are two types of encryption**

1. Symmetric key encryption
2. Public key encryption

**Advantages of encrypting data stored in the database:**

Following are some advantages of encrypting data stored in the database:

- Ensure guaranteed access to encrypted data by authorized users by automating storage and back-up for mission critical master encryption keys.
- Simplify data privacy compliance obligations and reporting activities through the use of a security-certified encryption and key management to enforce critical best practices and other standards of due care.

- Enforce separation of duties by isolating master encryption keys from encrypted data—reducing the threat of insider attacks.
- Maximize efficiency by reducing administration costs associated with managing keys in large-scale database environments with Thales' industry-leading Security World key management architecture.
- Deploy with confidence and accelerate implementation projects; Thales HSMs integrate easily with leading database management systems, featuring out-of-the-box integration with Transparent Data Encryption from Microsoft, and integration with other leading DBMS solutions via technology partners including Voltage and Prime Factors.

**Question-18: What is the role of the DBA with respect to security? 4 Marks**

**<u>Role of the DBA with respect to security:</u>**

The database administrator (DBA) plays an important role in enforcing the security-related aspects of a database design. In conjunction with the owners of the data, the DBA will probably also contribute to developing a security policy. The DBA has a special account, which we will call the system account, and is responsible for the overall security of the system. In particular the DBA deals with the following:

1. **Creating new accounts:** Each new user or group of users must be assigned an authorization id and a password. Note that application programs that access the database have the same authorization id as the user executing the program.

2. **Mandatory control issues:** If the DBMS supports mandatory control some customized systems for applications with very high security requirements (for example, military data) provide such support the DBA must assign security classes to each database object and assign security clearances to each authorization id in accordance with the chosen security policy.

The DBA is also responsible for maintaining the **audit trail**, which is essentially the log of updates with the authorization id (of the user who is executing the transaction) added to each log entry. This log is just a minor extension of the log mechanism used to recover from crashes. Additionally, the DBA may choose to maintain a log of all actions, including reads, performed by a user. Analysing such histories of how the DBMS was accessed can help prevent security violations by identifying suspicious patterns before an intruder finally succeeds in breaking in, or it can help track down an intruder after a violation has been detected.

In a nutshell, The DBA creates new accounts, ensures that passwords are safe and changed often, assigns mandatory access control levels, and can analyse the audit trail to look for security breaches. They can also assist users with their discretionary permissions.

# About the Authors

**Abu Saleh Musa Miah(abid) was born in Rangpur,Bangladesh in 1992**. He passed the M.Sc. and B.Sc. Engineering degree with Honours in Computer science and Engineering with FIRST merit Position Engineering first batch from University of Rajshahi ,Bangladesh. He also completed research on the field **Brain Computer Interfacing, Computer vision** specifically Blind source Separation. Currently he is working towards the Doctor of Philosophy(Ph.D) degree in the Department of Computer Science And Engineering University of Rajshahi, Bangladesh.

His research interests include **Brain Computer Interfacing**. Sparse signal recovery/compressed sensing, blind source separation, neuroimaging and computational and cognitive neuroscience.

**Email: abusalehcse.ru@gmail.com;**

**https://www.facebook.com/abusalehmusa.miah**