Unit 1 (Data Files and DBMS)

Data

1: Data refers to information or facts that are processed or stored by a computer system.
2: It can be any form of raw or unprocessed facts, numbers, text, images, audio, video, or other types of digital content.
3: In computer science, data is processed and manipulated using algorithms and programming languages.
4: Data is the foundation of computing and is used to perform various operations, computations, and analyses.
5: Various techniques and technologies, such as data structures, databases, data mining, and big data analytics, are employed to extract insights, patterns, and knowledge from data.

Files

1: A file is a named collection of data that is stored on a computer's storage device, such as a hard drive, solid-state drive, or network storage.

2: Files are used to organize and store information in a structured manner, making it easily accessible and retrievable by the computer's operating system and applications.

3: A file can contain various types of data, including text, images, audio, video, executable code, configuration settings, and more.

4: Files have attributes associated with them, such as file size, file type (extension), creation date, modification date, and permissions that control who can access or modify the file.

Operations of Files

Retrieval operation

1: Retrieval operation refers to the process of accessing and retrieving data from a file.

2: It involves reading the contents of a file and transferring them from the storage device to the computer's memory for further processing or display.

3: The retrieval operation allows programs and users to access the data stored in files, enabling them to view, analyze, or utilize the information contained within.

Update operation

Update operation changes the file by modifying the records, deleting the records and inserting new records. That is we actually modify the contents in file in the update operation.

Some common operations on update:

Find (Locate): The goal of this operation is to locate the record or records that

satisfy the search criteria. A block that contains the records is transferred to the

main memory and the records are searched. The first record that matches the search criteria is located and the search continues for other records until the end of file is reached.

Read: In read operation the contents of the records are copied from memory to a program variable or work area. This command in some cases advances the pointer to next record in the result set.

Read Next: Searches the next record that matches the selection condition and when found, the contents of the record are copied to the program variable or work area.

Modify: Also known as update. This command modifies the field values / data

of the current record then writes the modified record back to the disk.

Insert: Inserts a new record to the file.

Delete: Deletes the current record and updates the file on the disk to reflect the deletion.

Components of Database

Schema: A schema defines the logical structure and organization of a database.

Constraint: A constraint defines rules and conditions that must be met by the data stored in a database.

Relationship: A relationship represents an association or connection between two or more entities in a database.

Data item: A data item is the smallest unit of data in a database. It represents an indivisible piece of information.

Record: A record, also known as a row or tuple, is a collection of related data items representing a complete set of attributes for a specific instance or entity.

instance: It represents a unique entry in a table that corresponds to a particular entity. For example, a specific employee record in an employee table is an instance of the entity "employee".

Domain: A domain defines the set of possible values that an attribute can have. It represents the range of valid data for a specific attribute.

Attribute: An attribute is a characteristic or property of an entity

Entity: An entity represents a distinct object, concept, or item in the real world that is relevant to the database.

Relation "**Customer Details**"

Attributes	**Cust_name**	**cust_id**	**Cust_add**	**Balance**
Tuple/ Record	Jyoti	A123	Nashik	50000
	Sarika	B125	Pune	45000
	Manisha	C222	Nashik	30000

In above table / relation

Entity = Customer

Attributes= cust_name, cust_id, cust_add, balance.

Domains= cust_name (jyoti, sarika, manisha), cust_id (A123, B125, C222) etc.

Record / tuple= jyoti, A123, Nashik, 50000.

Unit 2 (Introduction to DBMS)

Definition of DBMS

1: DBMS stands for Database Management System.
2: It is a software system that facilitates the creation, organization, storage, retrieval, modification, and management of databases.
3: DBMS serves as an intermediary between users and the database, providing an interface to interact with the data stored in the database.
4: A DBMS offers a set of tools, functionalities, and services to efficiently store and manage large volumes of structured or unstructured data.
5: DBMS are widely used in various domains such as businesses, research, finance and healthcare to efficiently manage and process large amounts of data.
6: Some popular examples of DBMSs include Oracle Database, MySQL, Microsoft SQL Server, PostgreSQL, and MongoDB.

Features of DBMS that not available in File system

Transaction Management: Transaction management deals with the handling and execution of database transactions.

Concurrency Control: Concurrency control in DBMS deals with managing simultaneous access to the database by multiple users or applications.

Recovery Management: Recovery management in DBMS deals with the mechanisms and techniques employed to recover the database from failures, errors, or crashes.

Security Management: Security management in DBMS focuses on protecting the database and its contents from unauthorized access, tampering, or misuse.

Language Interface: The language(programming language) interface component provides a means for users or applications to interact with the DBMS and perform various database operations.

Difference between file system and DBMS

Difference between File System and DBMS

The file system is basically a way of arranging the files in a storage medium like a hard disk. The file system organizes the files and helps in the retrieval of files when they are required. File systems consist of different files which are grouped into directories. The directories further contain other folders and files. The file system performs basic operations like management, file naming, giving access rules, etc.

https://www.geeksforgeeks.org/difference-between-file-system-and-dbms/amp/https://www.geeksforgeeks.org/difference-between-file-system-and-dbms/amp/

Limitations of File system

Data Redundancy: In a file system, data redundancy can occur when the same data is stored in multiple files or locations. This redundancy leads to increased storage requirements.

Data Inconsistency: File systems lack built-in mechanisms to ensure data consistency.

Limited Data Sharing and Integration: File systems are designed primarily for individual applications or users. Sharing and integrating data across multiple applications or users can be complex and prone to errors.

Lack of Query and Data Manipulation Capabilities: File systems offer limited querying and data manipulation capabilities.

Scalability and Performance Challenges: File systems may face scalability and performance limitations as the volume of data grows.

Maintenance and Data Independence: File systems require manual efforts for maintenance tasks like backup, recovery, and data reorganization.

Atomicity: it is difficult to ensure atomicity in file processing system. For example; in online banking system suppose you are transferring $1000 from Account A to account B. And if a failure occurs during transaction there could be situation like $1000 is deducted from Account A and not credited in Account B.

Applications of DBMS

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

Airlines: For reservations and schedule information.

Universities: For maintaining student information, course registrations and grades etc.

Credit card transactions: For purchases on credit cards and generation of monthly statements.

Telecommunications: 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 purchase of financial instruments such as stocks and bonds.

Sales: For maintaining 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 paychecks.

Web based services: For taking web users feedback, responses, resource sharing online shopping etc.

Data abstraction

1: Data abstraction refers to the process of simplifying complex data structures by providing a conceptual model that hides unnecessary details and focuses on the essential aspects of the data.

2: It allows users and applications to interact with data at a higher level of understanding without being concerned with the underlying implementation complexities.

3: In the context of databases, data abstraction is achieved through the use of schemas.

4: A schema defines the logical and conceptual structure of the database, providing a way to organize and represent data.

Abstraction using schemas

External Schema (also known as User Schema or View Level)

1: The external schema represents the view of the database from the perspective of individual users or applications.

2: It defines the subset of data that is relevant to a specific user or group of users.

3: The external schema provides a personalized and customized view of the database, hiding irrelevant information and presenting data in a format that is convenient for the user's requirements.

4: Multiple external schemas can exist for different users or applications accessing the same database.

Conceptual Schema (also known as Logical Schema or Conceptual Level):

1: The logical schema represents the overall logical structure of the entire database.

2: It describes the entities, attributes, relationships, and constraints of the data model used in the database.

3: The logical schema provides a high-level, integrated view of the data that is independent of the specific physical implementation details.

4: It acts as an intermediary between the external schemas and the internal schema, providing a global understanding of the database structure.

Internal Schema (also known as Physical Schema or Storage Level):

1: The internal schema represents the physical storage and implementation details of the database on the underlying hardware and operating system.

2: It defines how the data is stored, indexed, and accessed on disk or other storage devices.

3: The internal schema is concerned with performance optimization, data storage formats, indexing strategies, and other low-level details of data organization.

Database users

Application Programmers: Application programmer is the person who is responsible for implementing the required functionality of database for the end user.

End Users: End users are those persons who interact with the application directly. They are responsible to insert, delete and update data in the database.

Naive Users: Naive users are those users who do not have any technical knowledge about the DBMS. They use the database through application programs by using simple user interface. They perform all operations by using simple commands (menus and buttons) provided in the user interface.

Sophisticated Users: Sophisticated users are the users who are familiar with the structure of database and facilities of DBMS. Such users can use a query language such as SQL to perform the required operations on databases.

Database Administrator: Database administrator is responsible for, managing the whole database

system. He designs creates and maintains the database. He manages the users who can access this database, and controls integrity issues. He also monitors the performance of the system and makes changes in the system as and when required.

DDL

1: DDL stands for Data Definition Language. It is a subset of SQL (Structured Query Language) used in database management systems to define and manage the structure and schema of a database.

2: DDL statements are responsible for creating, modifying, and deleting database objects such as tables, indexes, views, constraints, and stored procedures.

Here are some common DDL statements:

CREATE - to create objects in the database.

ALTER - alters the structure of the database.

DROP - delete objects from the database.

TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed

COMMENT - add comments to the data dictionary.

RENAME - rename an object.

DML

1: DML stands for Data Manipulation Language.

2: It is a subset of SQL (Structured Query Language) used in database management systems to perform operations on the data stored in the database.

3: DML statements are used to insert, retrieve, update, and delete data in database tables.

Here are the main DML statements:

SELECT - retrieve data from the a database.

INSERT - insert data into a table.

UPDATE - updates existing data within a table.

DELETE - deletes all records from a table, the space for the records remain.

CALL - call a PL/SQL or Java subprogram.

Procedural DML

1: Procedural DML is an approach where the user or application specifies the step-by-step procedure to manipulate the data.

2: In procedural DML, the user defines the specific sequence of operations to be performed on the data.

3: Examples of procedural DML include programming languages like PL/SQL (used in Oracle) or T-SQL (used in Microsoft SQL Server) that allow for writing procedural code to manipulate data.

4: Without procedural DML, the user has fine-grained control over the data manipulation process.

Non procedural DML

1: Non-procedural DML, is an approach where the user specifies the desired outcome or result of the data manipulation operation, but not the specific steps to achieve it.

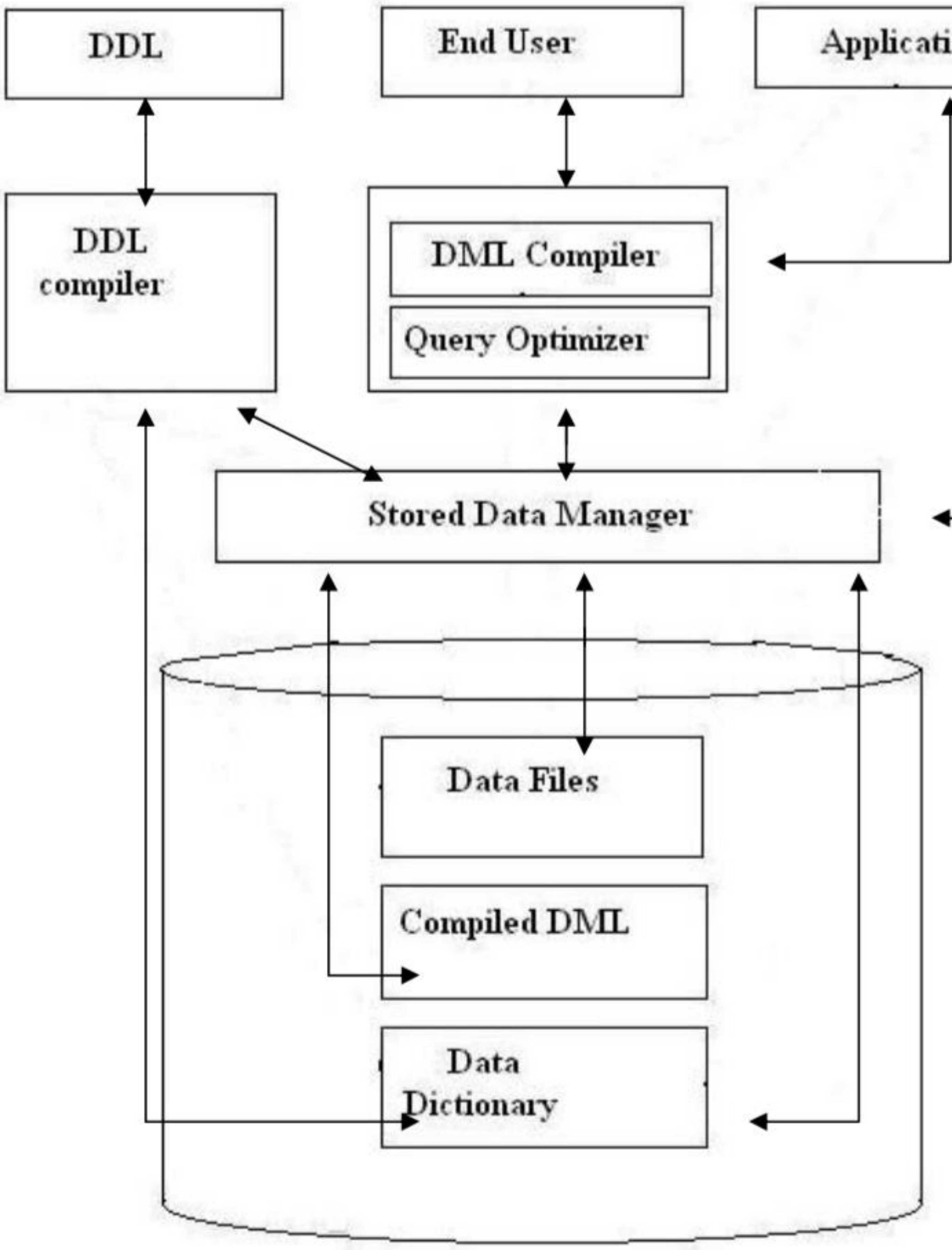
2: Non-procedural DML allows the user to specify what needs to be done rather than how it should be done.

