DATABASE SYSTEMS

S.PRABHAVATHI
ASSISTANT PROFESSOR
DEPARTMENT OF COMPUTER SEIENCE & IT
JAMAL MOHAMED COLLEGE
TRICHY – 620 020

UNIT II

Introduction to SQL(Structured Query Language): Overview of SQL – SQL Definition – Basic Structure of SQL Queries – Additional Basic Operations – Set Operations – # Null Values – Aggregate Functions # – Nested Sub-queries – Modification of the database - Intermediate SQL: Join Expression – Views. Database Design: Entity-Relationship Model – Constraints – Entity-Relationship Diagram.

1. Overview of Structured Query Language(SQL) :

SQL is a database computer language designed for the retrieval and management ofdata in a relational database. SQL stands for **Structured Query Language**. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language. Originally, SQL was called SEQUEL (Structured English QUEry Language). SQL uses the terms table, row, and column for the formal relational model terms relation, tuple, and attribute, respectively.

SQL uses the concept of a catalog—a named collection of schemas in an SQL environment. An SQL environment is basically an installation of an SQL-compliant RDBMS on acomputer system. A catalog always contains a special schema called INFORMATION_SCHEMA , which provides information on all the schemas in the catalog andall the element descriptors in these schemas.

SQL Constraints

- **NOT NULL** - Ensures that a column cannot have a NULL value
- **UNIQUE** - Ensures that all values in a column are different
- **PRIMARY KEY** - A combination of a NOT NULL and UNIQUE. Uniquely identifieseach row in a table
- **FOREIGN KEY** - Uniquely identifies a row/record in another table
- **CHECK** - Ensures that all values in a column satisfies a specific condition
- **DEFAULT** - Sets a default value for a column when no value is specified

SQL Commands

- DDL: Data Definition Language
- DML: Data Manipulation Language
- DCL: Data Control Language
- TCL:Transaction Control Language

DDL: The SQL DDL allows specification of not only a set of relations, but alsoinformation about each relation, including:

- The schema for each relation.
- The types of values associated with each attribute.
- The integrity constraints.
- The set of indices to be maintained for each relation.

Basic data types used are,

char	A fixed-length character string with user-specified length n. The fullform, character, can be used instead.
varchar	A variable-length character string with user-specified maximumlength n. The full form, character varying, is equivalent.
int	An integer (a finite subset of the integers that is machine dependent). Thefull form, integer, is equivalent.
float	A floating-point number, with precision of at least n digits.

Main Commands:

CREATE	Creates a new table, a view of a table, or other object in the database.
ALTER	Modifies an existing database object, such as a table.
DROP	Deletes an entire table, a view of a table or other objects inthe database.
Rename	Rename a table or its attribute.
Truncate	Operation that is used to mark the extents of a table for deallocation (empty for reuse)

Create

- To create database,
  CREATE DATABASE database_name

- To create Table Statement is used to create tables to store data. Integrity Constraintscan also be defined for the columns while creating the table.

  CREATE TABLE table name(Attribute1 datatype(No.),...Attribute n datatype(No));
  CREATE TABLE employee (id number(5),name char(20),dept(10));
- To create table constraint

  CREATE TABLE *table_name* (A*1 datatype constraint,A2 datatype constraint*,.....;

Primary key: The primary key attributes are required to be nonnull and unique.

CREATE TABLE Persons (PID int(10) NOT NULL PRIMARY KEY,LastName varchar(25),FirstName varchar(25));

Or

CREATE TABLE Persons (PID int (10) NOT NULL, LastName varchar(25), FirstNamevarchar(25), primary key(id));

Foreign Key

CREATE TABLE Orders (OrderID int NOT NULL PRIMARY KEY, OrderNumber int NOTNULL, PersonID int,FOREIGN KEY (PID) REFERENCES Persons(PID));

ALTER:

- To add column

  ALTER TABLE table_name ADD column_name datatype;
- To delete column

  ALTER TABLE table_name Drop column column_name;

DROP:

DROP DATABASE database_name;
DROP TABLE table_name;

RENAME:

- To rename a table

  RENAME TABLE tbl_name TO new_tbl_name**;**
- To rename a column

  ALTER TABLE table_name Rename column old column name to new name;

DML: The SQL commands that deals with the manipulation of data present in database belong to DML or Data Manipulation Language and this includes most of the SQL statements.

INSERT	used to insert data into a table
SELECT	used to retrieve data from the a database.
UPDATE	ud to update existing data
DELETE	used to delete records

INSERT

INSERT INTO *tablename* (*column1*, *column2*, ..)VALUES (*value1*, *value2*,.. .);

OR

INSERT INTO *table_name* VALUES (*value1*, *value2*, *value3*, ...);

SELECT

- To select a entire table
  SELECT * FROM table_name;
- To select column
  SELECT *column1*, *column2*, ...FROM *table_name;*
- To select rows
  SELECT *column1*, *column2*, ...FROM *table_name* WHERE *condition*;

UPDATE

UPDATE table_name SET column1 = value1, column2 = value2,... WHERE condition;

DELETE

- To delete all rows
  DELETE FROM *table_name*;
- To delete specific row
  DELETE FROM table_name WHERE condition;

DCL : DCL mainly deals with the rights, permissions and other controls of the database system

GRANT	gives user's access privileges to database.
REVOKE	withdraw user's access privileges given by using the GRANT command.

GRANT

GRANT privilege_name ON Table_name TO user_name;

Eg. GRANT SELECT ON employee TO user1

REVOKE

REVOKE privilege_name ON Table_name FROM user_name;

Eg. REVOKE SELECT ON employee FROM user1;

TCL

Transaction Control Language(TCL) commands are used to manage transactions in the database. These are used to manage the changes made to the data in a table by DML statements.

COMMIT	command is used to permanently save any transaction into the database.
ROLLBACK	command to rollback changes
SAVEPOINT	command is used to temporarily save a transaction so that you can rollbackto that point whenever required.

COMMIT:

Syntax- COMMIT;

ROLLBACK:

- The ROLLBACK command to rollback those changes, if they were not committed using the COMMIT command

  Rollback;

- The command restores the database to last committed state by using SAVEPOINTcommand.

  ROLLBACK TO savepoint_name;

SAVEPOINT:

SAVEPOINT savepoint_name;

2. Basic SQL Structure of SQL Queries :

The basic structure of an SQL query consists of three clauses: **select, from, and where**. The query takes as its input the relations listed in the from clause, operates on them as specified in the where and select clauses, and then producesa relation as the result.

