DATABASE MANAGEMENT SYSTEM LAB

(PCC-CS 691)

Last Revised

February, 2023

Compiled by

Dept. of CSE
Techno Main Salt Lake

DATABASE MANAGEMENT SYSTEM LAB

:INDEX:

DATABASE MANAGEMENT SYSTEM LAB SYLLABUS

Code: **PCC-CS691**

Contact: **4P**

Credits: **2**

Structured Query Language

1. Creating Database

- Creating Database
- Creating a Table
- Specifying Relational Data Types
- Specifying Constraints
- Creating Indexes

2. Table and Record Handling

- INSERT statement
- Using SELECT and INSERT together
- DELETE, UPDATE, TRUNCATE statements
- DROP, ALTER statements

3. Retrieving Data from a Database

- The SELECT statement
- Using the WHERE clause
- Using Logical Operators in the WHERE clause
- Using IN, BETWEEN, LIKE , ORDER BY, GROUP BY and HAVING

Clause

- Using Aggregate Functions
- Combining Tables Using JOINS
- Subqueries

4. Database Management

- Creating Views
- Creating Column Aliases
- Creating Database Users
- Using GRANT and REVOKE

Cursors in Oracle PL / SQL

Writing Oracle PL / SQL Stored Procedures

REFERENCE BOOKS

1. SQL/PLSQL Ivan Bayross
2. Fundamentals of database systems(Ramez Elmsari,Shamkant B.Navathe)
3. Database System Concepts (Avi Silberschatz · Henry F.Korth · S. Sudarshan)
4. Oracle Database 10g: The Complete Reference (Kevin Loney)
5. Oracle 9i: The Complete Reference (Kevin Loney, George Koch, and the Experts at TUSC)

Assignment List

Exp. No.	List of Experiments	Week No.
1.	Design an ER diagram for a Motor Vehicle Branch that administers driving tests and issues driver's licenses. Analyze the requirements by identifying the entities, attributes, relationships, keys, constraints etc. Apply extended entity-relationship features to the design. Defend your design with proper assumptions and justifications. Map the ER model into a relational model.	**Week1**
2.	Design an ER diagram for an application that models soccer teams, the games they play, and the players in each team. Analyze the requirements by identifying the entities, attributes, relationships, keys, constraints etc. Apply extended entity-relationship features to the design. Defend your design with proper assumptions and justifications. Map the ER model into a relational model.	**Week2**
3.	Design an ER diagram for an application that models an educational institute having several departments, faculty, students, projects, student hostels etc. Analyze the requirements by identifying the entities, attributes, relationships, keys, constraints etc. Apply extended entity-relationship features to the design. Defend your design with proper assumptions and justifications. Map the ER model into a relational model.	**Week3**
4.	i. Create tables for Client, Product, and Salesman with the attributes given, implementing DDL commands for specifying prime attributes, non-prime attributes, foreign keys, cardinalities, null values, constraints etc. and the data types. Implement DDL commands for drop, alter on the tables created. ii. Implement DML commands like populating the tables with data using insert command and retrieving data using simple queries in SQL. (Application of select, update, delete etc.)	**Week4**
5.	i. Create tables for Client, Product, Salesman, Sales__Order, and Sales_Order_Details and populate them. Retrieve data by writing queries in SQL using logical operators, aggregate operators, group by, having, order by clauses etc. ii. Create tables for Employee, Company and works and populate them. Retrieve data by writing nested queries in SQL using JOIN to combine tables and other operators like IN, BETWEEN, LIKE etc.	**Week5**
6.	i. Design an ER diagram for an application that models a car-insurance company whose customers own one or more cars each. Analyze the requirements by identifying the entities, attributes, relationships, keys, constraints etc. Apply extended entity-relationship features to the design. Defend your design with proper assumptions and justifications. Map the ER model into a relational model.	**Week6**

Exp. No.	List of Experiments	Week No.
	ii. Create tables, populate with data and construct queries (advanced) in SQL to extract information from the car insurance company's database.	
7.	i. Design an ER diagram for an application that models a hospital doctors treat patients, prescribe tests, monitor progress etc. Analyze the requirements by identifying the entities, attributes, relationships, keys, constraints etc. Apply extended entity-relationship features to the design. Defend your design with proper assumptions and justifications. Map the ER model into a relational model. ii. Create tables, populate with data and construct queries (advanced) in SQL to extract information from the car insurance company's database.	**Week7**
8.	i. Implement a PL/SQL block that will accept student id number from the user, and check is student attendance is less than 80% then display message that student cannot appear in exam. [Table: STUDENT (STUD_ID, primary key, STUD_NAME, STUD_ATT)]. ii. Implement a PL/SQL code block that will accept an account number from the user. Check if the user's balance is less than the minimum balance, only then deduct Rs.100 from the balance. The process is fired on the ACCT_MSTR table. [Table: ACCT_MSTR (ACCT_NO, ACCT_HOLDR_NAME, CURBAL]. iii. Implement a PL/SQL code block to calculate the area of a circle for a value of radius varying from 3 to 7. Store the radius and the corresponding values of calculated area in an empty table named AREAS, consisting of two columns Radius and Area. [Table: AREAS (RADIUS, AREA)]. iv. Implement a PL/SQL procedure that takes weight of an apple box as input from the user. ● If the weight is >= 10 kg, rate =Rs. 5/kg. ● If the weight is < 10 kg, rate = Rs. 7/kg. Calculate the cost of the apple box. Display the output on the screen. v. Implement a PL/SQL procedure to calculate the difference between highest salaried and lowest salaried employee. Store the information in a table. vi. Implement a PL/SQL block using cursor that will display the name, department and the salary of the first 3 employees getting lowest salary. [Table: Employee (ename, dept, salary)] vii. Implement a PL/SQL cursor that will update salary of all employees, such that, it allows an increment of 20% if the salary is less than 2000 otherwise	**Week8**

Exp. No.	List of Experiments	Week No.
	increment of Rs.1000. It should print old and new salary for all employees. [Table: Employee (ename, dept, salary)]	
9.	Consider the following relations and Draw the ER, EER Diagram, Relational Model and write the SQL statement for the following queries: Create the tables and insert 5 sets of records into each. employee (personname, street, city) works (personname, companyname, salary) company (companyname, city) manages (personname, managername) a) Find the names of all employees who work for Axis Bank. b) Find the names and cities of residence of all employees who work for Axis Bank. c) Find the names, street addresses, and cities of residence of all employees who work for Axis Bank and earn more than Rs.30000 per annum. d) Find all employees who live in the same city as the company for which they work is located. e) Find all employees who live in the same city and on the same street as their managers. f) Find all employees in the database who do not work for Axis Bank. g) Find all employees who earn more than every employee of Axis Bank. h) Assume that the companies may be located in several cities. Find all companies located in every city in which Axis Bank is located. i) Find all employees who earn more than the average salary of all employees of their company. j) Find the company that has the most employees. k) Find the company that has the smallest payroll.	**Week9**

Exp. No.	List of Experiments	Week No.
	l) Find those companies whose employees earn a higher salary, on average, than the average salary at Axis Bank. m) Modify the database so that ABC now lives in Kolkata. n) Give all employees of Axis Bank a 10 percent raise. o) Give all managers in the database a 10 percent raise. P) Give all managers in the database a 10 percent raise, unless the salary would be greater than Rs.300000.In such cases, give only a 3 percent raise. q) Delete all tuples in the works relation for employees of Axis Bank.	
10.	Consider the following tables: MATCH (match_id, team1, team2, ground, mdate, winner) PLAYER (p_id, lname, fname, country, yborn, bplace, ftest) BATTING (match_id, p_id, mts, order, out_type, fow, nruns, nballs, fours, sixes) BOWLING (match_id, p_id, novers, maidens, nruns, nwickets) 1. Draw the appropriate ER, EER and Relational model for the given data. 2. Write SQL expressions for the following: i) Find match ids of those matches in which player 27001 bats and makes more runs than he made at every match he played at Sydney. ii) Find player ids of players who have scored more than 30 in every ODI match that they have batted. iii) Find the ids of players that had a higher average score than the average score for all players when they played in Sri Lanka.	**Week10**
11.	A record company wishes to use a computer database to help with its operations regarding its performers, recordings and song catalogue. A requirements analysis has elicited the following information: • Songs have a unique song number, a non-unique title and a composition date. • A song can be written by a number of composers; the composer's full name is required. • Songs are recorded by recording artists (bands or solo performers).	**Week11**

Exp. No.	List of Experiments	Week No.
	• A song is recorded as a track of a CD. A CD has many songs on it, called tracks. CDs have a unique record catalogue number, • A title and must have a producer (the full name of the producer is required). • Each track must have the recording date and the track number of the CD. • A song can appear on many (or no) CDs, and be recorded by many different recording artists. The same recording artist might re-record the same song on different CDs. • A CD must have only 1 recording artist appearing on it. • CDs can be released a number of times, and each time the release date and associated number of sales is required. 1. Use this information to design an appropriate ER and relational model. 2. Compile DDL and DML commands on the database created. SQL:- i>Update number of recorded album to 4 for those artist who has recorded only 3. ii>Find all artists who have recorded at least two albums. iii>Find all writers who have only written one song. 3. PL/SQL i>Write Procedure to insert a new Contract into the Contract relation.	
12.	1> Create the following tables. Hotel (Hotel_No, Name, Address) Room (Room_No, Hotel_No, Type, Price) Booking (Hotel_No, Guest_No, Date_From, Date_To, Room_No) Guest (Guest_No, Name, Address) A. Populate the tables and answer the following query using SQL. 1. List the names and addresses of all guests in London, alphabetically ordered by name. 2. List all double or family rooms with a price below £40.00 per night, in ascending order of price. 3. List the bookings for which no date_to has been specified. 4. How many hotels are there?	**Week12**

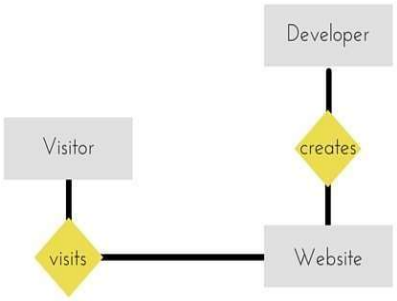
Exp. No.	List of Experiments	Week No.
	5. What is the average price of a room? 6. What is the total revenue per night from all double rooms? 7. How many different guests have made bookings for August? 8. List the details of all rooms at the Grosvenor Hotel, including the name of the guest staying in the room, if the room is occupied. 9. What is the total income from bookings for the Grosvenor Hotel today? 10. List the rooms that are currently unoccupied at the Grosvenor Hotel. B. Design an ER Model for an application where hotels are booked by guests wanting to go on a holiday in India or abroad. Your design should meet all requirements. Map into a relational model.	
	2> Consider the schema for Company Database: EMPLOYEE (SSN, Name, Address, Sex, Salary, SuperSSN, DNo) DEPARTMENT (DNo, DName, MgrSSN, MgrStartDate) DLOCATION (DNo,DLoc) PROJECT (PNo, PName, PLocation, DNo) WORKS_ON (SSN, PNo, Hours) A. Write SQL queries to …… 1. Make a list of all project numbers for projects that involve an employee whose last name is 'Scott', either as a worker or as a manager of the department that controls the project. 2. Show the resulting salaries if every employee working on the 'IoT' project is given a 10 percent raise. 3. Find the sum of the salaries of all employees of the 'Accounts' department, as well as the maximum salary, the minimum salary, and the average salary in this department.	

Exp. No.	List of Experiments	Week No.
	4. Retrieve the name of each employee who works on all the projects controlled by department number 5 (use NOT EXISTS operator). 5. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than Rs. 6,00,000. B. Write a program in PL/SQL to create a procedure to displays the GCD of nos. C. Write a program in PL/SQL to create a cursor displays the name and salary of each employee in the EMPLOYEES table whose salary is less than that specified by a passed-in parameter value.	

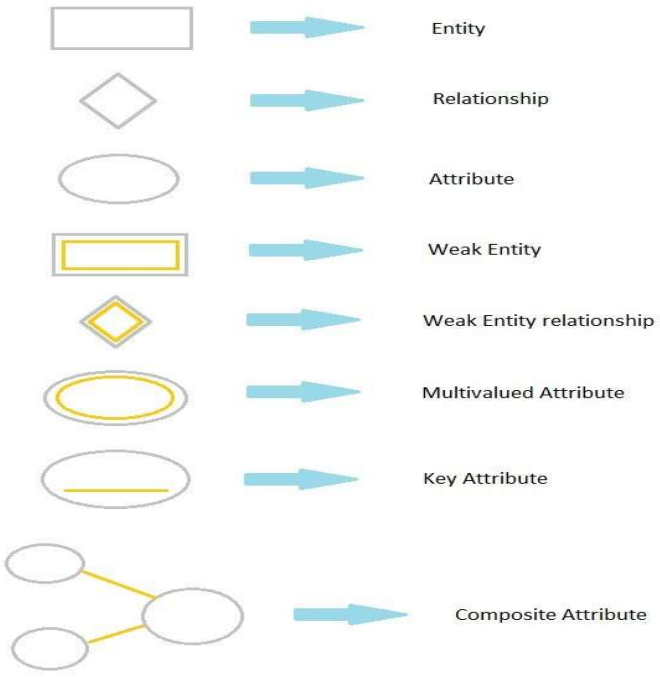
E-R Diagram

AIM: To study about the Entity, Attribute, Relationship, Generalization, Specialization, Aggregation for Entity-Relationship Diagram.

ER-Diagram is a visual representation of data that describes how data is related to each other.



Symbols and Notations



Components of E-R Diagram

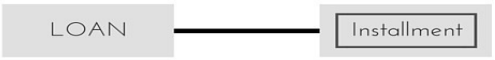
The E-R diagram has three main components.

1) **Entity**

An **Entity** can be any object, place, person or class. In E-R Diagram, an **entity** is represented using rectangles. Consider an example of an Organisation. Employee, Manager, Department, Product and many more can be taken as entities from an Organisation.

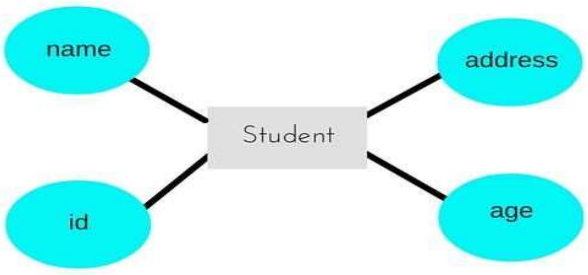
Weak Entity

Weak entity is an entity that depends on another entity. Weak entity doen't have key attribute of their own. Double rectangle represents weak entity.



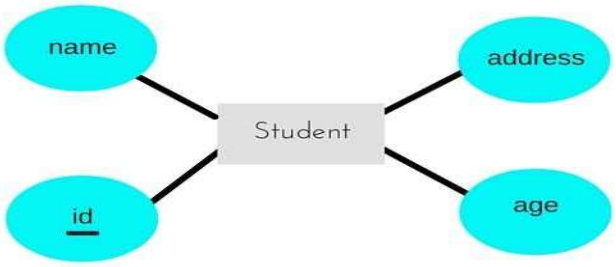
2) **Attribute**

An **Attribute** describes a property or characterstic of an entity. For example, Name, Age, Address etc can be attributes of a Student. An attribute is represented using eclipse.

