L.J. Polytechnic, Ahmedabad
Unit-1
Introduction to Database Management System

❖ Define: Data, Information, Records, Metadata, System Catalog

Answer:

Data:

Data means known facts, that can be recorded and have implicit meanings.

Information:

Information means processed or organized data.

Record:

A record is a collection of logically related fields.

Metadata:

A metadata is the data about data.

System Catalog:

A system catalog is a repository (or container) for a metadata.

❖ Explain Applications of DBMS

Answer:

- **Banking System:** We make thousands of transactions through banks daily and we can do this without going to the bank. So how banking has become so easy that by sitting at home we can send or get money through banks. That is all possible just because of DBMS that manages all the bank transactions.

- **Online Shopping:** Online shopping has become a trendy now-a-day. Everyone wants to shop from home. So, all these products are added and sold only with the help of DBMS. Purchase information, invoice bills, and payment, all of these are done with the help of DBMS.

- **Social Media Sites:** We all are on social media websites to share our views and connect with our friends. Daily millions of users signed up for these social media accounts like Facebook, Twitter, Pinterest, and Whatsapp. But how all the information of users is stored and how we become able to connect to other people, yes this is all because of DBMS.

- **Airline Reservation System or Railway Reservation System:** The airline needs DBMS to keep records of flights arrival, departure, and delay status. Same as it is Railway reservation system needs to keep up-to-date records and maintain train track off.

- **Library Management System:** There are thousands of books in the library so it is very difficult to keep record of all the books in a copy or register. So DBMS is used to maintain all the information related to book issuing dates, name of the book, author, and availability of the book.

❖ Differentiate DA and DBA and Explain Functions & Responsibilities of DBA

Answer:

Data Administrator	**Database Administrator**
The data administrator is a person in the organization who controls the data of the database.	The database administrator is a person in the organization who controls the design and the use of the database.
DA determines what data to be stored in database based on requirement of the organization.	DBA provides necessary technical support for implementing a database.
DA is involved more in the requirements gathering, analysis and design phase.	DBA is involved more in design, development, testing and operational phases.
DA is a manager or some senior level person in an organization who understands organizational requirements with respect to data.	DBA is a technical person having knowledge of database technology.
DA does not need to be a technical person.	DBA does not need to be a business person.

❖ Functions and Responsibility of DBA:

Answer:

1. **Schema Definition**
- The DBA defines the logical schema of the database.
- According to this schema, database will be developed to store required data for an organization.
2. **Storage Structure and Access Method Definition**
- The DBA decides how the data is to be represented in the stored database.
- Based on this, storage structure of the database and access methods of data is defined.
3. **Assisting Application Program**
- The DBA provides assistance to application programmers to develop application programs.
4. **Physical Organization Modification**
- The DBA modifies the physical organization of the database to reflect the changing needs of the organization or to improve performance.
5. **Approving Data Access**
- The DBA determines which user needs access to which part of the database.
- According to this, various types of authorization are granted to different users. This is required to prevent unauthorized access of a database.
6. **Monitoring Performance**
- The DBA monitors performance of the system. The DBA ensures that better performance is maintained by making changes in physical or logical schema if required.
7. **Backup and Recovery**
- Database is a valuable asset for any organization. It should not be lost or damaged.
- The DBA ensures this by periodically backing up the database on magnetic tapes or remote servers.
- In case of failures, such as flood or virus attack, database is recovered from this backup.

❖ Write short note on Data Dictionary.

Answer:

- A data dictionary contains metadata i.e data about the database.
- The data dictionary is very important as it contains information such as what is in the database, who is allowed to access it, where is the database physically stored etc.

- The users of the database normally don't interact with the data dictionary, it is only handled by the database administrators.
- The data dictionary in general contains information about the following −
- Names of all the database tables and their schemas.
- Details about all the tables in the database, such as their owners, their security constraints, when they were created etc.
- Physical information about the tables such as where they are stored and how.
- Table constraints such as primary key attributes, foreign key information etc.
- Information about the database views that are visible.
- The different types of data dictionary are −
- **Active Data Dictionary**
- If data dictionary is managed automatically by database management software, it is called as an active data dictionary.
- It is also called as Integrated data dictionary.
- It always consistent with the current definition and structure of the database, because it is managed by database itself.
- **Passive Data Dictionary**
- This is not as useful or easy to handle as an active data dictionary.
- A passive data dictionary is maintained separately to the database whose contents are stored in the dictionary. That means that if the database is modified the database dictionary is not automatically updated as in the case of Active Data Dictionary.
- So, the passive data dictionary has to be manually updated to match the database.
- This needs careful handling or else the database and data dictionary are out of sync.

❖ **Write a short note on Data Warehouse.**

Answer:

- “A data warehouse is a decision support database that is maintained separately from the organization’s operational database.”
- Data stored in data warehouse possess following characteristics:
  1. Subject-oriented
  2. Integrated
  3. Time-variant
  4. Non-volatile

1. **Subject-oriented**
   - Data warehouse is organized around major subjects or topics of an organization.
   - These subjects are those for which there may be a need to take some decision. For example, customer, products, sales, etc
2. **Integrated**
   - Data to be stored in a data warehouse might be coming from different locations, such as from different branches of a company or bank.
   - It is also possible that this data may have different structures according to their source.
   - All such kind of data is integrated in data warehouse. This means, it is converted to some homogenous structure.
3. **Time-variant**
   - Data warehouse provides information from a historical perspective.
   - Data stored in data warehouse contains time element to reflect timing of the data.

- So, data in data warehouse indicates what happened previous day, last week, last month, during past two years and so on.

4. **Non-volatile**
   - Data stored in data warehouse is never deleted or updated. New data is always added as a supplement to the database on a regular basis.

❖ Explain Advantages of File Oriented System

Answer:

1. **Back-up**

- It is possible to take faster and automatic back-up of database stored in files of computer-based systems.
- Computer systems provide functionalities to serve this purpose. It is also possible to develop specific application programs for this purpose.

2. **Compactness**

- It is possible to store data compactly.
- For example, to store all words of English Dictionary, only few kilobytes of memory is required in computer based systems.

3. **Data Retrieval**

- Computer based systems provide enhanced data retrieval techniques to retrieve data stored in files in easy and efficient way.

4. **Editing**

- It is easy to edit any information stored in computers in form of files.
- Specific application programs or editing software can be used for this purpose.

5. **Remote Access**

- In computer-based systems, it is possible to access data remotely.
- So, to access data, it is not necessary for a user to remain present at location where these data are kept.

6. **Sharing**

- Data stored in file of computer-based systems can be shared among multiple users at a same time.

❖ Explain Disadvantages of File-oriented System

Answer:

1. Data Redundancy:

- It is possible that the same information may be duplicated in different files.this leads to data redundancy results in memory wastage.

2. Data Inconsistency:

- Because of data redundancy, it is possible that data may not be in consistent state.

3. Difficulty in Accessing Data:

- Accessing data is not convenient and efficient in file processing system.

4. Limited Data Sharing:

- Data are scattered in various files and different files may have different formats and these files may be stored in different folders may be of different departments.
- So, due to this data isolation, it is difficult to share data among different applications.

5. Integrity Problems:

- Data integrity means that the data contained in the database in both correct and consistent. For this purpose, the data stored in database must satisfy correct and constraints.

6. Concurrent Access Anomalies:

- Multiple users are allowed to access data simultaneously, this is for the sake of better performance and faster response.

7. Security:

- Database should be accessible to users in limited way. Each user should be allowed to access data concerning his requirements only.

❖ Explain Advantages of Database Management System

1. Minimal Data Redundancy

- Due to centralized database, it is possible to avoid unnecessary duplication of information.
- For example, all the information about bank customer can be kept centralized. Both accounts – Saving and Current – can share this information.
- This prevents unnecessary duplication of customer information who has both type of accounts.

2. Improved Data Consistency

- Data inconsistency occurs due to data redundancy.
- For example, consider that customer information is maintained separately for both accounts.
- Now, if the address of some customer changes which has both kinds of accounts, it is possible that his/her address is updated for one account, leaving the other one as it is.
- With reduced data redundancy, such type of data inconsistency can be eliminated.

3. Efficient Data Access

- DBMS utilize a variety of techniques to retrieve data.
- For example, information about all customers or from some particular city can be retrieved easily by providing appropriate query statements.

4. Improved Data Sharing

- As database is maintained centrally, all authorized users and application programs can share this database easily.

5. Improved Data Integrity

- Data integrity means that the data contained in the database is both correct and consistent.
- For this purpose, the data stored in database must satisfy certain types of constraints (rules).
- For example, balance in an account should not be a negative value.

6. Guaranteed Atomicity

- Any operation in database must be atomic. This means, it must happen in its entirely or not at all.
- For example, a fund transfer from one account to another must happen in its entirely.

7. Improved Concurrent Access

- Multiple users are allowed to access data simultaneously.
- This is for sake of better performance and faster response.

8. Improved Security

- Database should be accessible to users in a limited way.
- Each user should be allowed to access data concerning his/her requirements only.
- For example, a customer can check balance only for his/her own account.

❖ Explain Disadvantages of Database Management System

- **Cost of Hardware and Software:** It requires a high speed of data processor and large memory size to run DBMS software.
- **Size:** It occupies a large space of disks and large memory to run them efficiently.
- **Complexity:** Database system creates additional complexity and requirements.
- **Higher impact of failure:** Failure is highly impacted the database because in most of the organization, all the data stored in a single database and if the database is damaged due to electric failure or database corruption then the data may be lost forever.

L.J. Polytechnic, Ahmedabad

Unit-2

Database Architecture

❖ **Define: Schema, Subschema, Instances**

Answer:

Schema: The overall logical design of database is known as Schema.

Subschema: The subset of schema and inherits the same property of schema is called as Subschema.

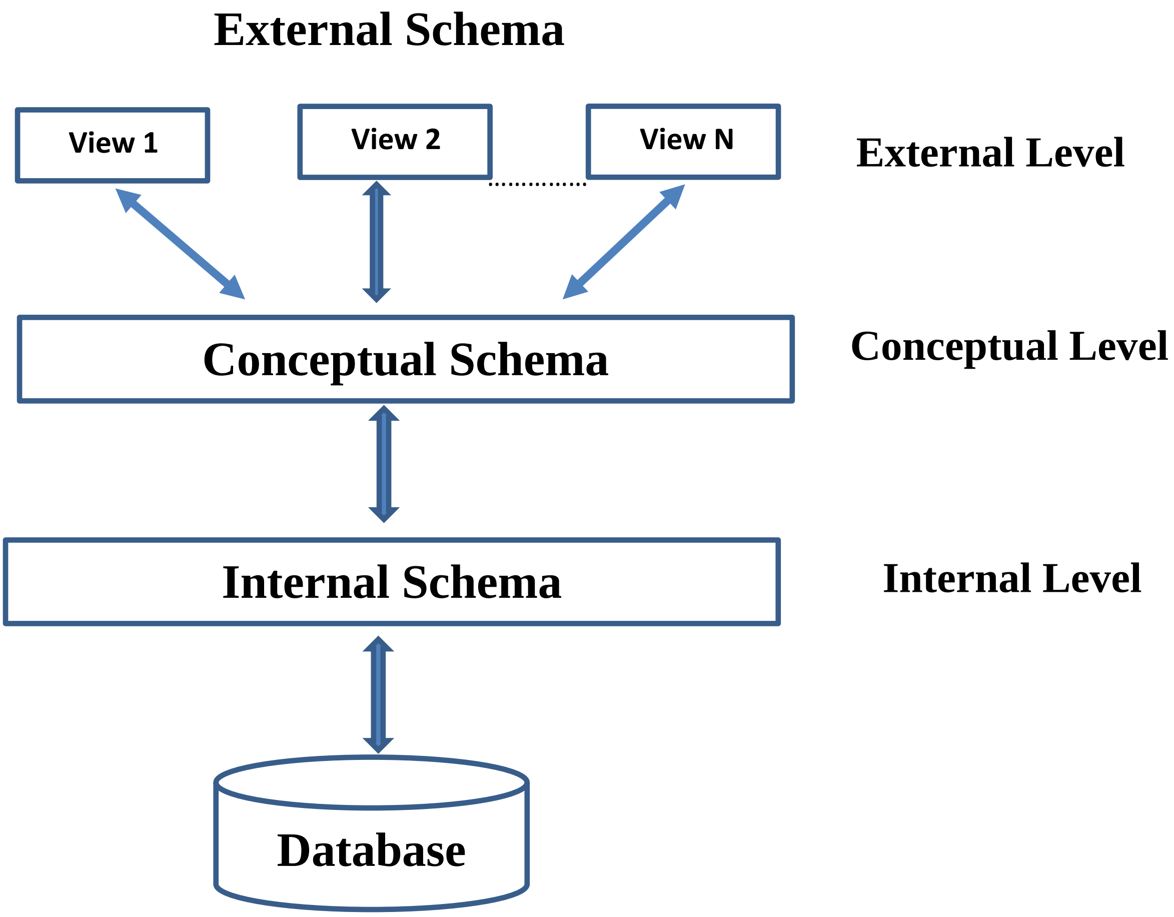
Instance: The collection of data items values or content of database at any point of time is called Instance.

❖ **Explain Three-tier ANSI SPARC Database Architecture**

Answer:

In 1975, American National Standards Institute-Standard Planning and Requirement Committee introduced three-tier or three-level architecture for the database system. The architecture of most of the commercial DBMSs available now-a-days is mostly based on ANSI-SPARC database architecture.