2.1Queries on a Single Relation

Let us consider the below table Faculty and DEPT,

FID	FNAME	DEPT ID	SALARY
1	JISY	1	35000
2	SANTHY	1	30000
3	SWETHA	2	25000

Faculty Table

DEPT ID	DEPTNAME	Block
1	CS	New
2	EC	New
3	EE	old

DEPT Table

Let us consider a simple query using our Faculty table, “Find the names of all instructors.

Select FNAME from Faculty;

The result is a relation consisting of a single attribute. If want to force the elimination of duplicates, we insert the keyword **distinct** after select.

We can rewrite the preceding query as:

FID	FNAME	DEPTNAME
1	JISY	1
2	SANTHY	1
3	SWETHA	2

Select distinct DEPTNAME from FACULTY;

DEPT
CS
EC

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

Select all DEPTNAME from DEPT;

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

For example,the query returns a relation that isthe same as the Faculty relation, except that the attribute salary is multiplied by 1.1.

select FID , FNAME, SALARY * 1.1 from Faculty;

SQL allows the use of the logical connectives and, or, and not in the where clause. The operands of the logical connectives can be expressions involving the comparison operators <, <=, >, >=, =, and <>.

select FNAME from Faculty where SALARY>30000;

2.2 Queries on Multiple Relations

Consider two tables,

CID	**NAME**	**Addr**
1	Manju	abc
2	Jisy	cde
3	Vishnu	efg
4	Meera	hij

OID	CID	AMOUNT
1	2	100
2	1	250
3	4	300
4	3	400

- Retrieve CID, Address and amount from relation CUSTOMER and ORDERwhose name= jisy

  SELECT CUSTOMER.CID, Addr, AMOUNT FROM CUSTOMER, ORDER WHERE CUSTOMERS.CID = ORDERS.CID and NAME='Jisy';

- Retrieve customer id, name, Address and amount from relation CUSTOMER and ORDER

  SELECT CUSTOMER.CID, NAME, Addr, AMOUNT FROM CUSTOMER, ORDERWHERE CUSTOMERS.CID = ORDERS.CID;

3. Additional Basic Operations:

String Operations

SQL specifies strings by enclosing them in single quotes, for example, ’Computer’. The SQL standard specifies that the equality operation on strings is case sensitive; as a result the expression “ ’computer’ = ’Computer’ ” evaluates to false.

SQL also permits a variety of functions on character strings, such as concatenating (using “ ||”), extracting substrings, finding the length of strings, converting strings to uppercase (using the function upper(s) where s is a string) and lowercase (using the function lower(s)), removing spaces at the end of the string (using trim(s)). Pattern matching can be performed on strings, using the operator **like**. We describe patterns by using two special characters:

- Percent (%): multiple character
- Underscore (_): single character.

SELECT *column1, column2, ...*FROM *table_name* WHERE *columnN* LIKE *pattern*;

Example:

SELECT * FROM CUSTOMER WHERE Name LIKE 'a%';

WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%_%'	Finds any values that start with "a" and areat least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" andends with "o"

ORDER BY

The ORDER BY statement in sql is used to sort the fetched data in either ascending or descending according to one or more columns. By default ORDER BY sorts the data in ascending order.

SELECT *column1, column2, ...*FROM *table_name* ORDER BY *column1,column2, ...* ASC/DESC;

Eg. SELECT NAME FROM CUSTOMER OR DER BY Name DESC;

NAME
Vishnu
Manju
Jisy

SQL BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates. The BETWEEN operator is inclusive: begin and end values are included.

SELECT *column_name(s)* FROM *table_name* WHERE *column_name*
BETWEEN *value1* AND *value2;*

SELECT * FROM ORDER WHERE AMOUNT BETWEEN 100 AND 350;

OID	CID	AMOUNT
1	2	100
2	1	250
3	4	300

4. <u>SET OPERATIONS:</u>

The SQL operations union, intersect, and except operate on relations andcorrespond to the mathematical set-theory operations ∪, ∩, and −. Consider two tables First and Second,

First

ID	Name
1	JISY
2	SANTHY

Second

ID	Name
3	SWETHA
2	SANTHY

UNION Operation:is used to combine the results of two or more SELECT statements. However it will eliminate duplicate rows from its resultset. In case of union, number of columns and datatype must be same in both the tables, on which UNION operation is being applied.

SELECT * FROM First UNION SELECT * FROM Second;

ID	Name
1	JISY
2	SANTHY
3	SWETHA

UNION ALL: This operation is similar to Union. But it also shows the duplicate rows.

SELECT * FROM First UNION ALL SELECT * FROM Second;

ID	Name
1	JISY
2	SANTHY
3	SWETHA
2	SANTHY

INTERSECT: Intersect operation is used to combine two SELECT statements, but it only retunsthe records which are common from both SELECT statements. In case of **Intersect** the number of columns and datatype must be same.

SELECT * FROM First INTERSECT SELECT * FROM Second;

ID	Name
2	SANTHY

Minus/ Except: It combines the result of two SELECT statements. Minus operator is used todisplay the rows which are present in the first query but absent in the second query.

SELECT * FROM First Except SELECT * FROM Second;

ID	Name
1	JISY

SELECT * FROM Second MINUS SELECT * FROM First ;

ID	Name
3	SWETHA

5. <u>Aggregate Functions:</u>

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

Basic Aggregation

SQL offers five built-in aggregate functions:

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

Select Aggregate fn(column name) from table_name where condition;

Stud

RollNo.	**Name**	**Mark**	**Dept**
1	A	40	cs
2	B	36	cs
3	C	28	ec
4	B	30	ec
5	F	46	ee
6	G	34	cs

AVG():

SELECT AVG(*column_name*) FROM *table_name* WHERE *condition*;

Select avg(Mark) from Stud;

COUNT(): The aggregate function count used to count the number of tuples in a relation.

Select Count(*) from Stud;

Count(*)
6

Select Count(*) from Stud where Name='B';

Count(*)
2

Select Count (Distinct Name) from Stud;

Name
A
B
C
F
G

MIN(): The MIN() function returns the smallest value of the columns.
Select Min(Mark) from Stud ;

Min
28

Max(): The MAX() function returns the largest value of the selected column.
Select Max(Mark) from Stud ;

Max
46

Sum(): The SUM() function returns the total sum of a numeric column.
Select sum(Mark) from Stud ;
Select sum(M1+M2) from Stud ;(also possible)

Aggregation with Group by:

The GROUP BY statement is often used with aggregatefunctions.
Select Aggregate fn(column name) from table_name **group by** column name;
Select Dept, Count(RollNo) from Stud Group By Dept;

Dept	Count
cs	3
ec	2
ee	1

Group by using the HAVING clause: Grouping data with certain condition.

