Unit 2

Relational Data Model Concepts

2.1 RELATIONAL DATA MODEL CONCEPTS

The relational model uses a collection of tables to represent both data and the relationship among those data.

- A table is a collection of rows and columns. Each column has a unique name.
- Each row in the table represents a collection of related data values.
- In the relational model, a row is called a tuple, a column is called an attribute and the table is called a relation.

Student

Attributes

Roll_No	Name	City	Age
1	Vijay	Gr. Noida	22
2	Santosh	Delhi	20
3	Gopal	Noida	23

Tuples

2.2 INTEGRITY CONSTRAINTS

Most database applications have certain integrity constraints that must hold for the data.

- The simplest type of integrity constraint involves specifying a data type for each data item.
- Integrity constrains can be classified as :

$$\begin{tikzcd}
 & \text{Entity integrity} \\
\text{Integrity constraint} \arrow[ur, dash] \arrow[dr, dash] & \\
 & \text{Referential integrity}
\end{tikzcd}$$

2.2.1 Entity Integrity

The entity integrity constraints states that no primary key value can be Null. This is because the primary key value is used to identify individual tuple in a relation, having Null values for the primary key implies that we cannot identify some tuples.

e.g., If two or more tuples has Null for their primary key, then we might not be able to distinguish them.

2.2.2 Referential Integrity

We wish to ensure that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another. This condition is called "Referential Integrity".

- This constraint establishes a relationship between records across a master and a detail tables.

This relationship ensures :

(i) Records cannot be inserted into a detail table if corresponding records in the master table do not exist.

(ii) Records of the master table cannot be deleted if corresponding records in the detail table exist.

2.3 DOMAIN CONSTRAINTS

It is a pool of values for which we can extract the values for set by taking some conditions. We have seen that a domain of possible values must be associated with every attribute. We see the number of standard domain types such as Integer types, character types and date/time types" defined in SQL.

- Domain constraints are the most elementary form of integrity constraint. They are tested easily by the system whenever a new data item is entered into the database. It is possible for several attributes to have the same domain.

e.g., The attributes customer-name and Employee-name might have the same domain. The set of all person names.

- The domain constraints not only allows us the test values inserted in the database, but also permits us to test queries to ensure that the comparison made make sense.

e.g., The create domain clause can be used to define new domains.

Create domain dollars numeric (12, 2) :

Create domain pounds numeric (12, 2) :

An attempt to assign a value of type dollars to a variable of type pounds would result in a syntax error, while both are of the same numeric type.

2.4 RELATIONAL ALGEBRA

Relational algebra is a procedural query language. It consists of a set of operations that take one or two relations as input and produce a new relation as their result.

The functional operations in the relational algebra are :

- Select
- Project
- Union
- Set difference
- Cartesian product
- Rename
- Intersection
- Division
- Join
- Natural Join

2.4.1 Select Operation

Select tuples that satisfy a given predicate. The notation of select operation is :

$$\sigma_P (R)$$

where $R \rightarrow$ input relation

$P \rightarrow$ predicate that is to be evaluated

e.g., Select those account which belongs the SBI.

$$\sigma_{\text{branch\_name} = \text{SBI}} \text{(account)}$$

Note : We can use following predicate.

(i) = (equal to)
(ii) < (less than)
(iii) > (greater than)
(iv) ≠ (not equal to)
(v) ≤ (less than or equal to)
(vi) ≥ (greater than equal to)
(vii) ∧ (and)

e.g., Select loans which has been taken from the SBI and whose amount is greater than 50,000 each.

Ans. $\sigma_{\text{branch\_name} = \text{SBI} \wedge \text{amount} > 50,000}$ (loan)

2.4.2 Project-Operation

The project operations is a uniary operation that returns its argument relation, with certain attributes left out.

Suppose, we want to list all loan numbers and the amount of the loan but do not care about the branch name. The project operation allows us to produce this relation.

It is denoted by π (Pi).

e.g., Find the names of all customers living in the New Delhi.

Ans. $\pi_{\text{customer\_name}}$ ($\sigma_{\text{customer\_city} = \text{"New Delhi"}}$ (Customer))

e.g., To find the loan number and amount of all loans.

Ans. $\pi_{\text{loan\_no, amount}}$ (loan)

2.4.3 Union Operation

The union of two sets combines all data that is appearing either one or both relations.

For the union operation $r \cup s$, the relation r and s must have the same number of attributes.

e.g., To find the names of all bank customers who have either an account or a loan or both.

Ans. Let the two relations are :

Depositor :

Customer_Name	Acc_No
Vijay	A–101
Gopal	A–102
Ajay	A–103
Alok	A–104
Sanjay	A–105

Borrower :

Customer_Name	Loan_No
Raj	L–17
Ravi	L–18
Ramesh	L–19
Santosh	L–20
Rohit	L–21

$\pi_{\text{Customer\_name}}^{\text{(Borrower)}} \cup \pi_{\text{Customer\_name}}^{\text{(Depositor)}}$

The query gives the result as :

Customer_Name
Vijay
Gopal
Ajay
Alok
Sanjay
Raj
Ravi
Ramesh
Santosh
Rohit

2.4.4 Set-Difference Operation

• The set-difference operation, allows us to find tuples that are in one relation but are not in another.

The expression $r-s$ produces a relation containing those tuples in r but not in s.

e.g., Find all customers of the bank who have an account but not a loan.

Ans. $\pi_{\text{Customer\_name}}^{\text{(Depositor)}} - \pi_{\text{Customer\_name}}^{\text{(Borrower)}}$

The result of this query will be

Customer_Name
Vijay
Gopal
Ajay
Alok
Sanjay

2.4.5 Cartesian Product Operation

The cartesian product operation, allows us to combine information from any two relations.

The cartesian product of relation r_1 and r_2 as $r_1 \times r_2$.

Ex. r_1

Roll_No	Name
1	Vijay
2	Gopal
3	Santosh
4	Sanjay

r_2

Subject
English
Math

$r_1 \times r_2$: The results of query will be

Roll_No	Name	Subject
1	Vijay	English
2	Gopal	English
3	Santosh	English
4	Sanjay	English
1	Vijay	Math
2	Gopal	Math
3	Santosh	Math
4	Sanjay	Math

2.4.6 Division Operation

The symbol of this operator is ÷ and it use for select "for all".
The division operator can apply as

A =

SNo	PNo
S_1	P_1
S_1	P_2
S_1	P_3
S_1	P_4
S_2	P_1
S_2	P_2
S_3	P_2
S_4	P_2

B =

PNo
P_2

C =

PNo
P_2
P_4

D =

PNo
P_1
P_2
P_4

A ÷ B =

SNo
S_1
S_2
S_3
S_4

A ÷ C =

SNo
S_1

A ÷ D =

SNo
S_1

2.4.7 Rename-Operation

In relational algebra, *A* rename is a unary operation written as

$$\rho_{a/b}(R)$$

where

- *a* and *b* are attribute names
- *R* is a relation.

The result is identical to *R* except that the *b* field in all tuples is renamed to an *a* field.

e.g., Consider the Employee relation and its renamed version :

Employee :

Name	Emp_Id
Vijay	3415
Gopal	2241

$\rho_{\text{Emp-Name/Name}}$ (Employee)

Emp_Name	Emp_Id
Vijay	3415
Gopal	2241

2.4.8 Join

The join operator allows the combining of two relations to form a single new relation.

For Ex. Emp. Table

Emp_Id	Name
100	Vijay
101	Gopal
102	Santosh
103	Sanjay

Salary Table

S_Id	Salary
100	15000
101	25000
102	20000
103	15000

Result of Emp Join Salary

Emp_Id	Name	S_Id	Salary
100	Vijay	100	15000
101	Gopal	101	25000
102	Santosh	102	20000
103	Sanjay	103	15000

2.4.8.1 Natural Join : Natural join is a dyadic operator that is written as R and S, where *R* and *S* are relations. The result of the natural join is the set of all combinations of tuples in *R* and *S* that are equal on their common attribute names.

For Ex. : Consider the tables employee and Dept and their natural join.

Employee

Name	Emp_Id	Dept_Name
Sumit	100	Finance
Sanjay	101	Sales
Raja	102	Finance
Sanjeev	103	Sales

Dept

Dept_Name	Manager
Finance	Gopal
Sales	Santosh
Production	Vijay

Employee Dept

Name	Emp_Id	Dept_Name	Manager
Sumit	100	Finance	Gopal
Sanjay	101	Sales	Santosh
Raja	102	Finance	Gopal
Sanjeev	103	Sales	Santosh

2.4.8.2 Semi Join : The semi join is similar to the natural join and written as $R \square S$, where R and S are relations. The result of the semi join is only the set of all tuples in R for which there is a tuple in S that is equal on their common attribute names.

For Ex. : Consider the tables employee and Dept and their semi join.

Employee

Name	Emp_Id	Dept_Name
Sanjay	100	Finance
Shally	101	Sales
Santosh	102	Finance
Sumit	103	Production

Dept

Dept_Name	Manager
Sales	Vijay
Production	Gopal

Employee □ Dept

Name	Emp_Id	Dept_Name
Shally	101	Sales
Sumit	103	Production

2.4.8.3 Anti Join : The anti join, written as $R \triangleright S$, where R and S are relations, is similar to the natural join, but the result of an anti join is only those tuples in R for which there is not a tuple in S that is equal on their common attribute names.

For Ex. : Consider the table employee and Dept and their anti join.

Employee

Name	Emp_Id	Dept_Name
Sumit	100	Finance
Sanjay	101	Sales
Raja	102	Finance
Sanjeev	103	Sales

Dept

Dept_Name	Manager
Sales	Vijay
Production	Santosh

Employee $\triangleright$ Dept

Name	Emp_Id	Dept_Name
Sumit	100	Finance
Raja	102	Finance

2.4.8.4 Outer join : The full outer join is written as $R = X = S$ where R and S are relations. The result of the full outer join is the set of all combinations of tuples in R and S that are equal on their common attribute names, in addition to tuple S that have on matching tuples in R and tuples in R that have no matching tuples in S in their common attribute names.

Ex. : Consider the table Employee and Dept and their full outer Join :

Employee

Name	Emp_Id	Dept_Name
Sumit	100	Finance
Sanjay	101	Sales
Raja	102	Finance
Sanjeev	103	Sales
Shally	104	Executive

Dept

Dept_Name	Manager
Sales	Vijay
Production	Santosh

Employee $= X =$ Dept

Name	Emp_Id	Dept_Name	Manager
Sumit	100	Finance	ω
Sanjay	101	Sales	Vijay
Raja	102	Finance	ω
Sanjeev	103	Sales	Vijay
Shally	104	Executive	ω
ω	ω	Production	Santosh

$\omega \rightarrow$ Null value

2.4.9 Projection

A projection is a uniary operation written as $\pi_{a_1, a_2, \ldots, a_n}(R)$, where $a_1, a_2, \ldots, a_n$ is the set of attribute names. The result of such projection is defined as the set that is obtained when all tuples in R are restricted to the set $(a_1, a_2, \ldots, a_n)$.

Example : The table employee (E)

ID	Name	Salary
1	Vijay	15000
5	Santosh	30000
7	Gopal	25000

SQL	Result		Relational Algebra
Select Salary from E	Salary		$^{\pi}$Salary (E)
	15000 30000 25000		
Select ID, Salary from E	ID	Salary	$^{\pi}$ID, Salary (E)
	1	15000	
	5	30000	
	7	25000	

Selection : The example selection

SQL	Result			Relational Algebra
Select * from E where salary < 30000	ID	Name	Salary	(E) $^{\sigma}$Salary < 30000
	1	Vijay	15000	
	7	Gopal	25000	
Select * from E where salary < 30000 and Id < 7	ID	Name	Salary	(E) $^{\sigma}$Salary < 30000 and ID < 7
	1	Vijay	15000	

2.5 RELATIONAL CALCULUS

The relational calculus are non procedural languages that represent the basic power required in a relational query language.

The relational calculus is used in design of commercial query language such as SQL, QBL.

A query in the tuple relational calculus is expressed as :

$$\{t/P(t)\}$$

that is, it is the set of all tuples t such that predicate P is true for t.

A tuple-relational-calculus formula is build up out of atoms. An atom has one of the following forms :

- $S \in r$, where S is a tuple variable and r is a relation.
- $S\,[x]\;\Theta\;u\,[y]$, where Θ is a comparison operator and S, u are tuple variables, x is an attribute on which S is defined and y is an attribute on which u is defined.
- $S\,[x]\;\Theta\;C$, where S is a tuple variable, x is an attribute on which S is define, Θ is comparison operator and C is a constraint in the domain of attribute x.

We build up formulae from atoms by using the following rules :

- An atom is a formula.
- If P_1 is a formula, then so are $\neg P_1$ and (P_1).
- If P_1 and P_2 are formulae, then so are $P_1 \vee P_2$, $P_1 \wedge P_2$ and $P_1 \Rightarrow P_2$.
- If $P_1\,(S)$ is a formula containing a free tuple variable S, and r is a relation, then

$$\exists\, S \in r\,(P_1\,(S)) \text{ and } \forall\, S \in r\,(P_1\,(S))$$

In the tuple relational calculus, these equivalences include the following three rules :

(1) $P_1 \wedge P_2$ is equivalent to $\neg\,(\neg\,(P_1) \vee \neg\,(P_2))$

(2) $\forall\, t \in r\,(P_1(t))$ is equivalent to $\neg\,\exists t \in r\,(\neg\,P_1(t))$

(3) $P_1 \Rightarrow P_2$ is equivalent to $\neg (P_1) \vee P_2$.