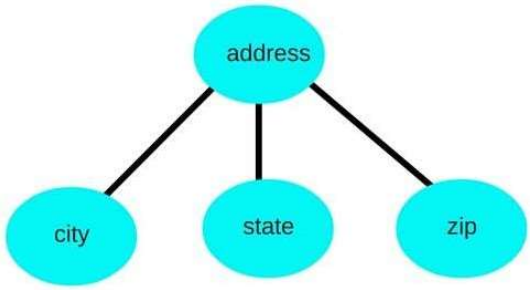


Key Attribute

Key attribute represents the main characterstic of an Entity. It is used to represent Primary key. Ellipse with underlying lines represent Key Attribute.



An attribute can also have their own attributes. These attributes are known as **Composite** attribute.



3) Relationship

A Relationship describes relations between **entities**. Relationship is represented using diamonds.

There are three types of relationship that exist between Entities.

- Binary Relationship
- Unary Relationship
- Recursive Relationship
- Ternary Relationship

Binary Relationship

Binary Relationship means relation between two Entities. This is further divided into three types.

1. **One to One :** This type of relationship is rarely seen in real world.

   The above example describes that one student can enroll only for one course and a course will also have only one Student. This is not what you will usually see in relationship.

2. **One to Many :** It reflects business rule that one entity is associated with many number of same entity. The example for this relation might sound a little weird, but this menas that one student can enroll to many courses, but one course will have one Student.

   The arrows in the diagram describes that one student can enroll for only one course.

3. **Many to One :** It reflects business rule that many entities can be associated with just one entity. For example, Student enrolls for only one Course but a Course can have many Students.

4. **Many to Many :**

   The above diagram represents that many students can enroll for more than one courses.

Unary Relationship

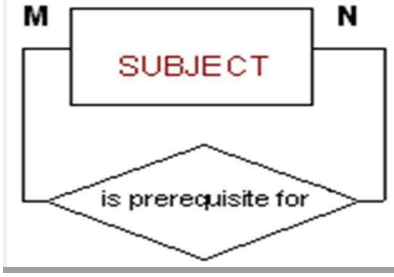
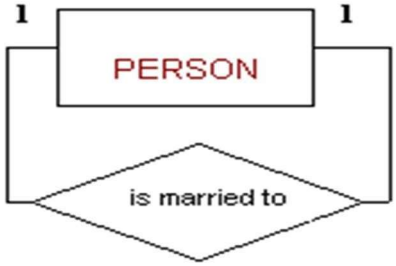
A unary relationship is when both participants in the relationship are the same entity.

For
Example:
Subjects may be prerequisites for other subjects.

Recursive Relationship

Recursive relationships occur within unary relationships. When an Entity is related with itself it is known as *Recursive Relationship*. The relationship may be one to one, one to many or many to many. That is the cardinality of the relationship is unary. The connectivity may be **1:1**, **1:M**, or **M:N**.

For example:

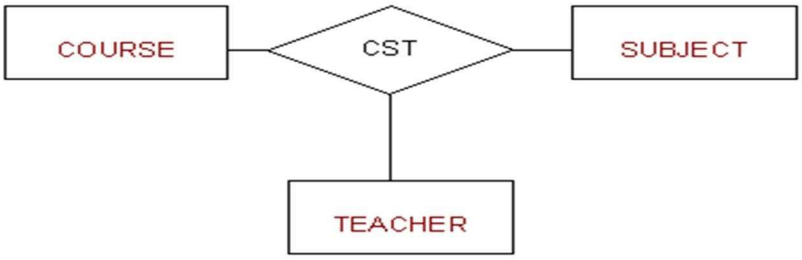
M:N unary relationship: A Subject may have many other Subjects as prerequisites and each Subject may be a prerequisite to many other Subjects	**1:M unary relationship:** An Employee may manage many Employees, but an Employee is managed by only one Employee.	**1:1 unary relationship:** A Person may be married to only one Person.
		

Ternary Relationship

A ternary relationship is when three entities participate in the relationship. Relationship of degree three is called Ternary relationship.

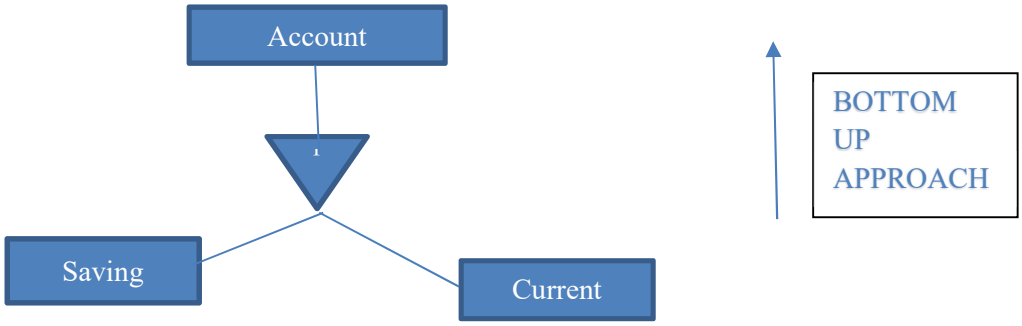
For Example:

The University might need to record which teachers taught which subjects in which courses.



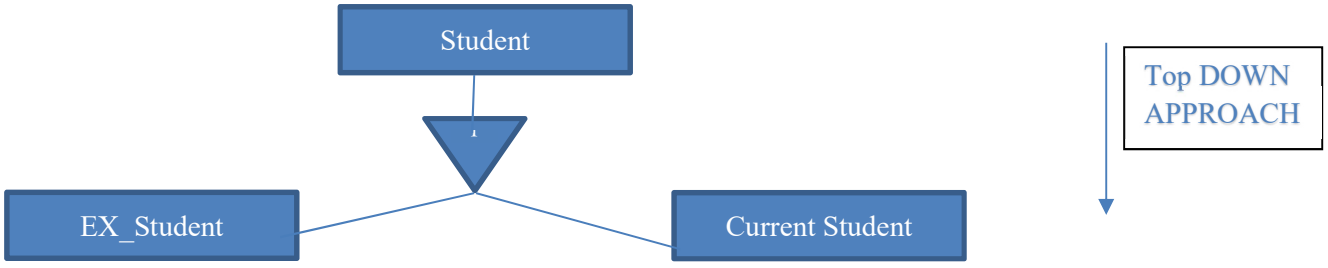
Generalization

Generalization is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entity to make further higher level entity.



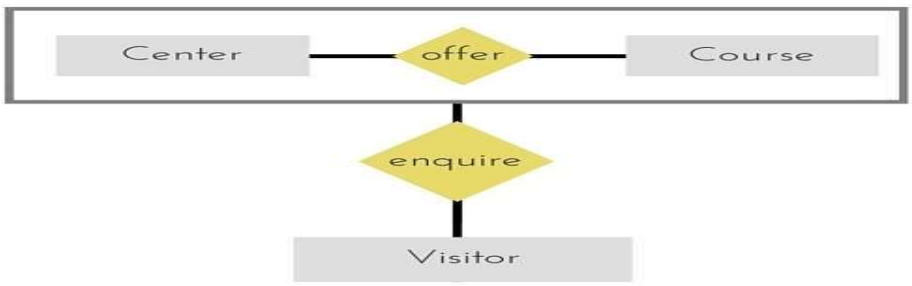
Specialization

Specialization is opposite to Generalization. It is a top-down approach in which one higher level entity can be broken down into two lower level entity. In specialization, some higher level entities may not have lower-level entity sets at all.



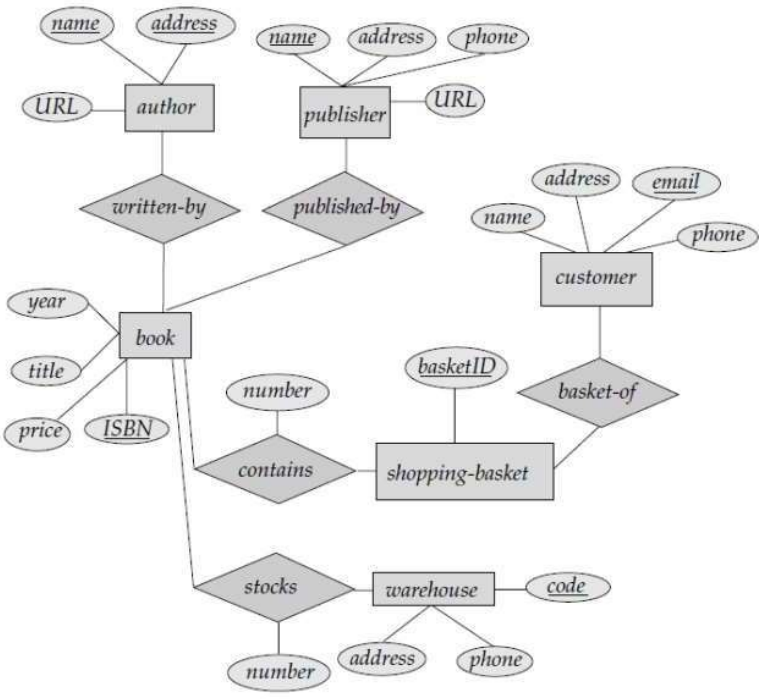
Aggregration

Aggregration is a process when relation between two entity is treated as a single entity. Here the relation between Center and Course, is acting as an Entity in relation with Visitor.



Examples::

1> Draw the E-R diagram which models an online bookstore.
Answer :



ER Diagram for Online BookStore

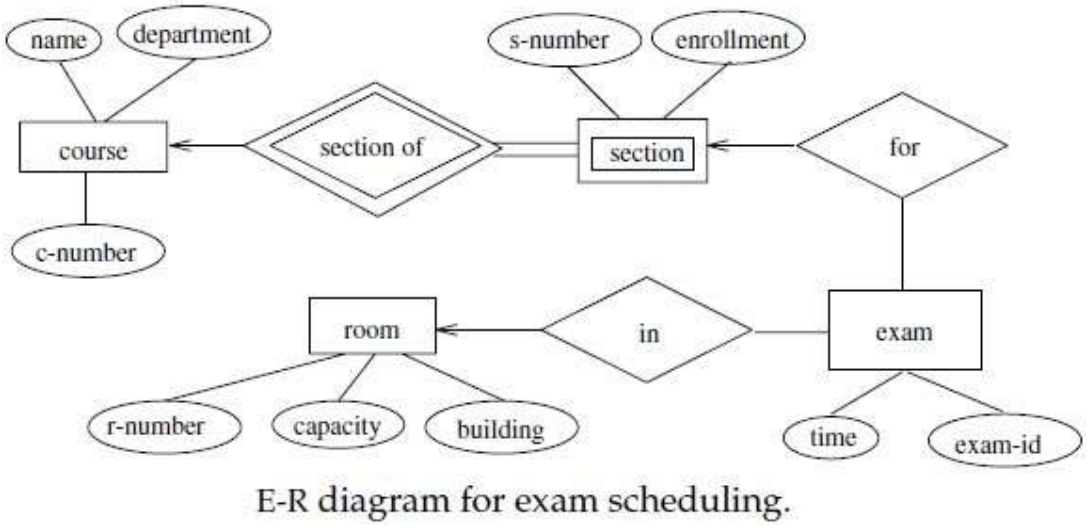
- Consider a university database for the scheduling of classrooms for -final exams. This database could be modeled as the single entity set exam, with attributes course-name,

section-number, room-number, and time. Alternatively, one or more additional entity sets could be defined, along with relationship sets to replace some of the attributes of the exam entity set, as

- course with attributes name, department, and c-number
- section with attributes s-number and enrollment, and dependent as a weak entity set on course
- room with attributes r-number, capacity, and building

Show an E-R diagram illustrating the use of all three additional entity sets listed.

Answer :

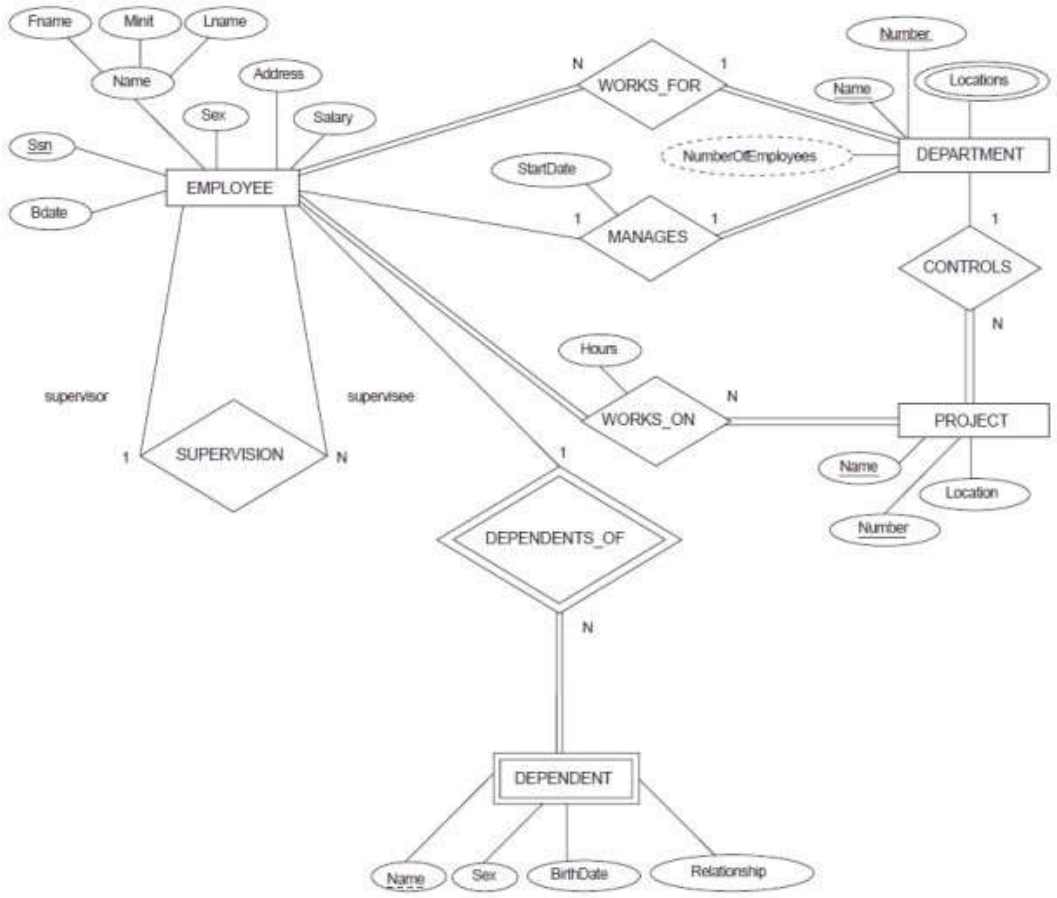


E-R diagram for exam scheduling.

2> Construct an ER Diagram for Company having following details :

- Company organized into DEPARTMENT. Each department has unique name and a particular employee who manages the department. Start date for the manager is recorded. Department may have several locations.
- A department controls a number of PROJECT. Projects have a unique name, number and a single location.
- Company's EMPLOYEE name, ssno, address, salary, sex and birth date are recorded. An employee is assigned to one department, but may work for several projects (not necessarily controlled by her dept). Number of hours/week an employee works on each project is recorded; The immediate supervisor for the employee.
- Employee's DEPENDENT are tracked for health insurance purposes (dependent name, birthdate, relationship to employee).

Answer :



➢ ER-to-Relational Mapping Algorithm

Step 1: Mapping of Regular Entity Types

Step 2: Mapping of Weak Entity Types

Step 3: Mapping of Binary 1:1 Relation Types

Step 4: Mapping of Binary 1:N Relationship Types.

Step 5: Mapping of Binary M:N Relationship Types.

Step 6: Mapping of Multivalued attributes.

Step 7: Mapping of N-ary Relationship Types.