SELECT column_name(s) FROM table_name
WHERE condition **GROUP BY** column name(s)**HAVING** condition

Select Dept, Count(RollNo) from Stud Group By Dept Having Mark>35;

Dept	Count
cs	2
ec	0
ee	1

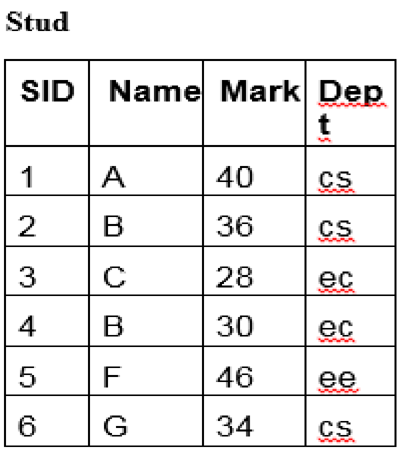
Aggregation with Null and Boolean Values

In general, aggregate functions treat nulls according to the following rule: All aggregate functions except count (*) ignore null values in their input collection. As a result of null values being ignored, the collection of values may be empty. The count of an empty collection is defined to be 0, and all other aggregate operations return a value of null when applied on an empty collection. A Boolean data type that can take values true, false, and unknown.

6. Nested sub-queries:

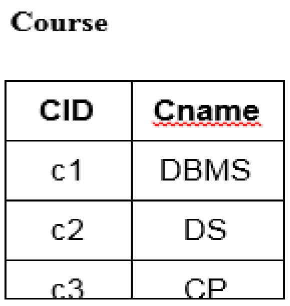
A Subquery or Inner query or a Nested query is a query within another SQL query and embedded within the WHERE clause. A subquery is a SELECT statement that is nested within another SELECT statement and which return intermediate results. The result of inner query is used in execution of outer query.

Consider the relation Stud, Course and Scourse



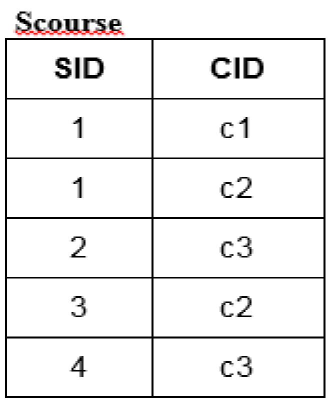
Stud

SID	Name	Mark	Dept
1	A	40	cs
2	B	36	cs
3	C	28	ec
4	B	30	ec
5	F	46	ee
6	G	34	cs



Course

CID	Cname
c1	DBMS
c2	DS
c3	CP



Scourse

SID	CID
1	c1
1	c2
2	c3
3	c2
4	c3

Independent Nested Queries:

In independent nested queries, query execution starts from innermost query to outermost queries. The execution of inner query is independent of outer query, but the result of inner query is used in execution of outer query. Various operators like IN, NOT IN, ANY, ALL etc are used in writing independent nested queries.

IN (Set Membership)

The IN connective for set membership, where the set is a collection of values producedby a select clause. The NOT IN connective for the absence of set membership.

SELECT column-names FROM table-name1 WHERE value IN (SELECT column-name FROM table-name2 WHERE condition)

Q1. If we want to find out SID who are enrolled in Cname 'DS' or 'DBMS'.

From Course table, we can find out CID for Cname 'DS' or DBMS' and we can use these CIDs for finding SIDs from Scourse TABLE.

STEP 1: Finding CID for Cname ='DS' or 'DBMS'

Select CID from Course where Cname = 'DS' or Cname = 'DBMS';

STEP 2: Using CID of step 1 for finding SID

Select SID from Scourse where CID **IN** (Select CID from Course where Cname'DS'or Cname = 'DBMS');

Q2. Find out names of STUDENTs who have either enrolled in 'DS' or 'DBMS', it can be doneas:

Select Name from Stud where SID **IN** (Select SID from Scourse where CID **IN** (SelectCID from Course where Cname = 'DS' or Cname = 'DBMS'));

Q3. If we want to find out SIDs of STUDENTs who have neither enrolled in 'DSA' nor in 'DBMS',it can be done as:

Select Name from Stud where SID **NOT IN** (Select SID from Scourse where CID **IN** (Select CID from Course where Cname = 'DS' or Cname = 'DBMS'));

Test for Empty Relations

SQL includes a feature for testing whether a subquery has any tuples in its result. The **exists** construct returns the value true if the argument subquery is nonempty. We can test for the nonexistence of tuples in a subquery by using the **not exists** construct

Q1. If we want to find out NAME of Student who are enrolled in CID ‘C1’

Select NAME from Stud where **EXISTS**(select * from Scourse where Stud.SID=Scourse.SIDand Scourse.CID=’C1’);

Correlated Query: With a normal nested subquery, the inner SELECT query runs first and executes once, returning values to be used by the main query. A **correlated subquery** is a subquery that uses values from the outer query.

Eg.SELECT employee_number, name FROM employees emp WHERE salary > (SELECT AVG(salary) FROM employees WHERE department = emp.department);

Test for the Absence of Duplicate Tuples

Unique constraint in SQL is used to check whether the sub query has duplicate tuples in it’s result. Unique construct returns true only if the sub query has no duplicate tuples, else it return false. We can test for the existence of duplicate tuples in a subquery by using the not unique construct.

Q1. Find course ID who enrolled in atleast one course?

SELECT Course.CID FROM Course WHERE UNIQUE (SELECT CID FROM Scourse where Scourse.CID=Course.CID);

ALL

The ALL operator returns TRUE if all of the subqueries values meet the condition.