For example : (1) Find the loan number for each loan of an amount greater than \$ 1200.

$\{t / \exists S \in \text{loan} \ (t\ [\text{loan-number}] = S\ [\text{loan-number}] \wedge S\ [\text{amount}] > 1200)\}$

(2) Find all customers who have a loan, an account or both at the bank.

$\{t / \exists S \in \text{borrower} \ (t\ [\text{customer-name}] = S\ [\text{customer-name}]) \vee \exists u \in \text{depositor} \ (t\ [\text{customer-name}] = u\ [\text{customer-name}])\}$

(3) Find all customers who have an account at the bank but do not have a loan from the bank.

$\{t \mid \exists u \in \text{depositor} \ (t\ [\text{customer-name}] = u\ [\text{customer-name}]) \wedge \neg \exists s \in \text{borrower} \ (t\ [\text{customer-name}] = s\ [\text{customer-name}])\}$

(4) Find the branch-name, loan-number, and amount from loan of over Rs. 12000.

$\{t \mid t \in \text{loan} \wedge t\ [\text{amount}] > 12000\}$

2.6 THE DOMAIN RELATIONAL CALCULUS

- The domain relational calculus are non procedural language that represent the basic power required in a relational query language.
- The domain relational calculus uses domain variables that take on values from an attributes domain.

Formal Definition : In the domain relational calculus is of the form :

$$\{\langle x_1, x_2, \ldots, x_n \rangle \mid P\ (x_2, x_2, \ldots, x_n)\}$$

where $x_1, x_2, \ldots, x_n$ are domain variables P represents a formula composed of atoms.

An atom in the domain relational calculus has one of the following forms :

- $< x_1, x_2, x_3, \ldots, x_n > \in r$, where r is a relation on n attributes and $x_1, x_2, \ldots, x_n$ are domain variables.
- $x \,\Theta\, y$, where x and y are domain variables and Θ is a comparison operator.
- $x \,\Theta\, c$, where x is a domain variable, Θ is comparison operator and c is a constraint in the domain of the attribute for which x is a domain variable.

We build up formulae from atoms by using the following rules :

- An atom is a formula.
- If P_1 is a formula, then so are $\neg P_1$ and (P_1).
- If P_1 and P_2 are formulae, then so are $P_1 \vee P_2, P_1 \wedge P_2$, and $P_1 \Rightarrow P_2$.
- If $P_1\ (x)$ is a formula in x, where x is a domain variable, then

$$\exists x\ (P_1(x)) \text{ and } \forall x\ (P_1\ (x))$$

Examples :

(1) Find the loan number, branch name, and amount for loans of over Rs. 12,000.

$\{<l, b, a> \mid <l, b, a> \in \text{loan} \wedge a > 12000\}$

(2) Find all loan numbers for loans with an amount greater than Rs. 12000

$\{<l> \mid \exists b, a\ (<l, b, a> \in \text{loan} \wedge a > 12000)\}$

(3) Find the names of all customers who have a loan from the SBI branch and find the loan amount

$\{<c, a> \mid \exists l <c, l> \in \text{borrower} \wedge \exists b\ (<l, b, a> \in \text{loan} \wedge b = \text{“SBI”})\}$

(4) Find the names of all customers who have a loan, an account, or both at the SBI branch.

$$\{<c> \mid \exists\, l\,(<c,l> \in \text{borrower} \wedge \exists\, b,a\,(<l,b,a> \in \text{loan} \wedge b = \text{“SBI”}))$$
$$\vee\, \exists\, a\,(<c,a> \in \text{depositor} \wedge \exists\, b,n\,(<a,b,n> \in \text{amount} \wedge b = \text{“SBI”}))\}$$

(5) Find the names of all customers who have an account at all the branches located in New Delhi.

$$\{<c> \mid \exists\, n\,(<c,n> \in \text{customer}) \wedge \forall\, x,y,z\,(<x,y,z> \in \text{branch} \wedge y = \text{“New Delhi”})$$
$$\Rightarrow \exists\, a,b\,\,(<a,x,b> \in \text{account} \wedge <c,a> \in \text{depositor}))\}$$

Q. *Retrieve the name and address of all employees who work for the 'computer' department.*

Ans. $\{qsv \mid (\exists\, z)\,(\exists\, l)\,(\exists\, m)$ (EMPLOYEE $(qrstuvwxyz)$ AND DEPARTMENT (l_{mno}) AND l = 'COMPUTER' AND $(m = z))\}$

2.7 INTRODUCTION TO SQL

Structured Query Language (SQL) is a language that provides an interface to relational database systems.

In common usage SQL also encompasses :

- DML (Data Manipulation Languages) for INSERTs, UPDATEs, DELETEs
- DDL (Data Definition Language) used for creating and modifying tables and other database structures.

Features of SQL :

(1) SQL can be used by a range of users, including those with little or no programming experience.
(2) It is a non procedural language.
(3) It reduces the amount of time required for creating and maintaining systems.
(4) It is an English-Like Language.

Components of SQL : There are following components of SQL.

(1) DDL

It is a set of SQL commands used to create, modify and delete data base structure but not data.

For examples :

(i) **CREATE :** To create objects in the database.

(ii) **ALTER :** Alters the structure of the database.

(iii) **DROP :** Delete objects from the database.

(iv) **TRUNCATE :** Remove all records from a table, including all spaces allocated for the records are removed.

(v) **COMMENT :** Add comments to the data dictionary.

(2) DML (Data Manipulation Language)

It is the area of SQL that allows changing data within the database.

For Examples :

(i) **INSERT :** Insert data into a table.

(ii) **UPDATE :** Updates existing data within a table.

(iii) **DELETE :** Deletes all records from a table, the space for the records remain.

(iv) **LOCK TABLE :** Control concurrency.

(3) DCL (Data Control Language)

It is the components of SQL statements that control access to data and to the database.

For Examples :

(i) **COMMIT** : Save work done

(ii) **SAVE POINT** : Identify a point in a transaction to which you can later roll back.

(iii) **ROLL BACK** : Restore database to original since the last COMMIT.

(iv) **GRANT/REVOKE** : Grant or take back permissions to or from the oracle users.

(v) **SET TRANSACTION** : Change transaction options like what rollback segment to use.

(4) DQL (Data Query Language)

It is the component of SQL statement that allows getting data from the database and imposing ordering upon it.

For example :

(1) **SELECT** : Retrieve data from the database.

2.7.1 Data Types

- Data types comes in several forms and sizes, allowing the programmer to create tables suited to the scope of the project.
- **CHAR (Size)** : This data type is used to store character strings values of fixed length. The maximum number of characters this data type can hold is 255 characters.

e.g., In case of 'Name Char (15)', then data held in the variable Name is only 15 characters in length.

- **VARCHAR (Size)/VARCHAR2 (size)** : This data type is used to store variable length alphanumeric data. The maximum type this can hold is 2000 characters.
- **NUMBER (P, S)** : The NUMBER data type is used to store numbers. The precision (*P*), determines the maximum length of the data, whereas the scale, (*S*), dertermines the number of places to the right of the decimal. The maximum precision (*P*), is 38 digits.
- **DATE** : This data type is used to represent date and time. The standard format is DD-MM-YY as in 26-June-07.
  The Date time stores date in the 24-hour format. By default, the time in a date field is 12 : 00 : 00 AM, if no time portion is specified.
- **LONG** : The LONG data type is used to store variable length character strings containing upto 2 GB.
  LONG data can be used to store arrays of binary data in ASCII format.
- **RAW/LONG RAW** : The RAW/LONG RAW data types is used to store binary data, such as digitized picture or image.
  RAW data type can have a maximum length of 255 bytes.
  LONG RAW data type can contain up to 2 GB.
- **ROWID (rowid)** : The format of the rowid is :

  BBBBBBB . RRRR . FFFFF

  where BBBBBBB is the block in the database file;
  RRRR is the row in the block;
  FFFFF is the data base file.
- **Boolean** : Valid in PL/SQL but this data type does not exist in oracle 8*i* or oracle 9*i*.

2.7.2 Types of SQL Commands

(1) The Create table Command :

Syntax : CREATE TABLE table name

(Column name datatype (size), column name data type (size),
column name datatype (size));

Example : Create a Employee Table.

CREATE TABLE Employee (E-ID number (6),
ENAME char (15), ADDRESS varchar (15),
CITY char (15), STATE char (15), PIN CODE number (6));

Output : Table Created

(2) Create a Student table

CREATE TABLE Student (Roll-No number (15), Name char (15),
Address varchar (15), Sex char (2), City char (15),
Phone number (15), State char (15));

Output : Table Created

2.7.3 Insertion of Data Into Tables

Once a table is created, the most natural thing to do is load this table with data to be manipulated later.

Syntax : INSERT INTO table name
(Column name 1, Column name 2 ...)
VALUES (expression, expression);

Example :

INSERT INTO Employee (E-ID, ENAME, ADDRESS, CITY,
STATE, PINCODE) VALUES (115, 'VIJAY', 'C-6 Gupta Road',
'New Delhi', 'DELHI');

Another method : INSERT INTO Employee VALUES
(& E-ID, '& NAME', '& ADDRESS', '& CITY', '& STATE');

Note :

(i) The character or varchar expression must be enclosed in single quotes (').

(ii) In the insert into SQL statement the columns and values have a one to one relationship.

2.7.4 Select Command

Once data has been inserted into a table, the next most logical operation would be to view what has been entered. This is achieved by SELECT SQL Verb.

(i) View global table data the syntax is :

SELECT * FROM table name;

e.g., SELECT * FROM Employee;

(ii) Retrieve ID, name, city of the employee

SELECT E-ID, ENAME, CITY FROM Employee;

Selected Columns and Selected Rows :

WHERE Clause

Syntax : SELECT * FROM table name
WHERE search condition;

e.g., (i) SELECT * FROM Employee WHERE
E-ID > 112;

(ii) SELECT Roll-No, Name FROM
Student WHERE Roll No. < = 150;

2.7.5 Elimination of Duplicates from the Select Statement

A table could contain duplicate rows. We can eliminate using select statement.

Syntax : SELECT DISTINCT Column name 1,
Column name 2 FROM table name;

Syntax : SELECT DISTINCT * FROM Table name

Example : (1) Select only unique rows from the table student :

SELECT DISTINCT * FROM Student;

2.7.6 Sorting Data in a Table

Oracle allows data from a table to be viewed in a sorted order. The rows retrieved from the table will be sorted in either ascending or descending order depending on the condition specified in the select statement.

Syntax : SELECT * FROM table name ORDER BY Column name 1, Column name 2 [Sort order];

Example : Retrieve all rows from student and display this data sorted on the value contained in the field Roll-No. in ascending order;

SELECT * FROM Student ORDER BY Roll-No;

Note : Oracle engine sorts in ascending order by default.

Example : For viewing the data in descending sorted order the word desc.

SELECT * FROM Student ORDER BY Roll-No desc;

2.7.7 Creating a Table from a Table

Syntax : CREATE TABLE Table name
[(Column name, Column name)]
AS SELECT column name, column name FROM Table name;

Example : Create a table student 1 from student.

CREATE TABLE Student 1
(SRoll-No, SName, Address, Sex, City, Ph.No, State)
AS SELECT Roll-No, Name, Address, Sex, City, Ph. No., State FROM Student;

2.7.8 Inserting Data Into a Table from Another Table

Syntax : INSERT INTO Table name SELECT column name, column name, FROM table name;

Example : Insert into table student 1 from the table student;

INSERT INTO student 1 SELECT Roll-No, Name, Address, Sex, City, Ph-No, State FROM Student;

Insertion of a Data Set Into a Table from Another Table

Syntax : INSERT INTO table name
SELECT column name, column name FROM table name
WHERE Column = Expression;

Example : Insert records into the table student 1 from the table student where the field Roll-No contains the value '115';

INSERT INTO Student 1 SELECT Roll-No, Name, Address, Sex, City, Ph-No, State FROM Student

WHERE Roll-No = '115';

2.7.9 Delete Operations

The DELETE commands deletes rows from the table that satisfies the condition provided by its WHERE clause, and returns the number of records deleted.

The Verb DELETE in SQL is used to remove rows from table. To remove

- All the rows from a table.

OR

- A select set of rows from a table.

Removal of All Rows :

Syntax : DELETE FROM table name;

Example : (1) Delete all rows from the table student

DELETE FROM Student;

Removal of a Specified Rows

Syntax : DELETE FROM table name WHERE search condition;

Example : Delete rows from the table student where the Roll-No > 115.

DELETE FROM student where Roll-No > 115;

2.7.10 Update Command

Updating the Contents of a Table : The UPDATE command is used to change or modify data values in a table.

To update :

- All the rows from a table.

OR

- A select set of rows from a table.

Updating of All Rows :

Syntax : UPDATE table name

```
          SET column name = expression,
              Column name = expression;
```

Example : Give every employee a bonus of 10%. Update the values held in the column net-salary.

```
UPDATE Employee
    SET Netsal = net-salary + Basic salary * 0.10;
```

Updating Records Conditionally :

Syntax : UPDATE table name

```
          SET column name = expression,
              column name = expression
      WHERE column name = expression;
```

Example : Update the table student change, the contents of the field name to 'Vijay Krishna' and the contents of field Address to 'Gr. Noida' for the record identified by the field Roll-No containing the value 115;

```
UPDATE student
          SET name = .'Vijay Krishna',
                    Address = 'Gr. Noida'
             WHERE Roll-No = 115;
```

2.7.11 Modifying the Structure of Tables

Adding New Columns :

Syntax : ALTER TABLE table name

```
          ADD (New column name data type/size)
            New column name data type (size...);
```

Example : Add the field Fax which is a field that can hold number upto 15 digits in length and Mobile-No, which is a field that can hold a number upto 10 digits in length.

```
ALTER TABLE student
     ADD (Mobile-No number (10), Fax number (15));
```

Modifying Existing Columns :

Syntax : ALTER TABLE table name

```
     MODIFY (column name, new data type (New size))
```

Example : Modify the field fax of the table student to now hold maximum of 25 character values.

```
ALTER TABLE student
               MODIFY (Fax Varchar (25));
```

Limitation of the ALTER TABLE : Using the ALTER TABLE clause the following tasks cannot be performed :

- Change the name of the table.
- Change the name of the column.
- Drop a column.
- Decrease the size of a column if table data exists.