Mapping EER Model Constructs to Relations

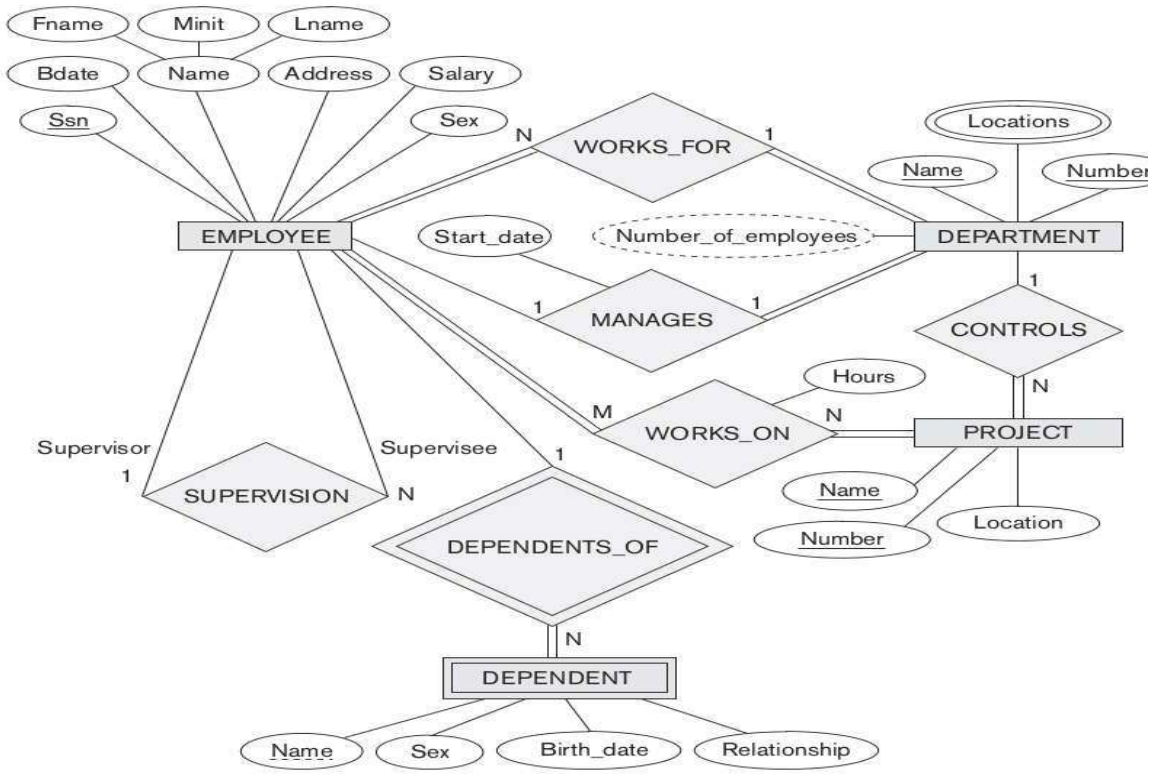
Step 8: Options for Mapping Specialization or Generalization.

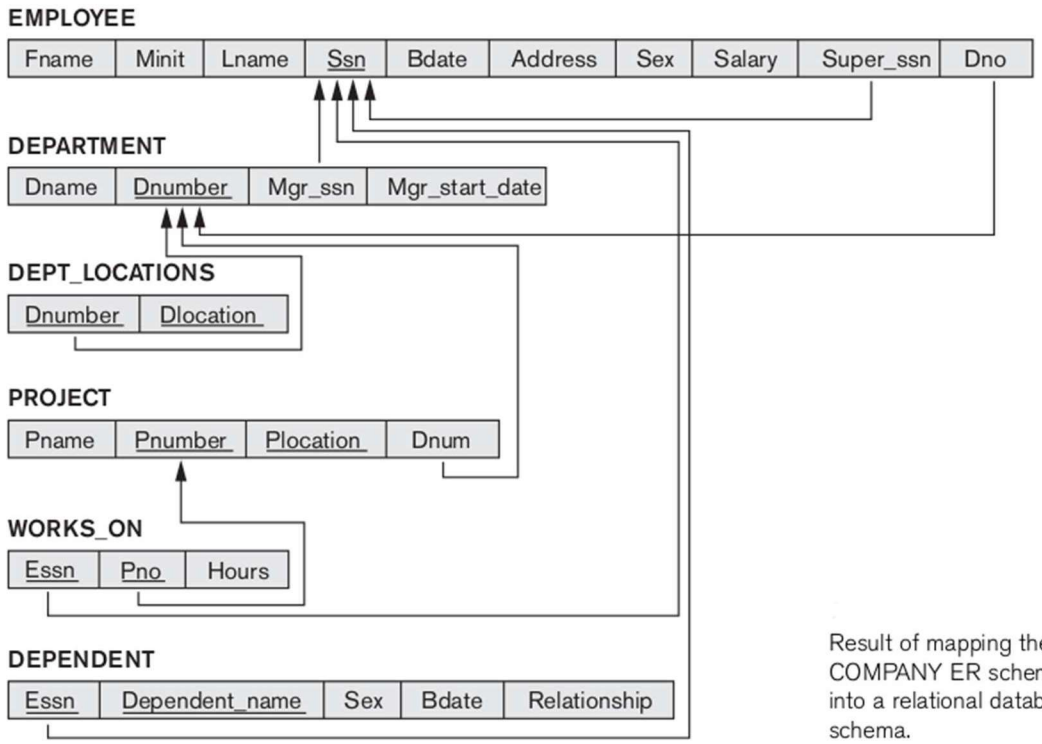
Step 9: Mapping of Union Types (Categories).

- **ER-to-Relational Mapping Algorithm**

- STEP 1: For each regular (strong) entity type E in the ER schema, create a relation R that includes all the simple attributes of E. Include only the simple component attributes of a composite attribute. Choose one of the key attributes of E as primary key for R. If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R.

NOTE: The foreign key and relationship attributes, if any, are not included yet at this step.





Result of mapping the COMPANY ER schema into a relational database schema.

- STEP 2: For each weak entity type W in the ER schema with owner entity type E, create a relation R, and include all simple attributes (or simple components of composite attributes) of W as attributes of R. In addition, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s); this takes care of the identifying relationship type of W. The primary key of R is the combination of the primary key(s) of the owner(s) and the partial key of the weak entity type W, if any.

Example: The DEPENDENT relation

NOTE: It is common to choose the *propagate* (CASCADE) option for the referential triggered action on the foreign key in the relation corresponding to the weak entity type.

- STEP 3: For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. Choose one of the relations—S, say—and include as foreign key in S the primary key of T. Include all the simple attributes (or simple components of composite attributes) of the 1:1 relationship type R as attributes of S.

NOTE: It is better to choose an entity type with total participation in R in the role of S. (**WHY?)**

Note: When both participations are total, an alternative mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation. (**WHY?)**

Example: The MANAGES relationship

- STEP 4: For each regular binary 1:N relationship type R, identify the relation S that represents the participating entity type at the N-side of the relationship type. Include as foreign key in S the primary key of the relation T that represents the other entity type participating in R. (**WHY?)**

Examples: WORKS_FOR, CONTROLS, and SUPERVISION

- STEP 5: For each binary M:N relationship type R, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types; their combination will form the primary key of S.

  Note: We cannot represent an M:N relationship type by a single foreign key attribute in one of the participating relations—as we did for 1:1 or 1:N relationship types . (**Why not?)**

  Example: WORKS_ON

NOTE: The *propagate* (CASCADE) option for the referential triggered action should be specified on the foreign keys in the relation corresponding to the relationship R, since each relationship instance has an existence dependency on each of the entities it relates. This can be used for both ON UPDATE and ON DELETE.

- STEP 6: For each multivalued attribute A , create a new relation R. This relation R will include an attribute corresponding to A, plus the primary key attribute K—as a foreign key in R—of the relation that represents the entity type or relationship type that has A as an attribute. The primary key of R is the combination of A and K. If the multivalued attribute is composite, we include its simple components.

Example: a new relation DEPT_LOCATIONS

NOTE: The propagate (CASCADE) option for the referential triggered action should be specified on the foreign key in the relation corresponding to the multivalued attribute for both ON UPDATE and ON DELETE.

- STEP 7: For each n-ary relationship type R, where n > 2, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity types.

- **Summary of the Mapping**

Correspondence between ER and Relational Models

ER Model	Relational Model
Entity type	"Entity" relation
1:1 or 1:N relationship type	Foreign key (or "relationship" relation)
M:N relationship type	"Relationship" relation and two foreign keys
n-ary relationship type	"Relationship" relation and n foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

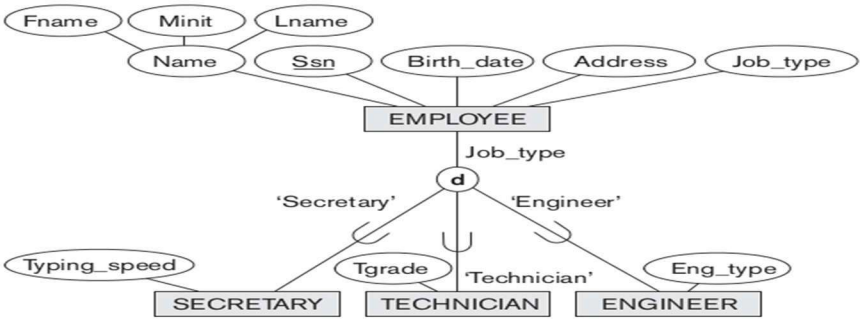
➢ From EER to Relational

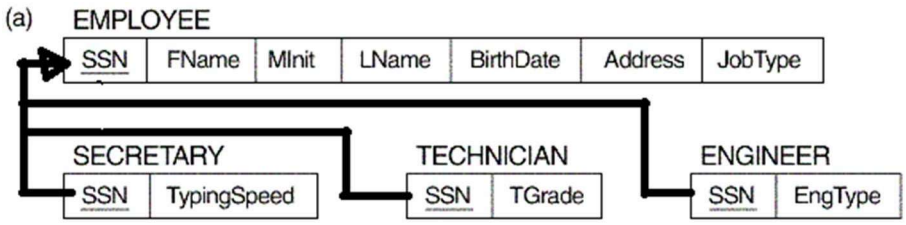
- STEP 8: Convert each specialization with m subclasses {S1, S2, . . ., Sm} and (generalized) superclass C, where the attributes of C are {k, a1, . . ., an} and k is the (primary) key, into relation schemas using one of the four following options:

  Option 8A: Create a relation L for C with attributes Attrs(L) = {k, a1, . . ., an} and PK(L) = k. Create a relation Li for each subclass Si, 1 1 i 1 m, with the attributes Attrs(Li) = {k}D {attributes of Si} and PK(Li) = k.

Example:

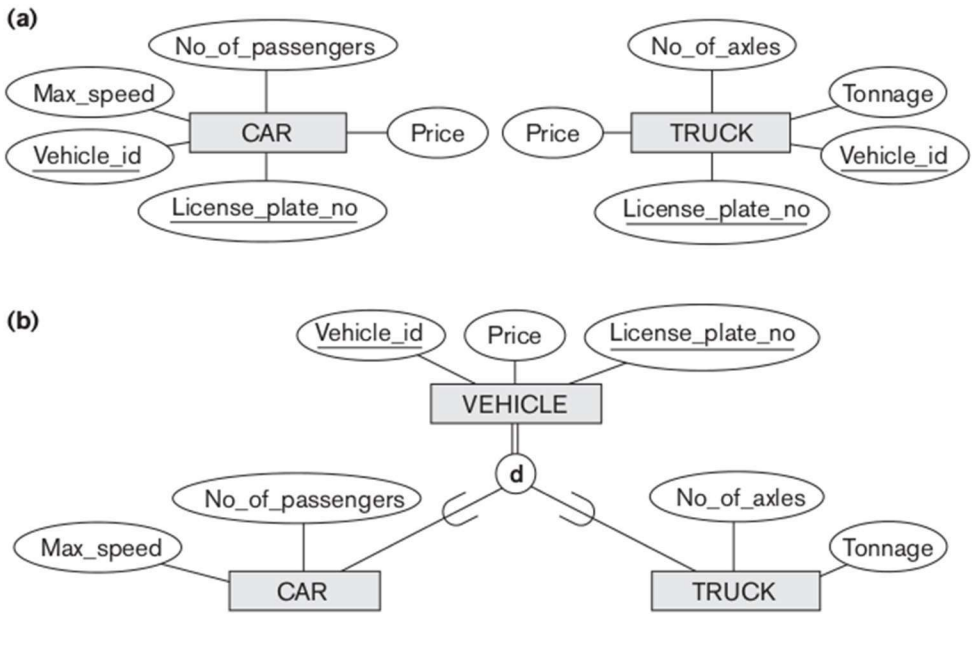
EER diagram notation for an attribute-defined specialization on Job_type.



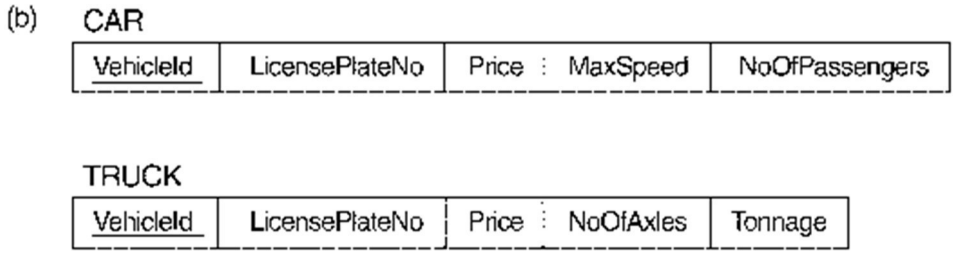


Option 8B: Create a relation Li for each subclass Si, 1 1 i 1 m, with the attributes Attrs(Li) = {attributes of Si}D {k, a1, . . ., an} and PK(Li) = k.

Example:

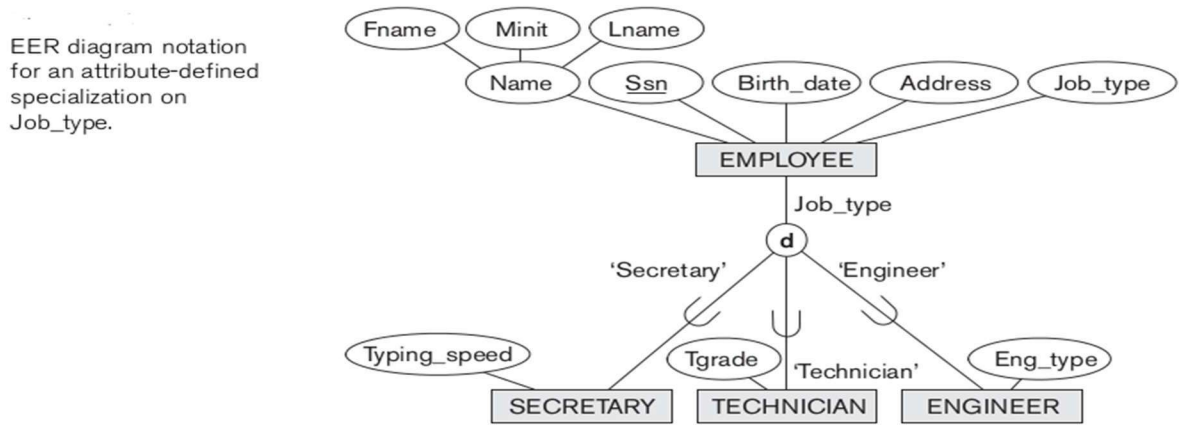


Generalization. (a) Two entity types, CAR and TRUCK. (b) Generalizing CAR and TRUCK into the superclass VEHICLE.