SELECT column-names FROM table-name WHERE column-name operator ALL(SELECT column-name FROM table-name WHERE condition)

Q1. Returns the names, Rollno of students whose mark is greater than the mark of all the students in department ec:

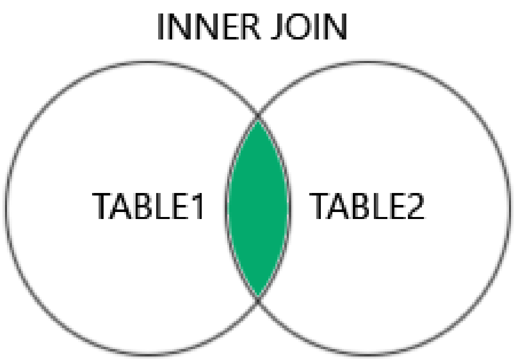
SELECT Name , SID FROM Stud WHERE Mark > ALL (SELECT Mark FROM Stud WHERE Dept =ec);

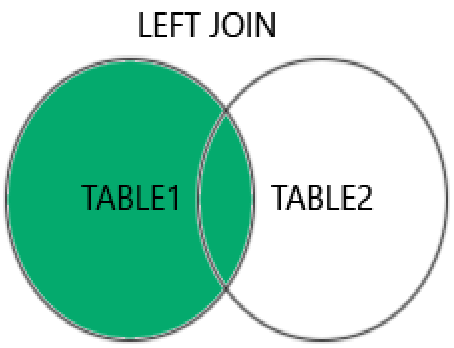
7. **JOIN Expression:**

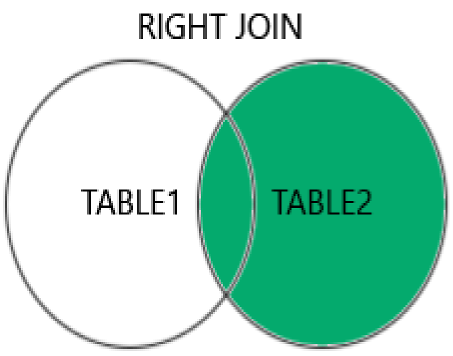
Types of SQL JOINs:

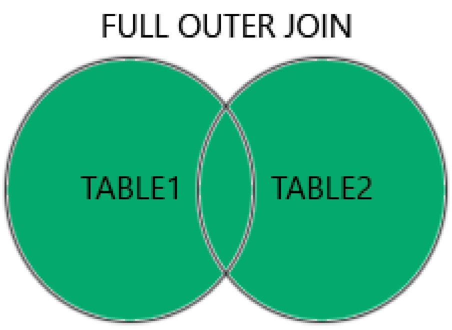
Here are the different types of the JOINs in SQL:

- (INNER) JOIN: Returns records that have matching values in both tables
- LEFT (OUTER) JOIN: Returns all records from the left table, and the matched records from the right table
- RIGHT (OUTER) JOIN: Returns all records from the right table, and the matched records from the left table
- FULL (OUTER) JOIN: Returns all records when there is a match in either left or right table









JOINING TABLES

`JOIN` combines data from two tables.

TOY		
toy_id	toy_name	cat_id
1	ball	3
2	spring	NULL
3	mouse	1
4	mouse	4
5	ball	1

CAT	
cat_id	cat_name
1	Kitty
2	Hugo
3	Sam
4	Misty

JOINtypically combines rows with equal values for the specified columns. **Usually**, one table contains a **primary key**, which is a column or columns that uniquely identify rows in the table (the cat_idcolumn in the cattable).
The other table has a column or columns that **refer to the primary key columns** in the first table (the cat_id column in the toytable). Such columns are **foreign keys**. The JOINcondition is the equality between the primary key columns in one table and columns referring to them in the other table.

JOIN

JOINreturns all rows that match the ONcondition.
JOINis also called INNER JOIN.

```
SELECT *
FROM toy JOIN cat
ON toy.cat_id = cat.cat_id;
```

toy_id	toy_name	cat_id	cat_id	cat_name
5	ball	1	1	Kitty
3	mouse	1	1	Kitty
1	ball	3	3	Sam
4	mouse	4	4	Misty

There is also another, older syntax, but it **isn't recommended**.
List joined tables in the FROM clause, and place the conditions in the WHEREclause.

```
SELECT *
FROM toy, cat
WHERE toy.cat_id = cat.cat_id;
```

JOIN CONDITIONS

The JOINcondition doesn't have to be an equality – it can be any condition you want. JOINdoesn't interpret the JOIN condition, it only checks if the rows satisfy the given condition.
To refer to a column in the JOINquery, you have to use the full column name: first the table name, then a dot (.) and the column name: `ON cat.cat_id = toy.cat_id`
You can omit the table name and use just the column name if the name of the column is unique within all columns in the joined tables.

NATURAL JOIN

If the tables have columns with **the same name**, you can use NATURAL JOINinstead of JOIN.

```
SELECT *
FROM toy
NATURAL JOIN cat;
```

cat_id	toy_id	toy_name	cat_name
1	5	ball	Kitty
1	3	mouse	Kitty
3	1	ball	Sam
4	4	mouse	Misty

The common column appears only once in the result table.
Note: NATURAL JOINis rarely used in real life.

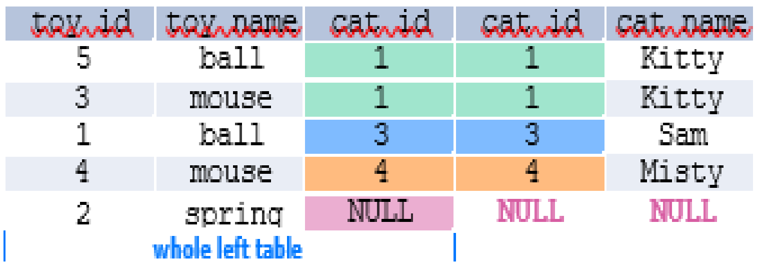
TOY		
toy_id	toy_name	cat_id
1	ball	3
2	spring	NULL
3	mouse	1
4	mouse	4
5	ball	1

CAT	
cat_id	cat_name
1	Kitty
2	Hugo
3	Sam
4	Misty

LEFT JOIN

LEFT JOINreturns all rows from the **left table** with matching rows from the right table. Rows without a match are filled with NULLs. LEFT JOINis also called LEFT OUTER JOIN.

```
SELECT *
FROM toy
LEFT JOIN cat
ON toy.cat_id = cat.cat_id;
```