2.7.12 RENAMING Command

To rename a table, the syntax is :

Syntax : RENAME old table name to New table name

Example : Rename the table Employee to Employee 1;

```
          RENAME Employee TO Employee 1;
```

2.7.13 Destroying Tables

Syntax : DROP TABLE table name;

Example : Destroy the table Employee and all the data held in it; DROP TABLE Employee;

DESCRIBE Command : To find information about the column defined in the table use the following syntax;

Syntax : DESCRIBE table name;

This command displays the column names, the data types and the special attributes connected to the table.

Example : Displays the columns and their attributes of the table student.

```
                    DESCRIBE Student;
```

2.7.14 Logical Operators

There are following logical operators used in SQL.

- **The AND Operator :** The oracle engine will process all rows in a table and display the result only when all of the conditions specified using the AND operator are satisfied.

Example : Retrieve the contents of the columns product-no, profit-percent, sell-price from the table product-master where the values contained in the field profit percent in between 10 and 20.

```
SELECT Product-no, profit-percent, sell-price
FROM Product-master
WHERE profit-percent > = 10 AND profit-percent < = 20;
```

- **The OR Operator :** The oracle engine will process all rows in a table and diplay the result only when any of the conditions specified using the OR operator are satisfied.

Example : Retrieve the all fields of the table student where the field Roll-No has the value 115 OR 200;

```
SELECT Roll-No, Name, Address, Sex, City, Ph-No,
State FROM Student WHERE (Roll-No = 115 OR Roll-No = 200);
```

- **The NOT Operator :** The oracle engine will process all rows in a table and diplay the result only when none of the conditions specified using the NOT operator are satisfied.

Example : Retrieve specified student information for the clients, who are NOT in 'New-Delhi' OR 'Noida';

```
SELECT Roll-No, Name, Address, City, State
                FROM Student WHERE NOT
          (City = 'New Delhi' OR City = 'Noida');
```

2.7.15 Range Searching

BETWEEN Operator :

Example : Retrieve Roll-No, Name, Address, Ph-No, State from the table student where the values contained within the field Roll-No is between 100 and 200 both inclusive.

```
SELECT Roll-No, Name, Address, Ph-No, State FROM Student
WHERE Roll-No BETWEEN 100 AND 200;
```

Pattern Matching :

The use of the LIKE predicate : The LIKE predicate allows for a comparison of one string value with another string value, which is not identical.

For the character data types :

The percent sign (%) matches any string.

The underscore (_) matches any single character.

Example : (1) Retrieve all information about students whose names begins with the letters 'vi' from student table.

```
SELECT * FROM Student
WHERE Name LIKE 'vi %';
```

(2) Retrieve all information about students where the second character of names are either 'V' or 'S'

```
SELECT * FROM Student
WHERE Name LIKE '_V%' OR
    Name LIKE '_S%';
```

The IN predicates : In case of value needs to be compared to a list of values then the IN predicate is used.

Example : Retrieve the Roll-No, Name, Address, City, Ph-No from the table student where name is either Vijay or Santosh or Gopal or Sanjay.

```
        SELECT Roll-No, Name, Address, City, Ph-No
FROM Student
  WHERE Name IN ('Vijay', 'Santosh', 'Gopal', 'Sanjay');
```

The NOT IN prediates : The NOT IN predicate is the opposite of the IN predicate. This will select all the rows where values do not match all of the values in the list.

Example :

```
SELECT Roll-No, Name, Address, City, Ph-No FROM Student
WHERE Name NOT IN ('Vijay', 'Santosh', 'Gopal', 'Sanjay');
```

2.7.16 UNIQUE KEY Constraint Defined at the Table Level

Syntax : UNIQUE (column name);

Example : Create a table student such that the unique key constraint on the column Roll-No is described as a table level constraint.

```
CREATE TABLE Student
(Roll-No number (5), Name char (15), Sex char (2),
Ph-No number (10), Address varchar (15), City char (10),
                    State char (15), UNIQUE (Roll-No));
```

2.7.17 PRIMARY KEY Constraint Defined at the Column Level

Syntax : PRIMARY KEY (column);

Example : Create a table student such that the primary key constraint on the column Roll-No is described as a table level constraint.

```
CREATE TABLE Student
(Roll-No number (5), Name char (15), Sex char (2),
Ph-No number (10), Address varchar (15), City char (10),
               State char (15), PRIMARY KEY (Roll-No));
```

2.7.18 FOREIGN KEY Constraint Defined at the Table Level

Syntax : FOREIGN KEY (column name [column name])

REFERENCES table name [column name [, column name]);

Example : Create table sales-order with primary key as detlorder-no and product-no and foreign key at table level as detloder-no referencing column order-no in the sales-order table.

```
            CREATE TABLE sales-order
(detlorder-no varchar (6), product no varchar (6),
            qty-order number (7), product-rate number (8, 2),
PRIMARY KEY (detlorder-no, product-no),
FOREIGN KEY (detlorder-no)
REFERENCE sales-order);
```

2.7.19 Aggregate Functions

Aggregae functions are functions that take a collection of values as input and return a single value. SQL offers five suit-in aggregate functions.

- Average : AVG

- Minimum : MIN
- MAXIMUM : MAX
- Total : SUM
- Count : COUNT :

Syntax : COUNT ([DISTINCT/ALL] expr)

Returns the number of rows where 'expr' is not NULL.

Example : SELECT COUNT (Product-no) "No. of product".
FROM product-master;

Output : No. of product = 10

AVG : Syntax : AVG ([DISTINCT/ALL]n)

Example : SELECT AVG (sell-price) "Average"
FROM product-master;

Output :
```
Average
(205.137)
```

MIN :

Syntax : MIN ([DISTINCT\ALL] expr)

Example : SELECT MIN (Bal-due) "Minimum Balance"
FROM client-master;

Output :
```
Minimum Balance
      3
```

MAX :

Syntax : MAX ([DISTINCT\ALL] expr)

Example : SELECT MAX (Bal-due) "Maximum"
FROM client-master;

Output :
```
Maximum
 25000
```

SUM :

Syntax : SUM ([DISTINCT\ALL]n)

Example : SELECT SUM (Bal-due) "Total Balance Due"
FROM client-master;

Output :
```
Total Balance Due
      30000
```

POWER :

Syntax : POWER (m, n)

Returns 'm' raised to 'nth' power 'n' must be an integer, else an error is returned.

Example : SELECT POWER (3,2) "RESULT =" FROM math;

Output : RESULT = 9

ABS :

Syntax : ABS (n)

Returns the absolute value of 'n'

Example : SELECT ABS (–10) "Absolute =" FROM math;

Output : Absolute = 10

LOWER :
Syntax : LOWER (char)
Returns char, with all letter in lowercase
Example : SELECT LOWER ('VIJAY KRISHNA') "Lower Case" FROM math;
Output :
```
 Lower Case
Vijay Krishna
```
UPPER :
Syntax : UPPER (char)
Returns char with all letters forced to upper case
Example : SELECT UPPER ('Vijay Krishna') "Result" FROM math;
Output :
```
    Result
VIJAY KRISHNA
```
INITCAP :
Syntax : INITCAP (char)
Returns string with the first letter in upper case
Example : SELECT INITCAP ('VIJAY KRISHNA') "Result" FROM math;
Output :
```
   Result
Vijay Krishna
```
SQRT :
Syntax : SQRT (n)
Returns square root of 'n'. If $n < 0$, NULL. SQRT returns a real result.
Example : SELECT SQRT (49) "Square root ="
FROM Math;
Output : Square Root = 7
LENGTH :
Syntax : LENGTH (char)
Returns the length of char.
Example : SELECT LENGTH ('VIJAY') "Length"
FROM Match;
Output :
```
Length
  5
```
LTRIM : Syntax : LTRIM (char [, set])
- Removes character from the left of char with initial characters removed upto the first character not in set.

Example : SELECT LTRIM ('VIJAY', 'V')
"Result" FROM Math;
Output :
```
Result
 IJAY
```
RTRIM : Syntax : RTRIM (char, [set])
- Returns char, with final characters removed after the last character not in the set.
- *Example :* SELECT RTRIM ('VIJAYA', 'A')

"Result" FROM Math;
Output :
```
Result
VIJAY
```

2.7.20 Subqueries

A subquery is a form of an SQL statement that appears inside another SQL statement. It is also turned as Nested query. The statement containing a subquery is called a parent statement. The parent statement uses the rows returned by the subquery.

It can be used by the following commands :

- To insert records in a target table.
- To create tables and insert records in the table created.
- To update records in a target table.
- To create view.
- To provide values for conditions in WHERE, HAVING, IN etc. used with SELECT, UPDATE, and DELETE statement.

Example : Retrieve all orders placed by client named 'VIJAY KRISHNA' from the sales-order table.

Table name : sales-order

Order_No	Client_No	Order_Date
A1901	B004	12–Apr–2007
A1902	B002	14–Apr–2007
A1903	B007	03–June–2007
A1904	B005	20–May–2007
A1905	B007	12–July–2007

Table name : client-master

Client_No	Name	Bal Due
B001	Ashok	400
B002	Gopal	300
B003	Sanjay	200
B004	Santosh	100
B005	Rahul	0
B006	Vivek	0
B007	VIJAY KRISHNA	0

```
SELECT * FROM Sales-order
        WHERE client-no = (SELECT client-no
FROM client-master
           WHERE Name = 'VIJAY KRISHNA');
```

Output :

Order_No	Client_No	Order_Date
A1903	B007	03–June–2007
A1905	B007	12–July–2007

2.7.21 Joins

Joining Multiple Tables (Equi Joins) : Sometimes we require to treat multiple tables as though they were a single entity. Then a single SQL sentence can manipulate data from all the tables to achieve this, we have to join tables. Tables are joined on columns that have the same data type and data width in the table.

Example : Retrieve the order numbers, client names and their order dates from the client-master and sales-order tables. The order date should be displayed in 'DD/MM/YY' format and sorted in ascending order.

Table name : sales-order

Order_No	Client_No	Order_Date
A1901	B006	12–Apr–2007
A1902	B002	14–Apr–2007
A1903	B001	03–June–2007
A1904	B005	20–May–2007
A1905	B004	12–July–2007
A1906	B001	

Table name : client-master

Client_No	Name	Bal Due
B001	Ashok	400
B002	Gopal	300
B003	Sanjay	200
B004	Santosh	100
B005	Rahul	0
B006	Vivek	0
B007	Vijay Krishna	0

```
SELECT order-no, to-char (order-date 'DD/MM/YY',) "Order Date"
FROM sales-order, client-master
WHERE client-master. client-no = sales-order.client-no
ORDER BY to-char (order-date, 'DD/MM/YY);
```

Output :

Order_No	Name	Order_Date
A1903		
A1906		
A1901		
A1904		
A1905		
A1902		

2.7.22 UNION Clause

The union clause merges the output of two or more queries into a single set of rows and columns.

Example : Retrieve the names of all the clients and salesman in the city of 'New Delhi' from the tables client-master and salesman-master.

Table name : client-master

Client_No	Name	City
A0001	Vijay Krishna	New Delhi
A0002	Gopal Krishna	Mumbai
A0003	Santosh Kumar	New Delhi
A0004	Sanjay Kumar	Calcutta
A0005	Ajay Kumar	Mumbai
A0006	Vishal	Varanasi
A0007	Vivek	Noida

Table name : salesman-master

Salesman_No	Name	City
B0001	Manish Kumar	New Delhi
B0002	Kiran Kumar	Mumbai
B0003	Nitin Kumar	New Delhi
B0004	Tushar Kumar	Calcutta

```
SELECT salesman-no "ID", name
                    FROM salesman-master
WHERE City = 'New Delhi'
UNION
SELECT client No "ID", Name
                    FROM client-master
WHERE City = 'New Delhi';
```

Output :

ID	Name
A0001	Vijay Krishna
A0003	Santosh Kumar
B0001	Manish Kumar
B0003	Nitin Kumar

The Restrictions on using a union are as follows :

- Number of columns in all the queries should be the same.
- The datatype of the columns in each query must be same.
- Unions cannot be used in subqueries.
- Aggregate functions cannot be used with union clause.

2.7.23 Intersect Clause

The output in an intersect clause will include only those rows that are retrieved by both the queries.

Example : Retrieve the salesman name in 'New Delhi' whose efforts have resulted into atleast one sales transaction.