Option 8C: Create a single relation L with attributes Attrs(L) = {k, a1, . . ., an} D {attributes of S1} D . . . D {attributes of Sm} D {t} and PK(L) = k. This option has the potential for generating a large number of null values.

Example:

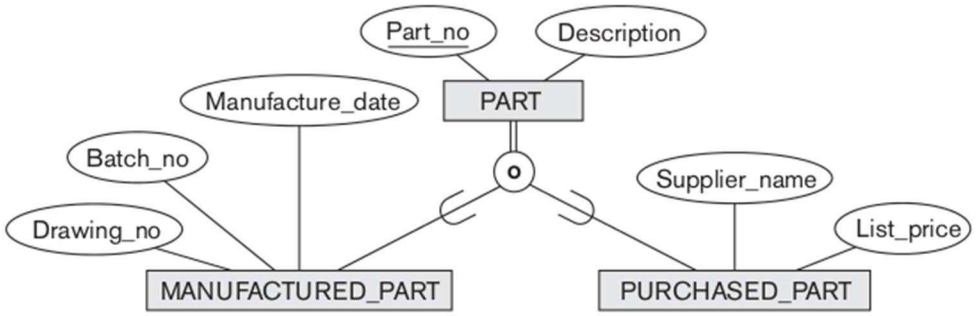


EER diagram notation for an attribute-defined specialization on Job_type.

(c) EMPLOYEE

SSN	FName	MInit	LName	BirthDate	Address	JobType	TypingSpeed	TGrade	EngType

Option 8D: Create a single relation schema L with attributes Attrs(L) = {k, a1, . . ., an} D {attributes of S1} D . . . D {attributes of Sm} D {t1, t2, . . ., tm} and PK(L) = k.
Example:



EER diagram notation for an overlapping (nondisjoint) specialization.

(d) PART

PartNo	Description	MFlag	DrawingNo	ManufactureDate	BatchNo	PFlag	SupplierName	ListPrice

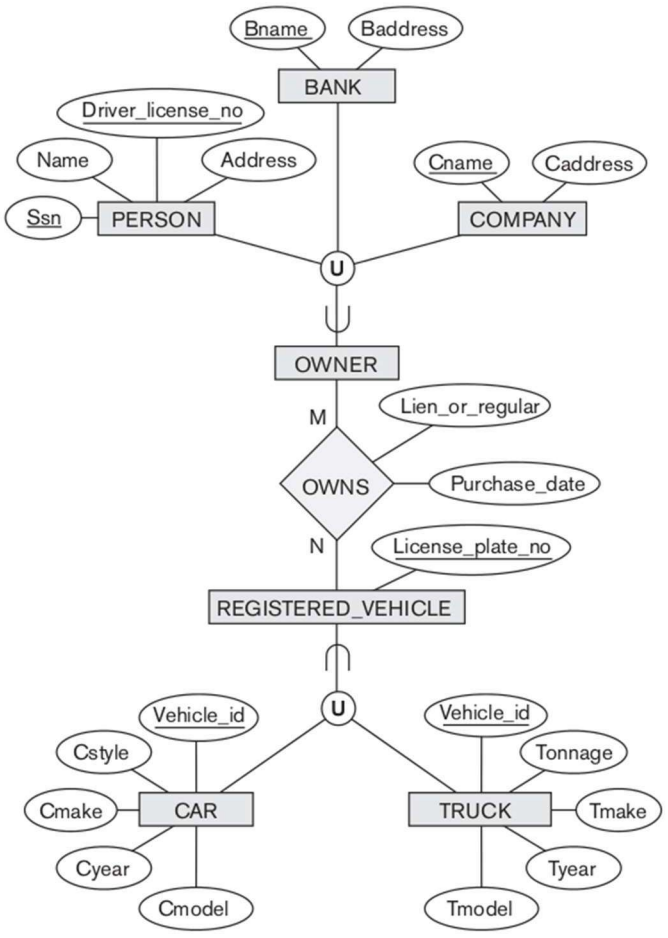
- **Mapping of *Categories* to Relations**

For mapping a category whose defining superclasses have different keys, it is customary to specify a new key attribute, called a *surrogate key*, when creating a relation to correspond to the category.

Union Type Category

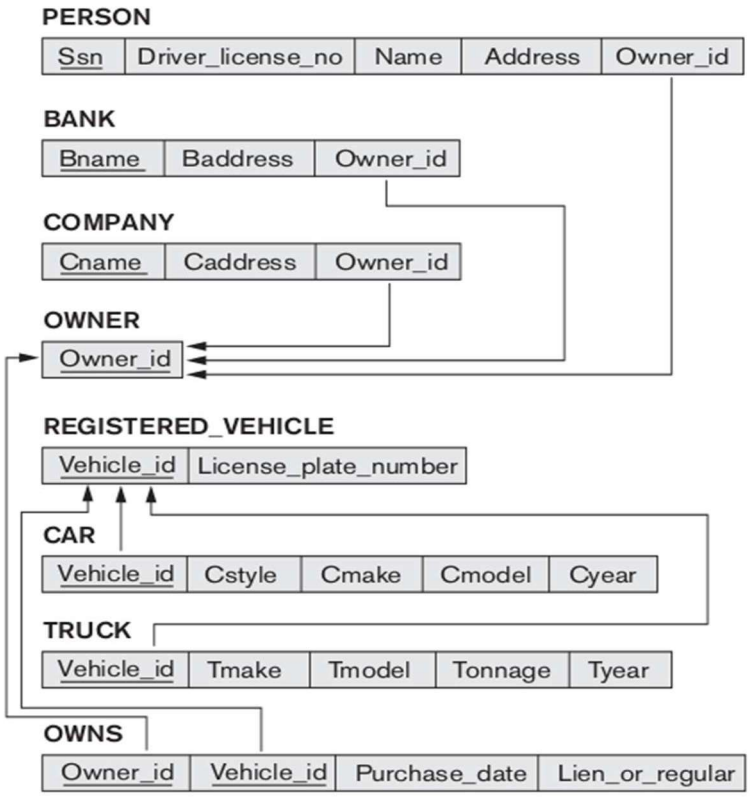
Step 9: Union Types

Let $C_1, C_2, \ldots, C_m$ be the entity types participating in the union and S be the union type. Create a relation for S. If the primary keys of the C_i relations differ, create a *surrogate key* k_s so that PK(S)=k_s, and also add k_s to each Attr(C_i) as a foreign key into S. If all the C_is have the same primary key type, use that as PK(S) instead.



Two categories (union types): OWNER and REGISTERED_VEHICLE.

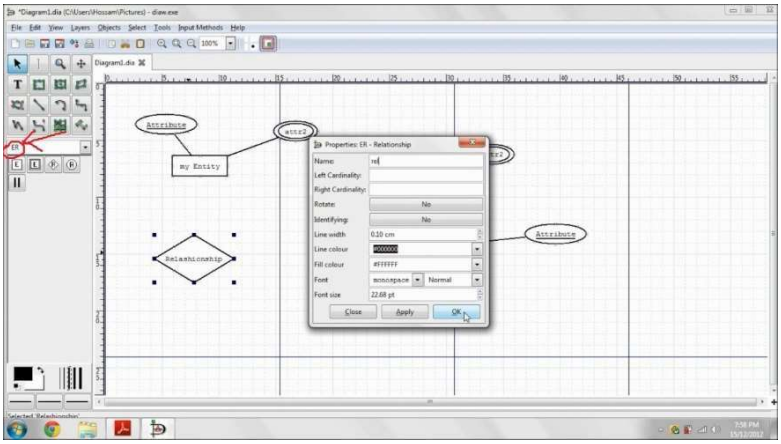
Mapping the EER categories (union types) :



To Draw the ER diagram use Dia software version 0.97.2.

Step 1: Write Dia in run window or select Dia from all programs.

Step 2:



Now select ER from drop down list and draw the ER and EER diagram using the given icons.

Here the 1st icon is use for Entity 

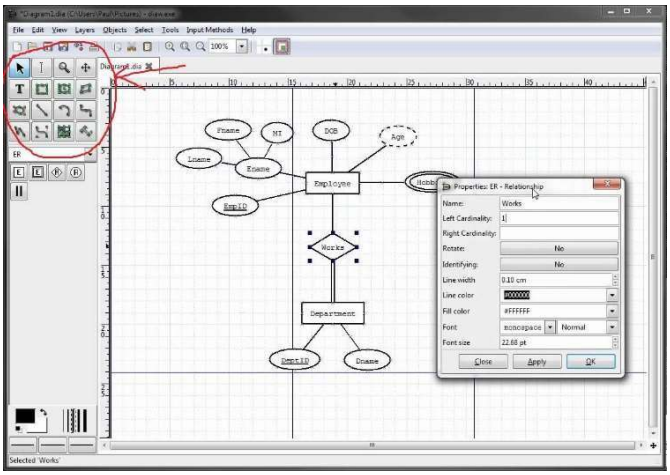
The 2nd icon is used for Weak Entity 

The 3rd icon is used for Relationship 

The 4th icon is used for Attributes 

And 5th icon is used for Connecting Lines 

But rest of this icons you also can use the selected icons if necessary



Now do the assignment below:

- **Assignment.1. ER Model to Relational Model**

  Design an ER diagram for a Motor Vehicle Branch that administers driving tests and issues driver's licenses. Analyze the requirements by identifying the entities, attributes, relationships, keys, constraints etc. Apply extended entity-relationship features to the design. Defend your design with proper assumptions and justifications. Map the ER model into a relational model.

- **Assignment.2. ER Model to Relational Model**

  Design an ER diagram for an application that models soccer teams, the games they play, and the players in each team. Analyze the requirements by identifying the entities, attributes, relationships, keys, constraints etc. Apply extended entity-relationship features to the design. Defend your design with proper assumptions and justifications. Map the ER model into a relational model.

- **Assignment.3. ER Model to Relational Model**

  Design an ER diagram for an application that models an educational institute having several departments, faculty, students, projects, student hostels etc. Analyze the requirements by identifying the entities, attributes, relationships, keys, constraints etc. Apply extended entity-relationship features to the design. Defend your design with proper assumptions and justifications. Map the ER model into a relational model.

SQL

AIM: To create a DDL commands to perform creation of table, alter, rename, drop, truncate and to study the various DML commands and implement them on the database.

SQL (Structured Query Language) is a database computer language designed for the retrieval and management of data in relational database management systems (RDBMS), database schema creation and modification, and database object access control management.SQL is a standard supported by all the popular relational database management systems in the market place. The basis data structure in RDBMS is a table.

Querying can be used to: To retrieve existing data from database.

- Get all data from the table
- Get selected columns from the table.
- Get selected rows from the table.
- Get selected columns of selected rows from the table.
- Get computed columns using char, number, data functions, general functions, and aggregating functions.
- Get data in multiple rows grouped on an aggregating function applied on one or more columns.
- Select specific aggregating data on multiple rows using having clause. Apply set operations like Union and Intersection on data sets of the same cardinality and type.
- Get data from multiple tables using Cartesian product, equality join, un-equal join, and outer join.
- Create views on physical data.

Various Data Types :

1. Character Datatypes:

- Char – fixed length character string that can varies between 1-2000 bytes
- Varchar / Varchar2 – variable length character string, size ranges from 1-4000 bytes.it saves the disk space(only length of the entered value will be assigned as the size of column)
- Long - variable length character string, maximum size is 2 GB

2. Number Datatypes :

Can store +ve,-ve,zero,fixed point,floating point with 38 precission.

- Number – {p=38,s=0}
- Number(p) - fixed point

- Number(p,s) –floating point (p=1 to 38,s= -84 to 127)

3. Date Datatype:
Used to store date and time in the table.

- DB uses its own format of storing in fixed length of 7 bytes for century, date, month year, hour, minutes, seconds
- Default data type is "dd-mon-yy"

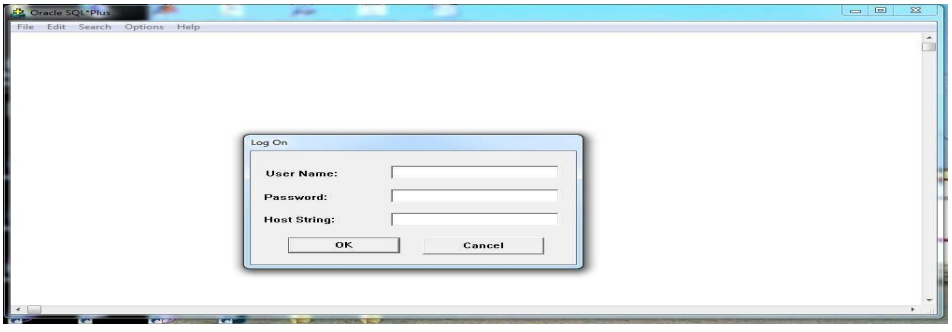
4. Raw Datatype: used to store byte oriented data like binary data and byte string.

5. Other :

- CLOB – stores character object with single byte character.
- BLOB – stores large binary objects such as graphics,video,sounds.
- BFILE – stores file pointers to the LOB's.

✓ How to write the SQL code using Oracle 10g.

Select SQL PLUS icon.
You will be prompt to login. Now write this,
User name is : cse6th<your university roll no>
Password is same as user name
Host string is orcl



IMPLEMENTATION OF DDL COMMANDS

1. Data Definition Language (DDL) commands in RDBMS.

It is used to communicate with database. DDL is used to:

- Create an object
- Alter the structure of an object
- To drop the object created.

The commands used are:

- Create
- Alter
- Drop
- Truncate

CREATE TABLE

The Create Table Command: - it defines each column of the table uniquely. Each has minimum of three attributes, a name , data type and size.

Syntax:

CREATE TABLE <TABLE NAME> (<COL1> <DATATYPE>(<SIZE>),<COL2> <DATATYPE><SIZE>));

Example:

1. CREATE TABLE EMP(EMPNO NUMBER(4) PRIMARY KEY, ENAME CHAR(10));

2. CREATE TABLE PROG20 (PNAME VARCHAR2(20) NOT NULL, DOJ DATE NOT NULL,
DOB DATE NOT NULL, SEX VARCHAR(1) NOT NULL, PROF1 VARCHAR(20),
PROF2 VARCHAR(20),SALARY NUMBER(7,2) NOT NULL);

3. CREATE TABLE EMP (EMPNO NUMBER(5), ENAME VARCHAR(15),
JOB CHAR(10) CONSTRAINT UNIK1 UNIQUE,
DEPTNO NUMBER(3) CONSTRAINT FKEY2 REFERENCES DEPT(DEPTNO));

Rules:

1. Oracle reserved words cannot be used.
3. Underscore, numerals, letters are allowed but not blank space.
3. Maximum length for the table name is 30 characters.
4. 2 different tables should not have same name.
5. We should specify a unique column name.
6. We should specify proper data type along with width.
7. We can include “not null” condition when needed. By default it is ‘null’.

ALTER TABLE

Alter command is used to:

1. Add a new column.
2. Modify the existing column definition.
3. To include or drop integrity constraint.

Syntax: alter table tablename add/modify (attribute datatype(size));

Example:

1. ALTER TABLE EMP ADD (PHONE_NO CHAR (20));
2. ALTER TABLE EMP MODIFY(PHONE_NO NUMBER (10));
3. ALTER TABLE EMP ADD CONSTRAINT Pkey1 PRIMARY KEY (EmpNo);

Modifying the structure of tables.

a) Add new columns

Syntax:

ALTER TABLE <TABLENAME> ADD(<NEWCOL> <DATATYPE(SIZE), <NEWCOL>DATATYPE(SIZE));

Example:ALTER TABLE EMP ADD(SAL NUMBER(7,2));

Dropping a column from a table.