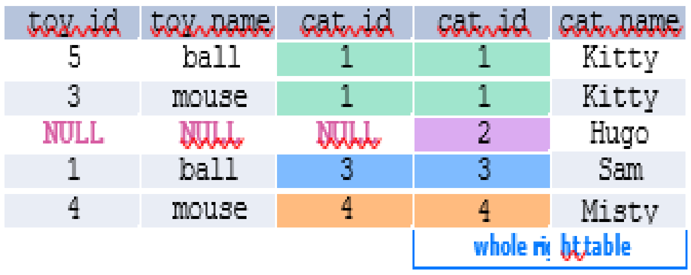
toy_id	toy_name	cat_id	cat_id	cat_name
5	ball	1	1	Kitty
3	mouse	1	1	Kitty
1	ball	3	3	Sam
4	mouse	4	4	Misty
2	spring	NULL	NULL	NULL

whole left table

RIGHT JOIN

RIGHT JOIN returns all rows from the **right table** with matching rows from the left table. Rows without a match are filled with NULLs. RIGHT JOINis also called RIGHT OUTER JOIN.

```
SELECT *
FROM toy
RIGHT JOIN cat
ON toy.cat_id = cat.cat_id;
```



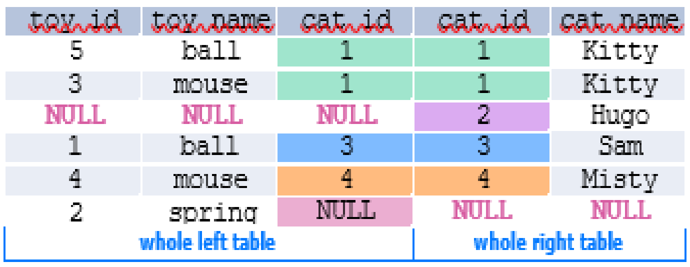
toy_id	toy_name	cat_id	cat_id	cat_name
5	ball	1	1	Kitty
3	mouse	1	1	Kitty
NULL	NULL	NULL	2	Hugo
1	ball	3	3	Sam
4	mouse	4	4	Misty

whole right table

FULL JOIN

FULL JOINreturns all rows from the **left table** and all rows from the **right table**. It fills the non-matching rows with NULLs. FULL JOINis also called FULL OUTER JOIN.

```
SELECT *
FROM toy
FULL JOIN cat
ON toy.cat_id = cat.cat_id;
```



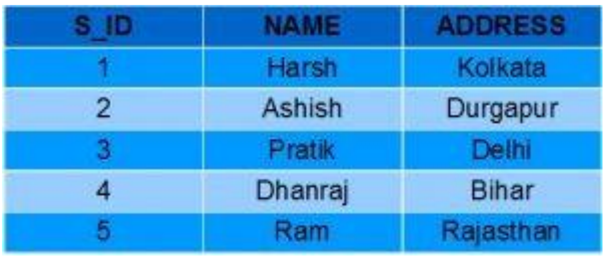
toy_id	toy_name	cat_id	cat_id	cat_name
5	ball	1	1	Kitty
3	mouse	1	1	Kitty
NULL	NULL	NULL	2	Hugo
1	ball	3	3	Sam
4	mouse	4	4	Misty
2	spring	NULL	NULL	NULL

whole left table whole right table

8. Views(Virtual Table):

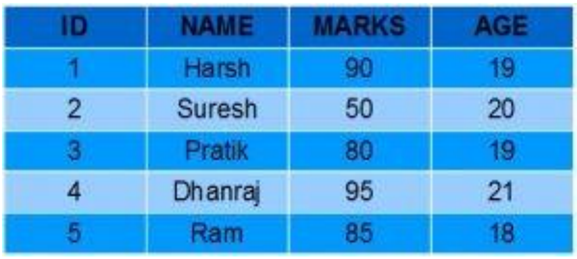
A view is a virtual table based on the result-set of an SQL statement. A view contains rows and columns, just like a real table. The fields in a view are fields from one or more real tables in the database. We can create a view by selecting fields from one or more tables present in the database. A View can either have all the rows of a table or specific rows based on certain condition. Views are a logical virtual table created by "select query" but the result is not stored any where in the disk and every time we need to fire the query when we need data, so alwayswe get updated or latest data from original tables.

SD



S_ID	NAME	ADDRESS
1	Harsh	Kolkata
2	Ashish	Durgapur
3	Pratik	Delhi
4	Dhanraj	Bihar
5	Ram	Rajasthan

SM



ID	NAME	MARKS	AGE
1	Harsh	90	19
2	Suresh	50	20
3	Pratik	80	19
4	Dhanraj	95	21
5	Ram	85	18

Creating Views

A View can be created from a single table or multiple tables.

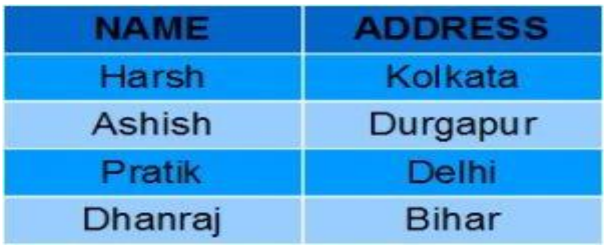
```
CREATE  VIEW  view_name  AS  SELECT  column1,  column2.....
FROM  table_name  WHEREcondition;
```

- **Creating View from a single table:**

  CREATE VIEW Details AS SELECT NAME, ADDRESS FROM SD WHERE S_ID < 5;

To see the data in the View, we can query the view in the same manner as we query a table.

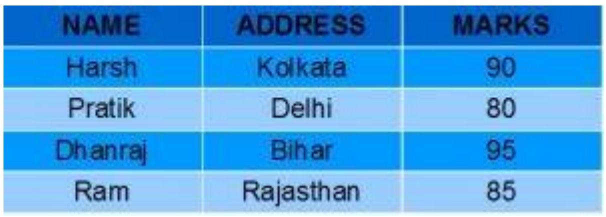
SELECT * FROM Details;



NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

- **Creating View from multiple tables**:

  CREATE VIEW Marks AS SELECT SD.NAME, SD.ADDRESS, SM.MARKS
  FROM SD, SM WHERE SD.NAME =SM.NAME;



NAME	ADDRESS	MARKS
Harsh	Kolkata	90
Pratik	Delhi	80
Dhanraj	Bihar	95
Ram	Rajasthan	85

UPDATING VIEWS

A view can be updated under certain conditions which are given below −

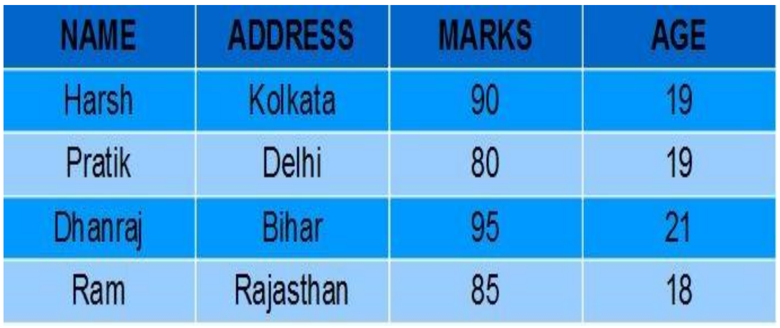
- The SELECT clause may not contain the keyword DISTINCT.
- The SELECT clause may not contain summary functions.
- The SELECT clause may not contain set functions.
- The SELECT clause may not contain set operators.
- The SELECT clause may not contain an ORDER BY clause.
- The query may not contain GROUP BY or HAVING.
- Calculated columns may not be updated.