Table name : salesman-master

Salesman_No	Name	City
A0001	Vijay Krishna	New Delhi
A0002	Santosh Kumar	Mumbai
A0003	Gopal Krishna	New Delhi
A0004	Sanjay Kumar	Noida

Table name : sales-order

Order_No	Order_Date	Salesman_No
B0001	12–Apr–2007	A0001
B0002	14–Apr–2007	A0003
B0003	03–June–2007	A0001
B0004	05–June–2007	A0004
B0005	02–July–2007	A0003
B0006	12–July–2007	A0002

```
SELECT Salesman_No, Name FROM salesman-master WHERE City = 'New Delhi'
INTERSECT
SELECT salesman-master.Salesman-No, Name FROM salesman-master, sales-order
WHERE salesman-master.Salesman-No = sales-order.Salesman-No;
```

Output :

Salesman_No	Name
A0001	Vijay Krishna
A0003	Gopal Krishna

Note : For the first query.

```
SELECT salesman No, Name
FROM salesman-master,
WHERE City = 'New Delhi';
```

Salesman_No	Name
A0001	Vijay Krishna
A0003	Gopal Krishna

For the second query

```
SELECT salesman-master.Salesman No, name
FROM salesman-master, sales-order
WHERE salesman-master.Salesman No = sales-order.Salesman No;
```

Salesman_No	Name
A0001	Vijay Krishna
A0003	Gopal Krishna
A0001	Vijay Krishna
A0004	Sanjay Kumar
A0003	Gopal Krishna
A0002	Santosh Kumar

2.7.24 MINUS CLAUSE

The Minus clause outputs the rows produced by the first query, after filtering the rows retrieve by the second query.

Example : Retrieve all the product numbers of non-moving items from the product-master table.

Table name : sales-order

Order_No	Product_No
A0001	B0001
A0001	B0004
A0001	B0006
A0002	B0002
A0002	B0005
A0003	B0003
A0004	B0001
A0005	B0006
A0005	B0004
A0006	B0006

Table name : Product-master

Product_No	Description
B0001	1.44 Drive
B0002	128 MB RAM
B0003	Keyboard
B0004	Mouse
B0005	Monitors
B0006	HDD
B0007	CD Drive
B0008	128 MB RAM
B0009	Monitors

```
SELECT Product_No FROM product-master
MINUS
SELECT Product_No FROM sales-order
```

Output : **Product No**

```
B0007
B0008
B0009
```

Note : for the first query

SELECT Product_No FROM Product-master

A :

Product_No
B0001
B0002
B0003
B0004
B0005
B0006
B0007
B0008
B0009

For the second query

SELECT Product_No FROM sales-order

B :

Product_No
A0001
A0004
A0006
A0002
A0005
A0003
A0001
A0006
A0004
A0006

Now

A – B =

Product_No
A0007
A0008
A0009

2.8 VIEWS

An interesting fact about a view is that it is stored only as a definition in Oracle's system catalogue. When a reference is made to a view; its definition is scanned, the base table is opened and the view created on top of the base table.

Hence a view holds no data at all, until a specific call to the view is made. This reduces redundant data. On the HDD to a very large extent. When a view is used to manipulate table data, the underlying base table will be completely invisible. This will give the level of data security required.

The reasons why views are
Created are :

- When Data security is required.
- When Data redundancy is to be kept to the minimum while maintaining data security.
- Views can provide logical data independence.
- Views provide "macro" capability.

Creation of views :

Syntax :

```
CREATE VIEW View name As
SELECT column name, column name
FROM table name
WHERE column name = expr. List;
GROUP BY grouping criteria
HAVING predicate
```

Example : (i) Create a view on the salesman-master table for the sales department.

```
CREATE VIEW VW-sales AS
SELECT * FROM Salesman-master;
```

(ii) Create a view on the client-master table for the Administration Department

```
CREATE VIEW VW-clientadmin AS
SELECT name, address, city, pin code State FROM client-master;
```

Renaming the columns of a view : The columns of the view can take on different names from the table columns, if required.

Example : CREATE VIEW VW-clientadmin AS SELECT name name 1, address address 1, city, pin code PIN, state FROM client-master;

2.9 INDEXES

Indexing a table is an 'access strategy', that is, a way to sort and search records in the table. Indexes are essential to improve the speed with which the records can be located and retrieved from a table.

- Indexing involves forming a two dimensional matrix completely independent of the table on which the index is being created.
- A column, which will hold sorted data, extracted from the table on which the index is being created.
- An address field that identifies the location of the record in the oracle database. This address field is called Rowid.
- When data is inserted in the table the oracle engine inserts the data value in the index.

For every data value held in the index the oracle engine inserts a unique rowid value. This rowid indicates exactly where the record is stored in the table.

Hence once the appropriate index data values have been located, the oracle engine locates an associated record in the table using the rowid found in the table.

Address Field in the Index : The address field of an index is called ROWID.

The ROWID format used by Oracle as follows

BBBBBBB.RRRR.FFFF

where FFFF is a unique number given by the oracle engine to each data file.

For Ex., database can be a collection of data files as follows :

Data File Name	Data File No	Size of Data File
Student-Ora	1	50 MB
Staff-Ora	2	10 MB
Temporcl-Ora	3	30 MB
Sysorcl-Ora	4	40 MB

BBBBBBB : Each data file is further divided into 'Data Block' and each block is given a unique number. The unique number assigned to the first data block in a data file 0.

Thus block number can be used to identify the data block in which a record is stored. BBBBBBB is the block number in which the record is stored.

RRRR : Each block can store one or more records. Thus each record in the data block is given a unique record number. The unique record number assigned to the first order in each data block is 0.

Thus record number can be used to identify a record stored in a block. RRRR is a unique record number.

Creation of Index : An index can be created on one or more column. Based on the number of column included in the index.

An index can be :

- Simple Index
- Composite Index

Simple Index : An index created on a single column of a table is called simple index.

Syntax : CREATE INDEX index name
ON table name (column name);

Example : create a simple index on a Roll-No columns of the student table.

```
CREATE INDEX IdX-Roll-No
    ON Student (Roll-No);
```

Composite Index : An index created on more than one column is called composite index.

Syntax : CREATE INDEX index name
ON table name (column name, column name);

Example : Create a composite index on the sales-order tables on column order-no and product-no.

```
CREATE INDEX idx-sales-order
ON sales-order (order-no, product-no)
```

2.10 ROW NUM IN SQL STATEMENT

For each row returned by a query, the ROW NUM pseudo column returns a number indicating the order in which oracle engine select the row from a table or set of joined rows.

First row selected has a ROW NUM of 1; The second has 2 and so on.

Limitation : ROW NUM can be used to limit the number of rows retrieved.

Example : Retrieve first 5 rows by using ROW NUM.

Table name : Student

Roll_No	Name
001	Vijay Krishna
002	Gopal Krishna
003	Santosh Kumar
004	Sanjay Kumar
005	Punit Kumar
006	Pravin Kumar
007	Pankaj Kumar
008	Tushar Kumar

```
SELECT ROW NUM, Roll-No, Name
FROM student
WHERE ROW NUM < 6;
```

Output :

RowNum	Roll_No	Name
1	001	Vijay Krishna
2	002	Gopal Krishna
3	003	Santosh Kumar
4	004	Sanjay Kumar
5	005	Punit Kumar

2.11 SEQUENCES

Oracle provides an object called a sequence that can generate numeric values. The value generated can have a maximum of 38 digits.

A sequence can be defined to

- Generate numbers in ascending or descending.
- Provide intervals between numbers.
- Caching of sequence number in memory.

Creating Sequences : The minimum information required for generating numbers using a sequence is.

- The starting number
- The maximum number that can be generated by a sequence.
- The increment value for generating the next number.

Syntax :

```
CREATE SEQUENCE Sequence name
[INCREMENT BY integer value
START WITH integer value
MAX VALUE integer value/
/NON MAX VALUE
MIN VALUE integer value/NON MIN VALUE
CYCLE/NO CYCLE
CACHE integer value/NO CACHE
ORDER/NO ORDER]
```

2.12 CURSOR

The Oracle Engine uses a work area for its internal processing in order to execute an SQL statement. This work area is called a cursor.

The data that is stored in the cursor is called the 'Active Data Set'.

The size of the cursor in memory is the size required to hold the number of rows in the Active Data Set.

e.g.,

SERVER RAM			
Active Data Set			
11	Vijay	Eng.	20000
12	Gopal	Eng.	20000
13	Santosh	Analyst	15000
14	Sanjay	Manager	15000

Contents of a cursor

When a user fires a select statement as :

```
SELECT EMP No, EName, Job, Salary
FROM Employee
WHERE Dept No = 20
```

The resultant data set in the cursor opened at server and will be displayed as shown above.

When a cursor is loaded with multiple rows via a query the oracle engine opens and maintain a row pointer. Depending on user requests to view data the row pointer will be relocated within the cursor's Active Data Set.

Types of Cursors : Cursors are classified depending on the circumstances under which they are opened.

These are following types

- Implicit Cursors
- Explicit Cursors
- **Implicit Cursors :** If the oracle engine for its internal processing has opened a cursor, they are known as Implicit cursors.
  That is a cursor which opens oracle engine for its internal processing is called Implicit cursor.
- **Explicit Cursor :** A user can also open a cursor for processing data as required. Such user defined cursors are known as 'Explicit Cursors'.

Attributes of Cursor : Both Implicit and Explicit cursors have four attributes.

(i) **% ISOPEN :** Returns TRUE if cursor is open, FALSE otherwise.

(ii) **% FOUND :** Returns TRUE if record was fetched successfully, FALSE otherwise.

(iii) **% NOT FOUND :** Returns TRUE if record was not fetched successfully FALSE otherwise.

(iv) **% ROW COUNT :** Returns number of records processed from the cursor.

Drawbacks of Implicit Cursors : The implicit cursor has the following drawbacks :

- It is less efficient than an explicit cursor.
- It is more vulnerable to data errors.

- It gives you less programmatic control.

Cursor Declaration : A cursor is defined in the declarative part of a PL/SQL block. This is done by naming the cursor and mapping it to a query. When a cursor is declared, the oracle engine is informed that a cursor of the said name needs to be opened. The declaration is only an intimation. There is no memory allocation at this point in time.

CURSOR cursor-name [([Paramater [, Parameter ... 1)]
[RETURN return-specification]
IS SELECT-Statement.
Where Cursor-name : The Name of Cursor
return-specification : An optional RETURN clause for the cursor.

SELECT-Statement : Any valid SQL SELECT statement.

The three commands used to control the cursor subsequently are open, fetch and close.

Fetch : A Fetch statement moves the data held in the Active Data Set into memory variable. The fetch statement is placed inside a loop ... End loop construct, which causes the data to be fetched into the memory variables and processed until all the rows in the Active Data Set are processed.

Syntax : CURSOR cursor name IS SELECT Statement;

Opening a Cursor : Opening a cursor executes the query and creates the active set that contains all rows, which meet the query search criteria.

An open statement retrieves records from a database table and places the records in the cursor. A cursor is opened in the server's memory.

Syntax : OPEN cursor name;

Closing A Cursor : The close statement disables the cursor and the active set becomes undefined. This will release the memory occupied by the cursor and its Data set both on the client and on the server.

Syntax : CLOSE Cursor Name.

2.13 DATABASE TRIGGERS

A trigger consists of PL/SQL code, which defines some action that the database should take when some database related event occurs.

The Oracle Engine allows us to define procedures that are implicitly executed when an insert, update or delete statement is issued against the associated table. These types of procedures are called data base triggers.

Use of Database Triggers : These are following

- A trigger can permit DML statement against a table only if they are issued, during regular business hours.
- A trigger can also be used to keep an audit trait of a table, along with the operation performed and the time on which the operation was performed.
- Enforce complex security authorizations.

Basic parts of Trigger : A trigger has three basic parts

- A triggering event or statement
- A trigger restriction
- A trigger action

Triggering Event or Statement : It is a SQL statement that causes a trigger to be fired. It can INSERT, UPDATE or DELETE Statement for a specific table.

Trigger Restriction : A trigger restriction specifies a Boolean expression that must be TRUE for the trigger to fire. It is an option available for triggers that are fired for each row.

A trigger restriction is specified using a WHEN clause.

Trigger Action : A trigger action is the PL/SQL code to be executed when a trigerring statement is encountered and any trigger restriction evaluates to TRUE.

Types of Triggers : These are following types :

- **Row Triggers :** A row trigger is fired each time a row in the table is affected by the triggering statement.

Example : If an UPDATE statement updates multiple rows of a table, a row trigger is fired once for each row affected by the UPDATE statement.

Row triggers should be used when some processing is required whenever a triggering statement affects a single row in a table.

Statement Triggers : A statement trigger is fired once on behalf of the triggering statement, independent of the number of rows the triggering statement affects.

Statement triggers should be used when a triggering statement affects rows in a table but the processing required is completely independent of the number of rows affected.

Before Triggers : BEFORE triggers execute the trigger action before the triggering statement. These types of triggers are commonly used in the following situation :

(i) BEFORE triggers are used when the trigger action should determine whether or not the triggering statement should be allowed to complete.

(ii) BEFORE triggers are used to derive specific column values before completing a triggering INSERT or UPDATE statement.

AFTER Triggers : AFTER triggers executes the trigger action after the triggering statement is executed.

These types of triggers are commonly used in the following situation :

(i) AFTER triggers are used when you want the triggering statement to complete before executing the trigger action.

(ii) If a BEFORE trigger is already present, an AFTER trigger can perform different actions on the same triggering statement.

Note : When a trigger is fired, an SQL statement inside the trigger's PL/SQL code block can also fire the same or some other trigger. This is called 'Cascading triggers'.

Database Triggers V/s Procedures

There are very few differences between database triggers and procedures

- Triggers do not accept parameters whereas procedures can.
- A trigger is executed implicitly by the oracle engine itself upon modification of an associated table or its data. To execute a procedure, it has to be explicitly called the user.

Difference between procedure and function :

- A function must return only one value back to the caller. While procedure can never return a value back to the caller.