ANSI-SPARC three-level database architecture is shown in figure.



Three-level Database Architecture

It contains three levels as shown in figure.

1. External Level
2. Conceptual Level
3. Internal Level

These three levels provide data abstraction means hide low level complexities from the end users. A database system should be efficient in performance and convenient in use. It is possible to use complex structures at internal level for efficient operations and make it simpler and convenient interface at external level.

The following sections describe three levels of architecture in detail.

1. **External Level:** At the external level, a database contains several schemas that sometimes called as subschema. The subschema is used to describe the different view of the database.
   An external schema is also known as view schema. Each view schema describes the database part that a particular user group is interested and hides the remaining database from that user group. The view schema describes the end user interaction with database systems.

2. **Conceptual Level:** The conceptual schema describes the design of a database at the conceptual level. Conceptual level is also known as logical level. The conceptual schema describes the structure of the whole database.
   The conceptual level describes what data are to be stored in the database and also describes what relationship exists among those data. In the conceptual level, internal details such as an implementation of the data structure are hidden. Programmers and database administrators work at this level.

3. **Internal Level:** The internal level has an internal schema which describes the physical storage structure of the database. The internal schema is also known as a physical schema. It uses the physical data model.
   It is used to define that how the data will be stored in a block. The physical level is used to describe complex low-level data structures in detail.

❖ **Write a short note on Data Independence.**

Answer:

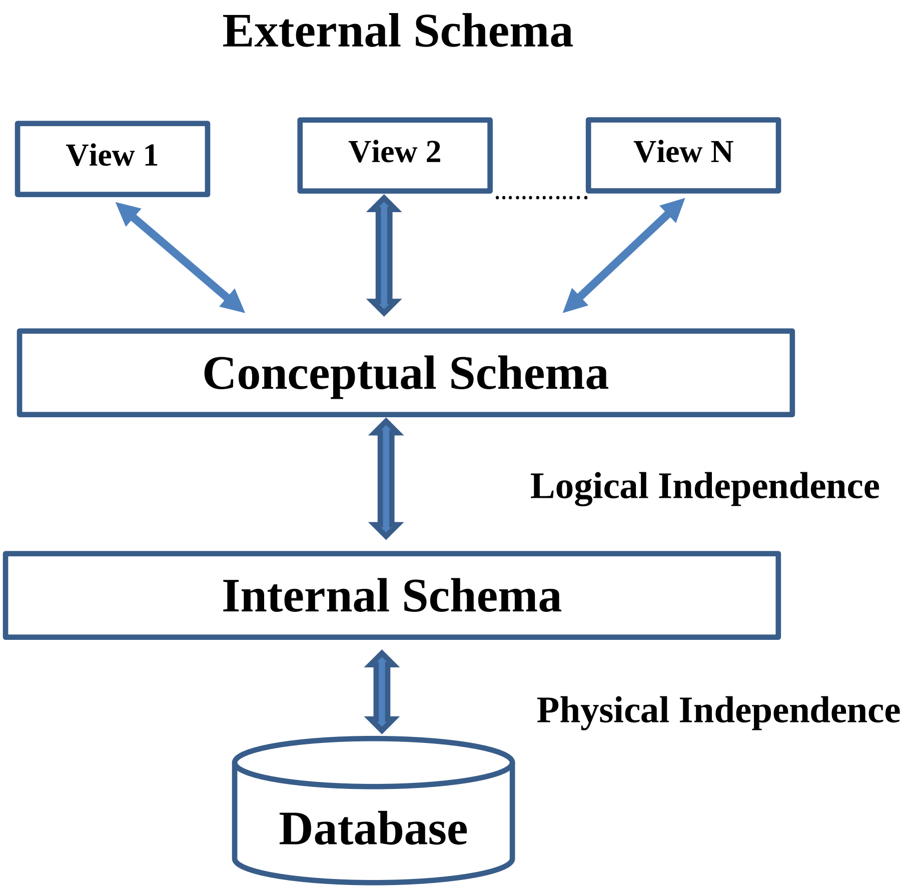
Data Independence is ability to modify the one schema definition without affecting another schema definition in the next higher level.

In other words, the application programs do not depend on any particular physical representation or access technique. The Data Independence in achieved by DBMS through the use of the three-tier architecture of the data abstraction.

There are two types of Data Independence as shown in figure.

Physical Data Independence is ability to modify the physical schema without affecting next higher level schema definition.

Logical Data Independence is ability to modify the logical schema definition without affecting next higher level schema definition.



- **Write a short note on Mapping in detail.**

Answer:

Process of transforming request and results between three level it's called Mapping.

There are the two types of Mappings:

Conceptual/Internal Mapping

External/Conceptual Mapping

Conceptual/Internal Mapping:

The conceptual/internal Mapping defines the correspondence between the conceptual view and the store database. It specifies how conceptual record and fields are represented at the internal level. It relates conceptual schema with internal schema.

If structure of the store database is changed. If changed is made to the storage structure definition-then the conceptual/internal Mapping must be changed accordingly, so that the conceptual schema can remain invariant. There could be one Mapping between conceptual and internal levels.

External/Conceptual Mapping:

The external/conceptual Mapping defines the correspondence between a particular external view and conceptual view. It relates each external schema with conceptual schema. The differences that can exist between these two levels are analogous to those that can exist between the conceptual view and the stored database.

Example: Fields can have different data types; fields and record name can be changed; several conceptual fields can be combined into a single external field. Any number of external views can exist at the same time; any number of users can share a given external view: different external views can overlap. There could be several Mapping between external and conceptual levels.

❖ **List and explain various types of Data Model.**

Answer:

A Data Model in Database Management System (DBMS), is the concept of tools that are developed to summarize the description of the database**.**

A Database model defines the logical design and structure of a database and defines how data will be stored, accessed and updated in a database management system.

Data Models are classified in four ways:

Hierarchical Model

Network Model

Entity-relationship Model

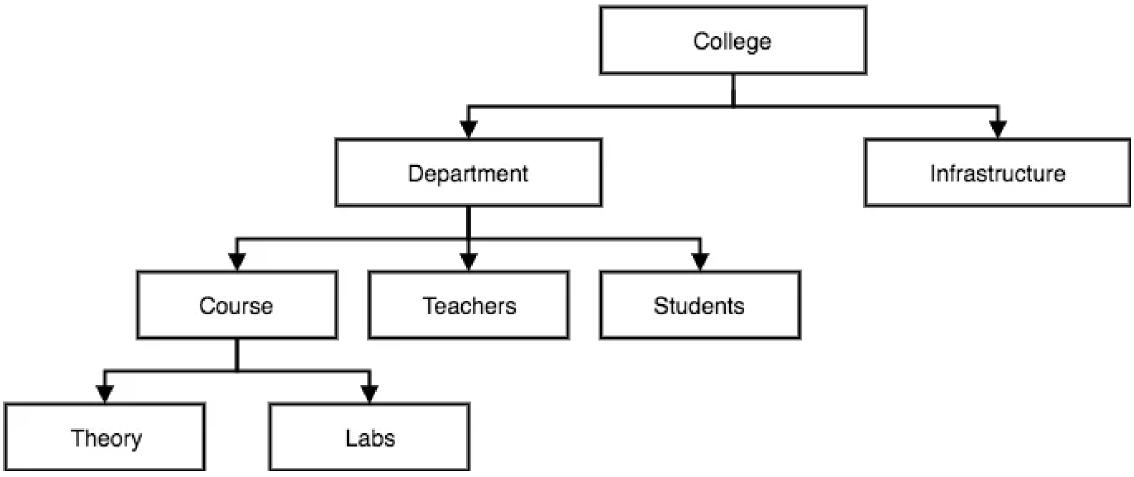
Relational Model

Hierarchical Model

This database model organises data into a tree-like-structure, with a single root, to which all the other data is linked. The hierarchy starts from the Root data, and expands like a tree, adding child nodes to the parent nodes.

In this model, a child node will only have a single parent node. This model efficiently describes many real-world relationships like index of a book, recipes etc.

In hierarchical model, data is organised into tree-like structure with one one-to-many relationship between two different types of data, for example, one department can have many courses, many professors and of-course many students.

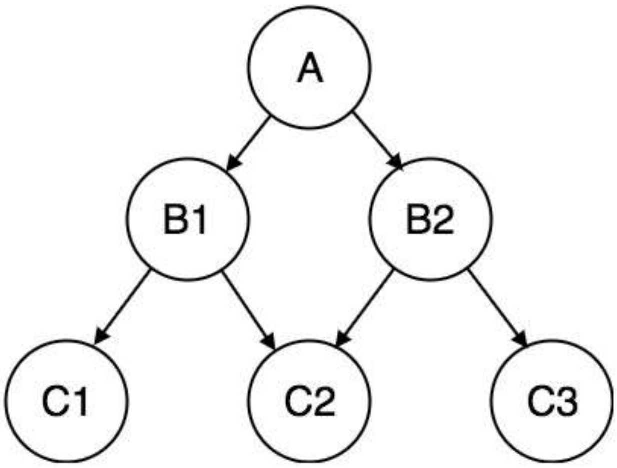


Network Model

This is an extension of the Hierarchical model. In this model data is organised more like a graph, and are allowed to have more than one parent node.

In this database model data is more related as more relationships are established in this database model. Also, as the data is more related, hence accessing the data is also easier and fast. This database model was used to map many-to-many data relationships.

This was the most widely used database model, before Relational Model was introduced.

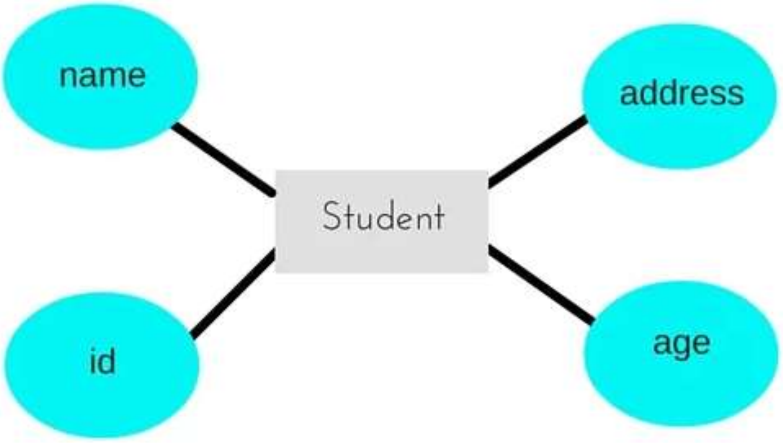


Entity-relationship Model

In this database model, relationships are created by dividing object of interest into entity and its characteristics into attributes.

Different entities are related using relationships.E-R Models are defined to represent the relationships into pictorial form to make it easier for different stakeholders to understand. This model is good to design a database, which can then be turned into tables in relational model.

Let's take an example, If we have to design a School Database, then **Student** will be an **entity** with **attributes** name, age, address etc. As **Address** is generally complex, it can be another **entity** with **attributes** street name, pincode, city etc, and there will be a relationship between them.

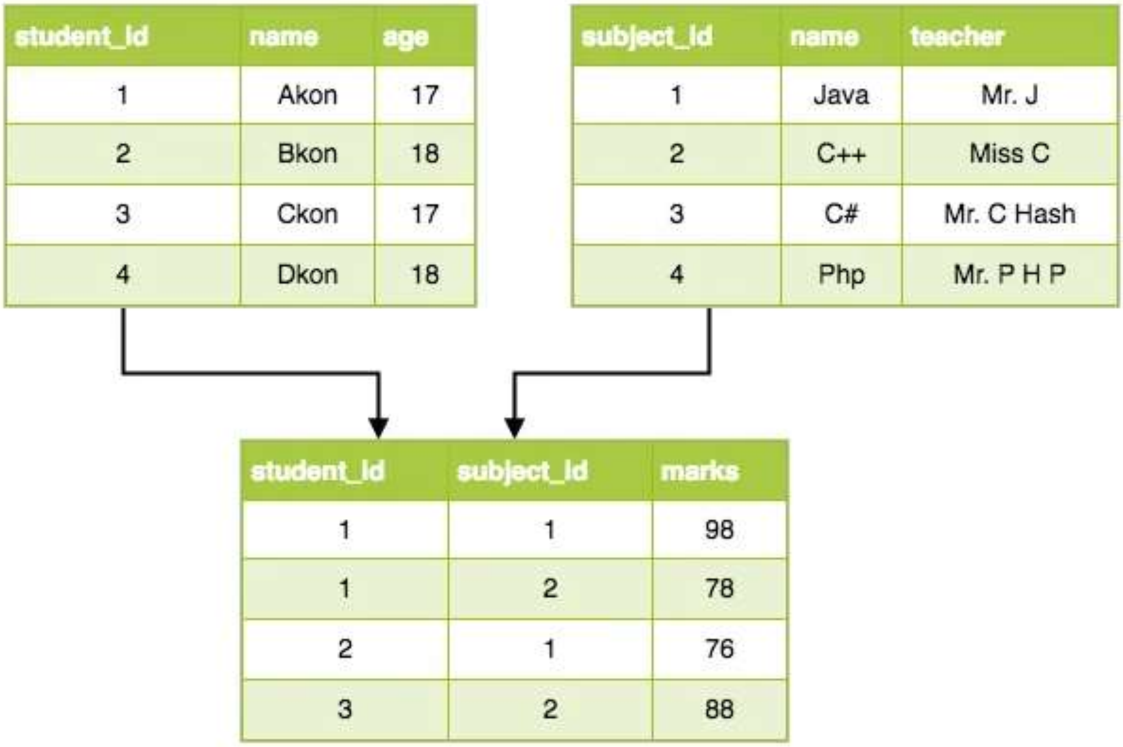


Relational Model

In this model, data is organised in two-dimensional tables and the relationship is maintained by storing a common field. This model was introduced by E.F Codd in 1970, and since then it has been the most widely used database model, in fact, we can say the only database model used around the world.