3: The database management system is responsible for determining the most efficient way to execute the operation.

4: SQL (Structured Query Language) is an example of a non-procedural DML.

5: With non-procedural DML, the user focuses on the desired data results rather than the underlying implementation details.

Structure of DBMS


DDL
End User
Applicati
DDL compiler
DML Compiler
Query Optimizer
Stored Data Manager
Data Files
Compiled DML
Data Dictionary

Main functions of Data manager

1: Controls DBMS information access that is stored on disk.

2: It also controls handling buffers in main memory.

3: It also enforces constraints to maintain consistency and integrity of the data.

4: It also synchronizes the simultaneous operations performed by the concurrent users.

5: It also controls the backup and recovery operations.

Importance of data dictionary

1: It improves the control of DBA over the information system and user's understanding of use of the system.

2: It helps in documenting the database design process by storing documentation of the result of every design phase and design decisions.

3: It helps in searching the views on the database definitions of those views.

4: It provides great assistance in producing a report of which data elements (i.e. data values) are used in all the programs.

5: It promotes data independence i.e. by addition or modifications of structures in the database application program are not effected.

Metadata

1: Metadata refers to the data that provides information about other data.

2: It is essentially "data about data." Metadata describes various characteristics and properties of the data, such as its structure, format, content, location, relationships, and context.

3: It provides additional information that helps in understanding, managing, organizing, and interpreting the actual data.

4: Metadata is critical for data governance, data quality, data integration, data discovery, and overall data management.

Unit 3 (Relational Data Models and Relational Algebra)

Three types of data modes

Object-based logical models :

Object-based logical models are a type of data model that represents the structure and relationships of data using objects.

In these models, data is organized into objects, which are instances of classes or entities, and the relationships between these objects are defined using attributes and associations. Object based logical model includes following sub-data models:

i) The Entity-Relationship (E-R) model:

It is a conceptual data model that represents the structure of a database in terms of entities, attributes, and relationships. It provides a graphical representation of the database schema, which helps in understanding and designing the database.

For example: let consider an entity "student" which have attributes Roll_n
class etc and "book" is another entity which has attributes book_code, boo
price, author etc. And a Relation called "Buy" shows the relationship betw
two entities. This is because the student can buy book or a book can be bo
student.
Here 'M' indicates Many to Many cardinality. I.e. one student can purcha
than one book and more than one student can purchase one book.

This model can also be shown by graphical method, as follows.

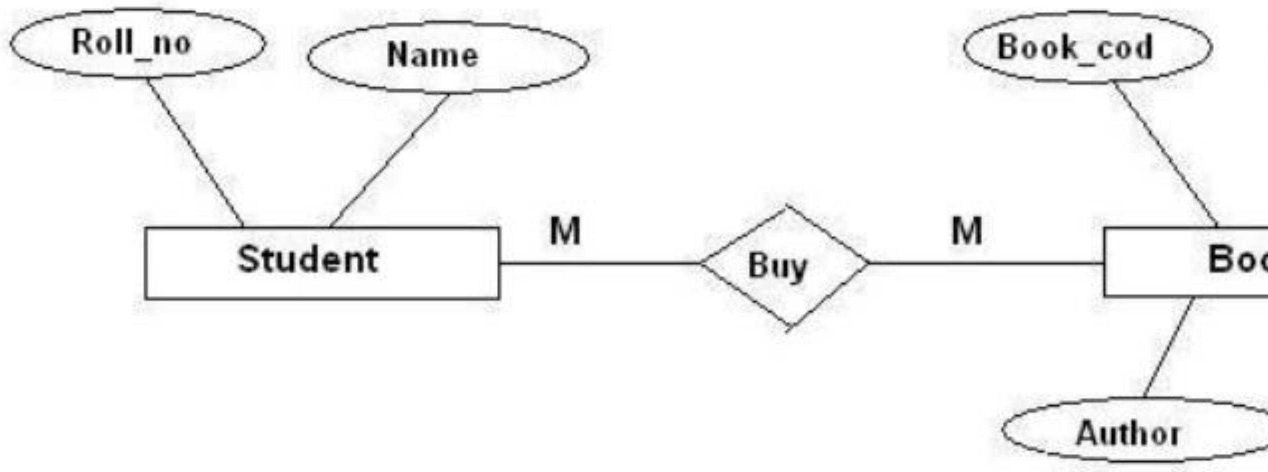


Figure2.1: A sample E-R diagram.

ii) object-oriented model:
In an object-oriented model, objects are the fundamental building blocks. An object represents a specific instance of a class, which is a blueprint or template defining the properties and behaviors of similar objects.
Record based Logical Models:
1:Record based logical model describe data at the conceptual and view levels.
Unlike object-oriented models, these models are used to specify overall logical structure of the database, and
2: Provide a higher-level description of the implementation.
3: It is named so because in this model the database is structured in fixed-format

records of several types where here each record type defines a fixed number of fields, or attributes.
The three types of record based models:
i)relational model
is a data model that organizes data into tables or relations.
It is based on mathematical set theory and logic, and it forms the foundation of most modern database management systems (DBMS).
ii) Network model
In the record-based network model, data is organized into records, which consist of fields or attributes.
Records are linked together through pointers or relationships to form complex networks.
The model allows for many-to-many relationships and supports more flexible data access compared to the hierarchical model.
iii) Hierarchical model
In a Hierarchical model, data is represented in the form of a tree. Data in a hierarchical model is represented by a collection of records, and relationships between the data are represented by links.
Physical Data Models:
1: A physical database model is similar to other data models which show all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships between tables.
2: This model specifies all tables and columns in database.
3:This model uses foreign keys to identify relationships between relations.
4: Because of some Physical considerations the physical data model is quite different from the logical data model.
5: Physical data model will be different for different databases. For example, In MySQL and SQL Server the data type we use for column may be different.
6: This model is used to describe data at the lowest level.

Integrity constraints

1: An integrity constraint is a condition that is enforced automatically by the

DBMS and that prevents the data from being stored in the database.

2: The DBMS enforces integrity constraints in that it only permits legal instances to be stored in the database.

3: The key constraint and the referential

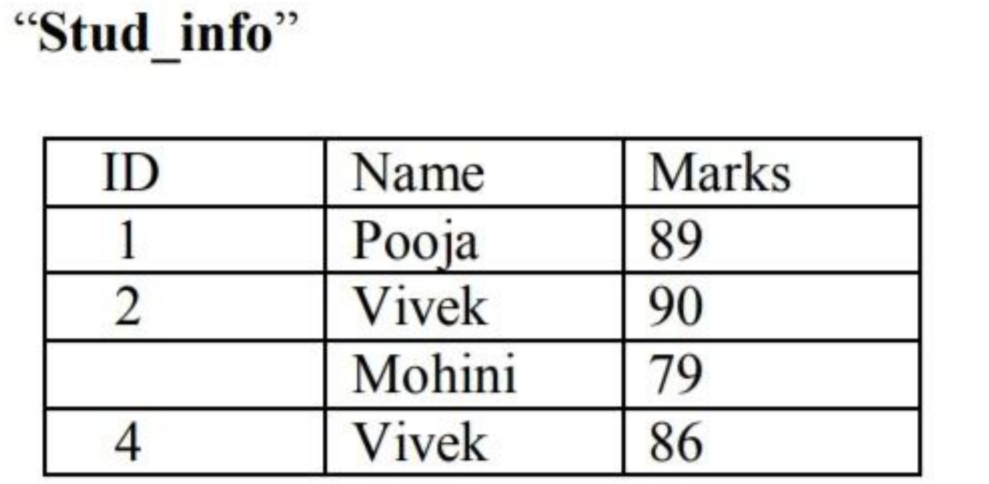
4: integrity constraints are identified as the two minimum constraints that must

be enforced by the DBMS.

NOT NULL Integrity Constraints

Null means the absence

of a value. A NOT NULL constraint requires a column of a table contain no null values.



“**Stud_info**”

ID	Name	Marks
1	Pooja	89
2	Vivek	90
	Mohini	79
4	Vivek	86

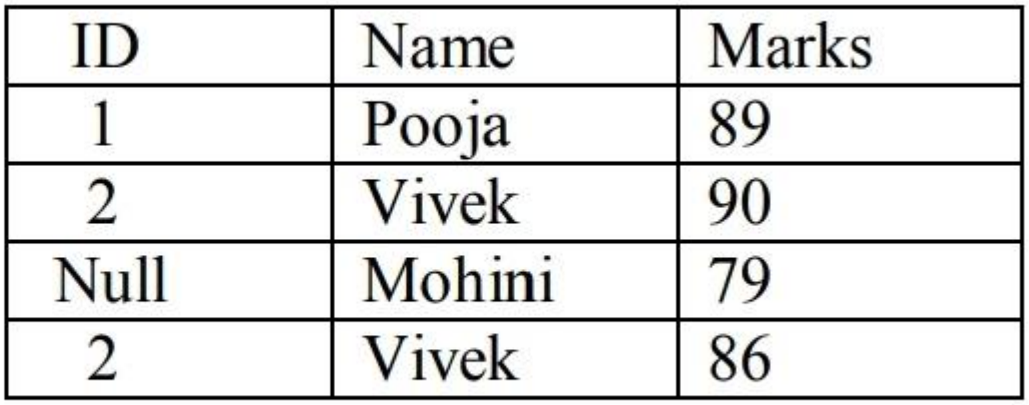
Figure 2.9: Example NOT NULL constraint

Entity Integrity constraint

The entity integrity constraint states that no primary key value can be null. This is

because the primary key value is used to identify individual tuples in a relation .

Having null value for the primary key implies that we cannot identify some tuples.



ID	Name	Marks
1	Pooja	89
2	Vivek	90
Null	Mohini	79
2	Vivek	86

Figure: 2.10 Relation "stud_info"

Referential Integrity or foreign key constraint

The referential integrity constraint is specified between two relations and is

used to maintain the consistency among tuples in the two relations. Informally, the

referential integrity constraint states that a tuple in one relation that refers to another

relation must refer to an existing tuple in that relation.

- Consider below example of a database that ha
integrity.

<u>"Stud"</u>

Stud_id	Stud_name
1	Pooja
2	Jaya
3	Akash
4	Viren

Delete record →

4	Viren

From " Stud"

○← ~~L~~ink broken

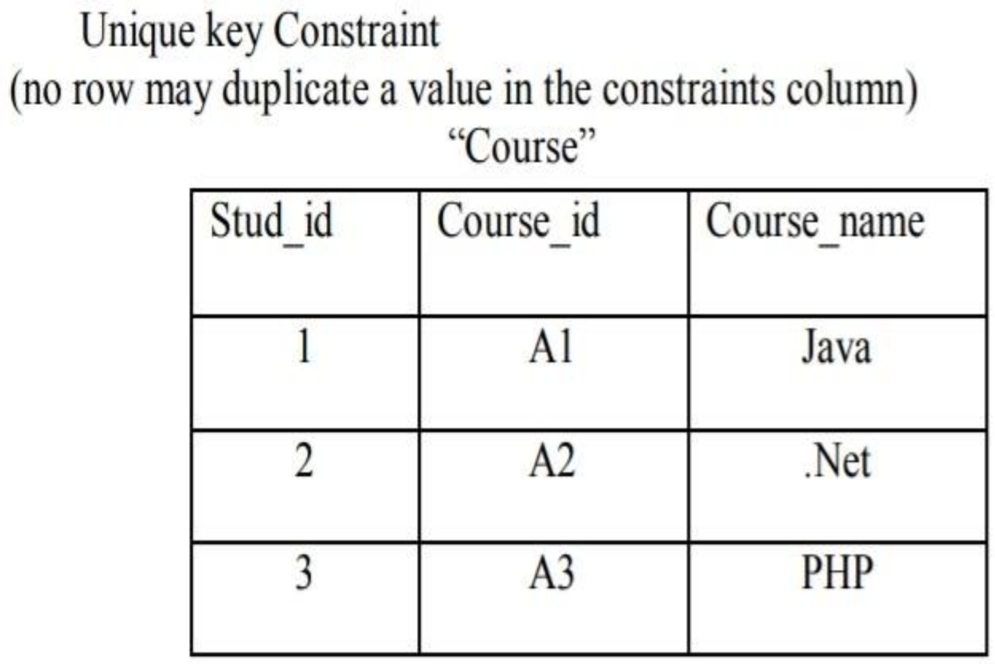
<u>"course"</u>

Stud_id	Course_id	C
3	A1	J

Unique Key Constraint

A UNIQUE key integrity constraint states that every value in a column or set

of columns (key) is unique—that is, no two rows of a table have duplicate values in a specified column or set of columns.



Unique key Constraint
(no row may duplicate a value in the constraints column)
"Course"

Stud_id	Course_id	Course_name
1	A1	Java
2	A2	.Net
3	A3	PHP

Figure: 2.13 Example of Unique key constraints

Unit 4 (Entity Relationship)

Entity Relationship

1: In a database management system (DBMS), an Entity-Relationship (ER) diagram is a graphical representation of the relationships between various entities within a database.
2: Entities are objects or concepts that have attributes and can be uniquely identified, such as customers, products, or orders.
3: Relationships depict how these entities interact, specifying associations like "owns," "is a part of," or "works with."
4: ER diagrams use symbols like rectangles for entities, diamonds for relationships, and lines to connect them.
5: These visual representations help designers and developers understand the structure of the database, facilitating the creation of efficient and well-organized databases for applications.

Data modelling

1: It is a Process, which is used to produce data model for describing data, its

relationships, and its constraints.