2.14 ORACLE PACKAGES

A package is an oracle object, which holds other objects within it. Objects commonly held within a package are procedures, functions, variables, constants, cursors and exceptions. It is a way of creating generic, encapsulated, re-usable code.

Component of an Oracle Package : A package has usually two components :

- A specification.
- A body

Specification : A packages specification declares the types, memory variable constants, exceptions, cursors, and sub programs that are available for use.

Body : A packages body full defines cursors, functions, and procedures and thus implements the specification.

Advantages of Packages : These are following advantages :

(i) Packages enable the organization of commercial applications into efficient modules.

(ii) Packages allow granting of privileges efficiently.

(iii) A package's public variables and cursors persist for the duration of the session. Therefore, all cursors and procedures that execute in this environment can share them.

(iv) Packages enable the overloading procedures and functions when required.

(v) Packages improve performance by loading multiple objects into memory at once.

(vi) Packages promote code reuse through the use of libraries that contain stored procedures and functions, thereby reducing redundant coding.

2.15 ASSERTIONS

An assertion is a predicate expressing a condition that we wish the database always to satisfy. Domain constraints and referential-integrity constraints are special forms of assertions. They are easily tested and apply to a wide range of database application.

An assertion in SQL takes the form

```
CREATE ASSERTION <assertion-name>
          Check <predicate>
```

Two examples of such constraints are :

- The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.
- Every loan has at least one customer who maintains an account with a minimum balance of Rs. 1000.00.

When an assertion is created, the system tests it for validity. If the assertion is valid, then any future modification to the database is allowed only if it does not cause that assertion to be violated.

Hence, assertions should be used with great care.

Solved Problems

Q.1. *What is a features of PL/SQL?*

Ans. Features of PL/SQL : These are following

(1) PL/SQL accepts ad hoc entry of statements

(2) It accepts SQL input from files.

(3) It provides a line editor for modifing SQL statements.

(4) It controls environmental settings.

(5) It formats query results into basic reports.

(6) It accesses local and remove databases.

Q.2. *What is difference between SQL and SQL * PLUS*

Ans.

SQL	SQL*Plus
(1) SQL is a language for communicating with the oracle server and other databases to access data.	SQL*Plus recognize SQL statements and sends them to the server.
(2) SQL is based on ANSI standard SQL.	SQL*Plus is the oracle proprietory interface for executing SQL statements.
(3) SQL manipulates data and table definitions in the database.	SQL*Plus does not allow manipulation of values in the database.
(4) SQL is entered into the SQL buffer on one or more lines.	SQL*Plus is entered one line at a time, not stored in the SQL buffer.
(5) It cannot be abbreviated.	It can be abbreviated.
(6) It uses a termination character to execute commands immediately.	It does not require termination characters; executes commands immediately.
(7) It uses functions to perform some formatting	It uses commands to format data.

Q.3. *Write the assertion for the following statement.* **(UPTU 2005-06)**

"Every loan has at least one customer who maintains an account with minimum balance of Rs. 1000 in banking system.

Ans. Create assertion balance-constraint check (not exists (select * from loan where not exists (select * from borrower, depositor, account

where loan.loan-number = borrower.loan-number

and borrower.customer-name = depositor.customer-name

and depositor.account-number = account.account-number

and account.balance > = 1000)));

Q.4. *Write the assertion for the following statement.*

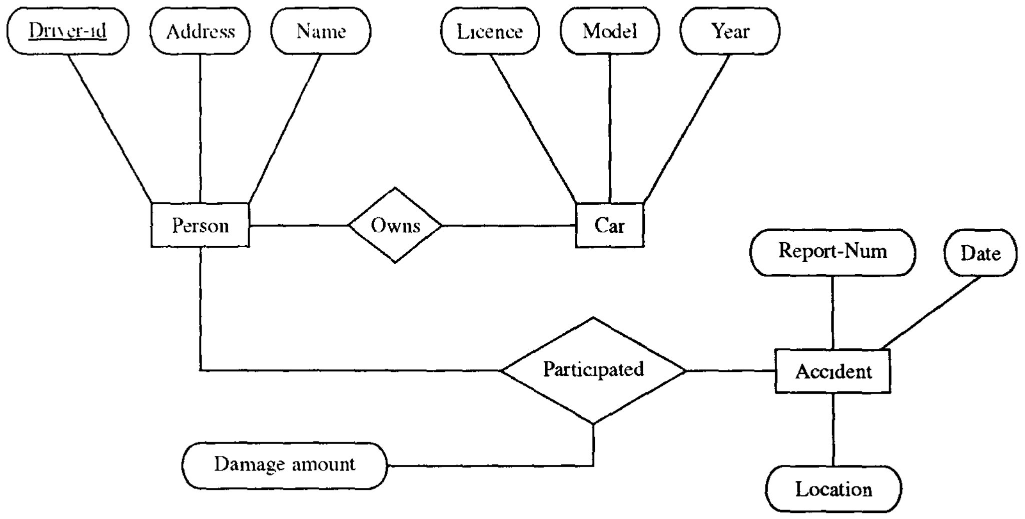
"The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch."

Ans. Create assertion sum-constraint check (not exists (select sum (amount) from loan where loan.branch-name = branch.branch-name)

> = (select sum (balance) from account where

account.branch-name = branch.branch-name)));

Q.5. *Design a relational data base corresponding to E-R diagram given* **(UPTU 2006)**



Ans. The relational database schema is as :

PERSON (driver_id, Name, address)

CAR (Licence, model, year)

ACCIDENT (Report-num, date, location)

OWNS (driver_id, licence)

PARTICIPATED (driver_id, report-num, licence, damage-amount)

EMPLOYEE (person-name, address, city)

COMPANY (company-name, city)

Q.6. *Consider the following relational database :* **(UPTU 2002, 03)**

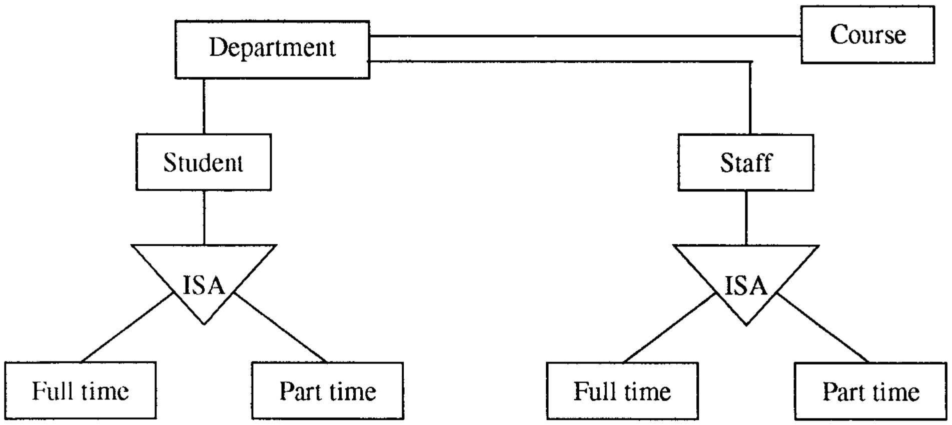
CUSTOMER (customer-name, street, customer city)
BRANCH (branch-name, assets, branch-city)
DEPOSIT (branch-name, account-number, customer name, balance)
Give SQL DDL definition of this data base.

Ans. CREATE TABLE CUSTOMER
(Customer-name char (20), street varchar (10), customer city char (15));
CREATE TABLE BRANCH
(Branch-name char (15), Assets varchar (20), Branch-city char (15));
CREATE TABLE DEPOSIT
(Branch-Name char (15), Account-number number (10),
Customer-name char (15), Balance number (10));

Q.7. *A university has many department. Each department may have full-time and part time students. Each department may float multiple courses for its own students. Each department has staff members who may be full time and part time.* **(UPTU 2005-06)**

Design a generalization, specialization hierarchy for the university.

Ans.



Q.8. *Consider the following scheme for PROJECT database*

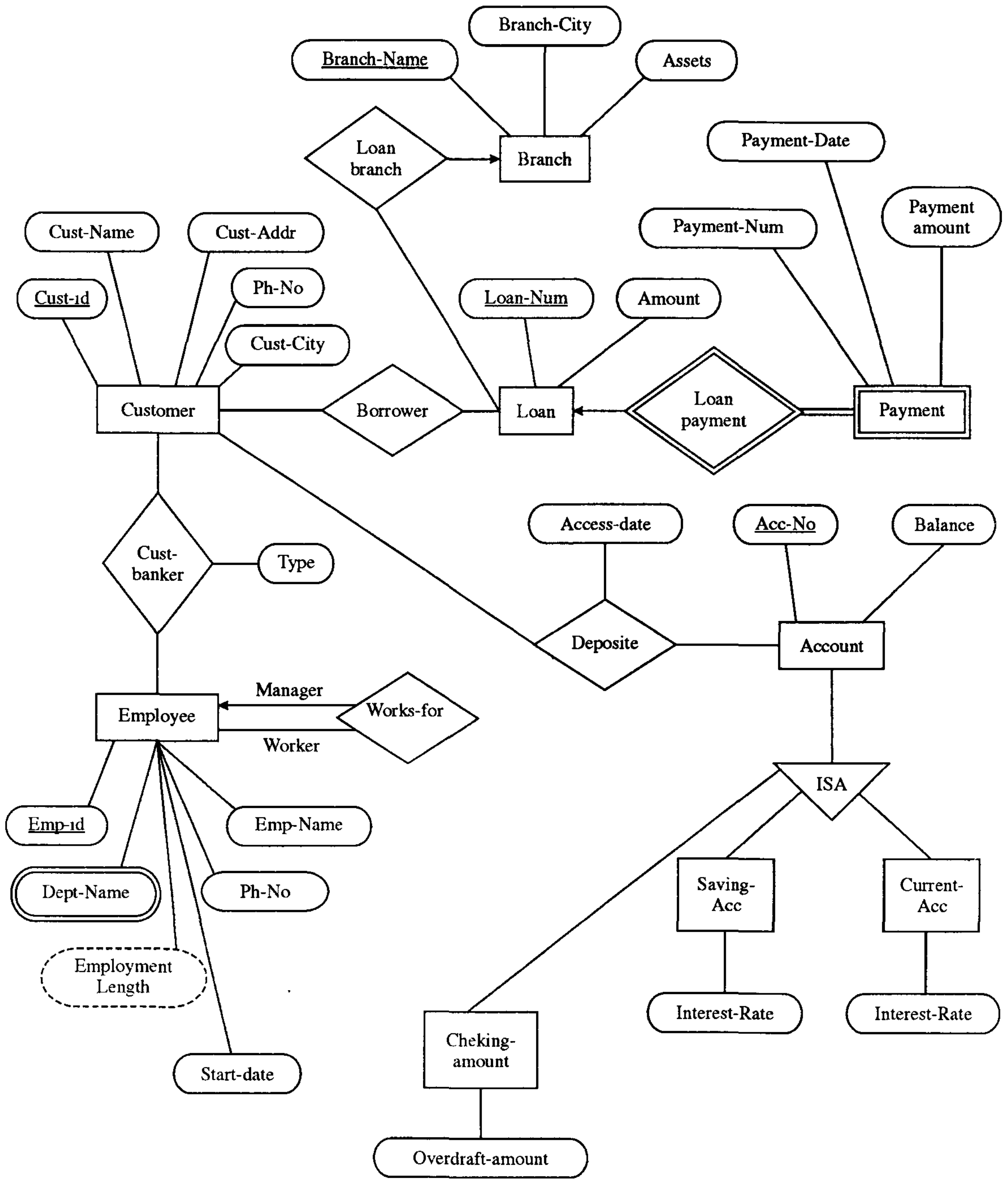
Project (Project-no, Project-name, Project-manager)
Employee (Employee-no, Employee-name)
Assigned to (Project-no, Employee-no)
Write SQL-DDL statement for PROJECT database
The SQL statement should clearly indicate the primary key and foreign keys. **(UPTU 2003-04)**

Ans. CREATE TABLE PROJECT
(Project-no, number (8) PRIMARY KEY, Project-name
varchar2 (20), Project-manager char (15));

- CREATE TABLE EMPLOYEE
  (Employee-no number (6) PRIMARY KEY, Employee-name char (30));
- CREATE TABLE ASSIGNED_TO
  (Project-no number (8), Employee-no number (6),
  PRIMARY KEY (Project-no, Employee-no),
  FOREIGN KEY (Project-no)
  REFERENCES (Project-1));

Q.9. *Draw E-R diagram for banking enterprise.* **(UPTU 2004, 05)**

Ans.