The basic structure of data in the relational model is tables. All the information related to a particular type is stored in rows of that table. Hence, tables are also known as relations in relational model.

In the coming tutorials we will learn how to design tables, normalize them to reduce data redundancy and how to use Structured Query language to access data from tables.



student_id	name	age
1	Akon	17
2	Bkon	18
3	Ckon	17
4	Dkon	18

subject_id	name	teacher
1	Java	Mr. J
2	C++	Miss C
3	C#	Mr. C Hash
4	Php	Mr. P H P

student_id	subject_id	marks
1	1	98
1	2	78
2	1	76
3	2	88

L.J. Polytechnic, Ahmedabad
Unit-3
Structured Query Language

- **Give full name of SQL and Explain different data types of oracle.**

Answer:

SQL: Structured Query Language

Data Types of Oracle:

1. Numerical Data Type
2. Character/String Data Type
3. Date Data Type
4. Binary Data Type

1. Numerical Data Type:

- Used to store zero, negative and positive numerical values. These values can be fixed-point or floating-point.
- Numbers can range between 1.0x10^{-130} and 1.0x10^{126}

2. Character/String Data Type:

1. CHAR (size):

- Stores character string of fixed length.
- Size represents the no. of characters (length) to be stored. Default size is 1.
- Maximum length is 255 characters.

2. VARCHAR (size)/ VARCHAR2 (size):

- Stores character strings of variable length.
- More flexible than CHAR.
- No default size will be considered. So, size must be specified explicitly.
- Maximum length is 2000 characters.

3. LONG:

- Stores large amount of character strings of variable length.
- Maximum length is up to 2 GB.
- Only one column per table can be defined as LONG.
- A LONG column cannot be indexed.
- A LONG column cannot be used as argument or return value in procedures or functions.

3. Date Data Type:

- Used to store date and time.
- The standard format is DD-MON-YY to store date, such as 01-APR-07.
- The current date and time can be retrieved using function SYSDATE.
- Addition and subtraction operation are possible using number constants and other dates. For example, SYSDATE + 7 will return same day of next week

4. Binary Data Type:

1. RAW:

- Stores binary type data.
- Maximum length is up to 255 bytes.

2. **LONG RAW:**

- Stores large amount of binary type data.
- Often referred as binary large object (BLOB).
- Maximum length is up to 2 GB.
- A LONG RAW column cannot be indexed.

❖ Explain DDL commands with suitable example.

Answer:

- It is a set of SQL commands used to create, modify and delete database objects such as tables, views, indices, etc.
- It is normally used by DBA and database designers.
- It provides commands like – CREATE, ALTER, DROP, TRUNCATE

CREATE:

Syntax:

```
Create table tablename(
ColumnName1 datatype(size),
ColumnName2 datatype(size),
:
:
ColumnNameN datatype(size)
);
```

Description:

- A tableName should be unique.
- A tableName and columnName must start with alphabet.
- TableName and columnName are not case sensitive.

Example:

Create an Account table having three columns for account number, balance, branch name.

Input:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3),
BALANCE NUMBER(9),
BNAME VARCHAR2(10)
);
```

Output:

```
Table Created
```

ALTER:

- The ALTER TABLE command can be used to alter the schema of a table.

Adding New Columns:

- This command adds a new column or columns in an exiting table.

Example:

Add new column acc_type in an Account table.

Input:

```
ALTER TABLE ACCOUNT
ADD (ACC_TYPE CHAR(10));
```

Output:

```
Table altered.
```

Dropping Columns:

- This command deletes an existing column from the table along with data held by the column.

Example:

Drop a column acc_type from the Account table.

```
ALTER TABLE ACCOUNT
DROP COLUMN ACC_TYPE;
```

Output:

```
Table altered.
```

Modifying Existing Columns:

- An existing column can be modified in one of the two ways:
  1. Change the **datatype** of a column
  2. Change the **size** of a column

Example:

Alter column *ano* to hold maximum 20 characters with datatype VARCHAR2 in table Account.

Input:

```
ALTER TABLE ACCOUNT
MODIFY(ANO VARCHAR(20));
```

Output:

```
Table altered.
```

DROP:

- The DROP TABLE command drops the specified table.
- This means, all records along with structure (or schema) of the table will be destroyed.

Example:

Remove the table Account along with the data held.

Input:

```
DROP TABLE ACCOUNT;
```

Output:

```
Table dropped.
```

TRUNCATE:

- The TRUNCATE TABLE command empties given table completely.
- Truncate operation drops and re-creates the tables.

Example:

Truncate the table Account.

Input:

```
TRUNCATE TABLE ACCOUNT;
```

Output:

```
Table truncated.
```

❖ Explain DML commands with suitable example.

Answer:

- It is a set of SQL commands used to insert, modify and delete data in a database.
- It is normally used by general users who are accessing database via pre-developed applications.
- It provides commands like - INSERT, UPDATE, DELETE, LOCK

INSERT:

- To insert user data into tables, “INSERT INTO…” SQL statement is used.
- This statement creates a new row (record) in a table, and stores the values into respective columns.

Syntax:

```
INSERT INTO tableName (column1, column2, …., columnN)
```

```
VALUES (expression1, expression2, …., expressionN);
```

- Characters and date constants are enclosed within *single* quotes.
- There is one to one mapping between column specified and expressions passed.

Example:

```
INSERT INTO ACCOUNT(ANO, BALANCE, BNAME)
VALUES ('A01',5000,'VVN');
```

Output:

```
1 row created.
```

UPDATE:

- The update command can be used to change or modify data values in a table.
- It can be used to update either all rows or a set of rows from a table.

Syntax:

```
UPDATE tableName
SET column1 = expression1, column2 = expression2
WHERE condition;
```

- UPDATE command updates rows from the table, and displays message regarding how many rows have been updated.
- The SET clause specifies which column data to modify.
- An expression can be a constant value, a variable, or some expression and it specifies new value for related columns.

Example:

Change branch name from ‘ksad’ to’ karamsad’ for each account which belongs to ‘ksad’ branch.

Input:

```
UPDATE ACCOUNT
SET BNAME='KARAMSAD'
WHERE BNAME='KSAD';
```

Output:

```
2 rows updated.
```

DELETE:

- The DELETE command can be used to remove either all rows of a table, or a set of rows from a table.

Example:

Delete all accounts which belong to ‘ksad’ branch.

Input:

```
DELETE FROM ACCOUNT
WHERE BNAME='KSAD';
```

Output:

```
2 rows deleted.
```

❖ **Explain DISTINCT, GROUP BY, ORDER BY, HAVING with purpose and example.**

Answer:

DISTINCT:

- The DISTINCT keyword removes duplicate rows from the result. It can be used only with SELECT statement.

Syntax:

```
SELECT DISTINCT column1, column2, …, column FROM tableName;
```

Example:

Display only branch names for an Account table with distinct.

Input:

```
SELECT DISTINCT BNAME FROM ACCOUNT;
```

Output:

```
BNAME
----------
VVN
ANAND
KSAD
```

Order By:

- This statement retrieves data in a sorted manner.
- Rows are sorted based on values of columns specified with ORDER BY clause.
- By default, order is considered an Ascending order. To sort data in descending order, it is necessary to specify DESC as an order.

Example:

Display all accounts according to their branch name.

Input:

```
SELECT * FROM ACCOUNT
ORDER BY BNAME;
```

Output:

```
ANO   BALANCE    BNAME
---   ----------   ----------
A03    7000        ANAND
A02    6000        KSAD
A04    8000        KSAD
A05    6000        VVN
A01    5000        VVN
```

Display all accounts sorted in descending order according to their balance.

Input:

```
SELECT * FROM ACCOUNT
ORDER BY BALANCE DESC;
```

Output:

```
ANO   BALANCE    BNAME
---   ----------   ----------
A04    8000        KSAD
A03    7000        ANAND
A05    6000        VVN
A02    6000        KSAD
A01    5000        VVN
```

Group By and Having:

- The GROUP BY clause groups records based on distinct values for specified columns.
- The HAVING clause filters groups based on the specified condition.

Example:

Display total balance of different accounts for 'vvn' branch.

Input:

```
SELECT BNAME, SUM(BALANCE) "TOTAL BALANCE" FROM ACCOUNT
GROUP BY BNAME
HAVING BNAME='VVN';
```

Output:

```
BNAME        TOTAL BALANCE
----------   -------------
VVN           22000
```

❖ Differentiate ORDER BY v/s GROUP BY.

ORDER BY	GROUP BY
It is used to group the rows that have the same values.	It sorts the result set either in ascending or descending order.
It controls the presentation of rows.	It controls the presentation of columns.
The attribute cannot be under aggregate function under GROUP BY statement.	The attribute can be under aggregate function under GROUP BY statement.
It is always used before the ORDER BY clause in the SELECT statement.	It is always used after the GROUP BY clause in the SELECT statement.
It is mandatory to use aggregate functions in the GROUP BY.	It is not mandatory to use aggregate functions in ORDER BY.
Here, the grouping is done based on the similarity among the row's attribute values.	Here, the result-set is sorted based on the column's attribute values, either ascending or descending order.
It may be allowed in CREATE VIEW statement.	It is not allowed in CREATE VIEW statement.

❖ Write short note on: Dual table, Sysdate, Arithmetic Operators, Logical Operators

Answer:

Dual Table:

- Dual table is a dummy table. It is provided by Oracle itself, i.e. it is in-built table.
- It contains only one row and one column with single value **x**.
- The dual table can be used to display output of operations which do not refer to any table.

Example:

Calculate the result of mathematical expression 5*3.

Input:

```
SELECT 5 * 3 FROM dual;
```

Output:

```
5 * 3
------
5
```

Sysdate:

- SYSDATE is a "pseudo column" that contains the current date and time.
- A pseudo column is a column that returns a value when it is selected, but it is not an actual column in the table, i.e. it does not belong to any specific table.

Example:

Display the current date.

Input:

```
SELECT SYSDATE FROM dual;
```

Output:

```
SYSDATE
--------------
14-FEB-13
```

Arithmetic Operators:

- Arithmetic operations can be performed while viewing table data, or while manipulating table data with insert, update or delete operation.
- SQL supports following operators for arithmetic operation.

+	Addition	-	Subtraction
*	Multiplication	/	Division
( )	Enclosed operation		

Example:

Input:

```
SELECT 5 * 3 FROM dual;
```

Output:

```
5 * 3
------
5
```

Logical Operators:

AND Operator:

- The AND operator is used to combine two or more conditions in WHERE and HAVING clauses.
- AND requires all the conditions to be true to consider entire clause true.

Example:

Find out the customer who belongs to ‘Anand’ city and has salary less than 17000.

Input:

```
SELECT * FROM EMPLOYEE
WHERE CITY = 'ANAND' AND SALARY < 17000;
```

Output:

```
EID   ENAME        BIRTHDATE    SALARY    CITY
----  ----------   ---------    ----------  ----------
E02   GOPI         15-AUG-83    15000     ANAND
```

OR Operator:

- The OR operator is also used to combine two or more conditions in WHERE and HAVING clauses.
- OR requires any one condition to be true to consider entire clause true.

Example:

Find out the customer who belongs to either ‘Anand’ city or ‘Surat’ city.

Input:

```
SELECT * FROM EMPLOYEE
WHERE CITY='ANAND' OR CITY='SURAT';
```

Output:

```
EID   ENAME       BIRTHDATE    SALARY    CITY
----  ----------  ---------    ----------  ----------
E02   GOPI        15-AUG-83    15000     ANAND
E04   VAISHALI    23-MAR-85    25000     SURAT
E05   LAXMI       14-FEB-83    18000     ANAND
```

NOT Operator:

- The NOT operator is used to negate the result of any condition or group of conditions.

Example:

Find out the customers who do not belong to ‘Anand’ city.

Input:

```
SELECT * FROM EMPLOYEE
WHERE NOT (CITY='ANAND');
```

Output:

```
EID   ENAME       BIRTHDATE    SALARY    CITY
----  ----------  ---------    ----------  ----------
E01   TULSI       26-JAN-82    12000     AHMEDABAD
E03   RAJSHREE    31-OCT-84    20000     VADODARA
E04   VAISHALI    23-MAR-85    25000     SURAT
```

❖ **Write short note on: Relational Operators, Range Searching Operators, Set Searching Operators, Character Operators**

Answer:

Relational Operators:

- SQL supports following relational operators to perform comparisons among values.

=	Equals	!= or < >	Not equals
<	Less than	< =	Less than or equal to
>	Greater than	> =	Greater than or equal to

- These relation operators compare expressions and return one of three values: True, False or Unknown.

Example:

Find out the customer who has salary less than 17000.

Input:

```
SELECT * FROM EMPLOYEE
WHERE SALARY < 17000;
```

Output:

```
EID   ENAME       BIRTHDATE    SALARY    CITY
----  ----------  ---------    ----------  ----------
E02   GOPI        15-AUG-83    15000     ANAND
```

Range Searching Operators:

BETWEEN Operator:

- Selects rows that contain values within a specified lower and upper limit.
- The lower and upper limit values must be linked with the keyword AND.

Example:

Display employees having salary between 15000 and 20000.

Input:

```
SELECT * FROM EMPLOYEE
WHERE SALARY BETWEEN 15000 AND 20000;
```

Output:

```
EID   ENAME       BIRTHDATE   SALARY   CITY
----  ----------  ---------   ----------  ----------
E02   GOPI        15-AUG-83   15000    ANAND
E03   RAJSHREE    31-OCT-84   20000    VADODARA
E05   LAXMI       14-FEB-83   18000    ANAND
E06   SHIVALI     05-SEP-84   20000
```

- To select data that do not fall in some particular range, NOT can be used with BETWEEN operator.

Example:

Input:

```
SELECT * FROM EMPLOYEE
WHERE SALARY NOT BETWEEN 15000 AND 20000;
```

Output:

```
EID   ENAME       BIRTHDATE   SALARY   CITY
----  ----------  ---------   ----------  ----------
E01   TULSI       26-JAN-82   12000    AHMEDABAD
E04   VAISHALI    23-MAR-85   25000    SURAT
```

Set Searching Operator:

IN Operator:

- Selects rows that contain any value given in a set.
- Can be used when there is a need to use multiple OR conditions.

Example:

Display employees coming from city ‘Ahmedabad’, ‘Vadodara’ or ‘Surat’.

Input:

```
SELECT * FROM EMPLOYEE
WHERE CITY IN ('AHMEDABAD','VADODARA','SURAT');
```

Output:

```
EID   ENAME       BIRTHDATE   SALARY   CITY
----  ----------  ---------   ----------  ----------
E01   TULSI       26-JAN-82   12000    AHMEDABAD
E03   RAJSHREE    31-OCT-84   20000    VADODARA
E04   VAISHALI    23-MAR-85   25000    SURAT
```

- To select data that do not belong to some particular set, NOT can be used with IN operator.

Example:

Display employees not coming from city ‘Ahmedabad’, ‘Vadodara’ or ‘Surat’.

Input:

```
SELECT * FROM EMPLOYEE
WHERE CITY NOT IN ('AHMEDABAD','VADODARA','SURAT');
```

Output:

```
EID    ENAME        BIRTHDATE    SALARY    CITY
----   ----------   ---------    ----------  ----------
E02    GOPI         15-AUG-83    15000     ANAND
E05    LAXMI        14-FEB-83    18000     ANAND
```

Character Operators:

LIKE Operator:

- Select rows that contain values similar to a given pattern.
- Can be used with character data type.
- Pattern can be formed by using two wild cards characters:
    i. % (Modulo) allows matching with any string having any number of characters, including zero.
    ii. _ (Underscore) allows matching with any single character.

Example:

Display employees whose name have ‘a’ as a second character.

Input:

```
SELECT * FROM EMPLOYEE
WHERE ENAME LIKE '_A%';
```

Output:

```
EID    ENAME        BIRTHDATE    SALARY    CITY
----   ----------   ---------    ----------  ----------
E03    RAJSHREE     31-OCT-84    20000     VADODARA
E04    VAISHALI     23-MAR-85    25000     SURAT
E05    LAXMI        14-FEB-83    18000     ANAND
```

❖ **Write sort note on (with example): Aggregate Functions, Numeric Functions**

Answer:

Aggregate Functions:

1. MAX (columnName)

- Returns maximum value for a given column.

Input:

```
SELECT MAX(SALARY) "MAX SALARY" FROM EMPLOYEE;
```

Output:

```
MAX SALARY
----------
```

```
    25000
```

2. MIN (columnName)

- Returns minimum value for given column.

Input:

```
SELECT MIN(SALARY) "MIN SALARY" FROM EMPLOYEE;
```

Output:

```
MIN SALARY
----------
    12000
```

3. SUM ([distinct / all] columnName)

- Returns sum of all values for a given column.

Example:

```
SELECT SUM(SALARY), SUM(DISTINCT SALARY) FROM EMPLOYEE;
```

Output:

```
SUM(SALARY)      SUM(DISTINCTSALARY)
-----------      -------------------
    110000       90000
```

4. AVG ([distinct / all] columnName)

- Returns average of all values for a given column.

Example:

Input:

```
SELECT AVG(SALARY), AVG(DISTINCT SALARY) FROM EMPLOYEE;
```

Output:

```
AVG(SALARY)      AVG(DISTINCTSALARY)
-----------      -------------------
 18333.3333      18000
```

5. COUNT (*)

- Returns number of rows in a table including duplicates and having null values.

Example:

Input:

```
SELECT COUNT(*) FROM EMPLOYEE;
```

Output:

```
COUNT(*)
----------
       6
```

6. COUNT ([distinct / all] columnName)

- Returns number of rows where column does not contain null value.

Example:

Input:

```
SELECT COUNT(*) "C*", COUNT (CITY) "CC", COUNT(DISTINCT CITY) "CDC"
FROM EMPLOYEE;
```

Output:

```
      C*         CC        CDC
---------- ---------- ----------
       6          5          4
```

Numeric Function:

1. **ABS (n)**

- Returns absolute value of 'n' This means, negative numbers are converted to positive numbers.

Example:

Input:

```
SELECT ABS(-25), ABS(25) FROM DUAL;
```

Output:

```
ABS(-25)    ABS(25)
----------  ----------
    25          25
```

2. **SQRT (n)**

- Returns square root of 'n'. Here, 'n' cannot be a negative number.

Example:

Input:

```
SELECT SQRT(25), SQRT(35) FROM DUAL;
```

Output:

```
SQRT(25)    SQRT(35)
----------  ----------
    5       5.91607978
```

3. **POWER (m, n)**

- Returns m raised to n^{th} power, i.e, returns m^n
- Also, n must be an integer value.

Example:

Input:

```
SELECT POWER(3,2), POWER(4,3) FROM DUAL;
```

Output:

```
POWER(3,2)  POWER(4,3)
----------  ----------
    9           64
```

4. **MOD (m, n)**

- Returns remainder of a m divided by n operation.

Example:

Input:

```
SELECT MOD(5,2), MOD(5,3) FROM DUAL;
```

Output:

```
MOD(5,2)    MOD(5,3)
----------  ----------
    1           2
```

5. **CEIL (n)**

- Returns the largest integer value that is greater than or equal to n.

Example:

Input:

```
SELECT CEIL(25.2), CEIL(25.7), CEIL(-25.2) FROM DUAL;
```

Output:

```
CEIL(25.2)    CEIL(25.7)    CEIL(-25.2)
----------    ----------    -----------
    26            26            -25
```

6. FLOOR (n)

- Returns the smallest integer value that is less than or equal to n.

Example:

Input:

```
SELECT FLOOR(25.2), FLOOR(25.7), FLOOR(-25.2) FROM DUAL;
```

Output:

```
FLOOR(25.2)        FLOOR(25.7)        FLOOR(-25.2)
-----------        -----------        ------------
    25                 25                 -26
```

7. ROUND (m, n)

- Returns m, rounded to n places to the right of a decimal point.
- If n is omitted, m is rounded to 0 places.
- If n is negative, m is rounded to n places to the left of a decimal point.
- n must be an integer value.

Example:

Input:

```
SELECT ROUND(157.732,2), ROUND(157.732), ROUND(157.732,-2) FROM DUAL;
```

Output:

```
ROUND(157.732,2)  ROUND(157.732)    ROUND(157.732,-2)
----------------  --------------    -----------------
      157.73                 158                200
```

8. TRUNC (m, n)

- If n is positive, m is truncated to n places to the right of a decimal point.
- If n is omitted, m is truncated to 0 places.
- If n is negative, m is truncated to n places to the left of a decimal point.

Example:

Input:

```
SELECT TRUNC(157.732,2), TRUNC(157.732), TRUNC(157.732,-2) FROM DUAL;
```

Output:

```
TRUNC(157.732,2)  TRUNC(157.732)    TRUNC(157.732,-2)
----------------  --------------    -----------------
      157.73           157               100
```

9. EXP (n)

- Returns e raised to nth power, i.e., returns e, where e = 2.071828183

Example:

Input:

```
SELECT EXP(2), EXP(3) FROM DUAL;
```

Output:

```
EXP(2)        EXP(3)
----------    ----------
 7.3890561    20.0855369
```

10. LN (n)

- Returns natural, or base e, logarithm of n.

Example:

Input:

```
SELECT LN(10), LN(2) FROM DUAL;
```

Output:

```
LN(10)        LN(2)
----------    ----------
2.30258509    0.693147181
```

11. LOG (b, n)

- Returns logarithm of the n in the base of b, or return Log $_b$ n.

Example:

Input:

```
SELECT LOG(10,5), LOG(10,100) FROM DUAL;
```

Output:

```
 LOG(10,5)           LOG(10,100)
----------           -----------
0.698970004                   2
```

12. COS (n), SIN (n), TAN (n)

- Returns trigonometric cosine, sine and tangent values for n; where n is an angle in radius.

Example:

Input:

```
SELECT COS(3.1415), SIN(3.1415), TAN(3.1415) FROM DUAL;
```

Output:

```
COS(3.1415)         SIN(3.1415)         TAN(3.1415)
-----------         -----------         -----------
        -1  .       0. 000092654        -0.00009265
```

13. COSH (n), SINH (n), TANH (n)

- Return hyperbolic cosine, sine and tangent values for n; where n is in radius.

Example:

Input:

```
SELECT COSH(3.1415), SINH(3.1415), TANH(3.1415) FROM DUAL;
```

Output:

```
COSH(3.1415)        SINH(3.1415)        TANH(3.1415)
------------        ------------        ------------
  11.5908833        11.5476654          0.996271387
```

14. SIGN (n)

- Returns -1, if n is negative; 0 if n is zero; and 1 if n is positive.

Example:

Input:

```
SELECT SIGN(-25), SIGN(0), SIGN(25) FROM DUAL;
```

Output:

```
 SIGN(-25)    SIGN(0)       SIGN(25)
----------    ----------    ----------
        -1             0             1
```

❖ **Write short note on (with example): Character Functions, Date Functions**

Answer:

Character Function:

1. **Length (str)**
   - Returns length, i.e. number of characters, of str.

Example:

Input:

```
SELECT LENGTH('INDIA') FROM DUAL;
```

Output:

```
LENGTH('INDIA')
---------------
          5
```

2. **Lower (str)**
   - Returns str with all letters in lower case.

Example:

Input:

```
SELECT LOWER('SACHIN Ramesh tendulkar') FROM DUAL;
```

Output:

```
LOWER('SACHINRAMESHTEND
----------------------
sachin ramesh tendulkar
```

3. **Upper (str)**
   - Returns str with all letters in upper case.

Example:

Input:

```
SELECT UPPER ('SACHIN Ramesh tendulkar') FROM DUAL;
```

Output:

```
UPPER('SACHINRAMESHTEND
----------------------
SACHIN RAMESH TENDULKAR
```

4. **Initcap (str)**
   - Returns str with the first letter of each word in upper case.

Example:

Input:

```
SELECT INITCAP ('SACHIN Ramesh tendulkar') FROM DUAL;
```

Output:

```
INITCAP('SACHINRAMESHTE
----------------------
Sachin Ramesh Tendulkar
```

5. **Substr (str, pos, length)**
   - Returns a portion of str, beginning at pos and going up to length characters.

Example:

Input:

```
SELECT SUBSTR ('SACHIN Ramesh tendulkar',8,6) FROM DUAL;
```

Output:

```
SUBSTR
------
Ramesh
```

6. **Lpad (str, n, str2)**
   - Returns str, left padded with str2 up to length n.

Example:

Input:

```
SELECT LPAD ('INDIA',10,'*') FROM DUAL;
```

Output:

```
LPAD('INDI
----------
*****INDIA
```

7. **Rpad (str, n, str2)**
   - Returns str, right padded with str2 up to length n.

Example:

Input:

```
SELECT RPAD ('INDIA',10,'*') FROM DUAL;
```

Output:

```
RPAD('INDI
----------
INDIA*****
```

8. **Ltrim (str, str2)**
   - Remove characters from the left of str. Characters will be removed up to the first character not in set.

Example:

Input:

```
SELECT LTRIM('Sumita','uSae') FROM DUAL;
```

Output:

```
LTRI
----
mita
```

9. **Rtrim (str, set)**
   - Remove characters from the right of str. Characters will be removed up to the first character not in set.

Example:

Input:

```
SELECT RTRIM('Sumita','tab') FROM DUAL;
```

Output:

```
RTRI
----
Sumi
```

10. **Translate (str, from_set, to_set)**
    - Characters of str that occur in from set are translated to the corresponding characters in the to_set.

Example:

Input:

```
SELECT TRANSLATE ('abc12efg3','1234','XYZW') FROM DUAL;
```

Output:

```
TRANSLATE
---------
abcXYefgZ
```

11. Replace (str, from_set, to_set)

- Replace is similar to translate, but it works on strings rather than set of characters.
- It replaces entire sub-string instead of individual characters as in translate.

Example:

Input:

```
SELECT REPLACE ('abc123efg','123','XYZ') FROM DUAL;
```

Output:

```
REPLACE('
---------
abcXYZefg
```

12. Ascii (char)

- Returns the ASCII code of a char.

Example:

Input:

```
SELECT ASCII ('a'), ASCII ('A') FROM DUAL;
```

Output:

```
ASCII('a')    ASCII('A')
----------    ----------
     97            65
```

Date Function:

1. ADD_MONTHS (date, n)

- Returns new date after adding n months in date specified by date.
- If n is a negative, then n month will be subtracted.

Example:

Input:

```
select add_months(SYSDATE, 3), add_months (SYSDATE, -3) from dual;
```

Output:

```
ADD_MONTH       ADD_MONTH
---------       ---------
23-OCT-19       23-APR-19
```

2. MONTHS_BETWEEN (date1, date2)

- Returns number of months between date1 and date2.
- Subtract date2 from date1 to find out the difference of months.

Example:

Input:

```
SELECT MONTHS_BETWEEN ('31-MAR-13', '31-DEC-12') FROM DUAL;
```

Output:

```
MONTHS_BETWEEN('31-MAR-13','31-DEC-12')
---------------------------------------
                                      3
```

3. LAST_DAY (date)

- Returns the last date of the month specified by date.

Example:

Input:

```
SELECT LAST_DAY ('14-FEB-13') FROM DUAL;
```

Output:

```
LAST_DAY(
---------
28-FEB-13
```

4. **NEXT_DAY (date, day)**

- Returns the date of next named week-day specified by day relative to date.

Example:

Input:

```
SELECT NEXT_DAY('31-JUL-13', 'SUNDAY') FROM DUAL;
```

Output:

```
NEXT_DAY(
---------
04-AUG-13
```

5. **ROUND (date, format)**

- Returns rounded date according to format.

Example:

Input:

```
SELECT ROUND (TO_DATE('31-DEC-12 03:30:45 PM', 'DD-MON-YY HH:MI:SS
PM')) FROM DUAL;
```

Output:

```
ROUND(TO_
---------
01-JAN-13
```

6. **TRUNC (date, format)**

- Returns truncated date according to format.

Example:

Input:

```
SELECT TRUNC (TO_DATE('31-DEC-12 03:30:45 PM', 'DD-MON-YY HH:MI:SS
PM')) FROM DUAL;
```

Output:

```
TRUNC(TO_
---------
31-DEC-12
```

7. **NEW_TIME (date, zone1, zone2)**

- Returns the date after converting it from time zone 1 to time zone 2.

Example:

Input:

```
SELECT NEW_TIME (TO_DATE('31-DEC-12 12:00:00 AM', 'DD-MON-YY
HH:MI:SS PM'), 'GMT', 'PST')  FROM DUAL;
```

Output:

```
NEW_TIME(
---------
30-DEC-12
```

❖ **Write sort note on (with example): Conversion Functions, Miscellaneous Functions**

Answer:

Conversion Function:

1. **Converting Character to Number:**

   TO_NUMBER (str):

   - Converts a value of a character data type, expressing a number, to NUMBER data type, i.e. converts CHAR or VARCHAR2 to NUMBER.
   - Returns equivalent numeric value to str.

Example:

Input:

```
SELECT TO_NUMBER ('1234.56') FROM DUAL;
```

Output:

```
TO_NUMBER('1234.56')
--------------------
            1234.56
```

2. **Converting Number to Character:**

   TO_CHAR (str):

   - Converts a numeric value to a character data type, using optional format.

Example:

Input:

```
SELECT TO_CHAR (123456,'09,99,999') FROM DUAL;
```

Output:

```
TO_CHAR(12
----------
 01,23,456
```

3. **Converting Date to Character:**

   TO_CHAR (date, format)

   - Converts a DATE value date to a CHAR value, using format.

Example:

Input:

```
SELECT TO_CHAR (SYSDATE, 'DD Month, YYYY') FROM DUAL;
```

Output:

```
TO_CHAR(SYSDATE,'DDMONTH,YYYY')
---------------------------------------------
23 July     , 2019
```

4. **Converting Character to Date:**

   TO_DATE (str, format)

   - Converts a character value, i.e. str, to a DATE value.

Example:

Input:

```
SELECT TO_DATE ('31 December, 2012', 'DD-MON-YY') FROM DUAL;
```

Output:

```
TO_DATE('
---------
31-DEC-12
```

Miscellaneous Function:

1. UID

- Returns an integer value corresponding to the UserID of the user currently logged in.

Example:

Input:

```
SELECT UID FROM DUAL;
```

Output:

```
       UID
----------
         5
```

2. USER

- Returns the user name of the user currently logged in.

Example:

Input:

```
SELECT USER FROM DUAL;
```

Output:

```
USER
------------------------------
SYSTEM
```

3. GREATEST (exp1, exp2, …, expN)

- Returns the greatest expression.
- Can be used with numeric, character as well as date related data.

Example:

Input:

```
SELECT GREATEST (11,32,7), GREATEST ('11','32','7') FROM DUAL;
```

Output:

```
GREATEST(11,32,7) G
----------------- -
               32 7
```

4. LEAST (exp1, exp2, …, expN)

- Similar to GREATEST function, but returns the least expression.

Example:

Input:

```
SELECT LEAST (11,32,7), LEAST ('11','32','7') FROM DUAL;
```

Output:

```
LEAST(11,32,7) LE
-------------- --
            7 11
```

5. **DECODE (value, if1, then1, if2, then2, …,else)**
   - DECODE is similar to if-then-else construct in programming languages.
   - A value is the value to be tested. It is compared with if part, and whenever match is found, corresponding then part is returned. If match not found with any of the if, then else will be returned.

Example:

Input:

```
SELECT DECODE ('SATUR', 'SUN', 'HOLIDAY', 'SATUR', 'HALFDAY', 'FULL
DAY') FROM DUAL;
```

Output:

```
DECODE(
-------
HALFDAY
```

6. **NVL (exp1, exp2)**
   - Returns exp2, if exp1 is null. Returns exp1, if it is not null.

Example:

Input:

```
SELECT NVL (5,10), NVL(null,10) FROM DUAL;
```

Output:

```
 NVL(5,10)   NVL(NULL,10)
----------   ------------
        5             10
```

7. **VSIZE (exp)**
   - Returns the storage size of exp in Oracle.

Example:

Input:

```
SELECT VSIZE(25), VSIZE('INDIA'), VSIZE(TO_DATE('31-DEC-12')) FROM
DUAL;
```

Output:

```
 VSIZE(25)   VSIZE('INDIA')      VSIZE(TO_DATE('31-DEC-12'))
----------   ----------------    -----------------------------------
2                   5                                          7
```

❖ **What are constraints? Give classification of Constraints.**

Answer:

"A constraint is a rule that restricts the values that may be present in the database."

- The data stored in database should be valid, correct and consistent. So, when data is manipulated, some rules (constraints) should be followed.
- Oracle provides various kinds of constraints to ensure validity, correctness and consistency of data in a database.
- These constraints can be broadly classified into two categories.
  1. Entity Integrity Constraint
  2. Referential Integrity Constraint

1. **Entity Integrity Constraint**

"Entity integrity constraints are constraints that restrict the values in row of an individual table." These constraints are further divided into two sub-categories:

1. **Domain Integrity Constraint**
2. **Key Constraint**

Domain Integrity Constraint:

- Specifies that the value of each column must belong to the domain of that column.
- For example, a balance for any account in banking system should not be negative value.
- Oracle provides two constraints – NOT NULL and CHECK – to provide domain integrity.

Key Constraint:

- Ensure that there must be some way to distinguish two different rows of the same table uniquely.
- Oracle provides two constraints – UNIQUE and PRIMARY KEY – to ensure this kind of uniqueness between two different rows.

2. **Referential Integrity Constraint**

"Referential integrity constraint specifies that a row in one table that refers to another row must refer to an existing row in that table."

- Oracle provides FOREIGN KEY constraint to ensure such kind of consistency.

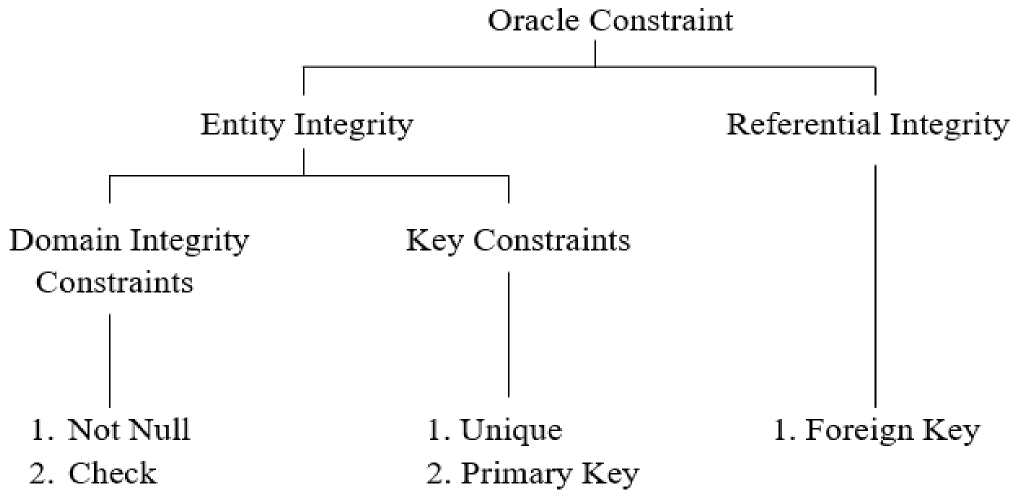


Figure: Classification of Oracle Constraints

❖ **Explain concept of Naming a constraint with example.**

Answer:

- It is also possible to assign some user defined name to a constraint.

Example:

Create an Account table having *ano* as a primary key and assign name 'prime_key' to this constraint.

Input:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3)  CONSTRAINT PRIME_KEY PRIMARY KEY,
BALANCE NUMBER (9),
BNAME CHAR(10)
);
```

Output:

```
Table created.
```

Example:

Create an Account table, having constraint named ‘for_key’ defining *bname* as a foreign key, referring to Branch table, defined at table level.

Input:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3) PRIMARY KEY,
BALANCE NUMBER (9),
BNAME CHAR(10),
constraint FOR_KEY FOREIGN KEY (BNAME) REFERENCES BRANCH (BNAME)
);
```

Output;

```
Table created.
```

❖ **Explain how to alter schema in case of constraints.**

Answer:

- ALTER TABLE can also be used to add, modify or drop constraint in an existing table.

Add Constraint:

- This command adds a new constraint in an existing table.

Example:

Consider that Account table doesn’t have any primary key. Alter this table by adding primary key on column *ano.*

Input:

```
ALTER TABLE ACCOUNT
ADD PRIMARY KEY (ANO);
```

Output:

```
Table altered.
```

Drop Constraint:

- A constraint can be dropped directly, or can be dropped by using pre-assigned name to the constraint.

Example:

Drop primary key from an Account table.

Input:

```
ALTER TABLE ACCOUNT
DROP PRIMARY KEY;
```

Output:

```
Table altered.
```

❖ **Explain Domain Integrity Constraints with example.**

Answer:

- Specifies that the value of each column must belong to the domain of that column.
- Oracle provides two constraints – NOT NULL and CHECK – to provide domain integrity.

NOT NULL Constraint:

- There may be records in table that do not contain any value for some fields.
- In other words, a NULL value represents an empty field.

- A NULL value indicates ‘not applicable’, ‘missing’, or ‘not known’.
- A NULL value is distinct from zero or other numeric value for numerical data.
- A NULL value is also distinct from a blank space for character data.
- But, sometimes it is required that a field cannot be left empty. For example, a balance column in Account table must have some value.
- A column, defined as a NOT NULL, cannot have a NULL value. In other words, such column becomes a mandatory column and cannot be left empty for any record.

Example:
Create an Account table having column balance as a mandatory column.
Input:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3),
BALANCE NUMBER (9) NOT NULL,
BNAME CHAR(10)
);
```

Output:

```
Table created.
```

Input:

```
INSERT INTO ACCOUNT
VALUES ('A01',NULL,'ANAND');
```

Output:

```
ERROR at line 2:
ORA-01400: cannot insert NULL into ("SYSTEM"."ACCOUNT"."BALANCE")
```

CHECK Constraint:

- The CHECK constraint is used to implement business rule. This constraint is also referred as business rule constraint.
- The CHECK constraint is bound to a particular column.
- Once a CHECK constraint is implemented, any insert or update operation on that table must follow this constraint, i.e. condition provided by this constraint.

Example:
Create an Account table which doesn’t allow negative values as a balance for any account.
Input:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3),
BALANCE NUMBER (9) CHECK (NOT (BALANCE < 0)),
BNAME CHAR(10)
);
```

Output:

```
Table created.
```

Input:

```
INSERT INTO ACCOUNT VALUES ('A01',-5000,'VVN');
```

Output:

```
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.SYS_C006994) violated
```

Example:
Create an Account table which accepts ‘vvn’,’ksad’ or ‘anand’ only as a branch name.
Input:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3),
BALANCE NUMBER (9),
BNAME CHAR(10),
CHECK ( BNAME IN ('VVN','KSAD','ANAND'))
);
```

Output:

```
Table created.
```

Input:

```
INSERT INTO ACCOUNT VALUES ('A01',5000,'AHMEDABAD');
```

Output:

```
ERROR at line 1:
ORA-02290: check constraint (SYSTEM.SYS_C006995) violated
```

❖ **Explain Key Constraints with example.**

Answer:

- Ensure that there must be some way to distinguish two different rows of the same table uniquely.
- Oracle provides two constraints – UNIQUE and PRIMARY KEY – to ensure this kind of uniqueness between two different rows.

Unique:

- There may be requirement that a column must have unique values. In other words, it is required that a column cannot contain duplicate values.
- For example, in Account table, all account numbers must be unique. This is required to identify all accounts, i.e. records stored in table, uniquely.
- A column, defined as a UNIQUE, cannot have duplicate values across all records.

Example:

Create Account table having column *ano* as a UNIQUE column.