We can use the CREATE OR REPLACE VIEW statement to add or remove fields from a view.

```
CREATE OR REPLACE VIEW view_name AS SELECT column1,coulmn2,..
FROM    table_name  WHERE condition;
```

For example, if we want to update the view **Marks** and add the field AGE to this View from **SM** Table, we can do this as:

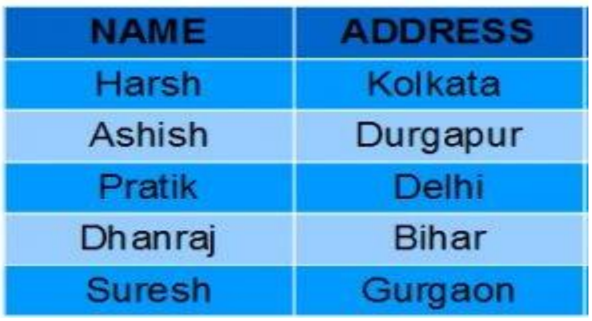
```
CREATE OR REPLACE VIEW Marks AS SELECT SD.NAME, SD.ADDRESS,

SM.MARKS,SM.AGE FROM SD,SM WHERE SD.NAME = SM.NAME;
```



NAME	ADDRESS	MARKS	AGE
Harsh	Kolkata	90	19
Pratik	Delhi	80	19
Dhanraj	Bihar	95	21
Ram	Rajasthan	85	18

Inserting a row

```
INSERT INTO View name(C1,C2...Cn) VALUES(V1,V2….Vn);
INSERT INTO Details(NAME, ADDRESS) VALUES("Suresh","Gurgaon");
```



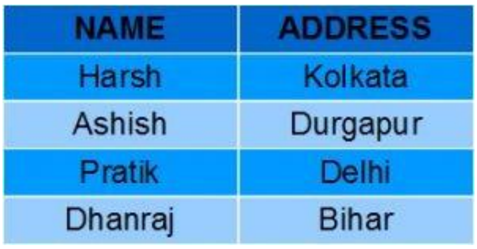
NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar
Suresh	Gurgaon

DELETING Rows

Deleting rows from a view is also as simple as deleting rows from a table. We can use the DELETE statement of SQL to delete rows from a view. Also deleting a row from a view first delete the row from the actual table and the change is then reflected in the view.

```
DELETE FROM view_name WHERE condition;
```

```
DELETE FROM Details WHERE NAME="Suresh";
```



NAME	ADDRESS
Harsh	Kolkata
Ashish	Durgapur
Pratik	Delhi
Dhanraj	Bihar

DELETING VIEWS

We can delete or drop a View using the DROP statement.

```
DROP VIEW view_name;
```

Materialized View

Materialized views are also the logical view of our data-driven by the select query but the resultof the query will get stored in the table or disk.

Materialized View vs View

View	Materialized View
Views query result is not stored in the disk or database	Materialized view allow to store the queryresult in disk or table.
when we create a view using any table, rowed of view is same as the original table	Materialized view rowed is different.
View we always get latest data	Materialized view we need to refresh the view for getting latest data.
Performance of View is less than Materializedview.	Performance of Materialized View is higherthan view.

9.ENTITY-RELATIONSHIP [E-R] MODEL:

ER model is a popular high level conceptual data model. Earlier commercial systems were based on the hierarchical and network approach. The ER model is a generalization of these models. It allows the representation of explicit constraints as well as relationships.

Even though the ER model has some means of describing the physical database model, it is basically useful in the design and communication of the logical database model.

In this model, objects of similar structure are collected into entity set. The relationship between entity set is represented by a named ER relationship and is 1:1, 1:M or M:N., mapping from one entity set to another. The database structure, employing the ER model is usually shown pictorially using ERDs.

ER model consist of basic objects, called entities and relationship among these objects. It represents overall logical structure of a database. The database structure, employing the ER model is usually shown pictorially using ERDs. Semantic modeling is known by many names including data modeling, Entity relationship modeling, entity modeling and object modeling. ER model was introduced by Chen in 1976 and refined in various ways by Chen and numerous others since that time.
It is one of the several semantic data models ER model is based on 3 basic concepts.

1) Entity sets
2) Relationship sets
3) Attributes

1.Entity sets: An Entity is a thing or object in the real world that is distinguishable from all other objects. It is an object of interest to an organization.
E.g. Each person in an enterprise Each student in the institute Loan in bank etc.
For instance a student has student-id which uniquely identifies one particular student in the institute. Similarly, loans can be thought of as entities and loan number at particular branch uniquely identifies a loan entity.

An entity may be concrete, such as a person or a book, or it may be abstract such as loan, holiday Objects of similar types are characterized by the same set of attributes or properties. Such similar objects form an entity set or entity type. Two objects are mutually distinguishable and this fact is represented in the entity set by giving them unique identifiers.

Objects are represented by their attributes and as objects are inter distinguishable, a subset of these attributes forms a primary key or a key for uniquely identifying an instance of an entity.

An Entity Set is a set of entities of the same type that share the same properties or attributes. The set of all persons who are students at a given institute can be defined as an entity set student. Similarly entity set loan might represent set of all loans awarded by a particular bank. The individual entities that constitute a set are said to be the extension of the entity set. Eg. Individual bank customers are the extension of the entity set customer.

2. Relationship sets: A relationship is an association among several entities .e.g. Relationship between loan and customer entities. A relationship set is a set of relationships of the same type. We define the relationship set borrower to denote the association between customers and bank loans that the customers have. When 2 entities are involved in relationship it is called binary relationship. When 3 entities are involved in it it is ternary relationship. When N entities are involved then it is N-ary relationship.