2: It also defines data elements, their structures and relationships between them.

3: We use data modeling to define and analyze data requirements that are needed.

4: It is a technique that is used to model data in a standard, reliable, expected manner so that it manages resources efficiently.

5: It is generally used in business and information technology to give a suggestion about how information is,

how it should be, how it is stored and used inside a business system.

Importance of Data models

1: It defines and analyzes data requirements.

2: It makes communication easy for application programmer, end user, and designer.

3: To define data elements, their structures and relationships between them.

4: It provides different view for data for end-users.

5: It organizes data for various users.

Three basic styles of data model

1:<u>Conceptual data models (CDM): </u>-

Conceptual data models also called domain models. It is a primary step in

constructing data model. It visualizes data structure of group.CDM is created by collecting all business requirement, .It contains all the main entities & relationships but does not consist of the complete information about attributes.

Eg. ER model

<u>2. Logical data models (LDMs): -</u>

LDMs means logical data model, which is used to give information of logical

entity types, the data attributes describing those entities, and the relationships between the entities. It includes all entities and relationships between them and also specifies all attributes for each entity .It also state Primary key & foreign key to identify relationship in entities.

Eg. Relational model

<u>3. Physical data models (PDMs): -</u>

PDMs means physical data model which explains all table structures, including column name, column data type, column constraints, primary key, foreign key, and relationships among tables. In this data model entities are transformed into tables. Relationship is transformed by using foreign key. Attribute is transformed into columns.

Eg. Table, Columns, Keys, Trigger

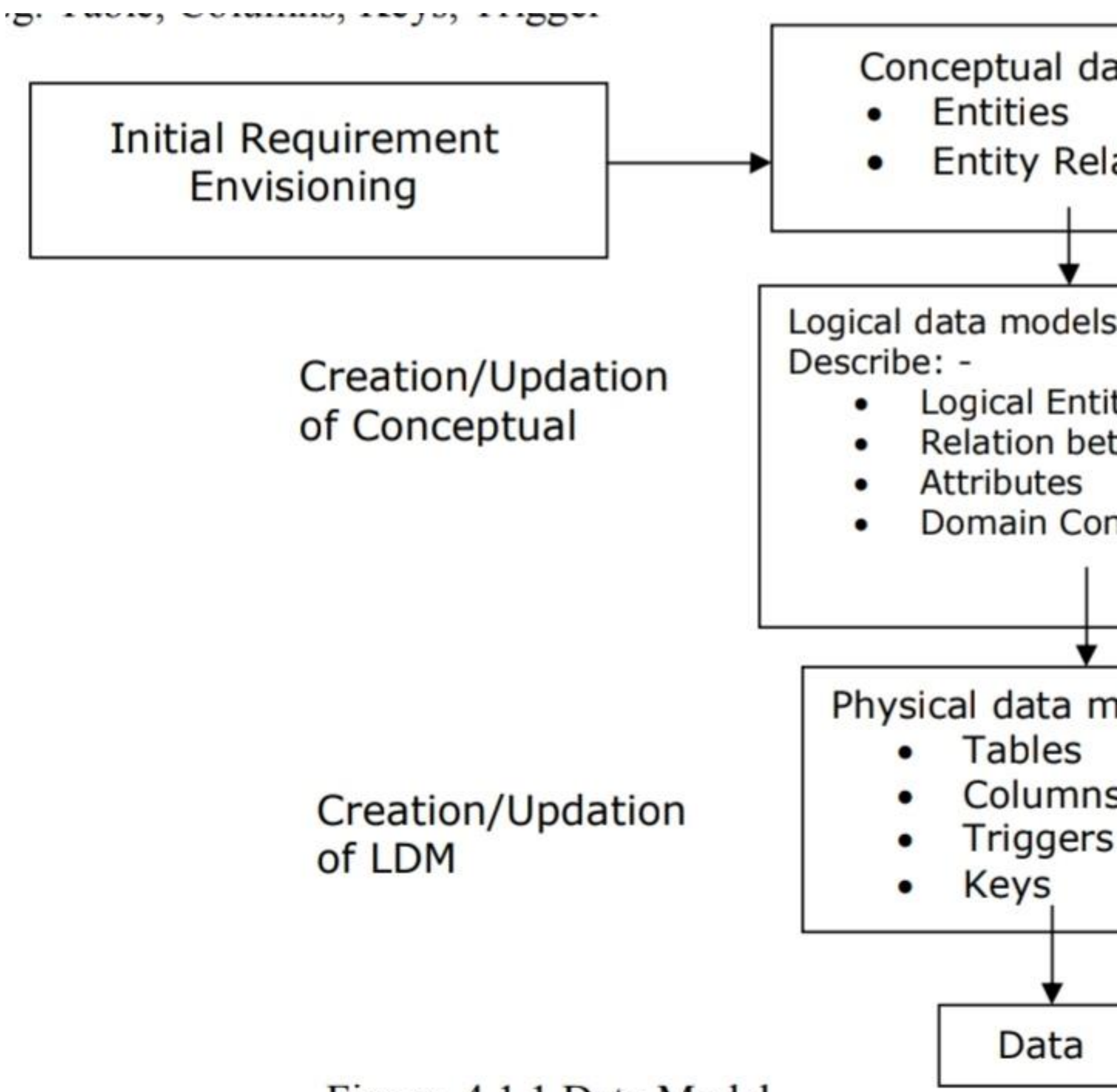


Figure. 4.1.1 Data Model

Components of ER Model

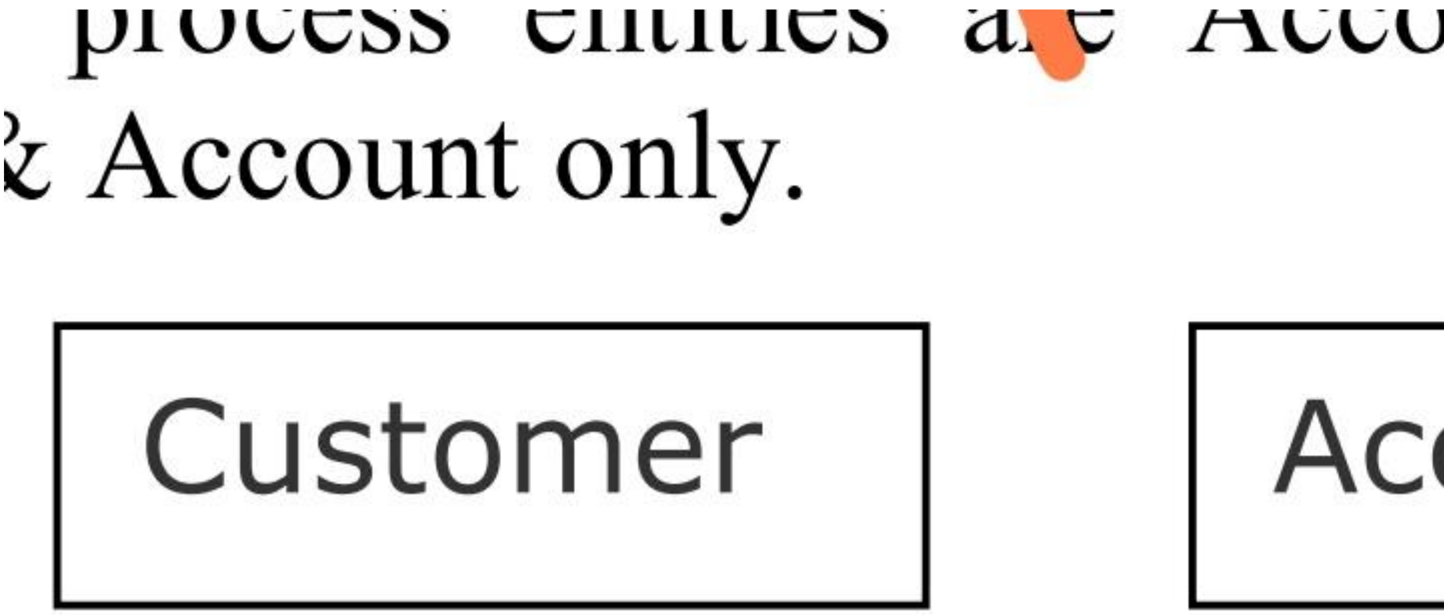
1: Entity

An entity is a thing or object in the actual world that is distinguishable from

all other objects. An entity is represented by rectangle containing the entity name.

Example: Student is entity having many properties name, address, roll no etc.

Example: In Bank System Customer withdraw money from bank, in terms of cash withdrawal process entities are Account & Customer so process is related with Customer & Account only.



2: Attributes:

1:An attribute is a property or characteristic.

2: An entity is represented by a set of attributes.

3: It corresponds to a field in a table.

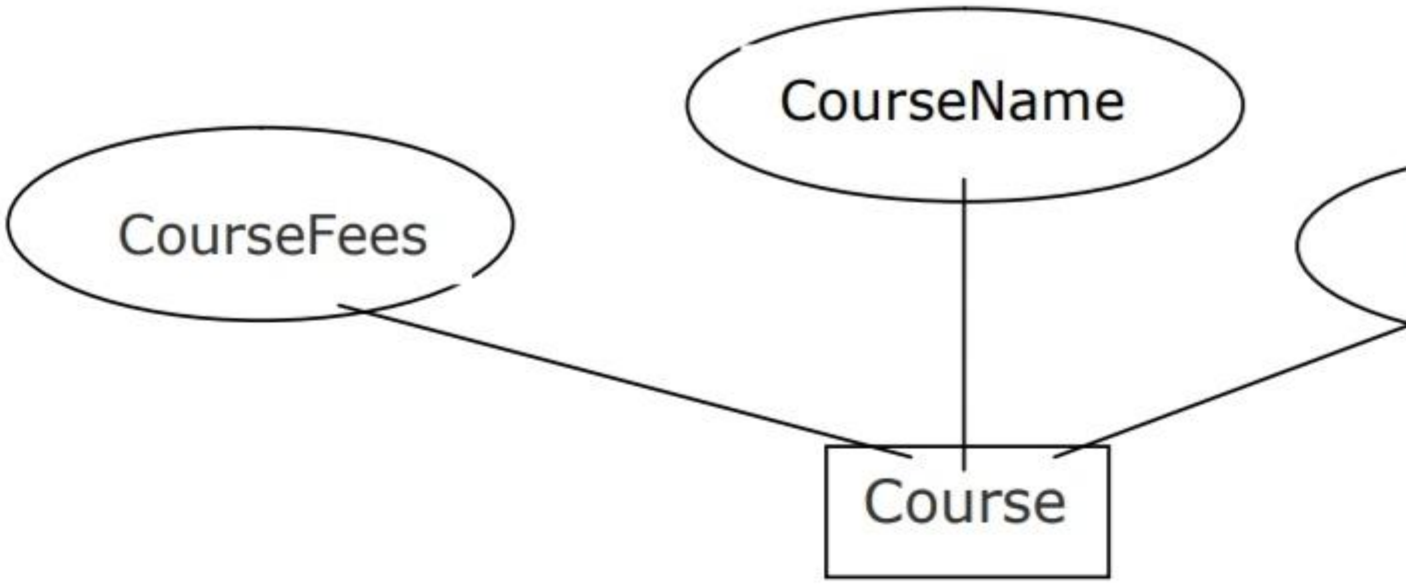
4: For each attribute there is a set of allowable values called the domain or value set of the attribute.

5. Attributes are represented by ovals and are linked to the entity with a line.

6: Each oval contains the name of the attribute it represents.

7: Example: Course= (CourseFees, CourseName, Duration).

8: Course Entity has a CourseFees, CourseName and Duration as attribute.



2.2 Attribute CourseFees, Duration, CourseName of course e

3: Entity Set:

1:In ER model a specific table row is referred as an entity instance or entity occurrence.

2: Each entity set has a key.

3: All entities in an entity set have the same set of attributes.

4: Thus entity set is a set of entities of the same type that share the same properties or attributes.

5: Set of entities of particular type is known as Entity Set.

d. Domain:

1: Attribute is characteristic or property used to correspond field in table.

2: Each attribute is connected with a set of values that is known as domain.

3: The domain defines the possible value that an attribute may grasp.

Weak entity

1: An entity, which is dependent on one or more entities for its existence, is known as Weak entity.

2: A weak entity is existence dependent. The existence of a weak entity is indicated by a double rectangle in the ER diagram.

3: An entity set that does not contain enough attributes to form a primary key is known as a weak entity set.

4: In weak Entity set it is vital to identify attributes of weak.entity set so we require a key it is known as discriminator or partial Key.

5: The.discriminator (or partial key) is used to categorize other attributes of a weak entity set.