Input: (Column Level)

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3) UNIQUE,
BALANCE NUMBER (9),
BNAME CHAR(10)
);
```

Table Level:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3),
BALANCE NUMBER (9),
BNAME CHAR(10),
UNIQUE(ANO)
);
```

Output:

```
Table created.
```

Input:

```
INSERT INTO ACCOUNT
VALUES ('A04',8000,'KSAD')
```

Output:

```
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C006988) violated
```

Primary Key:

"A primary key is a set of one or more columns used to identify each record uniquely in a column."

- A column, defined as a PRIMARY KEY, cannot have duplicate values across all records.

- A column, defined as a PRIMARY KEY, cannot have NULL value.
- Thus, a PRIMARY KEY constraint is a combination of NOT NULL and UNIQUE constraints.

Example:

Create Account table having column *ano* as a PRIMARY KEY column.

Input:(Column Level)

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3) PRIMARY KEY,
BALANCE NUMBER (9),
BNAME CHAR(10)
);
```

Table Level:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3),
BALANCE NUMBER (9),
BNAME CHAR(10),
PRIMARY KEY(ANO)
);
```

Output:

```
Table created.
```

Input:

```
INSERT INTO ACCOUNT VALUES ('A01',5000,'VVN');
```

Output:

```
ERROR at line 1:
ORA-00001: unique constraint (SYSTEM.SYS_C006990) violated
```

Input:

```
INSERT INTO ACCOUNT VALUES (NULL,5000,'VVN');
```

Output:

```
ERROR at line 1:
ORA-01400: cannot insert NULL into ("SYSTEM"."ACCOUNT"."ANO")
```

- ❖ **Explain Referential Integrity Constraints with example also explain use of On Delete Cascade.**

Answer:

- A FOREIGN KEY constraint, also referred as referential integrity constraint, is specified between two tables. This constraint is used to ensure consistency among records of the two tables.
- Consider the following two tables – Account and Branch – as shown in following.

Account:

Branch:

ano	**balance**	**Bname**
A01	5000	Vvn
A02	6000	Ksad
A03	7000	Anand
A04	8000	Ksad
A05	6000	vvn

bname	**Baddress**
vvn	Mota bazaar, VVNagar
ksad	Chhota bazaar, Karamsad
anand	Nana bazaar, Anand

- Here, bname is common between both tables and provides link between Account and Branch. It represents which account is related to which branch.

- A FOREIGN KEY constraint is used to maintain such kind of data consistency between two tables. To server this purpose, it uses concept of foreign key.
- "A foreign key is a set of one or more columns whose values are derived from the primary key or unique key of other table."
- In our case, an attribute *bname* in Account table is referred as foreign key as its values are derived from the branch table where it is defined as a primary key.
- The table, in which a foreign key is defined, is called a **foreign table**, **detail table** or **child table**. The table, of which primary key or unique key is referred, is called a **primary table**, **master table** or **parent table**.

Example:

Create an Account table in such a way that it contains column *bname* as a FOREIGN KEY referring to Branch table.

Input:

```
CREATE TABLE ACCOUNT(
ANO VARCHAR2(3) PRIMARY KEY,
BALANCE NUMBER (9),
BNAME CHAR(10) REFERENCES BRANCH(BNAME)
);
```

Output:

```
Table created.
```

- A table name in REFERENCES clause refers to the Master table.
- If *columnName* from REFERENCES clause is omitted, then the primary key is automatically referred of a table specified in this clause.

ON DELETE CASCADE Option:

- This option overcomes the restriction enforced on Master table for delete operation.
- With this option, when any record from Master table is deleted, all the corresponding records from Detail (Child) table are also deleted automatically.
- Consider that, this option has already been provided while defining a foreign key in Account table. Now, if record with branch name 'vvn' is deleted from Branch table, then all the records with 'vvn' as a branch name from Account table will also be deleted.

Example:

Create an Account table in such a way that it contains column *bname* as a FOREIGN KEY referring to Branch table defined at table level.

Input:

```
        CREATE TABLE ACCOUNT(
        ANO VARCHAR2(3) PRIMARY KEY,
        BALANCE NUMBER (9),
        BNAME CHAR(10),
        FOREIGN KEY (BNAME) REFERENCES BRANCH (BNAME)
        ON DELETE CASCADE
);
```

L.J. Polytechnic, Ahmedabad
Unit-4
Relational Model

❖ Basic terms of Relational Model

- **Domain**: It contains a set of atomic values that an attribute can take.

- **Attribute**: It contains the name of a column in a particular table. Each attribute Ai must have a domain.

- **Tuple:** A row or a record in a relation is referred as a Tuple.

- **Relational schema**: A Relational schema contains the name of the Relation and name of all columns or attributes.

- **Relational key**: In the Relational key, each row has one or more attributes. It can identify the row in the Relation uniquely.

- **Relational instance**: In the Relational database system, the relational instance is represented by a finite set of tuples. Relation instances do not have duplicate tuples.

- **Arity of a Relation:** The total numbers of attributes in relation is referred as Arity of Relation.
- For example, the arity of STUDENT relation is 4.

- **Cardinality of a Relation:** The total numbers of tuples in relation is referred as Cardinality of a Relation.
- For example, the cardinality of a STUDENT relation is 3.

- **Example**:
- STUDENT Relation

ENROLL_NO	NAME	CONTACT_NO	CITY
101	JIYA	7687864543	AHMEDABAD
102	KIYA	9897654512	ANAND
103	HIYA	8767675412	VVN

ENROLL_NO, NAME, CONTACT_NO, ADDRESS are attributes.
The total no. of attributes in a relation i.e arity of relation.
A row in relation is referred as tuple.
The total numbers of tuple in a relation are cardinality of a relation.
A domain of an attribute is a set of a permitted values for that attribute.

❖ Keys in Relational Model

- Keys in DBMS is an attribute or set of attributes which helps you to identify a row(tuple) in a relation(table). They allow you to find the relation between two tables.
- Keys help you uniquely identify a row in a table by a combination of one or more columns in that table. Key is also helpful for finding unique record or row from the table. Database key is also helpful for finding unique record or row from the table.

- **Super Key:**
- "A super key is a set of one or more attributes that allows to identify each tuple uniquely in a relation."
- For example, consider a relation Customer with attributes cid, cname, address and contact_no. In this relation, each tuple represents an individual customer. Here, the cid attribute can distinguish each tuple from another. So, cid is a super key for Customer relation.

- **Relation Key:**
- "A super key for which no subset is a super key is called a relation key."
- In other words, relation key is a minimal super key
- A relation key is sufficient to identify each and every tuple uniquely within a relation.
- A relation key cannot have null or duplicate values, as well as it does not contain any redundant attribute.
- Sometimes, there can be more than one relation keys for the same relation. For example, for Customer relation, {cid} as well as {cnam address} are relation keys.

- **Candidate Key:**
- "If a relation contains more than one relation keys, then, they each are called candidate key."
- For example, for Customer relation, {cid} as well as {cname, address} are relation keys. And so, they each are called candidate keys.

- **Primary Key:**
- "A primary key is a candidate key that is chosen by the database designer to identify tuples uniquely in a relation."
- For example, if database designers choose candidate key cid to identify all customers uniquely, then, cid is a primary key of the relation Customer.

- **Foreign Key:**
- "A foreign key is a set of one or more attributes whose values are derived from the key attributes of another relation."
- For example, consider the Account and Branch relations. Here, values for attribute *bname* of Account relation are derived from the Branch relation. So, an attribute *bname* in Account table is referred as foreign Key.

- **Alternate Key:**
- "An alternate key is a candidate key that is not chosen by the database designer to identify tuples uniquely in a relation."
- For example, if database designers choose candidate key cid to identify all customers uniquely, then, another candidate key {cname, address} is an alternate key.

❖ **Sub queries**

- A sub query is an SQL statement which appears inside another SQL statement.
- Here, a SELECT statement appears as a part of WHERE clause of some other SQL statement, and so, it is referred as a sub query.
- The statement containing a sub query is called a **parent** or **outer** query.

- The result of the parent query depends upon the result of the sub query.
- A sub query is also referred as a **nested query**.
- Consider following tables.

Customer:

cid	ano
C01	A01
C02	A02
C03	A03
C04	A04
C05	A05
C02	A04

Account_Holder:

ano	balance	bname
A01	5000	vvn
A02	6000	ksad
A03	7000	anand
A04	8000	ksad
A05	6000	vvn

Account:

cid	name
C01	Riya
C02	Jiya
C03	Diya
C04	Taral
C05	Saral

Example:

Find out the balance of an account which belongs to customer ‘C01’.

Input:

```
SELECT BALANCE FROM ACCOUNT
WHERE ANO IN(
SELECT ANO FROM ACCOUNT_HOLDER
WHERE CID='C01'
);
```

Output:

```
BALANCE
----------
    5000
```

Example:

Find out the balance of accounts which belongs to customer ‘Jiya’.

Input:

```
SELECT BALANCE FROM ACCOUNT
WHERE ANO IN(
SELECT ANO FROM ACCOUNT_HOLDER
WHERE CID IN(
SELECT CID FROM CUSTOMER
WHERE NAME='JIYA'
)
);
```

Output:

```
BALANCE
----------
    6000
    8000
```

Correlated Subquery:

- A sub query, which refers a column from a table in the parent query, is called a correlated sub query.
- A correlated sub query is evaluated once for each row processed by the parent statement.

Example:

Find out the accounts having maximum balance in branch to which it belongs. Display account number, balance and branch name for such accounts.

Input:

```
SELECT ANO, BALANCE, BNAME FROM ACCOUNT ACC
WHERE BALANCE IN (
SELECT MAX(BALANCE) FROM ACCOUNT
WHERE BNAME=ACC.BNAME
);
```

Output:

```
ANO   BALANCE    BNAME
---   ----------  ----------
A03    7000       ANAND
A04    8000       KSAD
A05    6000       VVN
```

❖ Relational Algebra

Relational algebra is a procedural query language, which takes instances of relations as input and yields instances of relations as output. It uses operators to perform queries. An operator can be either **unary** or **binary**. They accept relations as their input and yield relations as their output. Relational algebra is performed recursively on a relation and intermediate results are also considered relations.

The fundamental operations of relational algebra are as follows −

- **Select**
- **Project**
- **Union**
- **Set different**
- **Cartesian product**
- **Rename**

- **Select Operation (σ)**

It selects tuples that satisfy the given predicate from a relation.

Notation − $\sigma_p(r)$

Where **σ** stands for selection predicate and **r** stands for relation. *p* is prepositional logic formula which may use connectors like **and, or,** and **not**. These terms may use relational operators like − =, ≠, ≥, < , >, ≤.

For example −

$\sigma_{subject = "database"}(Books)$

Selects tuples from books where subject is 'database'.

$\sigma_{\text{subject = "database" and price = "450"}}(\text{Books})$

Selects tuples from books where subject is 'database' and 'price' is 450.

$\sigma_{\text{subject = "database" and price = "450" or year < "2022"}}(\text{Books})$

Selects tuples from books where subject is 'database' and 'price' is 450 or those books published before 2022.

- **Project Operation (∏)**

It projects column(s) that satisfy a given predicate.

Notation − $\prod_{A1}$, A_2, A_n (r)

Where A_1, A_2 , A_n are attribute names of relation **r**.

Duplicate rows are automatically eliminated, as relation is a set.

For example −

$\prod_{subject, author}$ (Books)

Selects and projects columns named as subject and author from the relation Books.

- **Union Operation (∪)**

It performs binary union between two given relations and is defined as −

r ∪ s = { t | t ∈ r or t ∈ s}

Notation − r U s

Where **r** and **s** are either database relations or relation result set (temporary relation).

For a union operation to be valid, the following conditions must hold −

- **r**, and **s** must have the same number of attributes.
- Attribute domains must be compatible.
- Duplicate tuples are automatically eliminated.

$\prod_{author}$ (Books) ∪ $\prod_{author}$ (Articles)

Projects the names of the authors who have either written a book or an article or both.

- **Set Difference (−)**

The result of set difference operation is tuples, which are present in one relation but are not in the second relation.

Notation − **r** − **s**

Finds all the tuples that are present in **r** but not in **s**.

$\prod_{author}$ (Books) − $\prod_{author}$ (Articles)

Provides the name of authors who have written books but not articles.

- **Cartesian Product (X)**

Combines information of two different relations into one.

Notation − r X s

Where **r** and **s** are relations and their output will be defined as −

r X s = { q t | q ∈ r and t ∈ s}

$\sigma_{author = 'tutorialspoint'}$(Books X Articles)

Yields a relation, which shows all the books and articles written by tutorialspoint.

- **Rename Operation (ρ)**

The results of relational algebra are also relations but without any name. The rename operation allows us to rename the output relation. 'rename' operation is denoted with small Greek letter **rho** ρ.

Notation − ρ_{x} (E)

Where the result of expression **E** is saved with name of **x**.

❖ Joins

A SQL Join statement is used to combine data or rows from two or more tables based on a common field between them. Different types of Joins are:

- Cross Join
- Inner Join
- Outer Join
- Self Join

❖ Cross Join (Cartesian Product, or Simple Join)

- The cross join combines every row from one table with every row in other table.
- The output of cross join is similar to Cartesian product, and so, this operation is also referred as Cartesian product.

Example:

Combine information from Account and Branch table.