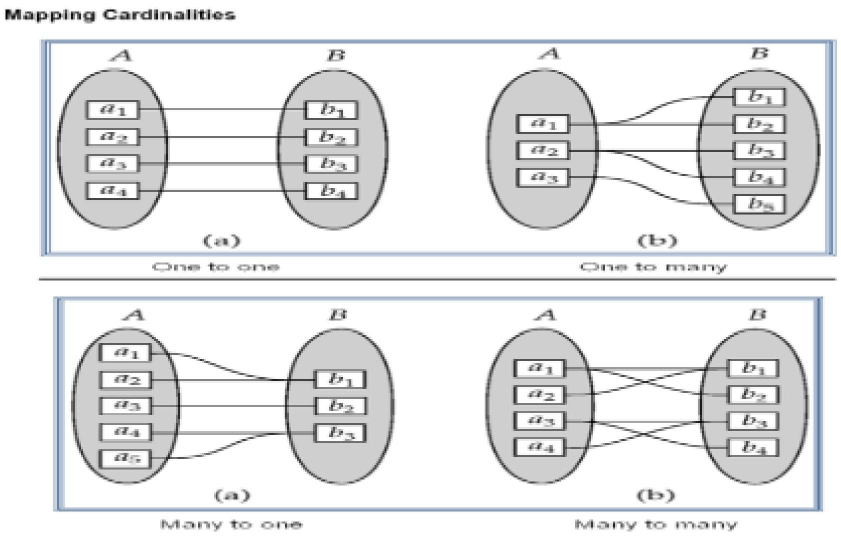
The association between entity sets is referred to as participation. The entities involved in a given relationship are said to be participants in that relationship. The function that an entity plays in a relationship is called that entities role. The relationship may also have attributes called descriptive attributes. A binary relationship can be one-to-one, one-to-many, many-to-many.

E.g. of one-to-one Relationship
1) person-spouse
2) person-driving license

E.g. of Many-to-one Relationship

1) Project-Manager
2) Employee-Department
3) Dependent-Employee

E.g. of Many-to-Many Relationships
1) Employee - Project
2)Supplier – Parts



Entities are distinguishable objects of concern and are modeled using their characteristics and attributes.

3.Attributes

Formally, an attribute of an entity set is a function that maps from the entity set into a domain. Associations exist between different attributes of an entity. An association between 2 attributes indicates that the values of the associated attributes are interdependent. Association that exists among the attributes of an entity is called attribute association. And that exist between entities is called a relationship. Formally, the association or interdependence between attributes is called functional dependency.

An entity is represented by a set of attributes. Attributes are properties of each member of entity set. E.g. possible attributes for customer entity set are customer-id, customer-name, customerstreet and customer-city. Each attribute has a value for each of its attributes. For each attribute, there is a set of permitted values, called the domain, or value set of that attribute. Eg.Domain of attribute loan -number might be the set of all strings of the form “L-n” where n is a positive integer.

Since an entity set may have several attributes each entity can be described by a set of (attribute, data value) pairs.

An attribute values describing an entity will constitute a significant portion of the data stored in the database. An attribute as used in ER model can be characterized by the following attribute types.

1) Simple and composite attributes
2) Single and multivalued attributes
3) Derived Attributes

Simple and composite attributes:

Simple attributes are attributes which are not further divided into subparts. On other hand composite attributes can be divided into subparts. (I.e. other attributes).
E.g. Attribute student-name could be structured as composite attribute consisting of first-name, middle-initial and last name.
E.g.Attribute address may consist of attributes street (street number, street name, apartment number), city, state and zip code.
Thus composite attributes help us to group together related attributes.

Single valued and multivalued attributes :

Attributes having single value for a particular entity are single valued. E.g. Loan-number.
Attributes that have a set of values for a specific entity are multivalued. E.g.

1) Employee entity set with the attribute **phone-number**. An employee may have zero, one or several phone numbers, and different employees may have different numbers of phones. This type of attribute is said to be multivalued.

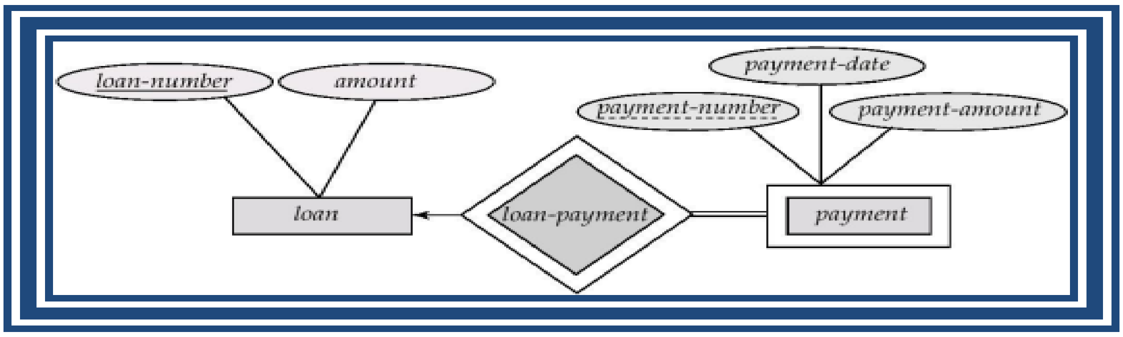
2) An attribute **dependent-name** of the employee set would be multivalued, since any particular employee may have zero, one or more dependents.

Derived attributes:

The value for this type of attribute can be derived from the values of other related attributes or entities. E.g. 1) Customer entity has an attribute age. If the customer entity also has an attribute **date-of –birth**, we can calculate age from the date-of-birth and the current date. Thus age is a derived attribute. (Here date-of-birth may be referred to as a base attribute or a stored attribute). The value of a derived attribute is not stored but is computed when required.

2) If the customer entity set has an attribute loans-held, which represents how many loans a customer has from the bank. We can derive the value for this attribute by counting the number of loan entities associated with that customer. An attribute takes null value when an entity does not have a value for it. E.g. Middle name Null may indicate not applicable –that is, that the value does not exist for the entity, or the attribute value is unknown (missing or not known). E.g.