Example:-If we consider cash withdraw system in Bank, we
transaction is executed on same account. But in this case we
amount as a primary key because its may happened that mor
executed on same account having same amount. To form
consider **Account_no**.
Weak Entity is denoted as below:-

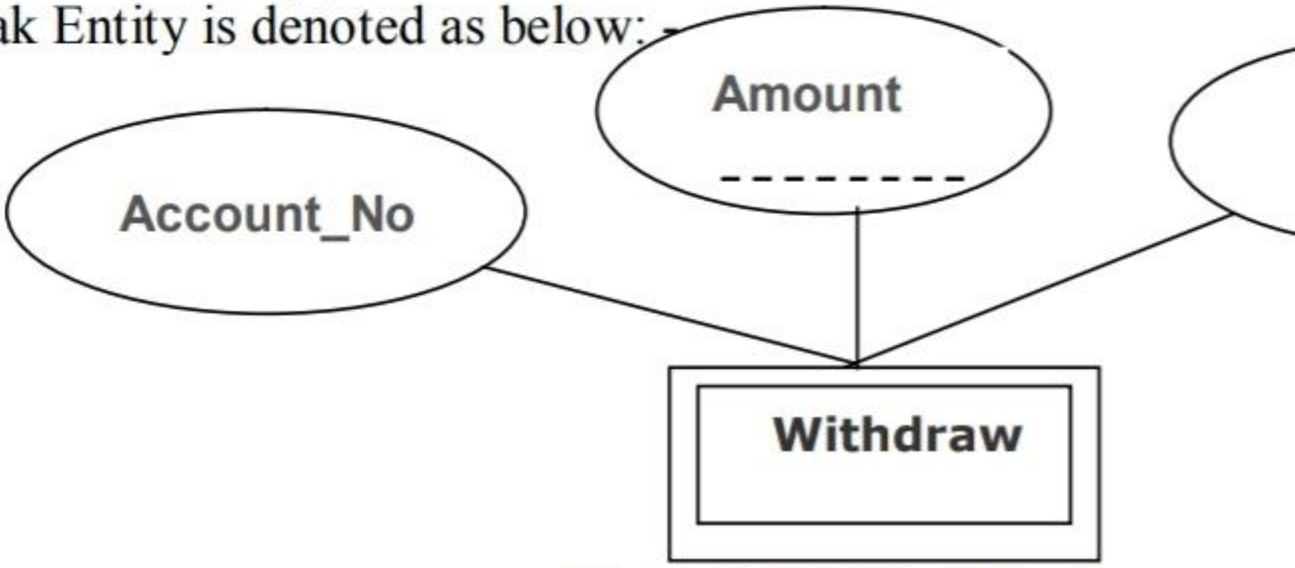


Figure.4.3.1 Weak Entity

Discriminator (or partial Key):

1: It is a set of attributes that uniquely identify weak entities and are connected to the same owner entity.

2: It is called as Discriminator or partial key.

3: Primary key of identifying entity set and the discriminator of weak entity set form the primary key of a weak entity set.

4: For example, a child entity is a weak entity because it relies on the parent entity in order for it to exist.

5: We underline the discriminator of a weak entity set by a dashed line in the ER diagram.

Strong entity

An entity set that has a primary key is known as strong entity set. A strong entity is independent of other entities and can exist on its own.

Example: Student is strong entity type because it associates with Primary key Roll_No

---

Recursive Entity

A recursive entity is one in which a relation can exist between occurrences of the same entity set. This occurs in a unary relationship.Example: In Organization many Employees managed by other employee's. Recursive entity is employee.

---

composite entity

Composite means thing composed of other things. Similar to that composite entity is composed of many parts. It is used to link the relationships that exist between many relationships. A composite entity is represented by a diamond shape within a rectangle in an ER Diagram.Example: A Student is identified by a RollNo and a Class.A Book is identified by a name of book, Author, Version, subject. Address is identified by Street name, City name

---

Attributes Type

In ER Model attributes can be classified into the following types.

When attribute consist of a single atomic value it is Simple attribute.i.e. Simple attribute cannot be subdivided. Example: - The attributes age, Gender etc is simple attributes.

Composite Attribute: -When attribute value is not atomic it is composite attribute. A composite attribute is an attribute that can be further subdivided. Example: The attribute ADDRESS can be subdivided into street, city, state, and zip code.Address: Street: City: State: Zip CodeName: ‘First Name: Middle Name: Last Name

Single Valued Attribute: -Attributes that have a single value are known as single valued attribute. Example:-A Person has ‘age’. Age have fixed value means multiple value is not possible for Age so Age is Single valued attribute .Single valued attribute can be simple or composite. Example:- Consider ‘Percentage’ in result system. It is a composite attribute .It is composed of main percentage part as well as decimal parts. It has fixed value. It is single valued attributes .Similarly ‘Customer Id ‘is a simple single valued attribute.

Multi Valued Attribute: -Multivalued attributes can have multiple values. Multivalued attributes are shown by a double line connecting to the entity in the ER diagram. Example: A customer can have multiple phone numbers, email id's etc A person may have several college degrees.Multi valued Entity is denoted as below: -

5. Stored Attributes: -An attribute that supplies a value to the related attribute is called as Stored Attribute. The value for the other attribute is derived from the stored attribute. Example: when calculating result Percentage that student obtain in examination is depends on the Grand total marks .In this case total marks is a stored attribute in which we store sum of all subject’s marks

Derived Attributes: -If value of attribute is derived from a stored attribute such type of attribute is called as derived attributes. We shown derived attributes name in Dotted oval.Example: In Result system First rank student is derived by the highest percentage i.e. student who obtained highest percentage is first ranker

Complex Attribute: A complex attribute is both composite and multi valued.i.e. Attribute have more than one value as well as it is composed of more than one attributes.Example: - Person Entity has Phone_No attribute, which is composite & mulitvalued. Means One person may have more than one phone number & Phone_No is may be composed of STD Code & Telephone number.

Null Attribute: -Some attributes are sometimes not necessary in future. In that case they may not have an applicable value for that attribute. The attributes having no value are called null.Example: EmailID. This attribute is not applicable every time.

Unit 5 (Normalizations)

Overview

1: Normalization in the context of databases refers to the process of organizing data in a relational database to reduce redundancy and dependency.
2: The goal of normalization is to minimize data anomalies, ensure data integrity, and improve the overall efficiency of the database design.
3: This process involves breaking down large tables into smaller, more manageable ones and establishing relationships between them.
4: By organizing data more efficiently, redundant information is minimized, which reduces the risk of inconsistencies and saves storage space.
5: Normalization helps maintain data integrity by minimizing anomalies such as update, insert, and delete anomalies.
6: The database becomes easier to maintain and update when it is well-organized through normalization.

Relational DB design

1: Understand the data requirements of the system.

2: Identify entities, their attributes, and the relationships between them.

3: Gather information about data access patterns and types of queries expected.

4: Create an Entity-Relationship Diagram (ERD) to represent the entities, attributes, and relationships.

5: Identify primary keys for each entity.

6: Normalize the data to at least the third normal form to reduce redundancy.

7: Translate the conceptual model into tables with columns, data types, and relationships.

8: Define primary and foreign keys.

9: Apply normalization techniques (1NF, 2NF, 3NF, etc.) to organize data and minimize redundancy.

10: Consider trade-offs and denormalization when necessary for performance optimization.

Decomposition (Small schema)

Decomposition, in the context of database design, refers to the process of breaking down a larger table into smaller tables to achieve normalization. It involves analyzing the dependencies between attributes and entities to organize data more efficiently and eliminate redundancy. Let's consider a small schema as an example for decomposition:

Suppose we have a simple schema representing information about students and courses they enroll in:

```
Student (StudentID, StudentName, Address, CourseID, CourseName,
Instructor)
```

In this schema, we have a combination of information related to both students and courses in the same table. To decompose this schema, we can identify the entities and normalize the data.
Step 1: Identify Entities
- Entities: Student, Course
Step 2: Create Separate Tables
1. **Student Table:**
```
 Student (StudentID, StudentName, Address)
```
2. **Course Table:**
```
Course (CourseID, CourseName, Instructor)
```

Step 3: Update Relationships
- Remove redundant attributes from the original table.
- Update relationships between tables using primary and foreign keys.
The decomposed tables would look like this:
Student Table:
```
Student (StudentID, StudentName, Address)
```
Course Table:
```
Course (CourseID, CourseName, Instructor)
```
Now, the relationships can be maintained using foreign keys:
Enrollment Table:
```
Enrollment (StudentID, CourseID)
```
In this new design

- The `Student` table contains information about students.
- The `Course` table contains information about courses.
- The `Enrollment` table establishes the many-to-many relationship between students and courses.

Lossy Decomposition

A lossy decomposition in the context of database design involves breaking down a relation or table into smaller relations in a way that results in the loss of information. Unlike the typical goal of normalization, which aims to eliminate redundancy and preserve all the original information, lossy decomposition intentionally sacrifices some data to achieve certain objectives, such as improving performance or simplifying queries.

One common scenario where lossy decomposition may be considered is denormalization. Denormalization involves introducing redundancy into a database design to improve query performance by reducing the need for joins and simplifying data retrieval. While this can enhance performance, it comes at the cost of increased storage space and the risk of data inconsistency if updates are not handled carefully.

Here's a simplified example of a lossy decomposition involving denormalization:

Suppose we have a normalized schema:

```
Employee (EmployeeID, EmployeeName, DepartmentID, DepartmentName,
Salary)
```

In this schema, the `DepartmentName` attribute is functionally dependent on `DepartmentID`. A lossy decomposition might involve creating a denormalized version with redundancy:

```
Employee (EmployeeID, EmployeeName, DepartmentID, Salary,
DepartmentName)
```

In this denormalized version, `DepartmentName` is stored redundantly with each employee record, making queries about the department of an employee more straightforward and potentially faster. However, it comes at the cost of increased storage space and the need for additional measures to ensure data consistency if department names change.

Lossless decomposition

1: Lossless decomposition in database design is a process of breaking down a relation or table into smaller relations without losing any information.

2: The objective is to preserve all the functional dependencies and integrity constraints that exist in the original table.

3: The primary goal is to maintain data consistency and accuracy while improving the overall structure of the database through normalization.

4: To achieve lossless decomposition, certain conditions must be satisfied.

5: The most common approach is to decompose the original relation into smaller relations in a way that allows the original data to be reconstructed through natural joins.

6: Two widely used algorithms to achieve lossless decomposition are the Boyce-Codd Normal Form (BCNF) and the Third Normal Form (3NF).

Functional Dependency

In the context of database design and normalization, functional dependencies describe the relationships between attributes in a relation (table). Let's discuss three types of dependencies: full dependency, partial dependency, and transitive dependency.

1. **Full Dependency:**

- A full dependency occurs when an attribute is functionally dependent on the entire primary key, and not on a part of it.

- In other words, every attribute in a non-prime attribute set is fully functionally dependent on the entire primary key.

Example:

Consider a relation `Employee` with attributes `(EmployeeID, EmployeeName, DepartmentID)`, where `EmployeeID` is the primary key. If `EmployeeName` depends on `EmployeeID` (the entire primary key) and not just a part of it, it's a full dependency.

```
EmployeeID -> EmployeeName
```

2. **Partial Dependency:**
   - A partial dependency occurs when an attribute is functionally dependent on only a part of the primary key, rather than the entire primary key.
   - It implies that the attribute depends on a subset of the primary key, and the relation is not in a fully normalized form.
   Example:
   Continuing with the `Employee` relation, if `EmployeeName` depends only on a part of the primary key (let's say `DepartmentID`), it's a partial dependency.

```
DepartmentID -> EmployeeName
```

   To resolve partial dependencies, the relation may need to be decomposed to eliminate these dependencies and achieve higher normalization.
3. **Transitive Dependency:**
   - A transitive dependency occurs when an attribute is functionally dependent on another non-prime attribute, rather than directly on the primary key.
   - It involves a chain of dependencies where an attribute depends on another attribute, which, in turn, depends on the primary key.
   Example:
   In the `Employee` relation, if `EmployeeName` depends on `DepartmentName` and `DepartmentName` depends on the primary key (`EmployeeID`), it's a transitive dependency.

```
EmployeeID -> DepartmentName -> EmployeeName
```

   To resolve transitive dependencies, normalization techniques, especially those targeting higher normal forms (e.g., Third Normal Form - 3NF or Boyce-Codd Normal Form - BCNF), can be applied.

Normalized Forms – Un – Normalized form, 1NF, 2NF, 3NF

Normalized forms are a set of guidelines in database design that help organize data in a relational database to eliminate redundancy and improve data integrity. The normalization process involves breaking down large tables into smaller, related tables based on certain rules and dependencies. Here are three commonly discussed normalized forms: Unnormalized Form (UNF), First Normal Form (1NF), and Third Normal Form (3NF).

1. **Unnormalized Form (UNF):**

   - UNF refers to a state where the data is not organized or normalized. It may contain duplicate rows and columns, making it prone to data anomalies and inconsistencies.

   Example of UNF:

```
   Employee (EmployeeID, EmployeeName, Department, DepartmentLocation)
```

   In this example, the department-related information is repeated for each employee, leading to redundancy.
2. **First Normal Form (1NF):**
   - A relation is in 1NF if it contains only atomic values (indivisible) and there are no repeating groups or arrays.
   Example of 1NF:
```
Employee (EmployeeID, EmployeeName, Department)
Department (Department, DepartmentLocation)
```
   In this 1NF example, the data is organized into two tables with atomic values.
3. **Second Normal Form (2NF):**
   - A relation is in 2NF if it is in 1NF and all non-prime attributes are fully functionally dependent on the entire primary key.
   Example of 2NF:
```
Employee (EmployeeID, EmployeeName, DepartmentID)
Department (DepartmentID, DepartmentName, DepartmentLocation)
```
   In this 2NF example, the relation is broken down into two tables, and all non-prime attributes (EmployeeName) depend on the entire primary key (EmployeeID).
4. **Third Normal Form (3NF):**
   - A relation is in 3NF if it is in 2NF and there are no transitive dependencies between non-prime attributes.
   Example of 3NF:
```
Employee (EmployeeID, EmployeeName, DepartmentID)
Department (DepartmentID, DepartmentName)
```
   In this 3NF example, transitive dependency (EmployeeName depending on DepartmentName) has been eliminated.

Each successive normal form builds upon the previous one, and higher normal forms (e.g., Boyce-Codd Normal Form - BCNF and Fourth Normal Form - 4NF) continue the process of normalization to eliminate further types of dependencies and anomalies. The goal is to create a well-structured database that minimizes redundancy, supports data integrity, and facilitates efficient data retrieval and maintenance.

De-Normalization

1: Denormalization is the process of intentionally introducing redundancy into a relational database by relaxing some of its normal forms.