Input:

```
SELECT * FROM ACCOUNT,BRANCH;
```

Output:

```
ANO   BALANCE    BNAME      BNAME      BADDRESS
---  ----------  ---------- -------    ------------------------------
A01    5000      VVN        VVN        Mota Bazar, VVNagar
A02    6000      KSAD       VVN        Mota Bazar, VVNagar
A03    7000      ANAND      VVN        Mota Bazar, VVNagar
A04    8000      KSAD       VVN        Mota Bazar, VVNagar
A05    6000      VVN        VVN        Mota Bazar, VVNagar
A01    5000      VVN        KSAD       Chhota Bazar, Karamsad
A02    6000      KSAD       KSAD       Chhota Bazar, Karamsad
A03    7000      ANAND      KSAD       Chhota Bazar, Karamsad
A04    8000      KSAD       KSAD       Chhota Bazar, Karamsad
A05    6000      VVN        KSAD       Chhota Bazar, Karamsad
A01    5000      VVN        ANAND      Nana Bazar, Anand
A02    6000      KSAD       ANAND      Nana Bazar, Anand
A03    7000      ANAND      ANAND      Nana Bazar, Anand
A04    8000      KSAD       ANAND      Nana Bazar, Anand
A05    6000      VVN        ANAND      Nana Bazar, Anand
```

❖ Inner Join (Equi and Non-equi Joins)

- The inner join combines only those records which contain common values in both tables.
- Thus, in contrast to cross join, inner join produces only consistent records in output.

Example:

Combine only consistent information from Account and Branch table.

Input:

```
select ANO, BALANCE, ACCOUNT.BNAME, BADDRESS from ACCOUNT,
BRANCH
WHERE ACCOUNT.BNAME = BRANCH.BNAME;
```

Output:

```
ANO   BALANCE    BNAME      BADDRESS
---   ----------  ----------  ------------------------------
A01    5000      VVN        Mota Bazar, VVNagar
A02    6000      KSAD       Chhota Bazar, Karamsad
A03    7000      ANAND      Nana Bazar, Anand
A04    8000      KSAD       Chhota Bazar, Karamsad
A05    6000      VVN        Mota Bazar, VVNagar
```

❖ Self Join:

- Sometimes, it is necessary to join a table with itself, as like, joining two separate tables. Such kind of join is called a self-join.
- Self-join is similar to inner-join, except that, a table is joined with itself. In self-join, each row of a table is combined with other rows of the same table to produce result.

Example:

Display employee id and name along with name of their manager.

Input:

```
SELECT EMP.EID, EMP.NAME "EMPNAME", MNGR.NAME "MNGRNAME"
FROM EMPLOYEE EMP, EMPLOYEE MNGR
WHERE EMP.MNGR_ID = MNGR.EID;
```

Output:

```
EID   EMPNAME    MNGRNAM
---   -------    -------
E05   SARAL      ZALAK
E04   TARAL      ZALAK
E03   FALAK      ZALAK
E01   PALAK      ZALAK
```

❖ Outer Join:

- The outer join operation is an extension of the inner join operation.

Example:

Combine the academic and hostel related information for all students.

Input:

```
SELECT ID, COLLEGE.NAME "NAME", DEPARTMENT, HOSTEL_NAME,
ROOM_NO FROM COLLEGE, HOSTEL
WHERE COLLEGE.NAME = HOSTEL.NAME;
```

Output:

```
ID  NAME      DEPARTMENT       HOSTEL_NAME        ROOM
--- --------  ----------       ------------------ ---
S02 Anisha    Computer         Kaveri Hostal      K01
S03 Nisha     I.T.             Godavari Hostal    G07
```

The outer join operation can be divided into three different forms:

1. Left outer join
2. Right outer join
3. Full outer join

1. Left Outer Join

- The left outer join retains all the records of the left table even though there is no matching record in the right table.

Example:

Input:

```
SELECT * FROM COLLEGE, HOSTEL
WHERE COLLEGE.NAME = HOSTEL.NAME(+);
```

Output:

```
NAME     ID  DEPARTMENT NAME     HOSTEL_NAME        ROOM
-------- --- ---------- -------  ------------------ ---
Anisha   S02 Computer   Anisha   Kaveri Hostal      K01
Nisha    S03 I.T.       Nisha    Godavari Hostal    G07
Manisha  S01 Computer
```

2. Right Outer Join

- The right outer join retains all the records of the right table even though there is no matching record in the left table.

Example:

Input:

```
SELECT * FROM COLLEGE, HOSTEL
WHERE COLLEGE.NAME(+) = HOSTEL.NAME;
```

Output:

```
NAME     ID  DEPARTMENT NAME     HOSTEL_NAME          ROOM
-------- --- ---------- -------  -------------------- ---
Anisha   S02 Computer   Anisha   Kaveri Hostal        K01
Nisha    S03 I.T.       Nisha    Godavari Hostal      G07
                        Isha     Kaveri Hostal        K02
```

3. Full Outer Join

- The full outer join retains all the records of both of the tables. It also pads null values whenever required.

Example:

Input:

```
SELECT * FROM COLLEGE, HOSTEL
WHERE COLLEGE.NAME = HOSTEL.NAME(+)
UNION
SELECT * FROM COLLEGE, HOSTEL
WHERE COLLEGE.NAME(+) = HOSTEL.NAME;
```

Output:

```
NAME      ID   DEPARTMENT NAME      HOSTEL_NAME           ROOM
--------  ---  ----------  -------  --------------------  ---
Anisha    S02  Computer    Anisha   Kaveri Hostal         K01
Manisha   S01  Computer
Nisha     S03  I.T.        Nisha    Godavari Hostal       G07
                           Isha     Kaveri Hostal         K02
```

❖ **Set operations**

- Consider following two tables Customer and Employee.

Customer:

cid	name
C01	Riya
C02	Jiya
C03	Diya
C04	Taral
C05	Saral

Employee:

eid	name	mngr_id
E01	Palak	E02
E02	Zalak	null
E03	Falak	E02
E04	Taral	E02
E05	Saral	E02

Union:

Query1 Union Query2

- The UNION clause merges the output of query1 and query2.
- The output will be as a single set of rows and columns.
- Output will contain all the records from query and query 2, but the common records will appear only once.

Example:

List out name of persons who are either customer or employee.

Input:

```
SELECT name FROM Customer
UNION
SELECT name FROM Employee;
```

Output:

```
Name
--------
Diya
Falak
Jiya
Palak
Riya
Saral
Taral
Zalak
```

Intersect:

query1 INTERSECT query2

- The INTERSECT clause combines the output of query1 and query2.
- Output will contain those records which are common to query1 and query2.

Example:

List out name of persons who are customer as well as employee.

Input:

```
SELECT name FROM Customer
INTERSECT
SELECT name FROM Employee;
```

Output:

```
NAME
------------
Saral
Taral
```

Minus:

query1 MINUS query2

- The MINUS clause combines the output of query1 and query2.
- Output will contain those records which are present in query1 but not present in query2.

Example:

List out name of persons who are customers but not employee.

Input:

```
SELECT name FROM Customer
MINUS
SELECT name FROM Employee;
```

Output:

```
NAME
-----------
Riya
Jiya
Diya
```

L.J. Polytechnic, Ahmedabad
Unit-5
Entity Relationship Model

Define the following terms.

Entity

- An entity is a thing or object or person in the real world that is distinguisable from all other object.
- E.g. book, student, employee, college etc…

Entity sets

- An entity set is a set of entities of same type that share the same properties or attributes.
- E.g. the set of all students in a college can be defined as entity set student.

Relationship

- Relationship is an association (connection) between several entities.
- Relationship between two entities is called binary relationship.
- E.g. book is issued by student where book and student are entities and issue is relation.

Relationship set

- Relationship set is a set of relationships of the same type.

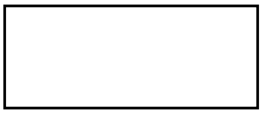
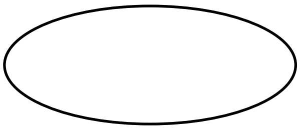
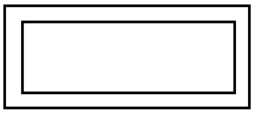
Attributes

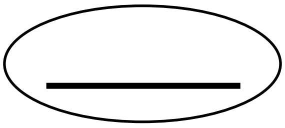
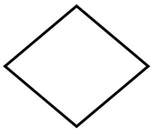
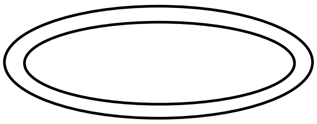
- Attributes are properties hold by each member of an entity set.
- E.g. entity is student and attributes of students are enrollmentno, name, address, cpi etc …

Explain Types of attributes.

- **Simple attribute:** It cannot be divided into subparts. E.g. cpi, rollno
  Composite attribute: It can be divided into subparts. E.g. address, name
- **Single valued attribute:** It has single data value. E.g. enrollmentno, birthdate
  Multi valued attribute: It has multiple data value. E.g. contactno (may have multiple phones)
- **Stored attribute:** It’s value is stored manually in database. E.g. birthdate
  Derived attribute: It’s value is derived or calculated from other attributes. E.g. age (can be calculated using current date and birthdate).

Draw various symbols using in E-R diagram.

Symbols	Represents as
	Entity
	Attribute
	Weak Entity

	Key Attribute
	Relationship
	Multi-Valued Attribute

Types of Attributes

- **Single Attribute**
  Single valued attributes have single data value for a particular entity.
- For example: SSN or Gender.

- **Multi-valued Attribute**
  Multi-valued attributes have multiple data values for a particular entity.
- For example: Contact_No.

- **Composite Attribute**
  Composite attributes can be divided into smaller sub-parts.
- For example: Address (Apt#, House#, Street, City, State, ZipCode, Country), or Name (FirstName: MiddleName, LastName).

- **Stored Attribute**
  Attribute that cannot be derived from other attributes are called as stored attributes.
- For example: Date of Birth.

- **Derived Attribute**
  The value for derived attribute can be derived from the values of the other related attributes.
- For example: Age.

- **Key Attribute**
  The value of attribute that can be uniquely identify i.e known as Key Attribute.
- For example: License no. of vehicle.

Types of Entity

- **Concrete Entity:**
  A concrete entity is a physically existing entity, such as a customer, an employee, or a book.

- **Abstract Entity:**
  An abstract entity has no physical existence, such as an account, a loan, or a subject.

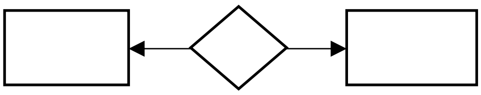
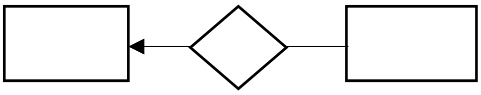
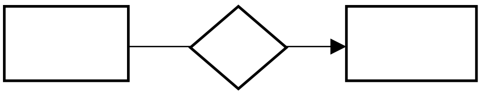
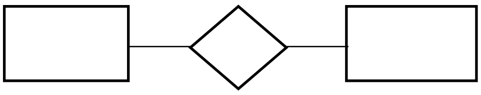
- **Strong Entity:**
  The strong entity has a primary key. Weak entities are dependent on strong entity. Its existence is not dependent on any other entity.

- **Weak Entity**

The weak entity in DBMS do not have a primary key and are dependent on the parent entity. It mainly depends on other entities.

Mapping Cardinality

- Defines the numerical attributes of the relationship between two entities or entity sets.
- Different types of cardinal relationships are:
- One-to-One Relationships
- One-to-Many Relationships
- Many-to-One Relationships
- Many-to-Many Relationships

Symbol	**Represents as**
	One to One
	One to Many
	Many to One
	Many to Many

Design E-R Model

Here are some best practice or example for Developing Effective E-R Diagrams.

1. Eliminate any redundant entities or relationships.

2. You need to make sure that all your entities and relationships are properly labelled.

3. There may be various valid approaches to an ER diagram. You need to make sure that the ER diagram supports all the data you need to store.

4. You should assure that each entity only appears a single time in the E-R Diagram.

5. Name every relationship, entity, and attribute are represented on your diagram.

6. Never connect relationships to each other.

7. You should use colours to highlight important portions of the E-R Diagram.

Problems with E-R Model

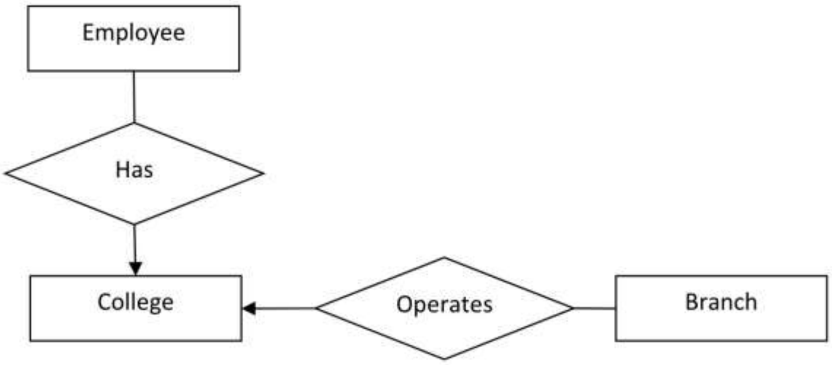
- The E-R model can result problems due to limitations in the way the entities are related in the relational databases. These problems are called connection traps. These problems often occur due to a misinterpretation of the meaning of certain relationships.

- Two main types of connection traps are called fan traps and chasm traps.