Syntax: ALTER TABLE <TABLENAME> DROP COLUMN <COL>;

Example:ALTER TABLE EMP DROP COLUMN SAL;

Modifying existing columns.

Syntax: ALTER TABLE <TABLENAME> MODIFY(<COL><NEWDATATYPE>(<NEWSIZE>));

Example:ALTER TABLE EMP MODIFY(ENAME VARCHAR2(15));

RENAMING THE TABLES

Syntax: RENAME <OLDTABLE> TO <NEW TABLE>;

Example: RENAME EMP TO EMP1;

DROP TABLE

It will delete the table structure provided the table should be empty.

Syntax: DROP TABLE <TABLENAME>;

Example: DROP TABLE PROG20; //Here prog20 is table name

TRUNCATE TABLE

If there is no further use of records stored in a table and the structure has to be retained then the records alone can be deleted.

Syntax: TRUNCATE TABLE <TABLE NAME>;

Example: TRUNCATE TABLE CUSTOMER;

DESC

This is used to view the structure of the table.

Syntax: DESC <TABLENAME>;

Example: DESC EMP;

NAME	NULL?	TYPE
EMPNO	NOT NULL	NUMBER(5)
ENAME	NOT NULL	VARCHAR(15)
JOB	NOT NULL	CHAR(10)
DEPTNO	NOT NULL	NUMBER(3)
PHONE_NO		NUMBER (10)

Sample Queries:

Q1. Create a table called EMP with the following structure.

Name	*Type*
EMPNO	*NUMBER(6)*
ENAME	*VARCHAR2(20)*
JOB	*VARCHAR2(10)*
DEPTNO	*NUMBER(3)*
SAL	*NUMBER(7,2)*

Allow NULL for all columns except ename and job.
Solution:
1. Understand create table syntax.
2. Use the create table syntax to create the said tables.
3. Create primary key constraint for each table as understand from logical table structure**.**
Ans: SQL> CREATE TABLE EMP(EMPNO NUMBER(6),ENAME VARCHAR2(20)NOT NULL,JOB VARCHAR2(10) NOT NULL, DEPTNO NUMBER(3),SAL NUMBER(7,2));
Table created.

Q2: Add a column experience to the emp table. Experience numeric null allowed.
Solution:
1. Learn alter table syntax.
2. Define the new column and its data type.
3. Use the alter table syntax.
Ans: SQL> ALTER TABLE EMP ADD(EXPERIENCE NUMBER(2));
Table altered.

Q3: Modify the column width of the job field of emp table.
Solution:
1. Use the alter table syntax.
2. Modify the column width and its data type.
Ans:
SQL> ALTER TABLE EMP MODIFY(JOB VARCHAR2(12));

Table altered.
SQL> ALTER TABLE EMP MODIFY(JOB VARCHAR(13));
Table altered.

Q4: Create dept table with the following structure.

Name	Type
DEPTNO	NUMBER(2)
DNAME	VARCHAR2(10)
LOC	VARCHAR2(10)

Deptno as the primarykey

Solution:
1. Understand create table syntax.
2. Decide the name of the table.
3. Decide the name of each column and its data type.
4. Use the create table syntax to create the said tables.
5. Create primary key constraint for each table as understand from logical table structure.

Ans: SQL> CREATE TABLE DEPT(DEPTNO NUMBER(2) PRIMARY KEY,DNAME VARCHAR2(10), LOCVARCHAR2(10));
Table created.

Q5: create the emp1 table with ename and empno, add constraints to check the empno value while entering (i.e) empno > 100.

Solution:
1. Learn alter table syntax.
2. Define the new constraint [columns name type]
3. Use the alter table syntax for adding constraints.

Ans: SQL> CREATE TABLE EMP1(ENAME VARCHAR2(10),EMPNO NUMBER(6) CONSTRAINT CHECK(EMPNO>100));
Table created.

Q6: drop a column experience to the emp table.

Solution:
1. Learn alter table syntax. Use the alter table syntax to drop the column.
Ans: SQL> ALTER TABLE EMP DROP COLUMN EXPERIENCE;
Table altered.

Q7: Truncate the emp table and drop the dept table

Solution: 1. Learn drop, truncate table syntax.
Ans: SQL> TRUNCATE TABLE EMP;
Table truncated.

IMPLEMENTATION OF DML AND DCL COMMANDS

DML COMMANDS

DML commands are the most frequently used SQL commands and is used to query and manipulate the existing database objects. Some of the commands are :

- Insert
- Select
- Update
- Delete.

INSERT Command

Insert Command This is used to add one or more rows to a table. The values are separated by commas and the data types char and date are enclosed in apostrophes. The values must be entered in the same order as they are defined.

Example:

First create a table named ***STD***

CREATE TABLE STD (SNO NUMBER(5),SNAME VARCHAR2(20), AGE NUMBER(5),
SDOB DATE,SM1 NUMBER(4,2),SM2 NUMBER(4,2),SM3 NUMBER(4,4));

To insert values into the STD table

Syntax:

INSERT INTO STD VALUES(101,"AAA",16,"03-JUL-88",80,90,98);

INSERT INTO STD VALUES(102,"BBB",18,"04-AUG-89",88,98,90);

SELECT Command

Select query is used to retrieve data from a tables. It is the most used SQL query. We can retrieve complete tables, or partial by mentioning conditions using WHERE clause.

Syntax:

SELECT column-name1, column-name2, column-name3, column-nameN from *table-name*;

Example:
Consider the following ***Student table****,*

S_id	S_Name	age	address
101	Adam	15	Noida
102	Alex	18	Delhi
103	Abhi	17	Rohtak
104	Ankit	22	Panipat

SELECT S_ID, S_NAME, AGE FROM STUDENT;

The above query will fetch information of S_ID, S_NAME and AGE column from Student table

S_ID	S_NAME	AGE
101	ADAM	15
102	ALEX	18
103	ABHI	17
104	ANKIT	22

Example to Select all Records from Table

A special character **asterisk** * is used to address all the data(belonging to all columns) in a query.

Select statement uses * character to retrieve all records from a table.

SELECT * FROM STUDENT;

The above query will show all the records of Student table that means it will show complete Student table as result.

S_ID	S_NAME	AGE	ADDRESS
101	ADAM	15	NOIDA
102	ALEX	18	DELHI
103	ABHI	17	ROHTAK
104	ANKIT	22	PANIPAT

Example to Select particular Record based on Condition
SELECT * FROM STUDENT **WHERE** S_NAME = 'ABHI';
103 ABHI 17 ROHTAK

Example to Perform Simple Calculations using Select Query

Conside the following **Employee** table.

EID	NAME	AGE	SALARY
101	ADAM	26	5000
102	RICKY	42	8000
103	ABHI	22	10000
104	ROHAN	35	5000

SELECT EID, NAME, SALARY+3000 FROM EMPLOYEE;

The above command will display a new column in the result, showing 3000 added into existing salaries of the employees.

EID	NAME	SALARY+3000
101	ADAM	8000
102	RICKY	11000
103	ABHI	13000
104	ROHAN	8000

UPDATE command

Update command is used to update a row of a table. A single column may be updated or more than one column could be updated.

Following is its general Syntax,
UPDATE *table-name* set column-name = value *where* **condition**;

Example:
UPDATE STUDENT SET AGE=18 WHERE S_ID=102;

S_ID	S_NAME	AGE
101	ADAM	15
102	ALEX	18
103	CHRIS	14

Example to Update multiple columns
UPDATE STUDENT SET S_NAME='ABHI',AGE=17 WHERE S_ID=103;

The above command will update two columns of a record.

S_ID	S_NAME	AGE
101	ADAM	15
102	ALEX	18
103	ABHI	17

DELETE command
Delete command is used to delete data from a table. Delete command can also be used with condition to delete a particular row. The delete command consists of a from clause followed by an optional where clause.

Following is its general syntax,
DELETE from *table-name*;

Example to Delete all Records from a Table
DELETE FROM STUDENT;
The above command will delete all the records from **Student** table.

Example to Delete a particular Record from a Table
Consider the following **Student** table

S_ID	S_NAME	AGE
101	ADAM	15
102	ALEX	18
103	ABHI	17

DELETE FROM STUDENT WHERE S_ID=103;
The above command will delete the record where s_id is 103 from **Student** table.

S_ID	S_NAME	AGE

101	ADAM	15
102	ALEX	18

Q1: Insert a single record into dept table.
Ans: SQL> INSERT INTO DEPT VALUES (1,'IT','THOLUDUR');

1 row created.

Q2: Insert more than a record into emp table using a single insert command.
Ans: SQL> INSERT INTO EMP VALUES(&EMPNO,'&ENAME','&JOB',&DEPTNO,&SAL);
Enter value for empno: 1
Enter value for ename: Mathi
Enter value for job: AP
Enter value for deptno: 1
Enter value for sal: 10000
old 1: insert into emp values(&empno,'&ename','&job',&deptno,&sal)
new 1: insert into emp values(1,'Mathi','AP',1,10000)

1 row created.

SQL> /
Enter value for empno: 2
Enter value for ename: Arjun
Enter value for job: ASP
Enter value for deptno: 2
Enter value for sal: 12000
old 1: insert into emp values(&empno,'&ename','&job',&deptno,&sal)
new 1: insert into emp values(2,'Arjun','ASP',2,12000)

1 row created.

SQL> /
Enter value for empno: 3
Enter value for ename: Gugan
Enter value for job: ASP
Enter value for deptno: 1
Enter value for sal: 12000
old 1: insert into emp values(&empno,'&ename','&job',&deptno,&sal)
new 1: insert into emp values(3,'Gugan','ASP',1,12000)

1 row created.

Q3: Update the emp table to set the salary of all employees to Rs15000/- who are working as

ASP.

Ans: SQL> SELECT * FROM EMP;

EMPNO	ENAME	JOB	DEPTNO	SAL
1	MATHI	AP	1	10000
2	ARJUN	ASP	2	12000
3	GUGAN	ASP	1	12000

SQL> UPDATE EMP SET SAL=15000 WHERE JOB='ASP';
2 rows updated.

SQL> SELECT * FROM EMP;

EMPNO	ENAME	JOB	DEPTNO	SAL
1	Mathi	AP	1	10000
2	Arjun	ASP	2	15000
3	Gugan	ASP	1	15000

Q4: Create a pseudo table employee with the same structure as the table emp and insert rows into the table using select clauses.

Ans: SQL> CREATE TABLE EMPLOYEE AS SELECT * FROM EMP;

Table created.

SQL> DESC EMPLOYEE;

Name	**Null?**	**Type**
EMPNO	NOT NULL	NUMBER(6)
ENAME	NOT NULL	VARCHAR2(20)
JOB	NOT NULL	VARCHAR2(13)
DEPTNO		NUMBER(3)
SAL		NUMBER(7,2)

Q5: select employee name, job from the emp table
Ans: SQL> SELECT ENAME, JOB FROM EMP;

ENAME	**JOB**
MATHI	AP
ARJUN	ASP

GUGAN	ASP
KARTHIK	PROF
AKALYA	AP
SURESH	LECT

6 rows selected.

Q6: Delete only those who are working as lecturer
Ans: SQL> SELECT * FROM EMP;

EMPNO	ENAME	JOB	DEPTNO	SAL
1	MATHI	AP	1	10000
2	ARJUN	ASP	2	15000
3	GUGAN	ASP	1	15000
4	KARTHIK	PROF	2	30000
5	AKALYA	AP	1	10000
6	SURESH	LECT	1	8000

6 rows selected.
SQL> DELETE FROM EMP WHERE JOB='LECT';
1 row deleted.
SQL> SELECT * FROM EMP;

EMPNO	ENAME	JOB	DEPTNO	SAL
1	Mathi	AP	1	10000
2	Arjun	ASP	2	15000
3	Gugan	ASP	1	15000
4	Karthik	Prof	2	30000
5	Akalya	AP	1	10000

Q7: List the records in the emp table orderby salary in ascending order.
Ans: SQL> SELECT * FROM EMP ORDER BY SAL;

EMPNO	**ENAME**	**JOB**	**DEPTNO**	**SAL**
1	Mathi	AP	1	10000
5	Akalya	AP	1	10000
2	Arjun	ASP	2	15000
3	Gugan	ASP	1	15000
4	Karthik	Prof	2	30000

Q8: List the records in the emp table orderby salary in descending order.
Ans: SQL> SELECT * FROM EMP ORDER BY SAL DESC;

EMPNO	ENAME	JOB	DEPTNO	SAL
4	Karthik	Prof	2	30000
2	Arjun	ASP	2	15000
3	Gugan	ASP	1	15000
1	Mathi	AP	1	10000
5	Akalya	AP	1	10000

Q9: Display only those employees whose deptno is 30.
Solution: Use SELECT FROM WHERE syntax.
Syntax. ANS: SQL> SELECT * FROM EMP WHERE DEPTNO=1;

EMPNO	ENAME	JOB	DEPTNO	SAL
1	Mathi	AP	1	10000
3	Gugan	ASP	1	15000
5	Akalya	AP	1	10000

Q10: Display deptno from the table employee avoiding the duplicated values.
Solution: 1. Use SELECT FROM syntax.
2.Select should include distinct clause for the deptno.
Ans: SQL> SELECT DISTINCT DEPTNO FROM EMP;

DEPTNO
1
2

INTEGRITY CONSTRAINT & WILD CARD CHARACTER

AIM: To study the various Constraint and wild card characters and implement them in the database

INTEGRITY CONSTRAINT

An integrity constraint is a mechanism used by oracle to prevent invalid data entry into the table. It has enforcing the rules for the columns in a table. The types of the integrity constraints are:

a) Domain Integrity
b) Entity Integrity
c) Referential Integrity

a) Domain Integrity

This constraint sets a range and any violations that takes place will prevent the user from performing the manipulation that caused the breach.It includes:

Not Null constraint:

While creating tables, by default the rows can have null value.the enforcement of not null constraint in a table ensure that the table contains values.

Principle of Null values:

- Setting null value is appropriate when the actual value is unknown, or when a value would not be meaningful.
- A null value is not equivalent to a value of zero.
- A null value will always evaluate to null in any expression.
- When a column name is defined as not null, that column becomes a mandatory i.e., the user has to enter data into it.
- Not null Integrity constraint cannot be defined using the alter table command when the table contain rows.

Example:

i. CREATE TABLE STUDENT(S_ID INT NOT NULL, NAME VARCHAR(60), AGE INT);

The above query will declare that the s_id field of Student table will not take NULL value.

ii. CREATE TABLE CUST(CUSTID NUMBER(6) NOT NULL, NAME CHAR(10));
ALTER TABLE CUST MODIFY (NAME NOT NULL);

This command will ensure that the user enters a value for the custid, name columns on the cust table, failing which it returns an error message.

Check Constraint

Check constraint can be defined to allow only a particular range of values. When the manipulation violates this constraint, the record will be rejected. Check condition cannot contain sub queries.

Example:

i. Using CHECK constraint at Table Level

CREATE TABLE STUDENT(S_ID INT NOT NULL **CHECK(S_ID > 0)**,NAME VARCHAR(60) NOT NULL, AGE INT);

The above query will restrict the s_id value to be greater than zero.

ii. Example using CHECK constraint at Column Level

ALTER TABLE STUDENT ADD CHECK(S_ID > 0);

Default constraint

The DEFAULT constraint is used to provide a default value for a column.
The default value will be added to all new records IF no other value is specified.