2: While normalization aims to minimize redundancy and improve data integrity, denormalization goes in the opposite direction, often for the purpose of improving query performance, simplifying data retrieval, and reducing the number of joins.

3: Denormalization can simplify the structure of the database and make queries more straightforward.

4: This is particularly beneficial in scenarios where complex joins or aggregations are common.

5: Denormalization can involve the creation of summary tables or caches that store aggregated or precomputed data.

6: This can be useful for scenarios where real-time data retrieval is not critical, and the emphasis is on performance.

Unit 6 (SQL)

SQL

1: SQL stands for Structured Query Language.
2: It is a programming language specifically designed for managing and manipulating relational databases.
3: SQL provides a standardized way to interact with databases and perform various operations such as querying data, inserting, updating, and deleting records, creating and modifying database structures, and controlling access to the data.
4: SQL is used by database management systems such as MySQL, Oracle, Microsoft SQL Server, PostgreSQL, and SQLite, among others.
5: These DBMSs provide the necessary tools and engines to execute SQL statements and manage the underlying database.

Uses of SQL

1: Querying data.

2: Inserting, updating, and deleting rows in a table.

3: Creating, replacing, altering, and dropping objects.

4: Controlling access to the database and its objects.

5: Provide database consistency and integrity.

SQL statements

1:Database Manipulation Language (DML): -

When we want to retrieve, store, modify, delete, insert and update data in

database we use DML Statements. DML statements are:

- SELECT
- INSERT
- UPDATE
- DELETE

2. Data Definition Language (DDL): -

When we want to create & modify structure of objects in a database, we use

DDL Statements. DDL statements are:

- CREATE
- ALTER
- DROP

3. Data Control Language (DCL): -

DCL statements are used for database administration.

- GRANT
- DENY (SQL Server Only)
- REVOKE

Data types in SQL

Data Type	Syntax	Storage Size	Explanation
Integer	integer	4 bytes	Integer numerical
Numeric	numeric (p, s)	5-17 bytes	Where p is a prec scale value.
decimal	decimal (p,s)	5-17 bytes	Where p is a prec scale value.
float	float(p)	4 Or 8 bytes	Where p is a prec

char(n)	char(n)	Max 8,000 Char	Fixed-length char
varchar(n)	varchar(n)	Max 8,000 Char	Variable-length cl
text	text	Max 2GB of text data	Variable-length cl
bit	bit(x)		Allows 0, 1, or N
date	date	8 bytes	Stores year, mont
datetime	datetime	8 bytes	Store Date & Tim format Eg. 2007-10-08 1
Time	time	3-5 bytes	Stores the hour, n values.
Image	Binary Value	Maximum 2GB	Image Variables

Basic Query structure

SQL is not case sensitive language. Like other languages all reserved words

are written in all capital letters.

SQL Query is divided into 3 parts: - Select, From & Where.

In **Select** part, we use to select attributes of tables i.e. Columns.

In **From** part, we use table’s name from which we want to select data.

In **where** part we apply condition for filter the data from tables.

SELECT

1: It is used to retrieve data from an SQL database. This process is also known asa query. The asterisk (*) can be used to SELECT all columns of a table. The SQLSELECT command is used as followsSyntax: -SELECT FROM

2: Sometimes some columns in tables have duplicate values and we want to avoid duplication .

3: We want to access distinct values, at that time we use SelectDistinct statement.

4: The DISTINCT keyword can be used to return only distinctvalues.Syntax: - SELECT DISTINCT column_name(s) FROM table name

And, OR Operator

And, OR Operator: In the WHERE clause, we can use a condition, but for specifying multiple criteria and multiple conditions, we use the AND, OR, IN, BETWEEN, and LIKE operators. The syntax for a compound condition is as follows:

```
SELECT "column_name" FROM "table_name" WHERE "Condition" {[AND|OR]
"Condition"}+
```

In the case of the AND operator, both conditions must be satisfied. If out of two, one condition is not satisfied, then the query won’t retrieve any row.
In the case of the OR operator, if one condition is satisfied out of both conditions, the query retrieves all rows for which the condition is satisfied.

IN operator

If you want to specify multiple values in a WHERE Clause, we used In operator.If you want to exclude the rows in your list, you used NOT IN at the place of IN operator.

The syntax for using the IN keyword is as follows:

```
SELECT "column_name" FROM "table_name"WHERE "column_name" IN
('value1', ‘value2’...)
```

You can use one or more values in the parenthesis in where clause of select query. Each value is separated by comma. We can use numerical or characters values.If only one value is present inside the parenthesis, it is equivalent to check for onlyone value.

BETWEEN Operator

In Operator help for selection from one or more distinct values, but if we wantcollection from a range, we used BETWEEN operator. It used to verify whether weaccess data between the values range we mention.The syntax for the BETWEEN clause is as follows:

```
SELECT "column_name" FROM "table_name"WHERE "column_name" BETWEEN
'value1' AND 'value2'
```

This will select all rows whose column has a value between 'value1' and'value2'.

LIKE Operator

Sometimes we want to access data from tables that matches specified patternin columns, in that case we use like operator. It is a method to check for matchingstrings. We can use wildcards ahead of the pattern as well as after the pattern.Two Wildcards are used (%) & (_).• The Percent Sign (%) is used to match zero or more characters.• Underscore (_) used to match a single character.

syntax:

```
SELECT column_name(s) FROM table_nameWHERE column_name LIKE pattern
```

SQL Order by Keyword

Sometime we want to sort the result-set by a specified column, at that timewe used ORDER BY keyword. In Order By Keyword by default it sort the records inascending order. If you want to sort in a descending order, you can use the DESCkeyword. It gives choices for sorting result set ascending or descending.SQL ORDER BY Syntax: -

```
SELECT column_name(s) FROM table_nameORDER BY column_name(s) ASC|DESC
```

Aggregate Functions

Sometimes we need summarization of large volumes of data, at that timeaggregate functions are used. SQL aggregate functions always return a single valuethat is calculated from values in a column. When mathematical operations must beperformed on all or a group of values aggregate functions is useful.

Useful aggregate functions:

AVG () - Returns the average value.

COUNT () - Returns the number of rows

• FIRST () - Returns the first value

• LAST () - Returns the last value

• MAX () - Returns the largest value

• MIN () - Returns the smallest value

• SUM () - Returns the sum

• NOW () -Returns the current system date and time.

• ROUND () -Used to round a numeric field to the number of decimalsspecified.

• FORMAT () -Used to format how a field is to be displayed.

Group By

When we want grouping the result dataset by certain database table columns,we use GROUP BY statement along with the SQL aggregate functions.Syntax: -

```
SELECT ColumnName, aggregate function (ColumnName)FROM table WHERE
ColumnName operator valueGROUP BY ColumnName
```

Having Clause

We can’t use where clause with aggregate functions so SQL HAVING clauseis used to perform action on groups. It is used with the SELECT clause to specify asearch condition for a group or aggregate. Having clause is applicable only for groupswhereas where is limited to individual rows, not for groups.Syntax:

```
SELECT column1, column2, ... column_n, aggregate_function
(expression)FROM tables WHERE predicatesGROUP BY column1, column2, ...
column_nHAVING condition1 ... condition_n;
```

CREATE

Data is stored in basic structure like Table in SQL. For creating new table inSQL we used Create Table Statements, it adds a new table to an SQL database. Tablesconsist of rows and columns. Create Table have following

```
CREATE TABLE "table_name"("column 1"
"data_type_for_column_1","column 2"
"data_type_for_column_2",...)
```

DROP

Drop table remove the table(s) from the database.Syntax: -

```
DROP TABLE [IF EXISTS] table_name1, table_name2, ...
```
If we want to check that before dropping any table it is exist or not we use IFEXISTS clause .This clause will drop the table only if it exists. If the table does notexist, it will create error.Example: -DROP TABLE IF EXISTS Student

---

Constraints

1: In general Constraints are like rules.

2: Sometime we want to restrict for doingsome operation on database so constraint are used to enforce the integrity betweencolumns and tables & for controlling values allowed in columns.

3: By using constraintwe can avoid user from inserting certain types of mismatched data values in the column(s).

4: Constraint ensures correctness and reliability of the data in the database.

Entity Integrity: When we want to check for no duplicate rows exist in a table weused entity integrity.

Domain Integrity: -It allows only legal entries for a specified column by check ontype, format, or the range of possible values.

Referential integrity: -it checks that rows cannot be deleted, which are used by otherrecords.

User-Defined Integrity: -When we want to check that various specific business rulesthat must be follow, no one can crash that rules at that time we use user definedintegrity. For enforce these categories of the data integrity we used appropriateconstrain.

1. Not Null: -Many times user didn’t enter data for field in table and we want to checkcolumn not to accept NULL values, we used NOT NULL constraint. NOT NULLConstraint enforces the field to accept a value. We Specify Not Null for

Columnwhere we want that user must enter data for that field. This means that you cannotinsert a new record, or update a record without adding a value to this field.

2. Primary Key: -A primary key is used to uniquely identify each row in a table. It can be used either when the table is created using create table statement or by changing theexisting table structure using alter table. It consists of one or more fields in a table.When multiple fields are used as a primary key, they are called a composite key.

3.Foreign Key: -A FOREIGN KEY in one table point to a PRIMARY KEY in another table. Itis used to make sure referential integrity of the data. When we want to generateforeign key we must have two tables one is parent table & child table. Both tableshave relationship with each other based on common column. A common column inboth tables relates parent table & child table.

4. UNIQUE constraint: -It used to check the uniqueness of the values in a set of columns, so noduplicate values are entered. It is used to enforce primary key *constraints.One/* Major difference between Unique & Primary Key Constraint is thatprimary key & unique both can't allow repeated value, but in case of primary key wecan't enter Null for that column, while in unique constraint we can enter Null Value.
5. CHECK constraint: -A CHECK constraint is used to limit the values that can be placed in acolumn. It is used to enforce domain integrity.

INSERT

When we want to insert records into a table in the database we used INSERTCommand. This statement comes in three forms.The first inserts data from a VALUES clause.Syntax: -

```
INSERT INTO table_name [(Col1,Col2,....)]VALUES
(Expression1,Experssion2, .... )
```

UPDATE

When we want to update information in a table we use update query. It is usedto set multiple expression value to multiple columns also used to update query withwhere clause for obtaining conditionality in query.Syntax: -

```
UPDATE table_name SET Col1 = expression1, Col2 = expression2, ....
[WHERE expression]
```

DELETE

The DELETE statement is used to delete records in a table. By default deleteStatement delete all records from table .By using where clause we can restrict queryby using condition in it.Syntax: -

```
DELETE FROM table_name [WHERE expression]
```

ALTER

Sometime it may happen that you create table and after some time you want tomodify its column name, Size, data type OR you may add, remove columns of tablesat that time you use Alter command. It used to add, remove and modify columns &constraints and also to drop columns from table or you can use to set defaultexpression to default value of column.

```
ALTER TABLE table_name DROP [COLUMN] column_name
```

Data Control Language (DCL)

Data Control Language (DCL) in SQL is a subset of SQL commands that are used to control access to data within a database. DCL commands primarily deal with the permissions and privileges associated with database objects. The two main DCL commands in SQL are `GRANT` and `REVOKE`.

1. **GRANT:**

   - `GRANT` is used to provide specific privileges or permissions to database users.

- It allows users to perform certain operations (such as SELECT, INSERT, UPDATE, DELETE) on specified database objects (tables, views, procedures, etc.).

- The syntax for the `GRANT` command is as follows:

```
 GRANT privilege_name
     ON object_name
     TO {user_name | PUBLIC | role_name} [WITH GRANT OPTION];
```

2. REVOKE:
   - `REVOKE` is used to take away or remove the privileges that have been granted from users or roles.
   - It is the opposite of the `GRANT` command.
   - The syntax for the `REVOKE` command is as follows:

```
     REVOKE privilege_name
     ON object_name
     FROM {user_name | PUBLIC | role_name};
```

These DCL commands are crucial for maintaining the security and integrity of a database by controlling who has access to what data and what operations they can perform. Keep in mind that the exact syntax and functionality might vary slightly between different database management systems (e.g., MySQL, PostgreSQL, Oracle).

Different operations on tables

1. Rename:

   - In SQL, you can use the `ALTER TABLE` statement to rename a table.

   - The basic syntax is:

```
ALTER TABLE old_table_name RENAME TO new_table_name;
```

2. Tuple Variables:
   - SQL doesn't use the concept of tuple variables in the way some programming languages might. However, in SQL, you work with rows or records in tables. You can use variables to store and manipulate data within stored procedures or scripts.
3. Set Operations:
   - Set operations in SQL involve combining or comparing the results of two or more queries.
   - Common set operations include:
     - UNION Operator:

```
SELECT column1, column2 FROM table1
```

```
 UNION
 SELECT column1, column2 FROM table2;
```
   - UNION ALL Operator:
     Similar to UNION, but it includes duplicate rows.
```
SELECT column1, column2 FROM table1
UNION ALL
SELECT column1, column2 FROM table2;
```
   - INTERSECT Operator:
     Returns rows that are common between two result sets.
```
SELECT column1, column2 FROM table1
INTERSECT
SELECT column1, column2 FROM table2;
```
   - MINUS (or EXCEPT) Operator:
     Returns rows from the first result set that are not present in the second result set.
```
SELECT column1, column2 FROM table1
MINUS
SELECT column1, column2 FROM table2;
```
4. String Operations:
  - SQL supports various string operations, such as concatenation, substring, and comparison.
  - Example of concatenation:
```
SELECT first_name || ' ' || last_name AS full_name FROM employees;
```
  - Example of substring:
```
  SELECT SUBSTRING(column_name FROM start_position FOR length) FROM
table_name;
```

  - Example of string comparison:
```
 SELECT * FROM employees WHERE first_name = 'John'
```

NULL

In SQL, a NULL value is a special marker used to indicate that a data value in a database does not exist in the dataset. A NULL value can represent an undefined or unknown state for a column in a table. Here are some key points about NULL values in SQL:

1. **Representation:** NULL is not the same as an empty string or zero. It is a unique marker that indicates the absence of a value.