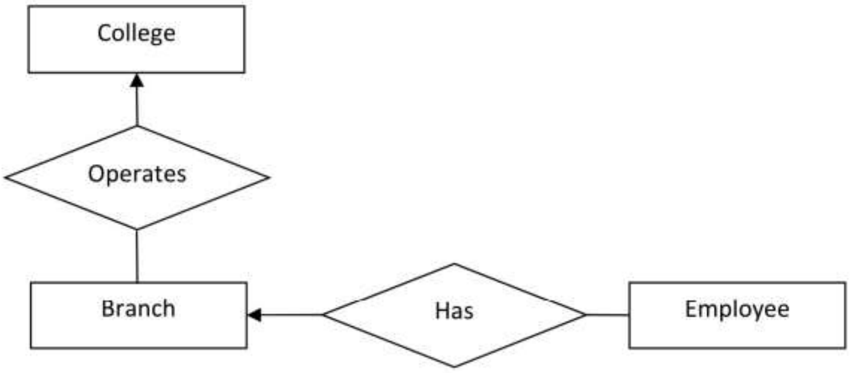
1. **Fan trap**
2. **Chasm trap**

1. **Fan trap**

- Fan trap occurs when two relationship sets, having mapping cardinality one to many, converge to single entity set.
- Problem:
- Suppose an college which operates various branches such as computer engineering, Electrical engineering, etc. Different kind of employees, such as lecturer,lab assistant, clerk etc. works in college. Also these employees belongs to specific branches.
- This diagram indicates that college has more than one employee and it operates more than one branch. The relationships sets has and operates are one to many types from college to employees and from college to branch. These both relationship sets coverage to single entity set College.



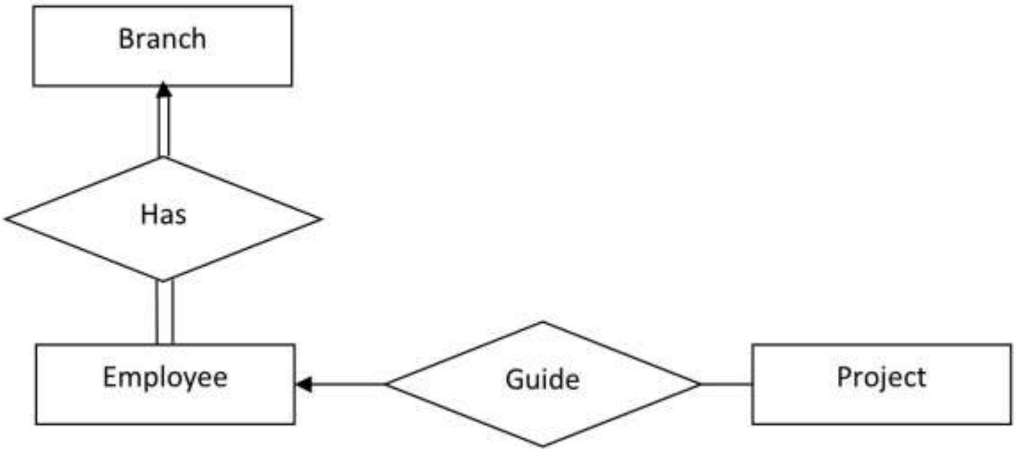
Solution: Restructured E-R Model.



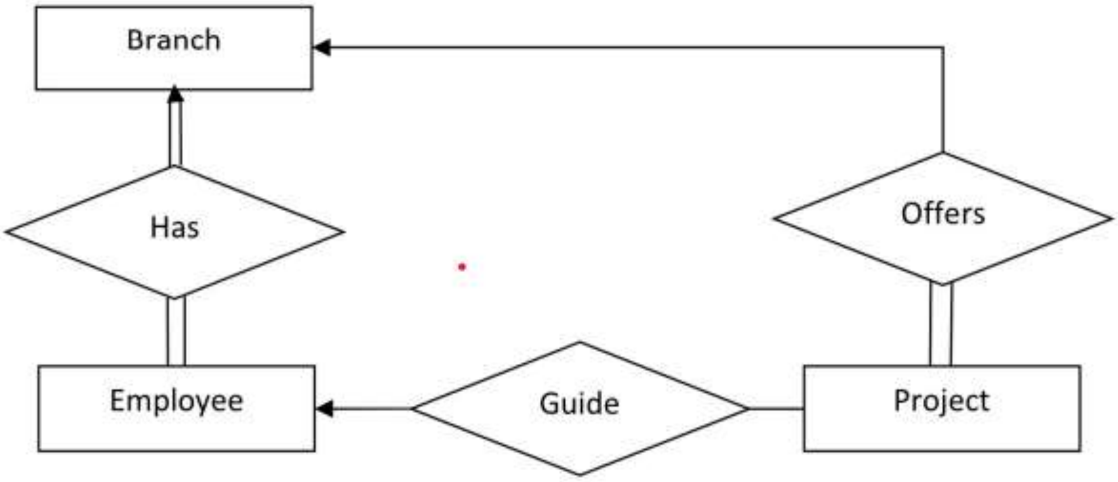
2. **Chasm trap**

- Chasm trap occurs when an E-R Diagram suggests the existence of a relationship between entity sets, but in real it doesn't exist.
- This is the possible when relationship set with partial participation forms the way of connection between entity sets.

- The relationship set Guide provides that employee provides guidance to Project. But all employees don't need to guide projects. And all projects don't be guided by some of the employees. So both entities are partially participate in Guide.
- Problem: If any project is associated with any employees it can't be associated with its branch.

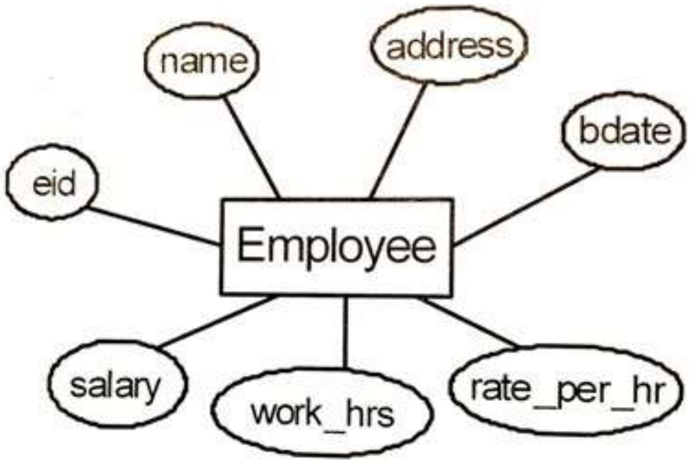


Solution: Add missing relationship

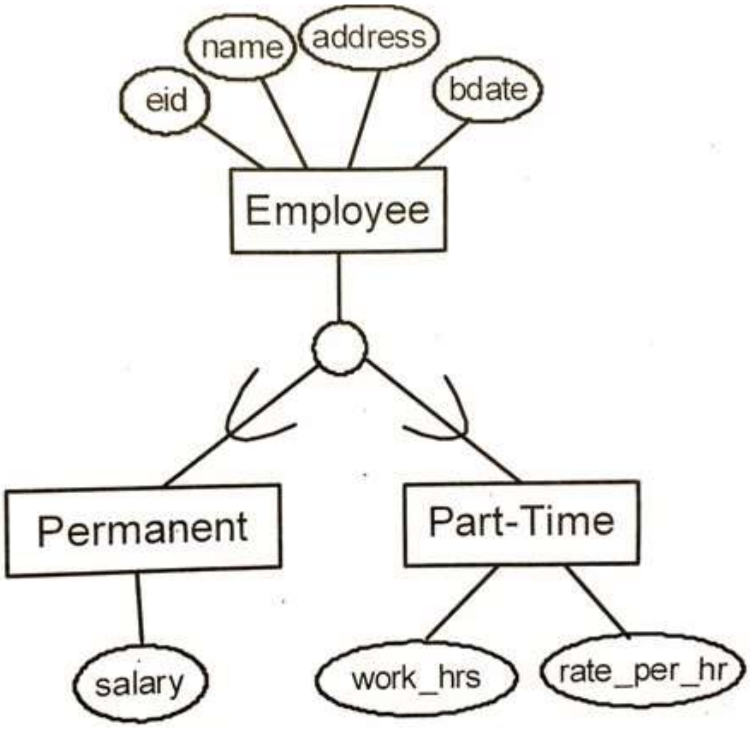


Specialization

- Specialization is the process of defining sub-classes of a super-class based on some distinguishing characteristics.
- Specialization is a top-down process to define super-class/sub-class relationship.
- It maximizes the differences between entities of the entity set and identifies unique characteristics for sub-sets of entities.
- For example, consider ***Employee*** entity set is shown in figure.

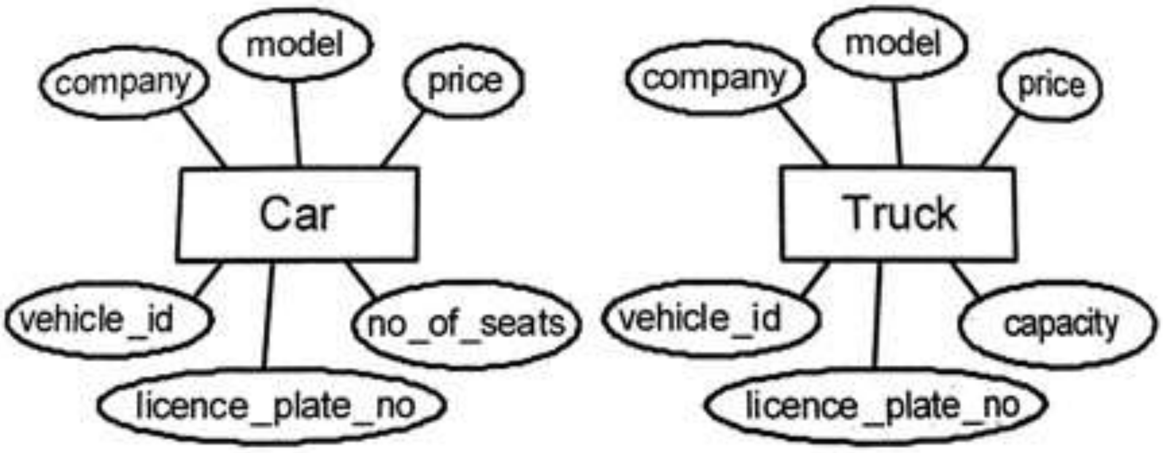


- Based on this, our entity set Employee can be further classified into two sub-sets, Permanent employees and Part-time employees. In other words, two sub-classes are derived from the single super-class. This process is called specialization.
- These relationships are shown in figure.

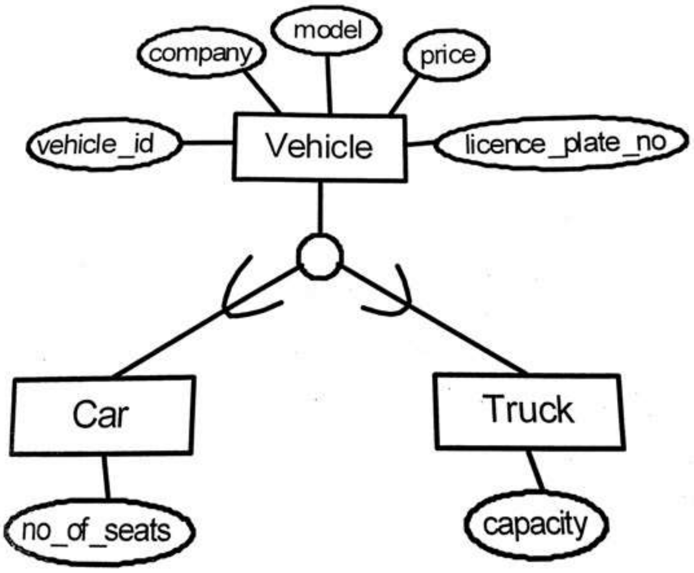


Generalization:

- Generalization is the process of defining a super-class from sub-classes based on some common characteristics.
- Generalization is a bottom-up process to define super-class/sub-class relationship.
- It minimizes the differences between entities of different entity sets by identifying common characteristics.
- For example, consider the two entity sets ***Car*** and ***Truck*** as given in following figure.



- Analysis of these two entity sets shows that they have many attributes in common.
- Based on this, another entity set, called ***Vehicle,*** can be defined as a super-class for two sub-classes Car and Truck.
- All the common attributes are moved to super-class Vehicle. While, sub-classes Car and Truck keep attributes which are unique to them.
- These relationships are shown in figure.
- Here, it is also said that sub-classes Car and Truck are generalized to Vehicle super-class.

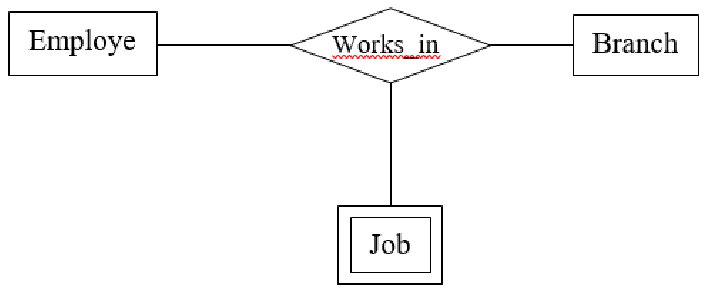


Weak Entity Set

- The entity set which does not have sufficient attributes to form a primary key is called as Weak entity set.

Example:

- Consider an entity set Payment which has three attributes: payment_number, payment_date and payment_amount.
- Although each payment entity is distinct but payment for different loans may share the same payment number.
- Thus, this entity set does not have a primary key and it is an entity set.
  - A weak entity set is represented by doubly outlined rectangles.


Employe
Works_in
Branch
Job