SQL DEFAULT on CREATE TABLE

The following SQL sets a DEFAULT value for the "CITY" column when the "PERSONS" table is created:

CREATE TABLE PERSONS (ID INT NOT NULL,LASTNAME VARCHAR2(55) NOT NULL,
FIRSTNAME VARCHAR2(55), AGE INT,CITY VARCHAR(55) DEFAULT 'SANDNES');

The DEFAULT constraint can also be used to insert system values, by using functions like GETDATE():

CREATE TABLE Orders (ID int NOT NULL, OrderNumber int NOT NULL,OrderDate date DEFAULT GETDATE());

b) Entity Integrity

Maintains uniqueness in a record. An entity represents a table and each row of a table represents an instance of that entity. To identify each row in a table uniquely we need to use this constraint. There are 2 entity constraints:

Unique key constraint

It is used to ensure that information in the column for each record is unique, as with telephone or drivers license numbers. It prevents the duplication of value with rows of a specified column in a set of column. A column defined with the constraint can allow null value.

If unique key constraint is defined in more than one column i.e., combination of column cannot be specified. Maximum combination of columns that a composite unique key can contain is 16.

Example:

```
CREATE TABLE CUST(CUSTID NUMBER(6) CONSTRAINT UNI UNIQUE, NAME CHAR(10));
ALTER TABLE CUST ADD(CONSTRAINT C  UNIQUE(CUSTID));
```

The above query specifies that ***custid*** field of ***cust*** table will only have unique value

Primary key constraint

A primary key avoids duplication of rows and does not allow null values.can be defined on one or more columns in a table and is used to uniquely identify each row in a table. These values should never be changed and should never be null.

A table should have only one primary key. If a primary key constraint is assigned to more than one column or combination of column is said to be composite primary key, which can contain 16 columns.

Example:

```
CREATE TABLE STUD(REGNO NUMBER(6)  CONSTRAINT PRIMARY KEY, NAME CHAR(20));
```

Example using PRIMARY KEY constraint at Table Level

CREATE TABLE STUDENT (S_ID INT **PRIMARY KEY**, NAME VARCHAR(60) NOT NULL, AGE INT);

The above command will creates a PRIMARY KEY on the s_id.

Example using PRIMARY KEY constraint at Column Level

ALTER TABLE STUDENT ADD PRIMARY KEY (S_ID);

The above command will creates a PRIMARY KEY on the s_id.

Note: Can't be defined using alter command when there is records in the table having null values.

c) Referential Integrity

It enforces relationship between tables. To establish parent-child relationship between 2 tables having a common column definition, we make use of this constraint. To implement this, we should define the column in the parent table as primary key and same column in the child table as foreign key referring to the corresponding parent entry.

Foreign key constraint

A column or combination of column included in the definition of referential integrity, which would refer to a referenced key.

To understand FOREIGN KEY, let's see it using two table.

CUSTOMER_DETAIL TABLE :

C_ID	CUSTOMER_NAME	ADDRESS
101	ADAM	NOIDA
102	ALEX	DELHI
103	STUART	ROHTAK

ORDER_DETAIL TABLE :

ORDER_ID	ORDER_NAME	C_ID
10	ORDER1	101
11	ORDER2	103
12	ORDER3	102

In **CUSTOMER_DETAIL** table, C_ID is the primary key which is set as foreign key in **ORDER_DETAIL** table. The value that is entered in C_ID which is set as foreign key in **ORDER_DETAIL** table must be present in **CUSTOMER_DETAIL** table where it is set as primary key. This prevents invalid data to be inserted into C_ID column of **ORDER_DETAIL** table.

Example using FOREIGN KEY constraint at Table Level

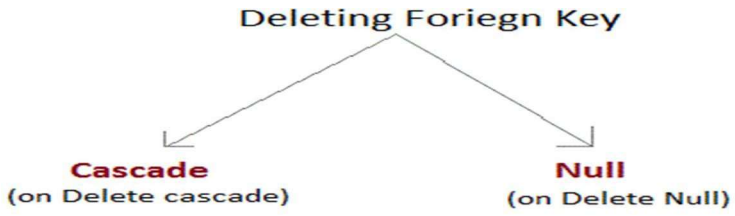
CREATE TABLE ORDER_DETAIL(ORDER_ID INT PRIMARY KEY, ORDER_NAME VARCHAR(60) NOT NULL, *C_ID INT* **FOREIGN KEY** REFERENCES **CUSTOMER_DETAIL**(*C_ID*));
In this query, C_ID in table ORDER_DETAIL is made as foriegn key, which is a reference of C_ID column of CUSTOMER_DETAIL.

Example using FOREIGN KEY constraint at Column Level

ALTER TABLE ORDER_DETAIL ADD **FOREIGN KEY** (C_ID) REFERENCES CUSTOMER_DETAIL(C_ID);

Behaviour of Foriegn Key Column on Delete

There are two ways to maintin the integrity of data in Child table, when a particular record is deleted in main table. When two tables are connected with Foriegn key, and certain data in the main table is deleted, for which record exit in child table too, then we must have some mechanism to save the integrity of data in child table.



- **On Delete Cascade :** This will remove the record from child table, if that value of foriegn key is deleted from the main table.
- **On Delete Null :** This will set all the values in that record of child table as NULL, for which the value of foriegn key is deleted from the main table.
- If we don't use any of the above, then we cannot delete data from the main table for which data in child table exists. We will get an error if we try to do so.
  ERROR : Record in child table exist

Referenced key constraint

It is a unique or primary key upon which is defined on a column belonging to the parent table.

Wildcard Characters

A wildcard character is used to substitute any other character(s) in a string.
Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
There are two wildcards used in conjunction with the LIKE operator:

- % - The percent sign represents zero, one, or multiple characters
- _ - The underscore represents a single character

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends 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 starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

Example

Consider a "Customers" table :

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Using the % Wildcard

The following SQL statement selects all customers with a City starting with "ber":

Example

```
SELECT * FROM Customers
WHERE City LIKE 'ber%';
```

The following SQL statement selects all customers with a City containing the pattern "es":

Example

```
SELECT * FROM Customers
WHERE City LIKE '%es%';
```

Using the _ Wildcard

The following SQL statement selects all customers with a City starting with any character, followed by "erlin":

Example

```
SELECT * FROM Customers
WHERE City LIKE '_erlin';
```

The following SQL statement selects all customers with a City starting with "L", followed by any character, followed by "n", followed by any character, followed by "on":

Example

```
SELECT * FROM Customers
WHERE City LIKE 'L_n_on';
```

Using the [charlist] Wildcard

The following SQL statement selects all customers with a City starting with "b", "s", or "p":

Example

```
SELECT * FROM Customers
WHERE City LIKE '[bsp]%';
```

The following SQL statement selects all customers with a City starting with "a", "b", or "c":

Example

```
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

Using the [!charlist] Wildcard

The two following SQL statements selects all customers with a City NOT starting with "b", "s", or "p":

Example

```
SELECT * FROM Customers
WHERE City LIKE '[!bsp]%';
```

Or:

Example

```
SELECT * FROM Customers
WHERE City NOT LIKE '[bsp]%';
```

Now do the assignment below:

- **Assignment. 4**.
  i. Create tables for Client, Product, and Salesman with the attributes given, implementing DDL commands for specifying prime attributes, non-prime attributes, foreign keys, cardinalities, null values, constraints etc. and the data types. Implement DDL commands for drop, alter on the tables created.
  ii. Implement DML commands like populating the tables with data using insert command and retrieving data using simple queries in SQL. (Application of select, update, delete etc.)

i> Create the following tables:

CLIENT_MASTER

Column Name	Data Type	Size	Default	Attributes
Client_no	Varchar2	6		Primary key / first letter must start with 'C'
Name	Varchar2	20		Not Null
City	Varchar2	15		
Pincode	Number	8		
State	Varchar2	15		
BalDue	Number	10,2		

PRODUCT_MASTER

Column Name	Data Type	Size	Default	Attributes
Product_no	Varchar2	6		Primary key / first letter must start with 'P'
Description	Varchar2	15		Not Null
QTY_ON_Hand	Number	8		Not Null
Sell_Price	Number	8,2		Not Null, can not be 0
Cost_Price	Number	8,2		Not Null, can not be 0

SALESMAN_MASTER

Column Name	Data Type	Size	Default	Attributes
Salesman_no	Varchar2	6		Primary key / first letter must start with 'S'
Salesman_name	Varchar2	20		Not Null
City	Varchar2	20		Not Null
Pincode	Number	8		Not Null
State	Varchar2	20		Not Null
Sal_Amt	Number	8,2		Not Null, can not be 0

SALES_ORDER

Column Name	Data Type	Size	Default	Attributes
Order_no	Varchar2	6		Primary key / first letter must start with 'O'
Client_no	Varchar2	6		Foreign key references Client_no of CLIENT_MASTER table
Order_date	Date			
Salesman_no	Varchar2	6		Foreign key references Salesman_no of SALESMAN_MASTER table

Dely_type	Char	1	F	Delivery: Part(P)/Full(F)
Dely_date	Date			

SALES_ORDER_DETAILS

Column Name	Data Type	Size	Default	Attributes
Order_no	Varchar2	6		Foreign key references Order_no of SALES_ORDER table
Product_no	Varchar2	6		Foreign key references Product_no of PRODUCT_MASTER table
Qty_disp	Number	8		
Product_rate	Number	10,2		

2. Insert 5 records in each table.

3. Answer the following queries:

1. Exercise on retrieving records from a table:
   i. Find out the names of all clients.
   ii. Retrieve the entire contents of the Client_Master table.
   iii. Retrieve the list of names, city and state of all clients.
   iv. List the various products available from the Product_Master table.
   v. List all clients who are located in Mumbai.
   vi. Find the names of salesman who have a salary equal to 3000.
   vii. Show the details of Product_Master according to Cost_Price in descending order.
   viii. Show different types of salary amounts of the salesman.
   ix. Show only unique product details.

ii>

2. Exercise on updating records in a table:
   i. Change the city of client no ‘C001’ to ‘Bangalore’.
   ii. Change the BalDue of client no ‘C006’ to Rs. 1000.
   iii. Change the cost price of ‘Trousers’ to Rs. 950.00.
   iv. Change the city of the salesman to ‘Pune’.
3. Add a column called ‘Telephone’ of data type number and size = 10 to the Client_Master table.
4. Change the size of Sell_Price column in Product_Master to 10, 2.
5. Drop the column Cost_Price from Product_Master.
6. Delete all salesmen from the Salesman_Master whose salaries are equal to Rs. 3500.
7. Delete all products from Product_Master where the quantity on hand is equal to 100.
8. Delete from Client_Master where the column state holds the value ‘Tamil Nadu’.
9. Change the name of the Salesman_Master table to Sman_Mast.
10. Destroy the table Client_Master along with its data.

Inbuilt functions in RDBMS

AIM: To study about various Inbuilt functions and implement them in the database.

Function accept zero or more arguments and both return one or more results. Both are used to manipulate individual data items. Operators differ from functional in that they follow the format of function name(arg..). An argument is a user defined variables or constants. Most operators accept at most 2 arguments while the structure of functions permit to accept 3 or more arguments. Function can be classifies into:-

1. SINGLE ROW FUNCTION
2. GROUP FUNCTIONS.

Single Row functions

A single row function or scalar function returns only one value for every row queries in table. Single row function can appear in a select command and can also be included in a where clause. The single row function can be broadly classified as,

o Date Function
o Numeric Function
o Character Function
o Conversion Function
o Miscellaneous Function

The example that follows mostly uses the symbol table "dual". It is a table, which is automatically created by oracle along with the data dictionary.

Date Function

They operate on date values and produce outputs, which also belong to date data type except for months, between, date function returns a number.

1. Add_month
This function returns a date after adding a specified date with specified number of months.
Syntax: Add_months(d,n); where d-date n-number of months
Example: Select add_months(sysdate,2) from dual;

2. last_day
It displays the last date of that month.
Syntax: last_day (d); where d-date
Example: Select last_day ('1-jun-2009') from dual;

3. Months_between

It gives the difference in number of months between d1 & d2.
Syntax: month_between (d1,d2); where d1 & d2 -dates

Example: Select month_between ('1-jun-2009','1-aug-2009') from dual;

4. next_day
It returns a day followed the specified date.
Syntax: next_day (d,day);
Example: Select next_day (sysdate,'wednesday') from dual

5. round
This function returns the date, which is rounded to the unit specified by the format model.
Syntax : round (d,[fmt]);
where d- date, [fmt] – optional. By default date will be rounded to the nearest day
Example: Select round (to_date('1-jun-2009','dd-mm-yy'),'year') from dual;
Select round ('1-jun-2009','year') from dual;

Numerical Functions

Command	Query	Output
Abs(n)	Select abs(-15) from dual;	15
Ceil(n)	Select ceil(55.67) from dual;	56
Exp(n)	Select exp(4) from dual;	54.59
Floor(n)	Select floor(100.2) from dual;	100
Power(m,n)	Select power(4,2) from dual;	16
Mod(m,n)	Select mod(10,3) from dual;	1
Round(m,n)	Select round(100.256,2) from dual;	100.26
Trunc(m,n)	Select trunc(100.256,2) from dual;	100.23
Sqrt(m,n)	Select sqrt(16) from dual;	4

Command	Query	Output
initcap(char);	select initcap("hello") from dual;	Hello
lower (char);	select lower ('HELLO') from dual;	hello
upper (char);	select upper ('hello') from dual;	HELLO
ltrim (char,[set]);	select ltrim ('cseit', 'cse') from dual;	it
rtrim (char,[set]);	select rtrim ('cseit', 'it') from dual;	cse
replace (char,search string, replace string)	select replace ('jack and jue', 'j', 'bl') from dual;	black and blue
substr (char,m,n);	select substr ('information', 3, 4) from dual;	form

Conversion Function

1. to_char()

Syntax: to_char(d,[format]);

This function converts date to a value of varchar type in a form specified by date format. If format is negelected then it converts date to varchar2 in the default date format.

Example: select to_char (sysdate, 'dd-mm-yy') from dual;

2. to_date()

Syntax: to_date(d,[format]);

This function converts character to date data format specified in the form character.

Example: select to_date('aug 15 2009','mm-dd-yy') from dual;

Group Functions

A group function returns a result based on group of rows.

1. avg

Example: select avg (total) from student;

2.max

Example: select max (percentagel) from student;

3.min

Example: select min (marksl) from student;

4. sum

Example: select sum(price) from product;

Count Function

In order to count the number of rows, count function is used.

1. count(*) – It counts all, inclusive of duplicates and nulls.

Example: select count(*) from student;

2. count(col_name)– It avoids null value.

Example: select count(total) from order;

3. count(distinct col_name) – It avoids the repeated and null values.

Example: select count(distinct ordid) from order;

Group by clause

This allows us to use simultaneous column name and group functions.
Example: Select max(percentage), deptname from student group by deptname;

Having clause

This is used to specify conditions on rows retrieved by using group by clause.
Example: Select max(percentage), deptname from student group by deptname having count(*)>=50;

Special Operators:

1. **In / not in** – used to select a equi from a specific set of values
2. **Any** - used to compare with a specific set of values
3. **Between / not between** – used to find between the ranges
4. **Like / not like** – used to do the pattern matching

Now do the assignment below:

- **Assignment. 5**.

i. Create tables for Client, Product, Salesman, Sales__Order, and Sales__Order_Details and populate them. Retrieve data by writing queries in SQL using logical operators, aggregate operators, group by, having, order by clauses etc.

ii. Create tables for Employee, Company and works and populate them. Retrieve data by writing nested queries in SQL using JOIN to combine tables and other operators like IN, BETWEEN, LIKE etc.

Do the following query on the same table of Assignment No. 4 for 5.1

Answer the following queries:

1. List the names of all clients having 'a' as the third letter in their names.
2. List the clients who stay in a city whose first letter is 'K'.
3. List all the clients who stay in 'Mumbai' or 'Kolkata'.
4. List all the clients whose BalDue is greater than value 1000.
5. List all information from the Sales_Order table for orders placed in the month of June.
6. List the order information for Client_no 'C00001' and 'C00003'.
7. List products whose selling price is greater than 500 and less than or equal to 750

8. Count the total number of order.
9. Determine the maximum and minimum product prices. Rename the output as max_price and min_price respectively.
10. Count the number of client who live in Kolkata.
11. Count the number of products having price less than or equal to 500.
12. List the order number and day on which clients placed their order.
13. List the Order_Date in the format 'DD-Month-YY'.
14. List the date, 20 days after today's date.
15. List name of the client who has maximum BalDue.
16. Find the difference between maximum BalDue and minimum BalDue.
17. Add Rs.1000/- with the salary amount of every salesmen.

Create the following tables and insert the values then do the queries for 5.2

employee: emp_no, name, dob, sex, address, salary
company: comp_no, name, address
works: emp_no, comp_no

1. List the employees who work for company 'C00002'
2. List the employees who work for company 'C00004'
3. List the employees who work for Clifford Corp
4. List the employees whose name ends with 'a'
5. List the employees born between 1999 and 2011

Nested Queries & Join Queries.

AIM: To know about various type of joining and nested queries

Nested Queries:

Nesting of queries one within another is known as a nested queries.

Subqueries

The query within another is known as a subquery. A statement containing subquery is called parent statement. The rows returned by subquery are used by the parent statement.

Example:

select ename, eno, address where salary >(select salary from employee where ename ='jones');

Types

1. Sub queries that return several values

Sub queries can also return more than one value. Such results should be made use along with the operators in and any.
Example: select ename, eno, from employee where salary <any (select salary from employee where deptno =10');

2. Multiple queries
Here more than one sub query is used. These multiple sub queries are combined by means of 'and' & 'or' keywords.

3. Correlated subquery
A subquery is evaluated once for the entire parent statement whereas a correlated subquery is evaluated once per row processed by the parent statement.
Example: select * from emp x where x.salary > (select avg(salary) from emp where deptno = x.deptno);
Above query selects the employees details from emp table such that the salary of employee is > the average salary of his own department.

Relating Data through Join Concept:

The purpose of a join concept is to combine data spread across tables. A JOIN clause is used to combine rows from two or more tables, based on a related column between them.
Syntax: select columns from table1, table2 where logical expression;

Different 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**: Return all records from the left table, and the matched records from the right table
- **RIGHT (OUTER) JOIN**: Return all records from the right table, and the matched records from the left table
- **FULL (OUTER) JOIN**: Return all records when there is a match in either left or right table
- **SELF JOIN**: Joining of a table to itself is known as self-join.

Simple Join or (Inner)Join

It is the most common type of join. It retrieves the rows from 2 tables having a common column and is further classified into

a) Equi-join
A join, which is based on equalities, is called equi-join.
Example: select * from item, cust where item.id=cust.id;

In the above statement, item-id = cust-id performs the join statement. It retrieves rows

from both the tables provided they both have the same id as specified by the where clause. Since the where clause uses the comparison operator (=) to perform a join, it is said to be *equijoin.* It combines the matched rows of tables.

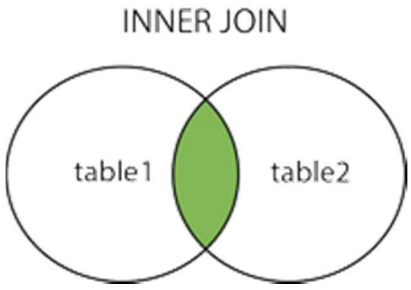
It can be used as follows:

- To insert records in the target table.
- To create tables and insert records in this table.
- To update records in the target table.
- To create views.

b) Non Equi-join
It specifies the relationship between columns belonging to different tables by making use of relational operators other than'='.
Example: select * from item, cust where item.id<cust.id;

Table Aliases
Table aliases are used to make multiple table queries shorted and more readable. We give an alias name to the table in the 'from' clause and use it instead of the name throughout the query.

SQL INNER JOIN Keyword
The INNER JOIN keyword selects records that have matching values in both tables.
Syntax
SELECT *column_name(s)*
FROM *table1*
INNER JOIN *table2* ON *table1.column_name = table2.column_name*;



Example
Orders table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3

10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

Customers table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

The following SQL statement selects all orders with customer information:

Example