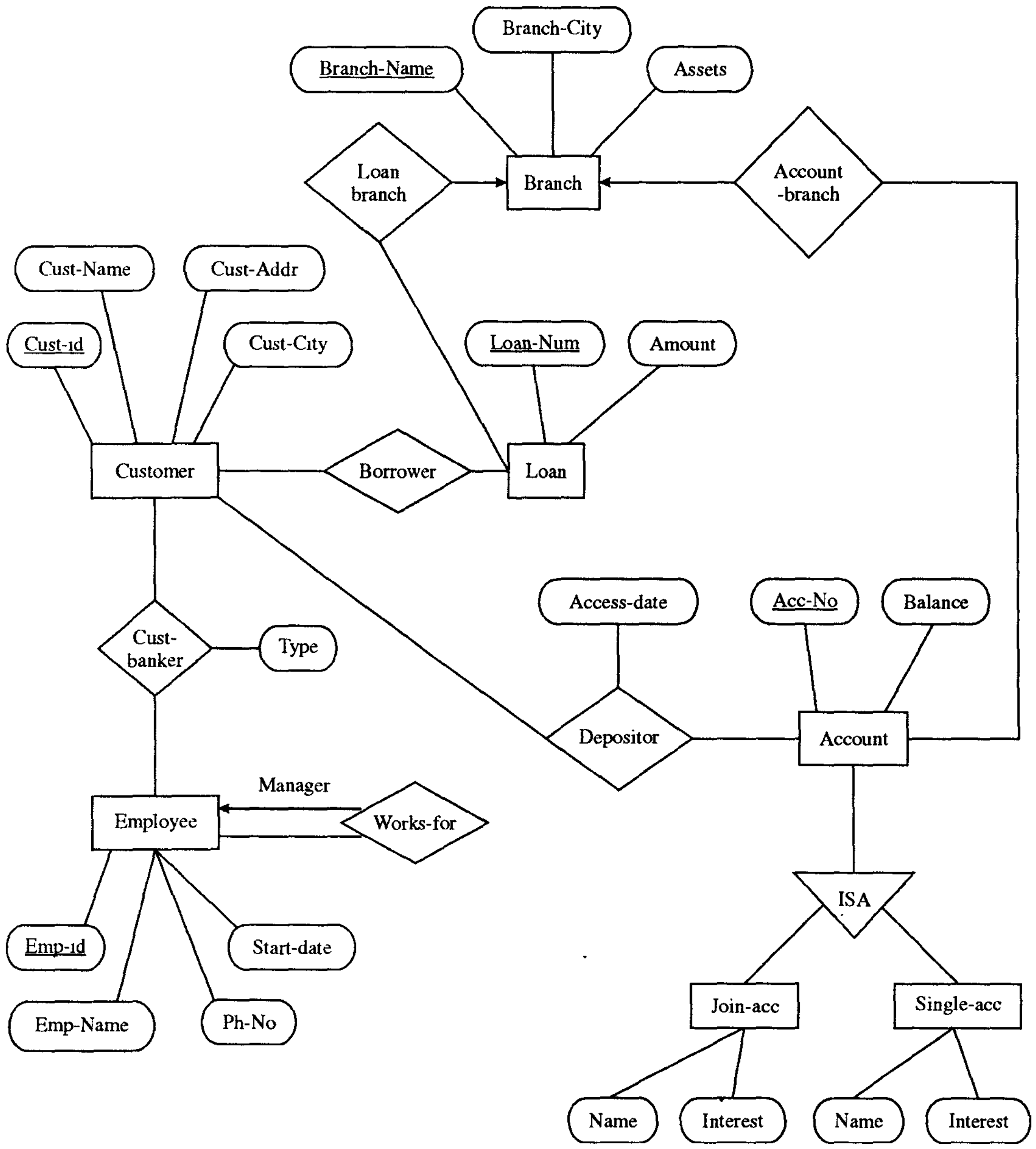


Q.10. *(a) Consider the following set of requirements for a bank databases :*

"A large bank has several branches at different places. Each branch maintains the account details of the customers. The customers may open join as well as single accounts. The bank also provides loan to the customer for different purposes. Bank keeps record of each transaction by the customer to his account. All of the branches have employees and some employees are managers".

Draw an E-R diagram that captures this information. **(UPTU 2002, 03)**

Ans.



(b) Transform this E-R diagram to relational database scheme.

Ans. Customer (customer-id, customer-name, customer-address, customer-city)

Loan (loan-num, amount)

Branch (branch-name, branch-city, assets)

Employee (emp-id, emp-name, ph-no, start-date)

Account (acc-num, balance)

Borrower (customer-id, loan-num)

Loan-branch (branch-name, loan-num)

Depositor (customer-id, acc-num)

Joint-acc (name, interest)

Single-acc (name, interest)

Examples of Relational Algebra

Q.11. *Select the employee tuples who's*

(a) DEPT_NO is 10

(b) SALARY is greater than 80,000

Ans. (a) $\sigma_{\text{DEPT\_NO} = 10}$ (Employee)

(b) $\sigma_{\text{SALARY} > 80{,}000}$ (Employee)

Q.12. *Select tuples for all employees in the EMPLOYEE who either work in DEPT_NO 10 and get annual salary of more than INR 80,000 or work in DEPT_NO 12 and get annual salary of more than INR 90,000.*

Ans. $\sigma_{(\text{DEPT-NO} = 10 \text{ AND SALARY} > 80{,}000) \text{ OR } (\text{DEPT-NO} = 12 \text{ AND SALARY} > 90{,}000)}$ (Employee)

Q.13. *List each employee's identification number (EMP_ID), Name, (EMP_NAME) and salary (SALARY).*

Ans. $\pi_{\text{EMP\_ID, EMP\_NAME, SALARY}}$ (Employee)

Q.14. *Retrieve the name (EMP_NAME) and salary (SALARY) of all employees in the relation EMPLOYEE who work in DEPT_NO 10.*

Ans. $\pi_{\text{EMP\_NAME, SALARY}}$ ($\sigma_{\text{DEPT} = 10}$) (Employee)

OR EMP−DEPT−10 ← ($\sigma_{(\text{DEPT-NO-10})}$) (Employee)

RESULTS ← ($\pi_{\text{EMP-NAME, SALARY}}$ (EMP–DEPT–10))

Q.15. *Retrieve the employees identification number of all employees who either work in DEPT–NO 10 or directly supervise an employee who work in DEPT–NO = 10.*

Ans. EMP–DEPT ← ($\sigma_{\text{DEPT-NO=10}}$ (Employee))

RESULT1 ← $\pi_{\text{EMP-ID}}$ (Employee)

RESULT2 (EMP–ID) ← $\pi_{\text{EMP-SUPERV}}$ (EMP–DEP–10)

FINAL RESULT ← RESULT1 ∪ RESULT2

Q.16. *Retrieve for each female employee (EMP–SEX = 'F') a list of the names of her dependents (EMP-DEPENDENT).*

Ans. FEMALE–EMP ← ($\sigma_{\text{EMP-SEX='F'}}$ (Employee))

ALL–EMP ← $\pi_{\text{EMP-ID, EMP-NAME}}$ (FEMALE_EMP)

DEPENDENTS ← ALL-EMP×EMP–DEPENDENT

ACTUAL–DEP ← ($\sigma_{\text{EMP-ID=FEPT-ID}}$ (DEPENDENTS))

FINAL-RESULT ← $\pi_{\text{EMP-NAME, DEPENDENT-NAME}}$ (ACTUAL_DEP)

Q.17. *Retrieve the name of the manager of each department (DEPT).*

Ans. DEPT–MANAGER ← DEPT $\quad$ MANAGER–ID = EMP–ID (Employee)

FINAL–RESULT ← $\pi_{\text{DEPT-NAME, EMP-NAME}}$ (DEPT_MANAGER)

Q.18. *Retrieve the name and address of all employees who work for the 'computer' department.*

Ans. COMP-DEP ← $\sigma_{\text{DNAME='COMPUTER'}}$ (Department)

COMP–EMPS ← (COMP-DEP $\quad$ DNUMBER = DNDEMPLOYEE)

RESULT ← $\pi_{\text{NAME, ADDRESS}}$ (COMP_EMPS)

Q.19. *For every project located in 'New Delhi', list the project numbers, the controlling department number and the department manager's last name, address and birth data.*

Ans. DELHI–PROJS ← $\sigma_{\text{PLOCATION='NEW DELHI'}}$ (Project)

CONTR–DEPT ← (DELHI–PROJS $\quad_{\text{DNUM=DNUMBER}}$ (Department))

PROJ–DEPT–MGR ← (CONTRO–DEPT $\quad_{\text{MGRSSN=SSN}}$ (Employee))

RESULT ← C $\pi_{\text{PNUMBER, DNUM, LNAME, ADDRESS, BDATE}}$ (PROJ_DEPT_MGR)

Q.20. *Find the name of employees who work on all the projects controlled by department number 10.*

Ans. DEPT–PROJS (PNO) ← π_{PNUMBER} ($\sigma_{\text{DNUM=10}}$ (Project))

EMP–PROJ (SSN, PNO) ← $\pi_{\text{ESSN, PNO}}$ (Work–On)

RESULT–EMP–SSN ← EMP–PROJ ÷ DEPT–PROJS

RESULT ← π_{NAME} (RESULT–MP–SSNS*EMPLOYEE)

Q.21. *Retrieve the names of employees who have no dependents.*

Ans. ALL–EMPS ← π_{SSN} (Employee)

EMP–WITH–DEPS (SSN) ← π_{ESSN} (Dependent)

EMP–WITHOUT–DEPS ← (ALL–EMPS–EMP–WITH–DEPS)

RESULT ← π_{NAME} (EMPS–WITHOUT–DEPS*EMPLOYEE)

Q.22. *List the name of managers who have at least one dependent.*

Ans. MGRS (SSN) ← π_{MGRSSN} (Department)

EMPS–WITH–DEPS (SSN) ← π_{ESSN} (Dependent)

MGRS–WITH–DEPS ← (MGRS ∩ EMPS–WITH–DEPS)

RESULT ← π_{NAME} (MGRS–WITH–DEPS*EMPLOYEE)

Q.23. *Explain the following terms briefly : Attribute, domain, entity, relationship, entity set, relationship set, one-to-many relationship, many-to-many relationship, participation constraint, overlap constraint, weak entity set, agreegation, and role indicator.*

Ans. Term explanations :

- *Attribute* – a property or description of an entity. A toy department employee entity could have attributes describing the employee's name, salary, and years of service.
- *Domain* – a set of possible values for an attribute.
- *Entity* – an object in the real world that is distinguishable from other objects such as the green dragon toy.

- *Relationship* – an association among two or more entities.
- *Entity set* – a collection of similar entitites such as all of the toys in the toy department.
- *Relationship set* – a collectioin of similar relationships.
- *one–to–many relationship* – a key constraint that indicates that one entity can be associated with many of another entity. An example of a one-to-many relationship is when an employee can work for only one department, and a department can have many employees.
- *Many–to–many relationship* – a key constraint that indicates that many of one entity can be associated with many of another entity. An example of a many-to-many relationship is employees and their hobbies : a person can have many different hobbies, and many people can have the same hobby.
- *Participation constraint* – a participation constraint determines whether relationship must involve certain entities. An example is if every department entity has a manager entity. Participation constraints can either be total or partial. A total participation constraint says that every department has a manager. A partial participation constraint says that every employee does not have to be a manager.
- *Overlap constraint* – within an ISA hierarchy, an overlap constraint determines whether or not two subclasses can contain the same entity.
- *Covering constraint* – within an ISA hierarchy, a covering constraint determines where the entities in the subclasses collectively include all entities in the superclass. For example, with an Employees entity set with subclasses HourlyEmployee and SalaryEmployee, does every Employee entity necessarily have to be within HourlyEmployee or SalaryEmployee?
- *Weak entity set* – an entity that cannot be identified uniquely without considering some primary key attributes of another identifying owner entity. An example is including Dependent information for employees for insurance purposes.
- *Aggregation* – a feature of the entity relationship model that allows a relationship set to participate in another relationship set. This is indicated on an ER diagram by drawing a dashed box around the aggregation.
- *Role indicator* – If an entity set plays more than one role, role indicators describe the different purpose in the relationship. An example is a single Employee entity set with a relation Reports–To that relates supervisors and subordinates.

Q. 24. *Consider the following information about a university database :*

Ans.

- Professors have an SSN, a name, an age, a rank, and a research specialty.
- Projects have a project number, a sponsor name (*e.g.*, NSF), a starting date, an ending date, and a budget.
- Graduate students have an SSN, a name, an age, and a degree program (*e.g.*, M.S. or Ph. D.).
- Each project is managed by one professor (known as the project's principal investigator).
- Each project is worked on by one or more professors (known as the project's co-investigators).
- Professors can manage and/or work on multiple projects.
- Each project is worked on by one or more graduate sutdents (known as the project's research assistants).
- When graduate students work on a project, a professor must supervise their work on the project. Graduate students can work on multiple projects, in which case they will have a (potentially different) supervisor for each one.

- Departments have a department number, a department name, and a main office.
- Departments have a professor (known as the chairman) who runs the department.
- Professors work in tone or more departments, and for each department that they work in, a time percentage is associated with their job.
- Graduate students have one major department in which they are working on their degree.