2. **Usage:** NULL can be used in any data type, including numeric, character, and date/time types.
3. **Comparison:** Comparisons with NULL using regular comparison operators (like =, <, >) will result in unknown or NULL. To check for NULL values, you should use the IS NULL or IS NOT NULL operators.

```
SELECT * FROM table_name WHERE column_name IS NULL;
```

Unit 7 (Transaction Management)

Transaction Concept

1: The execution of a program that includes database access operations is called the transaction.
2: A transaction is a unit of program execution that accesses and updates various data items.
3: Whenever we read from and/or write (update) a database, we create a transaction.
4: If the database operations in a transaction do not modify the data in the database, but only reads the data, then the transaction is called as the read-only transaction.
5: A transaction is a single unit of database operations that are either completed entirely or not performed at all.
6: The transaction should be completed or aborted, no intermediate states are allowed.
7: The transaction consists of simple SQL statements such as select, insert, delete, etc.

Properties of Transactions:

The atomic transaction has several properties. These are called as the ACID

properties.

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

2: *Consistency Preservation:* A correct execution of a transaction must take the database from one consistent state to another.

3: *Isolation*: A transaction should not make its updates visible to other transactions until it is committed. This property is useful in a multi-user database environment because several different users can access and update the database at the same time.

4: *Durability or Permanency:* Once a transaction changes the database and the changes are committed, these changes must never be lost because of the software or hardware failures.

Consistency and isolation

Consistency: This property ensures that a database transitions from one valid state to another. In other words, it guarantees that any data modifications or transactions conform to the predefined rules and constraints of the database. Consistency is maintained through techniques such as constraints, triggers, and validation rules to prevent data from becoming invalid or contradictory.

Isolation: Isolation ensures that multiple concurrent transactions do not interfere with each other while accessing and modifying data. Isolation levels, like Read Uncommitted, Read Committed, Repeatable Read, and Serializable, define the extent to which one transaction's changes are visible to others. Higher isolation levels provide stronger guarantees but may lead to performance trade-offs.

Together, consistency and isolation in a DBMS help maintain data accuracy, prevent anomalies, and manage the interactions between concurrent transactions to ensure the integrity of the database.

Atomicity and Durability

Transactions can be incomplete for three kinds of reason:

1) Firstly it can be aborted, or terminated unsuccessfully, by the DBMS because of some internal reason, during execution. If the transaction is aborted by DBMS, then it will restart automatically and execute as a new transaction.

2) Secondly, the system may crash while one or more transactions are in

progress. E.g. power failure.

3) Thirdly, a transaction may encounter an unexpected error and decide to

terminate itself.

DBMS maintains a record called log of all write operations performed on the database. To ensure the atomicity property, DBMS has to undo the actions of incomplete transactions. A transaction that terminates or aborts may leave the database in an inconsistent state. Thus DBMS has to remove the updates of such incomplete transactions by undoing the actions of those particular transactions.

The log is also useful in ensuring durability. If the system crashes before the

changes made by a completed transaction is written to disk, the log is used to recognize and restore the changes when the system restarts. The DBMS component, which ensures atomicity and durability, is called the recovery manager.

Transaction Terminology

1: *Commit:* A transaction that completes the execution successfully is called a committed transaction. The committed transaction should always take the database from one consistent state to another. The changes made by the committed transaction should be permanently stored in the database even if there is any system failure.

2: *Abort:* If there are no failures then all the transactions complete successfully. But transaction may not always complete its execution successfully then the transaction is called as aborted.

3: *Roll back:* If we want to obey the atomicity property then all the changes made by the aborted transactions must be undone. When we undo the changes of a transaction we say that the transaction has been rolled back.

4: *Restart:* If a transaction is aborted because of hardware or software failure, a transaction restarts as a new transaction

5: *Kill:* A transaction is killed, if there is some internal logic problem, or input

problem or output problem.

6: *Throughput:* It is the average number of transactions completed in a given amount of time.

7: *Average response time:* It is the average time taken by a transaction to complete after it has been submitted.

Transaction States

The transaction may be in one of the following states while executing:

1) Active: This is the initial state. The transaction is in this state while it is

executing.

2) Partially Committed: The entire transaction has been executed, but not yet

committed.

3) Failed: The transaction cannot execute normally.

4) Aborted: The transaction is rolled back and the database is stored to its prior state.

5) Committed: After a successful completion.

5) **Committed**: After a successful completion.

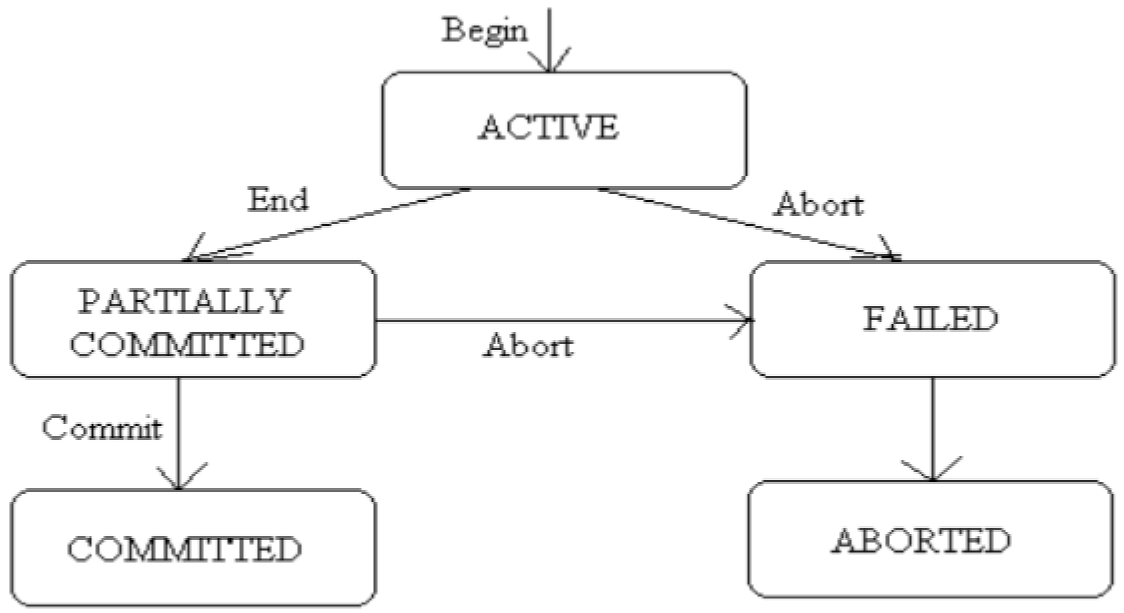


Figure 6.3: Transaction States.

1: A transaction starts its execution in active state.

2: When the transaction completes, it enters the partially committed state. **3:** Here, the transaction has finished the execution, but some recovery protocols need to ensure that a system failure will not result in an inability to record the changes of the transactions permanently.

4: Once this check is successful, the transaction is said to have reached the commit point and enters the committed state.

5: Once a transaction is committed then it has finished the

execution successfully and all its changes must be recorded permanently in the database.

6: The database system writes enough information to disk. So, that even in case of failure the updates of a transaction can be re-created when the system restarts.

7: When this information is written then the transaction enters the committed state.

8: A transaction enters a failed state when the DBMS finds that the transaction is not executing normally. Such a transaction is rolled back. Then, it enters the aborted state.

9: After a transaction aborts, it may restart or get killed.

Concurrent Execution of Transactions

1: Concurrent execution of transactions in a database refers to the simultaneous processing of multiple transactions.

2: Concurrency control mechanisms, such as locking, isolation levels, and timestamp-based methods, are employed to maintain data integrity and consistency while preventing conflicts.

3: Deadlock detection and resolution strategies are used to address situations where transactions block each other.

4: Achieving serializability ensures that concurrent transactions have the same effect as if they were executed sequentially.

5: Various database systems use different approaches, like Multi-Version Concurrency Control (MVCC) or optimistic concurrency control, to balance performance and data integrity.

6: Effective concurrency management is vital for multi-user database systems' reliability and performance.

Operations on a Transaction

1: *Read-item (X):* Reads the database item named X into a program variable. The Read-item (X) includes the following steps:

- Find the address of disk block that contains item X.
- Copy the contents of disk block into a buffer in main memory.
- Copy the item X from the buffer to the program variable named X.

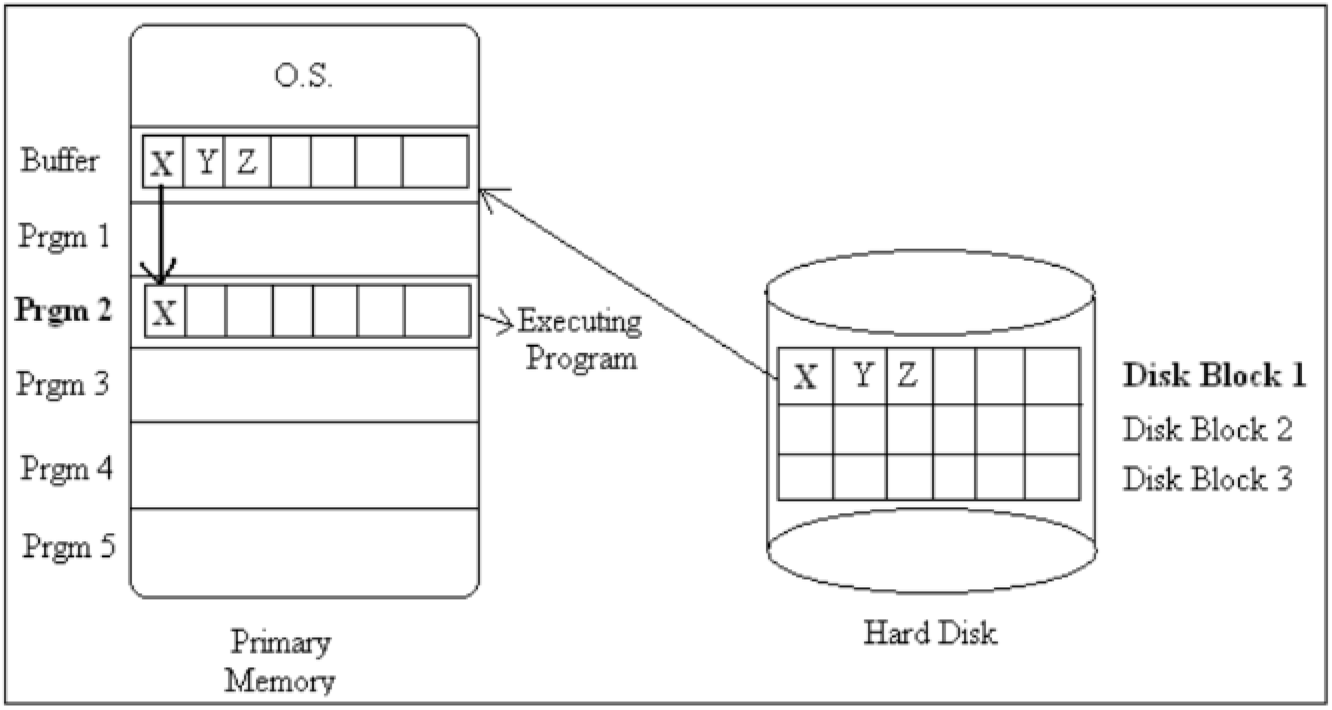


Figure 6.6: Read item (X) operation

2: _Write-item (X):_ Writes the value of a program variable X into the database item named X. Write-item (X) includes the following steps:

- Find the address of the disk block that contains item X.
- Copy the contents of disk block into a buffer in main memory.
- Copy item X from program variable named X into its correct location in
- the buffer.
- Store the updated block from the buffer back to disk.

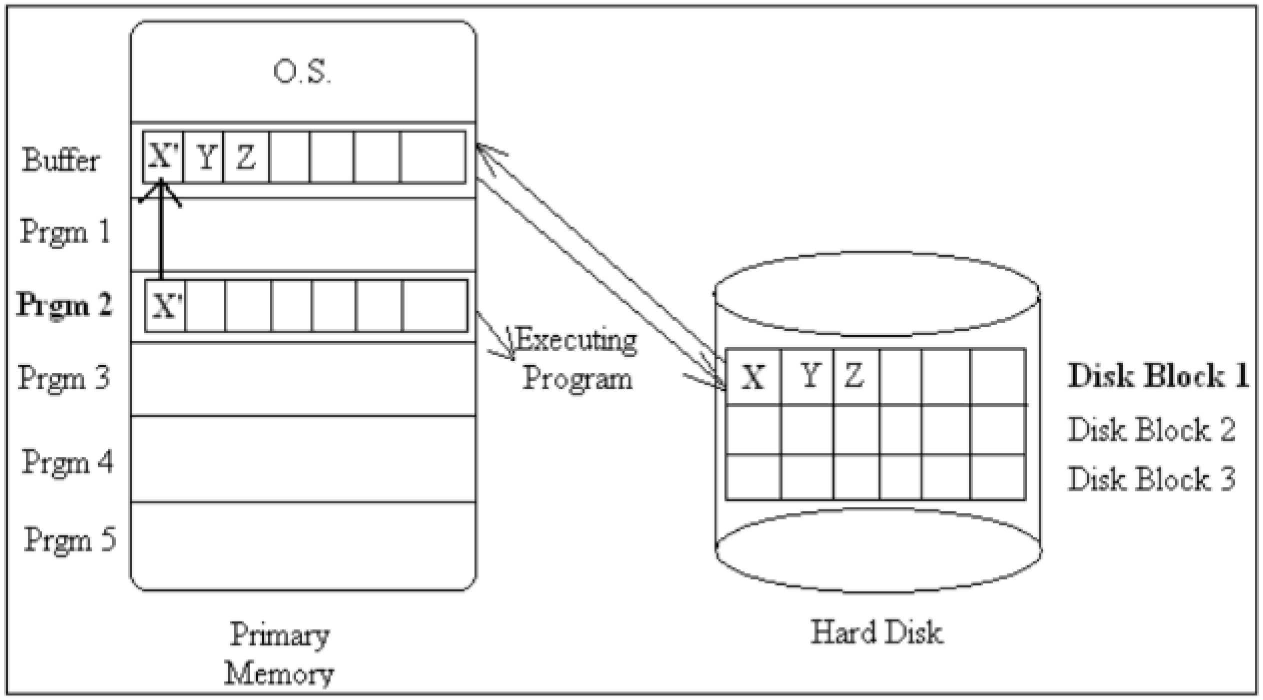


Figure 6.7: Write-item (X) operation.