```
SELECT Orders.OrderID, Customers.CustomerName
FROM Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID;
```

Note: The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the "Orders" table that do not have matches in "Customers", these orders will not show!

To Join Three Tables:

The following SQL statement selects all orders with customer and shipper information:

Example

```
SELECT Orders.OrderID, Customers.CustomerName, Shippers.ShipperName
FROM ((Orders
INNER JOIN Customers ON Orders.CustomerID = Customers.CustomerID)
INNER JOIN Shippers ON Orders.ShipperID = Shippers.ShipperID);
```

LEFT (OUTER) JOIN

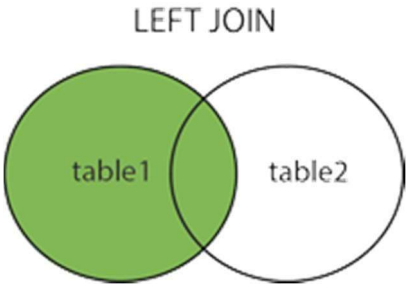
SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matched records from the right table (table2). The result is NULL from the right side, if there is no match.

Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2 ON table1.column_name = table2.column_name;
```

Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.



Example

"Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

"Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

SQL LEFT JOIN Example

The following SQL statement will select all customers, and any orders they might have:

Example

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID
ORDER BY Customers.CustomerName;
```

Note: The LEFT JOIN keyword returns all records from the left table (Customers), even if there are no matches in the right table (Orders).

RIGHT (OUTER) JOIN

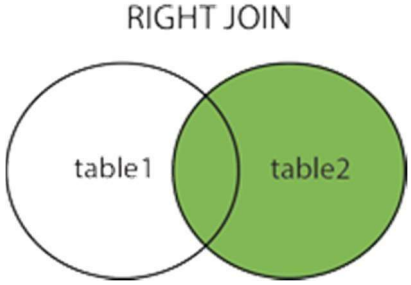
SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matched records from the left table (table1). The result is NULL from the left side, when there is no match.

Syntax

SELECT *column_name(s)*
FROM *table1*
RIGHT JOIN *table2* ON *table1.column_name = table2.column_name*;

Note: In some databases RIGHT JOIN is called RIGHT OUTER JOIN.



Example

"Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

"Employees" table:

EmployeeID	LastName	FirstName	BirthDate	Photo
1	Davolio	Nancy	12/8/1968	EmpID1.pic
2	Fuller	Andrew	2/19/1952	EmpID2.pic
3	Leverling	Janet	8/30/1963	EmpID3.pic

SQL RIGHT JOIN Example

The following SQL statement will return all employees, and any orders they might have have placed:

Example

```
SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
FROM Orders
```

RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID
ORDER BY Orders.OrderID;

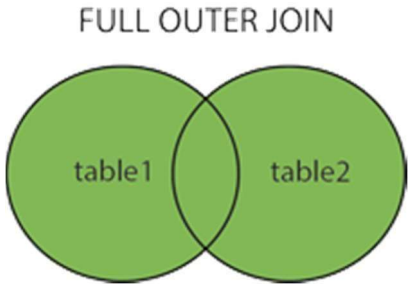
Note: The RIGHT JOIN keyword returns all records from the right table (Employees), even if there are no matches in the left table (Orders).

FULL (OUTER) JOIN

SQL FULL OUTER JOIN Keyword
The FULL OUTER JOIN keyword return all records when there is a match in either left (table1) or right (table2) table records.

Note: FULL OUTER JOIN can potentially return very large result-sets!

Syntax
SELECT *column_name(s)*
FROM *table1*
FULL OUTER JOIN *table2* ON *table1.column_name = table2.column_name*;



Example
"Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

"Orders" table:

OrderID	CustomerID	EmployeeID	OrderDate	ShipperID
10308	2	7	1996-09-18	3
10309	37	3	1996-09-19	1
10310	77	8	1996-09-20	2

SQL FULL OUTER JOIN Example
The following SQL statement selects all customers, and all orders:

```
SELECT Customers.CustomerName, Orders.OrderID
FROM Customers
FULL OUTER JOIN Orders ON Customers.CustomerID=Orders.CustomerID
ORDER BY Customers.CustomerName;
```

A selection from the result set may look like this:

CustomerName	OrderID
Alfreds Futterkiste	10308
Ana Trujillo Emparedados y helados	10365
Antonio Moreno Taquería	10382

Note: The FULL OUTER JOIN keyword returns all the rows from the left table (Customers), and all the rows from the right table (Orders). If there are rows in "Customers" that do not have matches in "Orders", or if there are rows in "Orders" that do not have matches in "Customers", those rows will be listed as well.

Self Join

Joining of a table to itself is known as self-join. It joins one row in a table to another. It can compare each row of the table to itself and also with other rows of the same table.

Syntax

```
SELECT column_name(s)
FROM table1 T1, table1 T2
WHERE condition;
```

Example
"Customers" table:

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany

2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico

The following SQL statement matches customers that are from the same city:

```
SELECT A.CustomerName AS CustomerName1, B.CustomerName AS CustomerName2,
A.City
FROM Customers A, Customers B
WHERE A.CustomerID <> B.CustomerID
AND A.City = B.City
ORDER BY A.City;
```

Now do the assignments below:

- ***Assignment.6.***

i. Design an ER diagram for an application that models a car-insurance company whose customers own one or more cars each. Analyze the requirements by identifying the entities, attributes, relationships, keys, constraints etc. Apply extended entity-relationship features to the design. Defend your design with proper assumptions and justifications. Map the ER model into a relational model.

ii. Create tables, populate with data and construct queries (advanced) in SQL to extract information from the car insurance company's database.Consider a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.

iii. Enter at least 5 sets of records in each table form created in part (ii).

iv. Write and run the following SQL queries for your database:

a. Find the total number of people who owned cars that were involved in accidents in 2010.
b. Find the number of accidents in which the cars belonging to "XYZ" were involved.
c. Add a new accident to the database; assume any values for required attributes.
d. Delete the model 'Scorpio belonging to "ABC".
e. Update the damage amount for the car with license number "AIBPC2010" in the accident with report number "FIR271" to Rs. 5000.

➢ *Assignment.7.*

I. Design an ER diagram for an application that models a hospital doctors treat patients, prescribe tests, monitor progress etc. Analyze the requirements by identifying the entities, attributes, relationships, keys, constraints etc. Apply extended entity-relationship features to the design. Defend your design with proper assumptions and justifications. Map the ER model into a relational model.

II. Create tables, populate with data and construct queries (advanced) in SQL to extract information from the hospital doctor's database.

III. Consider the following relations run the following SQL queries **:**

Doctor(SSN, FirstName, LastName, Specialty, YearsOfExperience, PhoneNum)

Patient(SSN, FirstName, LastName, Address, DOB, PrimaryDoctor_SSN)

Medicine(TradeName, UnitPrice, GenericFlag)

Prescription(Id, Date, Doctor_SSN, Patient_SSN)

Prescription_Medicine(Prescription Id, TradeName, NumOfUnits)

1. List the trade name of generic medicine with unit price less than $50.
2. List the first and last name of patients whose primary doctor named 'John Smith'.
3. List the first and last name of doctors who are not primary doctors to any patient.
4. For medicines written in more than 20 prescriptions, report the trade name and the total number of units prescribed.
5. List the SSN of patients who have 'Aspirin' and 'Vitamin' trade names in one prescription.
6. List the SNN of distinct patients who have 'Aspirin' prescribed to them by doctor named 'John Smith'.
7. List the first and last name of patients who have no prescriptions written by doctors other than their primary doctors.

PL/SQL

Aim: To combine database language and procedural programming language.

PL/SQL - FUNCTION

Creating a Function

A standalone function is created using the **CREATE FUNCTION** statement. The simplified syntax for the **CREATE OR REPLACE PROCEDURE** statement is as follows −

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter_name [IN | OUT | IN OUT] type [, ...])]
RETURN return_datatype
{IS | AS}
BEGIN
  < function_body >
END [function_name];
```

Where, *function-name* specifies the name of the function.

- [OR REPLACE] option allows the modification of an existing function.
- The optional parameter list contains name, mode and types of the parameters. IN represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- The function must contain a **return** statement.
- The *RETURN* clause specifies the data type you are going to return from the function.
- *function-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone function.

This function returns the total number of CUSTOMERS in the customers table.

```
CREATE OR REPLACE FUNCTION totalCustomers
RETURN number IS
  total number(2) := 0;
BEGIN
  SELECT count(*) into total
  FROM customers;

  RETURN total;
END;
/
```

Output: Function Created.

Calling a Function

Following program calls the function **totalCustomers** from an anonymous block −

```
DECLARE
   c number(2);
BEGIN
   c := totalCustomers();
   dbms_output.put_line('Total no. of Customers: ' || c);
END;
/
```

Output:Total no. of Customers: 6

PL/SQL procedure successfully completed.

PL/SQL Recursive Functions

The following program calculates the factorial of a given number by calling itself recursively −

```
DECLARE
   num number;
   factorial number;

FUNCTION fact(x number)
RETURN number
IS
   f number;
BEGIN
   IF x=0 THEN
      f := 1;
   ELSE
      f := x * fact(x-1);
   END IF;
RETURN f;
END;

BEGIN
   num:= 6;
   factorial := fact(num);
   dbms_output.put_line(' Factorial '|| num || ' is ' || factorial);
```

```
END;
/
```

Output: Factorial 6 is 720

PL/SQL - CURSOR

A **cursor** is a pointer to this context area. PL/SQL controls the context area through a cursor. A cursor holds the rows (one or more) returned by a SQL statement. The set of rows the cursor holds is referred to as the **active set**.

You can name a cursor so that it could be referred to in a program to fetch and process the rows returned by the SQL statement, one at a time. There are two types of cursors −

- Implicit cursors
- Explicit cursors

Implicit Cursors

Implicit cursors are automatically created by Oracle whenever an SQL statement is executed, when there is no explicit cursor for the statement. Programmers cannot control the implicit cursors and the information in it.

Whenever a DML statement (INSERT, UPDATE and DELETE) is issued, an implicit cursor is associated with this statement. For INSERT operations, the cursor holds the data that needs to be inserted. For UPDATE and DELETE operations, the cursor identifies the rows that would be affected.

In PL/SQL, you can refer to the most recent implicit cursor as the **SQL cursor**, which always has attributes such as **%FOUND, %ISOPEN, %NOTFOUND**, and **%ROWCOUNT**. The SQL cursor has additional attributes, **%BULK_ROWCOUNT** and **%BULK_EXCEPTIONS**, designed for use with the **FORALL** statement. The following table provides the description of the most used attributes −

S.No	Attribute & Description
1	**%FOUND** Returns TRUE if an INSERT, UPDATE, or DELETE statement affected one or more rows or a SELECT INTO statement returned one or more rows. Otherwise, it returns FALSE.

2	**%NOTFOUND** The logical opposite of %FOUND. It returns TRUE if an INSERT, UPDATE, or DELETE statement affected no rows, or a SELECT INTO statement returned no rows. Otherwise, it returns FALSE.
3	**%ISOPEN** Always returns FALSE for implicit cursors, because Oracle closes the SQL cursor automatically after executing its associated SQL statement.
4	**%ROWCOUNT** Returns the number of rows affected by an INSERT, UPDATE, or DELETE statement, or returned by a SELECT INTO statement.