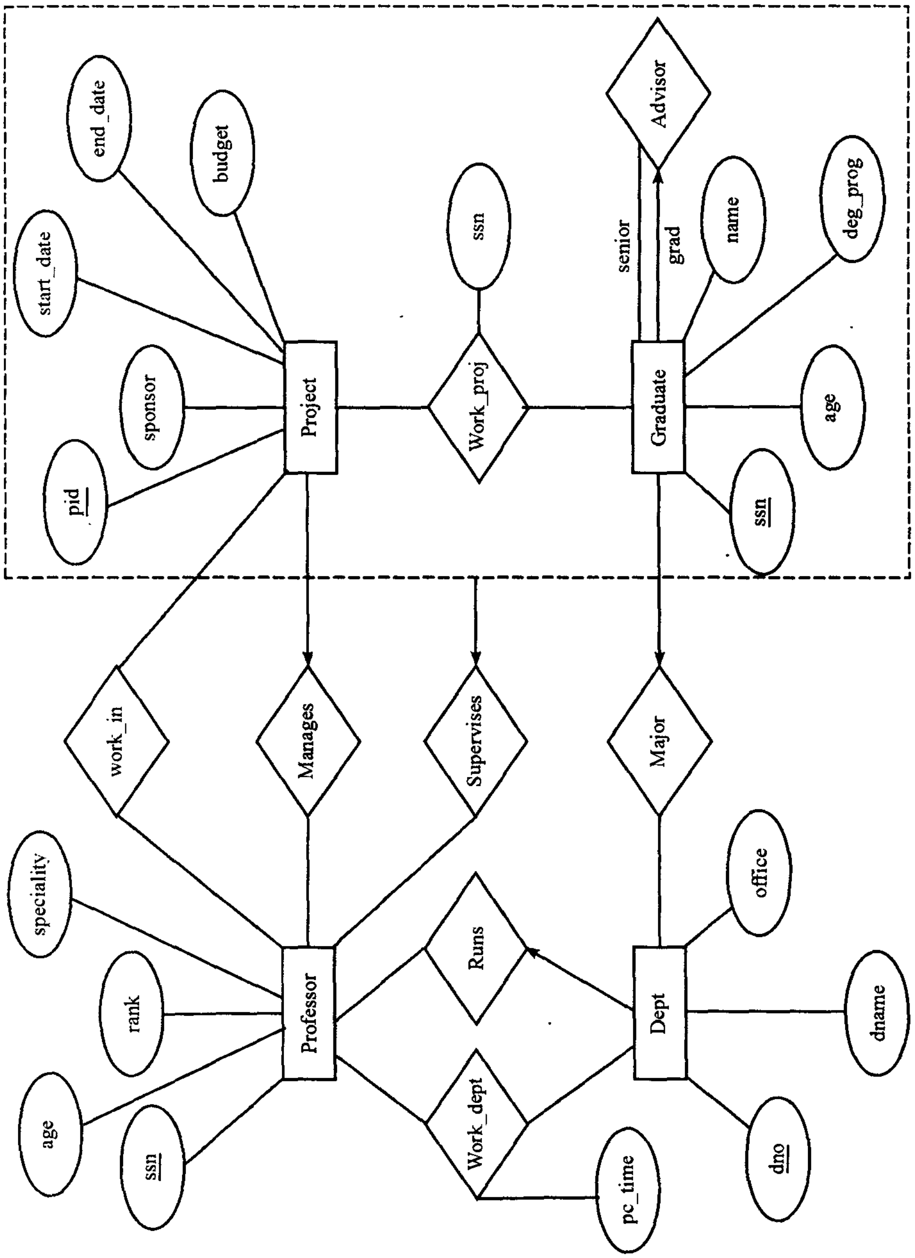


Fig. ER Diagram

- Each graduate student have another, more senior graduate student (known as a student advisor) who advises him or her on what courses to take.

Design and draw an ER diagram that captures the information about the university. Use only the basic ER model here; that is, entities, relationships, and attributes. Be sure to indicate any key and participation constraints.

Q. 25. *Notown Records has decided to store information about musicians who perform on its albums (as well as other company data) in a database. The company has wisely chosen to hire you as a database designer (at your usual consulting fee of* $2500/*day).*

- *Each musician that records at Notown has an SSN, a name, an address, and a phone number. Poorly paid musicians often share the same address, and no address has more than one phone.*
- *Each instrument used in songs recorded at Notown has a unique identification number, a name (e.g., guitar, synthesizer, flute) and a musical key (e.g., C, B-flat, E-flat).*
- *Each album recorded on the Notown label has a unique identification number, a title, a copyright data, a format (e.g., CD or MC), and an album identifier.*
- *Each song recorded at Notown has a title and an author.*
- *Each musician may play several instruments, and a given instrument may be played by several musicians.*
- *Each album has a number of songs on it, but no song may appear on more than one album.*
- *Each songs is performed by one or more musicians, and a musician may perform a number of songs.*
- *Each album has exactly one musician who acts as its producer. A musician may produce several alumbs, of course.*

Ans.

Design a conceptual schema for Notown and draw an ER diagram for your schema. The preceding information describes the situation that the Notown database must model. Be sure to indicate all key and cardinality constraints and any assumptions you make. Identify any contrainst you are unable to capture in the ER diagram and briefly explain why you could not express them.

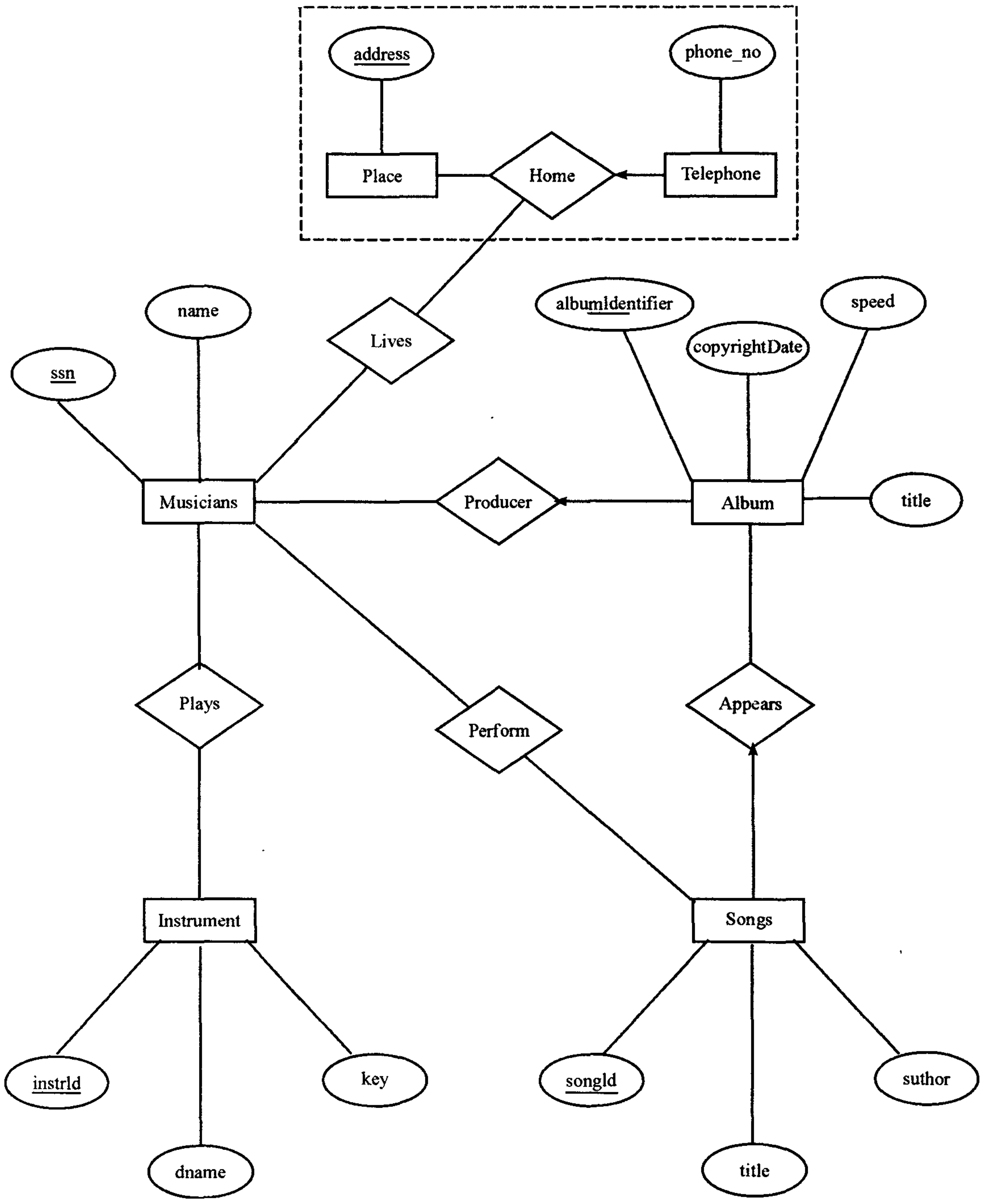


Fig. ER diagram

Q.26. *The prescriptions-R-X chain of pharmacies has offered to give you a free lifetime supply of medicine if you design its database. Given the rising cost of health care, you agree. Here's the information that you gather :*

- Patients are identified by an SSN, and their names, addresses, and ages must be recorded.
- Doctors are identified by an SSN. For each doctor, the name, specialty, and years of experience must be recorded.
- Each pharmaceutical company is identified by name and has a phone number.
- For each drugs, the trade name and formula must be recorded. Each drug is sold by a given pharmaceutical company, and the trade name identifies a drug uniquely from among the products of that company. If a pharmaceutical company is deleted, you need not keep track of its products any longer.
- Each pharmacy has a name, address, and phone number.
- Every patient has a primary physician. Every doctor has at least one patient.
- Each pharmacy sells several drugs and has a price for each. A drug could be sold at serveral pharmacies, and the price could vary from one pharmacy to another.
- Doctors prescribe drugs for patients. A doctor could prescribe one or more drugs for several patients, and a patient could obtain prescriptions from several doctors. Each prescription has a date and a quantity associated with it. You can assume that, if a doctor prescribes the same drug for the same patient more than once, only the last such prescription needs to be stored.
- Pharmaceutical companies have long-term contracts with pharmacies. A pharmaceutical company can contract with several pharmacies, and a pharmacy can contract with several pharmaceutical companies. For each contract, you have to store a start date, an end date, and the text of the contract.
- Pharmacies appoint a supervisor for each contract. There must always be a supervisor for each contract, but the contract supervisor can change over the lifetime of the contract.

1. Draw an ER diagram that captures the preceding information. Identify any constraints not captured by the ER diagram.
2. How would your design change if each drug must be sold at a fixed price by all pharmacies?
3. How would your design change if the design requirements change as follows : if a doctor prescribes the same drug for the same patient more than once, several such prescriptions may have to be stored.

Ans.

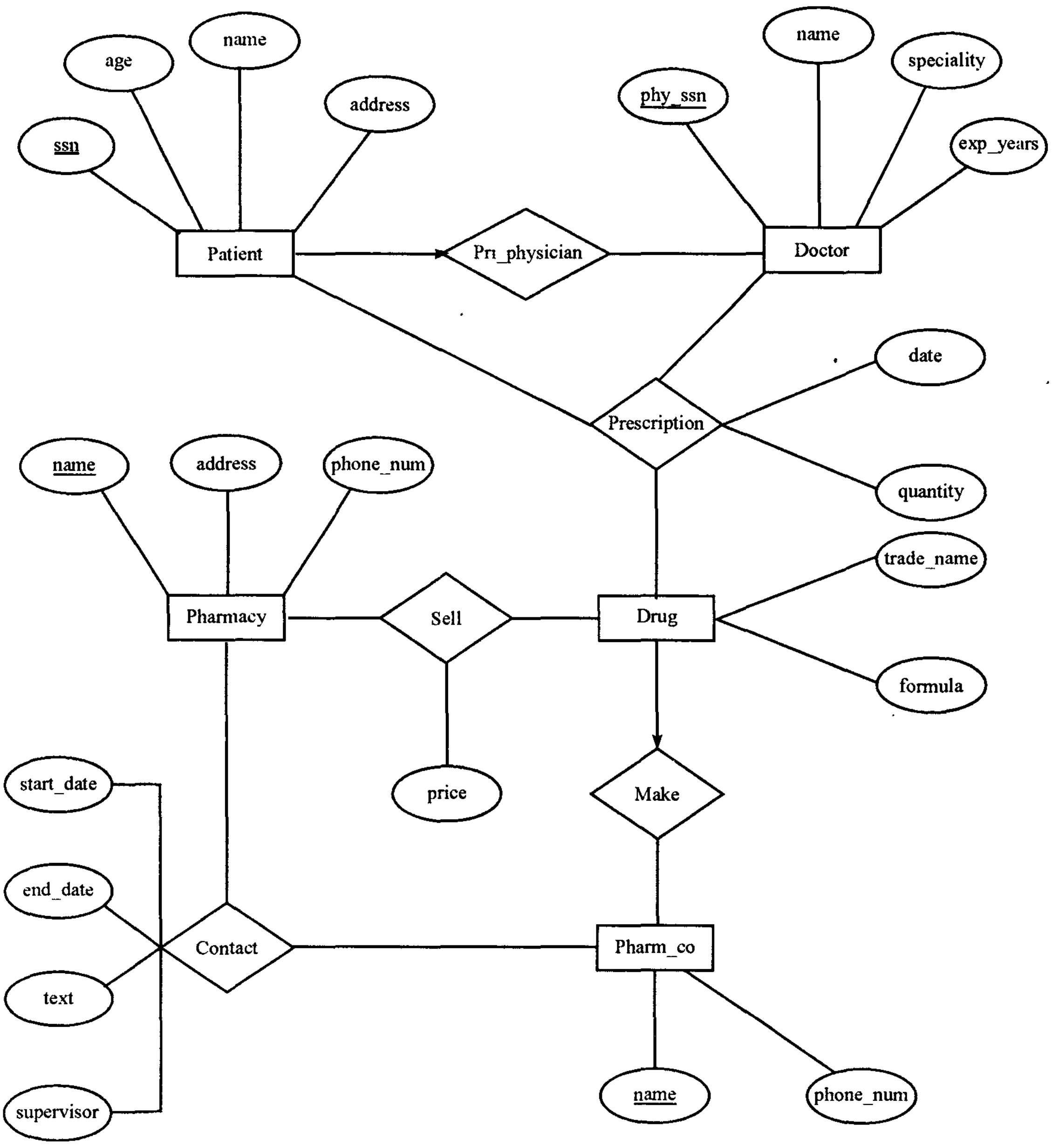


Fig. DR diagram

Q. 27. *Suppose that we have a ternary relationship R between entity sets A, B and C such that A has a key constraint and total participation and B has a key constraint; these are the only constraints. A has attributes a1 and a2, with a1 being the key; B and C are similar. R has no descriptive attributes. Write SQL statement that create tables corresponding to this information so as to capture as many of the constraints as possible. If you cannot capture some constraint, explain why ?*

Ans. The following SQL statements create the corresponding relations.

```
CREATE TABLE A     ( al CHAR (10),
                     a2 CHAR(10),
                     b1 CHAR(10),
                     cl CHAR(10),
                     PRIMARY KEY (al),
                     UNIQUE (bl),
                     FOREIGN KEY (bl) REFERENCES B,
                     FOREIGN KEY (cl) REFERENCES C);

CREATE TABLE B     ( b1 CHAR(10),
                     b2 CAHR(10),
                     PRIMARY KEY (bl));

CREATE TABLE C     ( b1 CHAR(10),
                     c2 CHAR(10),
                     PRIMARY KEY (cl));
```

The first SQL statement folds the relationship R into table A and thereby guarantees the participation constraint.