---

concurrency control

1: In multi-user systems, programs are executed at the same time.

2: Concurrency control is a mechanism used to ensure that the programs do not interfere with other executing programs.

3: The major task performed by concurrency control protocols is to manage the concurrent operations on the database by executing programs.

4: Database is an application that manages data and allows fast storage and retrieval of that data.

5: Multiple users can concurrently share the data stored in the database.

6: The database can remain consistent if all the programs are only reading the data in the database.

7: The problem arises when some programs are reading the data in the database and some program is updating the same data in the database, leaving the database in an inconsistent state.

Multiple Update Problem

Multiple Update problem occurs when two transactions access the same

database item and are interleaved in a way, which results in an incorrect final value of the database item.

Suppose you have a banking application with an account balance stored in a database. Two users, User A and User B, both want to transfer money from their accounts to a shared account. The current balance of the shared account is $100.

1. *Initial State:*

- Shared Account Balance: $100

2. *Transactions:*

- Transaction 1 (User A): User A wants to transfer $50 to the shared account.

- Transaction 2 (User B): Simultaneously, User B wants to transfer $30 to the shared account.

3. *Execution:*

- Both transactions read the current balance of the shared account, which is $100.

4. *Multiple Update Problem:*

- Both transactions proceed with their updates simultaneously:

  - Transaction 1: $100 + $50 = $150 (new balance)

  - Transaction 2: $100 + $30 = $130 (new balance)

5. *Inconsistency:*

- The final state of the shared account is inconsistent because the total amount transferred ($150 + $130 = $280) exceeds the initial balance.

Uncommitted Dependency (Dirty Read) Problem

The Uncommitted Dependency problem, also known as Dirty Read, is a phenomenon that occurs in a database when one transaction reads data that has been modified by another transaction but has not yet been committed. In other words, a transaction reads uncommitted changes made by another transaction, and these changes might be rolled back later, leading to inconsistencies.

Let's illustrate the Dirty Read problem with an example:

1. **Initial State:**
   - Account Balance: $100
2. **Transactions:**
   - **Transaction 1 (User A):** User A wants to transfer $50 to another account.
   - **Transaction 2 (User B):** Simultaneously, User B checks the account balance of user A.
3. **Execution:**
   - Both transactions start with the initial account balance of $100.
4. **Dirty Read:**
   - **Transaction 1 (User A):** User A subtracts $50, making the new balance $50. However, this change is not yet committed.
   - **Transaction 2 (User B):** User B reads the uncommitted balance of $50.
5. **Rollback:**
   - Transaction 1 encounters an issue and decides to roll back the changes.
6. **Inconsistency:**
   - Even though User B saw a balance of $50 temporarily, the final state of the database should have remained $100 after the rollback. However, due to the Dirty Read, User B had an inconsistent view of the data.

To prevent Dirty Reads, databases typically use isolation levels and concurrency control mechanisms. Isolation levels, such as "Read Committed" or "Serializable," determine the degree to which one transaction is isolated from the effects of other concurrently executing transactions. Using a higher isolation level helps avoid Dirty Reads but may impact performance.

Incorrect Analysis Problem

The "Incorrect Analysis Problem" refers to a situation where a transaction reads data that has been modified by another transaction during its execution, leading to an incorrect analysis or decision based on outdated or inconsistent information. This problem is one of the issues addressed by concurrency control mechanisms in database management systems.

Let's illustrate the Incorrect Analysis Problem with an example:

1. **Initial State:**
   - User A's account balance: $200
2. **Transactions:**
   - **Transaction 1 (User A):** User A checks their account balance and decides to transfer $100 to another account.
   - **Transaction 2 (User B):** Simultaneously, User B checks User A's account balance.
3. **Execution:**
   - Both transactions start with the initial account balance of $200.
4. **Modification by Transaction 1:**
   - **Transaction 1 (User A):** User A subtracts $100 from their account, making the new balance $100.
5. **Incorrect Analysis by Transaction 2:**
   - **Transaction 2 (User B):** User B, unaware of the modification made by Transaction 1, reads the outdated balance of $200.
6. **Inconsistency:**
   - User B makes decisions or analysis based on the outdated information, leading to an incorrect understanding of the actual state of User A's account.

Concurrency control mechanisms, such as locking, timestamp-based methods, or multi-version concurrency control, aim to prevent the Incorrect Analysis Problem by ensuring that transactions are properly synchronized. These mechanisms help maintain the consistency and integrity of the database by regulating access to data, preventing transactions from reading outdated or inconsistent information.

Schedules

1: A schedule in a database dictates the chronological order of transaction execution.

2: Serial schedules are straightforward, executing transactions one after another, while concurrent schedules allow overlapping, improving performance.

3: Conflict serializable schedules ensure equivalent results as a serial execution, addressing conflicts arising from read-write or write-write operations.

4: Serializable schedules include conflict serializable ones. Recoverable schedules guarantee that committed transactions depend on prior committed transactions, and cascadeless schedules prevent reading uncommitted data.

5: Two-Phase Locking (2PL) is a common concurrency control protocol, requiring transactions to acquire and release locks to prevent conflicts, ensuring data consistency, integrity, and isolation.

6: A schedule in a database dictates the chronological order of transaction execution.

7: Serial schedules are straightforward, executing transactions one after another, while concurrent schedules allow overlapping, improving performance.

8: Conflict serializable schedules ensure equivalent results as a serial execution, addressing conflicts arising from read-write or write-write operations.

9: Serializable schedules include conflict serializable ones.

10: Recoverable schedules guarantee that committed transactions depend on prior committed transactions, and cascadeless schedules prevent reading uncommitted data.

11: Two-Phase Locking (2PL) is a common concurrency control protocol, requiring transactions to acquire and release locks to prevent conflicts, ensuring data consistency, integrity, and isolation.

ACID Properties

<u>Atomicity:</u> A transaction is an atomic unit of processing; it is either

performed in its entirety or not performed at all.

Consistency Preservation: A correct execution of a transaction must take

the database from one consistent state to another.

Isolation: A transaction should not make its updates visible to other

transactions until it is committed.

Durability: Once a transaction changes the database and the changes are

committed, these changes must never be lost.

Unit 8 (PL / SQL)

introduction to PL/SQL

1: PL/SQL (Procedural Language/Structured Query Language) is a programming language designed specifically for managing and manipulating relational database management system (RDBMS) data.
2: It is an extension of SQL and adds procedural constructs found in programming languages, making it a powerful tool for building database-driven applications.
3: PL/SQL is commonly used with Oracle Database, but it is also supported by other relational database systems.
4: PL/SQL code is organized into blocks.
5: A block consists of three parts: Declaration, Execution, and Exception Handling.
6: Declaration: Variables, constants, and cursors are declared in this section.
7: Execution: Business logic and SQL statements are executed in this section.
8: Exception Handling: Errors are handled in this section.

Advantages of PL/SQL

1: PL/SQL allows the use of procedural constructs such as loops, conditionals, and exception handling, which are not available in standard SQL.

2: PL/SQL supports modular programming by allowing you to organize code into procedures, functions, packages, and triggers.

3: This enhances code reusability and maintainability.

4: PL/SQL provides robust error handling mechanisms, allowing you to catch and handle exceptions.

5: PL/SQL provides control over transactions using features such as COMMIT and ROLLBACK.

6: PL/SQL allows for the optimization of code execution through features like bulk processing and the use of cursors.

7: Triggers enable the automatic execution of code in response to specific events, such as data modifications or database events.

PL/SQL Architecture

1: PL/SQL architecture involves a compiler translating code into bytecode and an execution engine interpreting it.

2: Blocks, the fundamental units, can be anonymous or named (procedures, functions, packages).

3: PL/SQL seamlessly integrates with the SQL engine, allowing the combination of procedural and SQL statements.

4: A server process manages execution on the server side, interacting with the database server.

5: Cursor management handles result sets, and robust exception handling deals with errors.

6: Transaction control statements like COMMIT and ROLLBACK ensure data integrity.

7: Triggers respond to specific events. Understanding this architecture empowers developers to create efficient, modular, and maintainable code for database-centric applications.

PL/SQL Data types

1. **Scalar Data Types:**
   - **VARCHAR2(n):** Variable-length character string with a maximum length of 'n' characters.
   - **NUMBER:** Numeric data type for integers and decimals.
   - **DATE:** Represents date and time information.
   - **BOOLEAN:** Represents Boolean values (TRUE, FALSE, or NULL).

2. **Composite Data Types:**
    - **Record:** A structure containing fields, each with its own data type.
    - **Table:** An ordered collection of elements of the same data type.
    - **VARRAY:** Variable-size array.
    - **Nested Table:** Unordered collection of elements.
3. **Reference Data Types:**
    - **REF CURSOR:** A data type representing the memory address of a query result set, used for dynamic SQL and stored procedures.
4. **Large Object Data Types:**
    - **CLOB (Character Large Object):** Stores large amounts of character data.
    - **BLOB (Binary Large Object):** Stores large amounts of binary data.
    - **BFILE:** Stores binary file references.
5. **ROWID and UROWID:**
    - **ROWID:** Represents the address of a row in a table.
    - **UROWID:** Similar to ROWID but is universal, not tied to a specific table.
6. **PL/SQL-specific Data Types:**
    - **BOOLEAN:** Represents Boolean values (TRUE, FALSE, or NULL).
    - **PLS_INTEGER:** Integer data type optimized for arithmetic operations.
    - **BINARY_INTEGER:** Similar to PLS_INTEGER, used for integer calculations.
7. **Cursor Variables:**
    - **CURSOR:** Used for processing the result sets of queries.

Variable and Constants

In PL/SQL, variables and constants are essential elements for storing and managing data within programs. Here's a brief overview of variables and constants:

Variables:

- **Declaration:** Variables are declared using the `DECLARE` keyword within PL/SQL blocks.
- **Syntax:** `variable_name data_type;`
- **Example:** `DECLARE my_variable NUMBER;`
- **Initialization:** Variables can be initialized at the time of declaration or later in the program.

Constants:

- **Declaration:** Constants are declared using the `CONSTANT` keyword.
- **Syntax:** `CONSTANT constant_name data_type := value;`
- **Example:** `CONSTANT pi CONSTANT NUMBER := 3.14159;`
- **Initialization:** Constants must be initialized at the time of declaration and cannot be changed later in the program.

```
DECLARE
  my_variable NUMBER := 10;
  constant_value CONSTANT NUMBER := 5;
BEGIN
  my_variable := my_variable + constant_value;
  -- Further code using my_variable
END;
```

Using Built_in Functions

`CONCAT` **or** `||`: Concatenates two strings.

```
DECLARE
  first_name VARCHAR2(20) := 'John';
  last_name VARCHAR2(20) := 'Doe';
  full_name VARCHAR2(50);
BEGIN
  full_name := first_name || ' ' || last_name;
END;
SUBSTR: Extracts a substring from a string.
DECLARE
  original_string VARCHAR2(20) := 'Hello, World!';
  substring VARCHAR2(5);
BEGIN
  substring := SUBSTR(original_string, 1, 5); -- Extracts 'Hello'
END;
```

Numeric Functions:

- `ROUND`: Rounds a numeric value to a specified number of decimal places.

```
DECLARE
  num NUMBER := 3.14159;
  rounded_num NUMBER;
BEGIN
  rounded_num := ROUND(num, 2); -- Rounds to 2 decimal places
END;
```

Conversion Functions:

- `TO_NUMBER`, `TO_DATE`, `TO_CHAR`: Convert data types.

```
DECLARE
  num_str VARCHAR2(10) := '123';
  num NUMBER;
  date_str VARCHAR2(20) := '2022-01-01';
  date_value DATE;
BEGIN
  num := TO_NUMBER(num_str);
  date_value := TO_DATE(date_str, 'YYYY-MM-DD');
END;
```

Conditional and Unconditional Statements

1. IF Statement:

  - The `IF` statement allows for conditional execution of a block of code.

```
  DECLARE
       x NUMBER := 10;
     BEGIN
       IF x > 5 THEN
         -- Code to execute if x is greater than 5
       END IF;
     END;
```

2. IF-ELSE Statement:
  - The `IF-ELSE` statement extends the `IF` statement to include an alternative block of code to execute if the condition is not true.

```
     DECLARE
       x NUMBER := 3;
     BEGIN
       IF x > 5 THEN
         -- Code to execute if x is greater than 5
       ELSE
         -- Code to execute if x is not greater than 5
       END IF;
     END;
```

3. ELSIF Clause:
  - The `ELSIF` clause allows you to check multiple conditions sequentially.

```
 DECLARE
       x NUMBER := 3;
     BEGIN
       IF x > 5 THEN
         -- Code to execute if x is greater than 5
       ELSIF x = 5 THEN
         -- Code to execute if x is equal to 5
       ELSE
         -- Code to execute if x is less than 5
       END IF;
     END;
```
Unconditional Statements:
1. LOOP Statement:
  - The `LOOP` statement allows for the execution of a block of code repeatedly.

```
DECLARE
       counter NUMBER := 1;
     BEGIN
       LOOP
         -- Code to execute repeatedly
         counter := counter + 1;
         EXIT WHEN counter > 5; -- Exit the loop when the condition is
met
       END LOOP;
     END;
```
2. FOR Loop:
  - The `FOR` loop is used to iterate over a range of values.

```
 DECLARE
       total NUMBER := 0;
     BEGIN
       FOR i IN 1..5 LOOP
         -- Code to execute for each iteration
         total := total + i;
       END LOOP;
     END;
```
3. WHILE Loop:
  - The `WHILE` loop repeats a block of code as long as a specified condition is true.
```
  DECLARE
       counter NUMBER := 1;
     BEGIN
       WHILE counter <= 5 LOOP
         -- Code to execute while the condition is true
         counter := counter + 1;
       END LOOP;
     END;
```
4. EXIT Statement:
   - The `EXIT` statement is used to exit a loop or block of code prematurely based on a condition.

```
DECLARE
       counter NUMBER := 1;
     BEGIN
       LOOP
         -- Code to execute repeatedly
         EXIT WHEN counter > 5; -- Exit the loop when the condition is
met
         counter := counter + 1;
       END LOOP;
     END;
```

---

if then statement

if then statement is the most simple decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not. Syntax:

```
if condition then
-- do something
end if;
```
Here, condition after evaluation will be either true or false. if statement accepts boolean values – if the value is true then it will execute the block of statements below it otherwise not. if and endif consider as a block here.