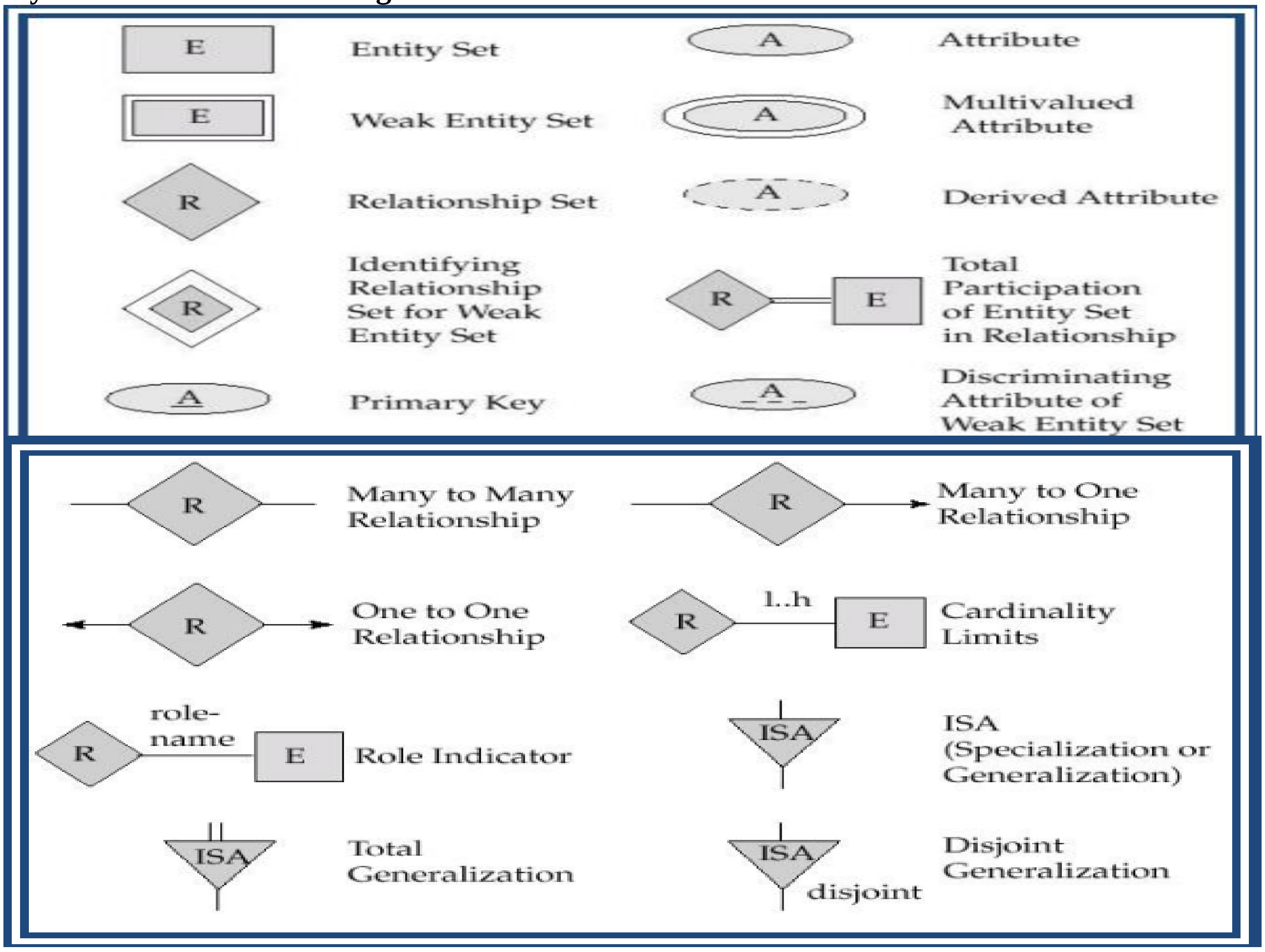
3) A null value for apartment number attribute could mean that the address does not include an apartment number (not applicable), that an apartment number exists but we do not know what it is (missing) or it is not part of customer address (unknown)
Entity:
An entity set that has a primary key is termed as strong entity set. E.g. Employee
An entity that does not have sufficient attributes to form a primary key is termed as a weak entity set. E.g. Dependent.
Thus weak entity is an entity that is existence dependent on some other entity, in the sense that it can not exist if that other entity does not also exist. So here if a given employee is deleted, all dependents of that employee must be deleted too. Entities belonging to a weak entity type are identified by being related to specific entities from another entity type in combination with some of their attribute values. We call this other entity type the identifying owner (strong entity) and we call the relationship type that relates a weak entity type to its owner the identifying relationship of the weak entity type. A weak entity cannot be identified without an owner entity.



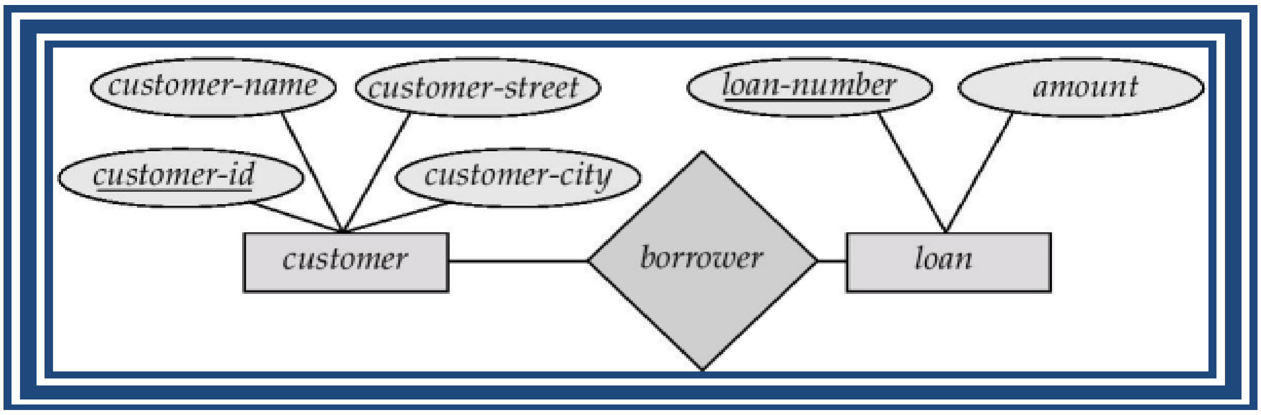
10.Entity Relationship diagram (ERD):

1)An entity is shown as a rectangle
2) A diamond represents the relationship among a number of entities, which are connected to the diamond by lines.
3) The attributes, shown as ovals, are connected to the entities or relationship by lines.
4) Diamonds, ovals and rectangles are labeled. The type of relationship existing between the entities is represented by giving the cardinality of the relationship on the line joining
The relationship to the entity.

Symbols used while drawing ERD:



Example of Entity Relationship diagram (ERD)


customer-name
customer-street
loan-number
amount
customer-id
customer-city
customer
borrower
loan