Any SQL cursor attribute will be accessed as **sql%attribute_name** as shown below in the example.

Example

We will be using the CUSTOMERS table we had created and used in the previous chapters.

Select * from customers;

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
+----+----------+-----+-----------+----------+
```

The following program will update the table and increase the salary of each customer by 500 and use the **SQL%ROWCOUNT** attribute to determine the number of rows affected −

```
DECLARE
   total_rows number(2);
BEGIN
   UPDATE customers
   SET salary = salary + 500;
```

```
   IF sql%notfound THEN
      dbms_output.put_line('no customers selected');
   ELSIF sql%found THEN
      total_rows := sql%rowcount;
      dbms_output.put_line( total_rows || ' customers selected ');
   END IF;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

6 customers selected

PL/SQL procedure successfully completed.

If you check the records in customers table, you will find that the rows have been updated −

Select * from customers;

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2500.00 |
|  2 | Khilan   |  25 | Delhi     |  2000.00 |
|  3 | kaushik  |  23 | Kota      |  2500.00 |
|  4 | Chaitali |  25 | Mumbai    |  7000.00 |
|  5 | Hardik   |  27 | Bhopal    |  9000.00 |
|  6 | Komal    |  22 | MP        |  5000.00 |
+----+----------+-----+-----------+----------+
```

Explicit Cursors

Explicit cursors are programmer-defined cursors for gaining more control over the **context area**. An explicit cursor should be defined in the declaration section of the PL/SQL Block. It is created on a SELECT Statement which returns more than one row.

The syntax for creating an explicit cursor is −

CURSOR cursor_name IS select_statement;

Working with an explicit cursor includes the following steps −

- Declaring the cursor for initializing the memory
- Opening the cursor for allocating the memory
- Fetching the cursor for retrieving the data
- Closing the cursor to release the allocated memory

Declaring the Cursor

Declaring the cursor defines the cursor with a name and the associated SELECT statement. For example −

```
CURSOR c_customers IS
  SELECT id, name, address FROM customers;
```

Opening the Cursor

Opening the cursor allocates the memory for the cursor and makes it ready for fetching the rows returned by the SQL statement into it. For example, we will open the above defined cursor as follows −

```
OPEN c_customers;
```

Fetching the Cursor

Fetching the cursor involves accessing one row at a time. For example, we will fetch rows from the above-opened cursor as follows −

```
FETCH c_customers INTO c_id, c_name, c_addr;
```

Closing the Cursor

Closing the cursor means releasing the allocated memory. For example, we will close the above-opened cursor as follows −

```
CLOSE c_customers;
```

Example

Following is a complete example to illustrate the concepts of explicit cursors &minua;

```
DECLARE
  c_id customers.id%type;
  c_name customerS.No.ame%type;
  c_addr customers.address%type;
  CURSOR c_customers is
    SELECT id, name, address FROM customers;
BEGIN
  OPEN c_customers;
  LOOP
  FETCH c_customers into c_id, c_name, c_addr;
```

```
    EXIT WHEN c_customers%notfound;
    dbms_output.put_line(c_id || ' ' || c_name || ' ' || c_addr);
  END LOOP;
  CLOSE c_customers;
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

1 Ramesh Ahmedabad
2 Khilan Delhi
3 kaushik Kota
4 Chaitali Mumbai
5 Hardik Bhopal
6 Komal MP

PL/SQL procedure successfully completed.

PL/SQL – Procedures

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

A subprogram can be created −

- At the schema level
- Inside a package
- Inside a PL/SQL block

At the schema level, subprogram is a **standalone subprogram**. It is created with the CREATE PROCEDURE or the CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.

A subprogram created inside a package is a **packaged subprogram**. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement. We will discuss packages in the chapter **'PL/SQL - Packages'**.

PL/SQL subprograms are named PL/SQL blocks that can be invoked with a set of parameters. PL/SQL provides two kinds of subprograms −

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

Parts of a PL/SQL Subprogram

Each PL/SQL subprogram has a name, and may also have a parameter list. Like anonymous PL/SQL blocks, the named blocks will also have the following three parts −

S.No	Parts & Description
1	**Declarative Part** It is an optional part. However, the declarative part for a subprogram does not start with the DECLARE keyword. It contains declarations of types, cursors, constants, variables, exceptions, and nested subprograms. These items are local to the subprogram and cease to exist when the subprogram completes execution.
2	**Executable Part** This is a mandatory part and contains statements that perform the designated action.
3	**Exception-handling** This is again an optional part. It contains the code that handles run-time errors.

Creating a Procedure

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

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

Where,

- *procedure-name* specifies the name of the procedure.
- [OR REPLACE] option allows the modification of an existing procedure.

- The optional parameter list contains name, mode and types of the parameters. **IN** represents the value that will be passed from outside and OUT represents the parameter that will be used to return a value outside of the procedure.
- *procedure-body* contains the executable part.
- The AS keyword is used instead of the IS keyword for creating a standalone procedure.

Example

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

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

Outuput: Procedure created.

Executing a Standalone Procedure

A standalone procedure can be called in two ways −

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

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

EXECUTE greetings;

The above call will display −

Hello World

PL/SQL procedure successfully completed.

The procedure can also be called from another PL/SQL block −

```
BEGIN
   greetings;
END;
/
```

The above call will display −

Hello World

PL/SQL procedure successfully completed.

Deleting a Standalone Procedure

A standalone procedure is deleted with the **DROP PROCEDURE** statement. Syntax for deleting a procedure is −

DROP PROCEDURE procedure-name;

You can drop the greetings procedure by using the following statement −

DROP PROCEDURE greetings;

Parameter Modes in PL/SQL Subprograms

The following table lists out the parameter modes in PL/SQL subprograms −

S.No	Parameter Mode & Description
1	**IN** An IN parameter lets you pass a value to the subprogram. **It is a read-only parameter**. Inside the subprogram, an IN parameter acts like a constant. It cannot be assigned a value. You can pass a constant, literal, initialized variable, or expression as an IN parameter. You can also initialize it to a default value; however, in that case, it is omitted from the subprogram call. **It is the default mode of parameter passing. Parameters are passed by reference**.
2	**OUT** An OUT parameter returns a value to the calling program. Inside the subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it. **The actual parameter must be variable and it is passed by value**.
3	**IN OUT** An **IN OUT** parameter passes an initial value to a subprogram and returns an updated value to the caller. It can be assigned a value and the value can be read.

	The actual parameter corresponding to an IN OUT formal parameter must be a variable, not a constant or an expression. Formal parameter must be assigned a value. **Actual parameter is passed by value.**

✓ IN & OUT Mode Example 1

This program finds the minimum of two values. Here, the procedure takes two numbers using the IN mode and returns their minimum using the OUT parameters.

```
DECLARE
   a number;
   b number;
   c number;
PROCEDURE findMin(x IN number, y IN number, z OUT number) IS
BEGIN
   IF x < y THEN
      z:= x;
   ELSE
      z:= y;
   END IF;
END;
BEGIN
   a:= 23;
   b:= 45;
   findMin(a, b, c);
   dbms_output.put_line(' Minimum of (23, 45) : ' || c);
END;
/
```

When the above code is executed at the SQL prompt, it produces the following result −

Minimum of (23, 45) : 23

PL/SQL procedure successfully completed.

✓ IN & OUT Mode Example 2

This procedure computes the square of value of a passed value. This example shows how we can use the same parameter to accept a value and then return another result.

```
DECLARE
```

```
  a number;
PROCEDURE squareNum(x IN OUT number) IS
BEGIN
 x := x * x;
END;
BEGIN
  a:= 23;
  squareNum(a);
  dbms_output.put_line(' Square of (23): ' || a);
END;
/
```

Output: Square of (23): 529

- Methods for Passing Parameters

Actual parameters can be passed in three ways −

I. Positional notation
II. Named notation
III. Mixed notation

- Positional Notation

In positional notation, you can call the procedure as −

findMin(a, b, c, d);

In positional notation, the first actual parameter is substituted for the first formal parameter; the second actual parameter is substituted for the second formal parameter, and so on. So, **a** is substituted for **x, b** is substituted for **y, c** is substituted for **z** and **d** is substituted for **m**.

- Named Notation

In named notation, the actual parameter is associated with the formal parameter using the **arrow symbol (=>)**. The procedure call will be like the following −

findMin(x => a, y => b, z => c, m => d);

- Mixed Notation

In mixed notation, you can mix both notations in procedure call; however, the positional notation should precede the named notation.

The following call is legal −

findMin(a, b, c, m => d);

However, this is not legal:

findMin(x => a, b, c, d);

➢ *Assignment.8.*

i. Implement a PL/SQL block that will accept student id number from the user, and check is student attendance is less than 80% then display message that student cannot appear in exam. [Table: STUDENT (STUD_ID, primary key, STUD_NAME, STUD_ATT)].

ii. Implement a PL/SQL code block that will accept an account number from the user. Check if the user's balance is less than the minimum balance, only then deduct Rs.100 from the balance. The process is fired on the ACCT_MSTR table. [Table: ACCT_MSTR (ACCT_NO, ACCT_HOLDR_NAME, CURBAL].

iii. Implement a PL/SQL code block to calculate the area of a circle for a value of radius varying from 3 to 7. Store the radius and the corresponding values of calculated area in an empty table named AREAS, consisting of two columns Radius and Area. [Table: AREAS (RADIUS, AREA)].

iv. Implement a PL/SQL procedure that takes weight of an apple box as input from the user.
- If the weight is >= 10 kg, rate =Rs. 5/kg.
- If weight is < 10 kg, rate = Rs. 7/kg.

Calculate the cost of the apple box. Display the output on the screen.

v. Implement a PL/SQL procedure to calculate the difference between highest salaried and lowest salaried employee. Store the information in a table.

vi. Implement a PL/SQL block using cursor that will display the name, department and the salary of the first 3 employees getting lowest salary. [Table: Employee (ename, dept, salary)]

vii. Implement a PL/SQL cursor that will update salary of all employees, such that, it allows an increment of 20% if the salary is less than 2000 otherwise increment of Rs.1000. It should print old and new salary for all employees. [Table: Employee (ename, dept, salary)]

➢ *Assignment.9.*

Consider the following relations and Draw the ER, EER Diagram, Relational Model and write the SQL statement for the following queries:

Create the tables and insert 5 sets of records into each.

employee (personname, street, city)

works (personname, companyname, salary)

company (companyname, city)

manages (personname, managername)

a) Find the names of all employees who work for Axis Bank.

b) Find the names and cities of residence of all employees who work for Axis Bank.

c) Find the names, street addresses, and cities of residence of all employees who work for Axis Bank and earn more than Rs.30000 per annum.

d) Find all employees who live in the same city as the company for which they work is located.

e) Find all employees who live in the same city and on the same street as their managers.

f) Find all employees in the database who do not work for Axis Bank.

g) Find all employees who earn more than every employee of Axis Bank.

h) Assume that the companies may be located in several cities. Find all companies located in every city iin which Axis Bank is located.

i) Find all employees who earn more than the average salary of all employees of their company.

j) Find the company that has the most employees.

k) Find the company that has the smallest payroll.

l) Find those companies whose employees earn a higher salary, on average, than the average salary at Axis Bank.

m) Modify the database so that ABC now lives in Kolkata.

n) Give all employees of Axis Bank a 10 percent raise.

o) Give all managers in the database a 10 percent raise.

P) Give all managers in the database a 10 percent raise, unless the salary would be greater than Rs.300000.In such cases, give only a 3 percent raise.

q) Delete all tuples in the works relation for employees of Axis Bank.

➢ *Assignment.10.*

Consider the following tables:
MATCH (match_id, team1, team2, ground, mdate, winner)
PLAYER (p_id, lname, fname, country, yborn, bplace, ftest)
BATTING (match_id, p_id, mts, order, out_type, fow, nruns, nballs, fours, sixes)
BOWLING (match_id, p_id, novers, maidens, nruns, nwickets)

1. Draw the appropriate ER, EER and Relational model for the given data.
2. Write SQL expressions for the following:

i) Find match ids of those matches in which player 27001 bats and makes more runs than he made at every match he played at Sydney.

ii) Find player ids of players who have scored more than 30 in every ODI match that they have batted.

iii) Find the ids of players that had a higher average score than the average score for all players when they played in Sri Lanka.

➢ *Assignment.11.*

A record company wishes to use a computer database to help with its operations regarding its performers, recordings and song catalogue. A requirements analysis has elicited the following information: Songs have a unique song number, a non-unique title and a composition date. A song can be written by a number of composers; the composer's full name is required. Songs are recorded by recording artists (bands or solo performers). A song is recorded as a track of a CD. A CD has many songs on it, called tracks. CDs have a unique record catalogue number, a title and must have a producer (the full name of the producer is required). Each track must have the recording date and the track number of the CD. • A song can appear on many (or no) CDs, and be recorded by many different recording artists. The same recording artist might re-record the same song on different CDs. A CD must have only 1 recording artist appearing on it. CDs can be released a number of times, and each time the release date and associated number of sales is required.

1. Use this information to design an appropriate ER and relational model.
2. Compile DDL and DML commands on the database created.

SQL:-

i>Update number of recorded album to 4 for those artist who has recorded only 3.

ii>Find all artists who have recorded at least two albums.

iii>Find all writers who have only written one song.

PL/SQL

1. i>Write Procedure to insert a new Contract into the Contract relation.

➢ *Assignment.12.*

1> Create the following tables.
Hotel (Hotel_No, Name, Address)
Room (Room_No, Hotel_No, Type, Price)
Booking (Hotel_No, Guest_No, Date_From, Date_To, Room_No)
Guest (Guest_No, Name, Address)
Populate the tables
Answer the following query using SQL.
1. List the names and addresses of all guests in London, alphabetically ordered by name
2. List all double or family rooms with a price below £40.00 per night, in ascending order of price.
3. List the bookings for which no date_to has been specified.
4. How many hotels are there?
5. What is the average price of a room?
6. What is the total revenue per night from all double rooms?
7. How many different guests have made bookings for August?
8. List the details of all rooms at the Grosvenor Hotel, including the name of the guest staying in the room, if the room is occupied.
9. What is the total income from bookings for the Grosvenor Hotel today?
10. List the rooms that are currently unoccupied at the Grosvenor Hotel.

Design an ER Model for an application where hotels are booked by guests wanting to go on aholiday in India or abroad. Your design should meet all requirements. Map into a relationalmodel.

2> A. Consider the schema for Company Database:

EMPLOYEE (SSN, Name, Address, Sex, Salary, SuperSSN, DNo)

DEPARTMENT (DNo, DName, MgrSSN, MgrStartDate)

DLOCATION (DNo,DLoc)

PROJECT (PNo, PName, PLocation, DNo)

WORKS_ON (SSN, PNo, Hours)

Write SQL queries to ……

1. Make a list of all project numbers for projects that involve an employee whose last name is 'Scott', either as a worker or as a manager of the department that controls the project.

2. Show the resulting salaries if every employee working on the 'IoT' project is given a 10 percent raise.

3. Find the sum of the salaries of all employees of the 'Accounts' department, as well as the maximum salary, the minimum salary, and the average salary in this department

4. Retrieve the name of each employee who works on all the projects controlled by department number 5 (use NOT EXISTS operator).

5. For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than Rs. 6,00,000.

B. Write a program in PL/SQL to create a procedure to displays the GCD of nos.

C.Write a program in PL/SQL to create a cursor displays the name and salary of each employee in the EMPLOYEES table whose salary is less than that specified by a passed-in parameter value.