---

if – then- else

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won’t. But what if we want to do something else if the condition is false. Here comes the else statement. We can use the else statement with if statement to execute a block of code when the condition is false. Syntax:-

```
if (condition) then
-- Executes this block if
-- condition is true
else
-- Executes this block if
-- condition is false
```

---

nested-if-then

A nested if-then is an if statement that is the target of another if statement. Nested if-then statements mean an if statement inside another if statement. Yes, PL/SQL allows us to nest if statements within if-then statements. i.e, we can place an if then statement inside another if then statement. Syntax:-

```
if (condition1) then
-- Executes when condition1 is true
if (condition2) then
-- Executes when condition2 is true
end if;
end if;
```

---

if-then-elsif-then-else ladder

Here, a user can decide among multiple options. The if then statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed. Syntax:-

```
if (condition) then
--statementelsif (condition) then
--statement
.
.
else
--statement
endif
```

Selection Case (CASE statement)

The `CASE` statement in PL/SQL is used to perform conditional logic. It's similar to a switch statement in other programming languages. Here's a simple example:

```
DECLARE
   grade CHAR := 'B';
BEGIN
   CASE grade
      WHEN 'A' THEN
         DBMS_OUTPUT.PUT_LINE('Excellent');
      WHEN 'B' THEN
         DBMS_OUTPUT.PUT_LINE('Good');
      WHEN 'C' THEN
         DBMS_OUTPUT.PUT_LINE('Average');
      ELSE
         DBMS_OUTPUT.PUT_LINE('Fail');
   END CASE;
END;
```

Simple Case (CASE statement)

The CASE statement can also be used in a simplified form for simple equality comparisons:

```
DECLARE
   day_number NUMBER := 3;
BEGIN
   DBMS_OUTPUT.PUT_LINE(
      CASE day_number
         WHEN 1 THEN 'Sunday'
         WHEN 2 THEN 'Monday'
         WHEN 3 THEN 'Tuesday'
         WHEN 4 THEN 'Wednesday'
         WHEN 5 THEN 'Thursday'
         WHEN 6 THEN 'Friday'
         WHEN 7 THEN 'Saturday'
         ELSE 'Invalid day'
      END CASE
   );
END;
```

GOTO Label

While GOTO statements are available in PL/SQL, it's generally advised to avoid them due to their potential to create spaghetti code. However, if you still want to use it, here's an example:

```
DECLARE
   x NUMBER := 10;
BEGIN
   IF x > 5 THEN
      GOTO my_label;
   END IF;
   DBMS_OUTPUT.PUT_LINE('This line will be skipped');
   <<my_label>>
   DBMS_OUTPUT.PUT_LINE('GOTO statement jumped here');
END;
```

EXIT

The `EXIT` statement is used to exit a loop. Here's an example with a simple loop:

```
DECLARE
   counter NUMBER := 1;
BEGIN
   LOOP
      EXIT WHEN counter > 5;
      DBMS_OUTPUT.PUT_LINE('Counter: ' || counter);
      counter := counter + 1;
   END LOOP;
END;
```

Iterative Statements in PL/SQL

Iterative control Statements are used when we want to repeat the execution of one or more statements for specified number of times.

There are three types of loops in PL/SQL:

Simple Loop

A Simple Loop is used when a set of statements is to be executed at least once before the loop terminates. An EXIT condition must be specified in the loop, otherwise the loop will get into an infinite number of iterations. When the EXIT condition is satisfied the process exits from the loop.

General Syntax to write a Simple Loop is:

```
LOOP
   statements;
   EXIT;
   {or EXIT WHEN condition;}
END LOOP;
```

These are the important steps to be followed while using Simple Loop.
1) Initialise a variable before the loop body.
2) Increment the variable in the loop.
3) Use a EXIT WHEN statement to exit from the Loop. If you use a EXIT statement without WHEN condition, the statements in the loop is executed only once.

While Loop

A WHILE LOOP is used when a set of statements has to be executed as long as a condition is true. The condition is evaluated at the beginning of each iteration. The iteration continues until the condition becomes false.

The General Syntax to write a WHILE LOOP is:

```
WHILE <condition>
LOOP statements;
END LOOP;
```

Important steps to follow when executing a while loop:
1) Initialise a variable before the loop body.
2) Increment the variable in the loop.
3) EXIT WHEN statement and EXIT statements can be used in while loops but it's not done oftenly.

FOR Loop

A FOR LOOP is used to execute a set of statements for a predetermined number of times. Iteration occurs between the start and end integer values given. The counter is always incremented by 1. The loop exits when the counter reachs the value of the end integer.

The General Syntax to write a FOR LOOP is:

```
FOR counter IN val1..val2
  LOOP statements;
END LOOP;
```

Important steps to follow when executing a For loop:

1) The counter variable is implicitly declared in the declaration section, so it's not necessary to declare it explicity.
2) The counter variable is incremented by 1 and does not need to be incremented explicitly.
3) EXIT WHEN statement and EXIT statements can be used in FOR loops but it's not done often.

Procedures

A **subprogram** is a program unit/module that performs a particular task. These subprograms are combined to form larger programs. This is basically called the 'Modular design'. A subprogram can be invoked by another subprogram or program which is called the **calling program**.

PL/SQL provides two kinds of subprograms −

- **Functions** − These subprograms return a single value; mainly used to compute and return a value.
- **Procedures** − These subprograms do not return a value directly; mainly used to perform an action.

Creating a Procedure

A procedure is created with the **CREATE OR REPLACE PROCEDURE** statement. The simplified syntax for the CREATE OR REPLACE PROCEDURE statement is as follows −

```
CREATE [OR REPLACE] PROCEDURE procedure_name
 [(parameter_name [IN | OUT | IN OUT] type [, ...])]
 {IS | AS}
 BEGIN
< procedure_body >
 END procedure_name;
```

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.
- The optional parameter list contains name, mode and types of the parameters. **IN** represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.

- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

Example

The following example creates a simple procedure that displays the string 'Hello World!' on the screen when executed.

```
CREATE OR REPLACE PROCEDURE greetings
 AS
BEGIN
    dbms_output.put_line('Hello World!');
 END;
 /
```

Executing a Standalone Procedure

A standalone procedure can be called in two ways –

- Using the **EXECUTE** keyword
- Calling the name of the procedure from a PL/SQL block

The above procedure named **'greetings'** can be called with the EXECUTE keyword as –

```
EXECUTE greetings;
```

Exceptions

In PL/SQL, exceptions are used to handle errors that might occur during the execution of a program. An exception is an abnormal condition that arises during the execution of a block of code. PL/SQL provides a mechanism to handle these exceptions, allowing the program to gracefully respond to errors.

```
DECLARE
   -- Declaration section (optional)
   variable1 NUMBER;
BEGIN
   -- Executable section
   -- PL/SQL statements that may cause exceptions
EXCEPTION
   WHEN exception_name1 THEN
      -- Handle exception 1
   WHEN exception_name2 THEN
```

```
      -- Handle exception 2
   WHEN OTHERS THEN
      -- Handle any other exceptions
END;
/
```

- **Declaration section (optional):** This section is used to declare variables, constants, types, and cursors that will be used within the block. It is optional and depends on the requirements of your code.
- **Executable section:** This is the main part of the block where you write the PL/SQL statements that may raise exceptions.
- **EXCEPTION:** This keyword introduces the exception-handling section. It is followed by one or more `WHEN` clauses, each specifying an exception condition.
- **WHEN exception_name THEN:** This clause is used to handle a specific exception. `exception_name` is the name of the predefined or user-defined exception. For example, `NO_DATA_FOUND` is a predefined exception.
- **WHEN OTHERS THEN:** This clause is optional and handles any other exceptions that are not explicitly caught by the previous `WHEN` clauses. It is a catch-all for unhandled exceptions.

---

database trigger

In PL/SQL, a database trigger is a set of instructions associated with a specific event that occurs in the database. Triggers are automatically executed (or "triggered") in response to events such as INSERT, UPDATE, DELETE, or DDL (Data Definition Language) statements. They are useful for enforcing business rules, maintaining data integrity, and automating certain actions when specific events occur.

Here is the basic syntax for creating a PL/SQL database trigger:

```
CREATE [OR REPLACE] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF} -- Event type
{INSERT | UPDATE | DELETE | {DROP | ALTER | CREATE} ON table_name}
[FOR EACH ROW | FOR EACH STATEMENT]
DECLARE
   -- Declarations section (optional)
BEGIN
   -- Trigger body: PL/SQL statements
   -- This section contains the logic to be executed when the trigger
is fired
```

```
EXCEPTION
   -- Exception handling (optional)
END trigger_name;
/
```

- **CREATE [OR REPLACE] TRIGGER:** This clause is used to create a new trigger. The `OR REPLACE` option allows you to modify an existing trigger without dropping it first.
- **trigger_name:** This is the name of the trigger.
- **BEFORE | AFTER | INSTEAD OF:** This clause specifies when the trigger should be fired in relation to the triggering event.
  - `BEFORE`: The trigger fires before the event (e.g., BEFORE INSERT).
  - `AFTER`: The trigger fires after the event (e.g., AFTER UPDATE).
  - `INSTEAD OF`: Used with views to specify that the trigger should replace the default action of the triggering event.
- **INSERT | UPDATE | DELETE | {DROP | ALTER | CREATE} ON table_name:** This clause specifies the event or DDL statement that triggers the execution of the trigger. You can choose one or more events based on your requirements.
- **FOR EACH ROW | FOR EACH STATEMENT:** This clause determines whether the trigger fires once for each affected row (e.g., FOR EACH ROW in an UPDATE) or once for each triggering statement (e.g., FOR EACH STATEMENT).
- **DECLARE:** The declarations section is optional and is used for declaring variables, constants, and other elements that may be used in the trigger body.
- **BEGIN:** The beginning of the trigger body, where you place the PL/SQL statements that define the logic to be executed when the trigger is fired.
- **EXCEPTION:** The exception-handling section is optional and is used to handle exceptions that may occur during the execution of the trigger.
- **END trigger_name:** Marks the end of the trigger body.

---

file input/output

In PL/SQL, file input/output operations are typically performed using the `UTL_FILE` package. The `UTL_FILE` package provides procedures and functions to read from and write to operating system files.

Here's a basic overview of how file input/output works in PL/SQL:

Open the File: Use `UTL_FILE.FOPEN` to open the file for reading. This function returns a file handle.

```
DECLARE
   file_handle UTL_FILE.FILE_TYPE;
BEGIN
   file_handle := UTL_FILE.FOPEN('DIRECTORY_NAME', 'FILE_NAME.txt',
'R');
END;
```

Read from the File: Use `UTL_FILE.GET_LINE` to read a line from the file.

```
DECLARE
   file_handle UTL_FILE.FILE_TYPE;
   file_line VARCHAR2(100);
BEGIN
   file_handle := UTL_FILE.FOPEN('DIRECTORY_NAME', 'FILE_NAME.txt',
'R');
   UTL_FILE.GET_LINE(file_handle, file_line);
END;
```

Close the File: Always close the file after reading.

```
DECLARE
   file_handle UTL_FILE.FILE_TYPE;
BEGIN
   file_handle := UTL_FILE.FOPEN('DIRECTORY_NAME', 'FILE_NAME.txt',
'R');
   -- Read from the file
   UTL_FILE.FCLOSE(file_handle);
END;
```

Write to the File: Use `UTL_FILE.PUT_LINE` to write a line to the file.

```
DECLARE
   file_handle UTL_FILE.FILE_TYPE;
BEGIN
   file_handle := UTL_FILE.FOPEN('DIRECTORY_NAME', 'FILE_NAME.txt',
'W');
   UTL_FILE.PUT_LINE(file_handle, 'This is a line of text.');
END;
```