Q. 28. *Consider the university database from Quest 24 and the ER diagram you designed. Write SQL statements to create the corresponding relations and capture as many of the constraints as possible. If you cannot capture some constraints, explain why ?*

Answer. The following SQL statements create the corresponding relations.

```
1. CREATE TABLE Professors (     prof_ssn CHAR(10),
                                 name CHAR(64),
                                 age NUMBER(5),
                                 rank NUMBER(5),
                                 speciality CHAR(64),
                                 PRIMARY KEY (prof_ssn));

2. CREATE TABLE Depts (          dno NUMBER(5),
                                 dname CHAR(64),
                                 office CHAR(10),
                                 PRIMARY KEY (dno));

3. CREATE TABLE Runs (           dno NUMBER(5),
                                 prof_ssn CHAR(10),
                                 PRIMARY KEY (dno, prof_ssn),
                              FOREIGN KEY (prof_ssn) REFERENCES
                                                      Professors,
                              FOREIGN KEY (dno) REFERENCES Depts);

4. CREATE TABLE Work_Dept (   dno NUMBER(5),
                              prof_ssn CHAR(10),
                              pc_time NUMBER(6),
                              PRIMARY KEY (dno, prof_ssn),
                              FOREIGN KEY (prof_ssn) REFERENCES
                                                      Professors,
                              FOREIGN KEY (dno) REFERENCES Depts);
```

Observe that we would need check constraints or assertions in SQL to enforce the rule that Professors work in at least one department.

```
5. CREATE TABLE Project ( pid NUMBER(5),
                              sponsor CHAR (32),
                              start_date DATE,
                              budget NUMBER(8, 4),
                              PRIMARY KEY (pid))

6. CREATE TABLE Graduates ( grad_ssn CHAR(10),
                              age NUMBER(5),
                              name CHAR(64),
                              deg_prog CHAR(32),
                              major NUMBER(10),
                              PRIMARY KEY (grad_ssn),
                              FOREIGN KEY (major) REFERENCES Depts);
```

Note that the Major table is not necessary since each Graduate has only one major and so this can be an attribute in the Graduates table.

```
7. CREATE TABLE Advisor ( senior_ssn CHAR(10),
                              grad_ssn CHAR(10),
                              PRIMARY KEY (senior_ssn, grad_ssn),
                              FOREIGN KEY (senior_ssn)
                              REFERENCES Graduates (grad_ssn),
                              FOREIGN KEY (grad_ssn) REFERENCES
                                                      Graduates);
```

```
8. CREATE TABLE Manages ( pid INTEGER,
                          prof_ssn CHAR(10),
                          PRIMARY KEY (pid, prof_ssn),
                          FOREIGN KEY (prof_ssn) REFERENCES
                                                  Professors,
                          FOREIGN KEY (pid) REFERENCES Projects)
```

```
9. CREATE TABLE Work_In ( pid INTEGER,
                          prof_ssn CHAR(10),
                          PRIMARY KEY (pid, prof_ssn),
                          FOREIGN KEY (prof_ssn) REFERENCES
                                                  Professors,
                          FOREIGN KEY (pid) REFERENCES Profjects)
```

Observes that we cannot enforce the participation constraint for Projects in the Work_In table without check constraints or assertions in SQL.

```
10. CREATE TABLE Supervises ( prof_ssn CHAR(10),
                              grad_ssn CHAR(10),
                              pid INTEGER,
                              PRIMARY KEY (prof_ssn, grad_ssn, pid),
                              FOREIGN KEY (prof_ssn) REFERENCES
                                                      Proffessors,
                              FOREIGN KEY (grad_ssn) REFERENCES
                                                      Graduates,
                              FOREIGN KEY (pid) REFERENCES Projects)
```

Note that we do not need an explicit table for the Work_Proj relation since every time a Graduate works on a Project, he or she must have a Supervisor.

Q.29. Consider the Notown database from Quest 25. You have decided to recommend that Notown use a relational database system to store company data. Show the SQL statements for creating relations corresponding to the entity sets and relationship sets in your design. Identify any constraints in the ER diagram that you are unable to capture in the SQL statements and briefly explain why you could not express them.

Ans. The following SQL statements create the corresponding relations.

```
1. CREATE TABLE Musicians ( ssn CHAR(10),
                            name CHAR(30),
                            PRIMARY KEY (ssn));
```

```
2. CREATE TABLE Instruments (     instrId CHAR(10),
                                  dname CHAR(30),
                                  key CHAR(5),
                                  PRIMARY KEY (instrId));

3. CREATE TABLE Plays (           ssn CHAR(10),
                                  instrId NUMBER(5),
                                  PRIMARY KEY (ssn, instrId),
                                  FOREIGN KEY (ssn) REFERENCES Musicians,
                                  FOREIGN KEY (instrId) REFERENCES
                                                            Instruments);

4. CREATE TABLE Songs_Appears (   songId NUMBER(8),
                                  author CHAR(30),
                                  title CHAR(30),
                                  albumIdentifier NUMBER(10) NOT NULL,
                                  PRIMARY KEY (songId),
                                  FOREIGN KEY (albumIdentifier)
                                  References Album_Producer);

5. CREATE TABLE Telephone_Home (  Phone CHAR(11),
                                  address VARCHAR(30),
                                  PRIMARY KEY (phone),
                                  FOREIGN KEY (address) REFERENCES Place);

6. CREATE TABLE Lives (           ssn CHAR(10),
                                  phone NUMBER(11),
                                  address VARCHAR(30),
                                  PRIMARY KEY (ssn, address),
                                  FOREIGN KEY (phone, address)
                                  References Telephone_Home,
                                  FOREIGN KEY (ssn) REFERENCES Musicians);

7. CREATE TABLE Place (           address VARCHAR(30));

8. CREATE TABLE Perform (         songId NUMBER(8),
                                  ssn CHAR(10),
                                  PRIMARY KEY (ssn, songId),
                                     FOREIGN KEY (songId) REFERENCES Songs,
                                     FOREIGN KEY (ssn) REFERENCES Musicians);

9. CREATE TABLE Album_producer (      albumIdentifier NUMBER(8),
                                     ssn CHAR(10),
                                     copyrightDate DATE,
                                     speed NUMBER(5),
                                     title CHAR(30),
                                     PRIMARY KEY (albumIdentifier),
                                     FOREIGN KEY (ssn) references Musicians);
```

Q.30. Consider the ER diagram that you designed for the Prescriptions-R-X chain of pharmacies in Quest 26. Define relations corresponding to the entity sets and relationship sets in your design using SQL.

Ans. The statements to create tables corresponding to entity sets Doctor, Pharmacy, and Pharm_co are straightforward and omitted. The other required tables can be created as follows :

```
1. CREATE TABLE Pri_Phy_Patient (ssn CHAR(11),
                                     name CHAR(20),
                                     age NUMBER(5),
                                     address VARCHAR(20),
                                     phy_ssn CHAR(11),
                                     PRIMARY KEY (ssn),
                                     FOREIGN KEY (phy_ssn) REFERENCES
                                                                  Doctor);

2. CREATE TABLE prescription (       ssn CHAR(11),
                                     phy_ssn CHAR(11),
                                     date CHAR(11),
                                     quantity NUMBER(5),
                                     trage_name CHAR(20),
                                     pharm_id CHAR(11),
                                     PRIMARY KEY (ssn, phy_ssn),
                                     FOREIGN KEY (phy_ssn) REFERENCES Doctor,
                                     FOREIGN KEY (trade_name, pharm_id)
                                     References Make_Drug)

3. CREATE TABLE Make_Drug (          trade_name CHAR(20),
                                     pharm_id CHAR(11),
                                     PRIMARY KEY (trade_name, pharm_id),
                              FOREIGN KEY (trade_name)
                                                REFERENCES Drug,
                              FOREIGN KEY (pharm_id) REFERENCES
                                                      Pharm_co);
4. CREATE TABLE Sell (        price NUMBER(8),
                              name CHAR(10),
                              trade_name CHAR(10),
                              PRIMARY KEY (name, trade_name),
                              FOREIGN KEY (name) REFERENCES Pharmacy,
                              FOREIGN KEY (trade_name) REFERENCES
                                                         Drug);

5. CREATE TABLE Contract (    name CHAR(20),
                              pharm_id CHAR(11),
                              start_date CHAR(11),
                              end_date CHAR(11),
                              text CHAR(10000),
                              supervisor CHAR(20),
                              PRIMARY KEY (name, pharm_id),
                              FOREIGN KEY (name) REFERENCES
                                                       Pharmacy,
                              FOREIGN KEY (pharm_id) REFERENCES
                                                       Pharm_co)
```

Q.31. *Briefly answer the following questions based on this schema :*

Emp(<u>*eid* : integer</u>, *ename* : string, *age* : integer, *salary* : real)

works (<u>*eid* : integer, *did* : integer</u>, *pct_time* : integer)

Dept (<u>*did* : integer</u>, *budget* : real, *managerial* : integer)

1. Suppose you have a view SeniorEmp definedas follows :

```
CREATE VIEW SeniorEmp (sname, sage, salary)
             AS SELECT E.ename, E.age, E.salary
                  FROM Emp E
                  WHERE E.age > 50;
```

Explain what the system will do to process the following query :

```
SELECT S.snamc
FROM SeniroEmp S
WHERE S.salary > 100,000;
```

2. Give an example of a view on Emp that could be automatically updated by updating Emp.
3. Give an example of a view on Emp that would be impossible to update (automatically) and explain why your example presents the update problem that it does.

Ans. The answer to each questions is given below.

1. The system will do the following :

```
SELECT S.name
FROM        (SELECT E.ename AS name, E.age, E.salary
             FROM Emp E
             WHERE E.age > 50 ) AS S
WHERE S.salary > 100000;
```

2. The following view on Emp can be updated automatically by updating Emp:

```
CREATE VIEW SeniorEmp (eid, name, age, salary)
            AS SELECT E.eid, E.ename, E.age, E.salary
                   FROM Emp E
                   WHERE E.age > 50;
```

3. The following view cannot be updated automatically because it is not clear which employee records will be affected by a given update :

```
CREATE VIEW AvgSalaryBy Age (age, avgSalary)
            AS SELECT E.eid, AVG (E.salary)
                   FROM Emp E
                   GROUP BY E.age;
```

Q. 32. *Explain the statement that relational algebra operators can be composed. Why is the ability to compose operators important?*

Ans. Every operator in relational algebra accepts one or moı ı ·lation instances as arguments and the result is always an relation instance. So, the argument of one operator could be the result of another operator. This is important because, this makes it easy to write complex queries by simply composing the relational algebra operators.

Q. 33. *Consider the following schema :*

Suppliers (<u>sid : integer</u>, *sname* : string, *address* : string)

Parts (<u>pid : integer</u>, *pname* : string, *color* : string)

Catalog (<u>sid : integer, pid : integer</u>, *cost* : real)

The key fields are underlined, and the domain of each field is listed after the field name. Therefore *sid* is the key for Suppliers, *pid* is the key for Parts, and *sid* and *pid* together form the key for Catalog. The Catalog relation lists the prices charged for parts by suppliers. Write the following queries in relational algebra, tuple relational calculus, and domain relational calculus :

1. Find the *names* of suppliers who supply some red part.
2. Find the *sids* of suppliers who supply some red or green part.
3. Find the *sids* of suppliers who supply some red part or are at 221 Packet Street.
4. Find the *sids* of suppliers who supply some red part and some green part.
5. Find the *sids* of suppliers who supply every part.
6. Find the *sids* of suppliers who supply every red part.
7. Find the *sids* of suppliers who supply every red or green part.
8. Find the *sids* of suppliers who supply every red part or supply every green part.
9. Find *paris* of *sids* such that the supplier with the first *sid* charges more for some part than the supplier with the second *sid*.
10. Find the *pids* of parts supplied by at least two different suppliers.
11. Find the *pids* of the most expensive parts supplied by suppliers named Yosemite Sham.
12. Find the *pids* of parts supplied by every supplier at less than \$200. (If any supplier either does not supply the part or charges more than \$200 for it, the part is not selected.)

Ans. In the answers below RA refers to Relational Algebra, TRC refers to Tuple Relational Calculus and DRC refers to Domain Relational Calculus.

1. ■ RA

$\pi_{\text{sname}}(\pi_{\text{sid}}((\pi_{\text{pid}}\sigma_{\text{color = 'red'}}\, Parts) \bowtie Catalog) \bowtie Suppliers)$

■ TRC

$\{T \mid \exists T1 \in Suppliers\ (\exists X \in Parts\ (X.color = \text{' red'} \wedge \exists Y \in Catalog$
$(Y.pid = X.pid \wedge Y.sid = T1.sid)) \wedge T.sname = T1.sname)\}$

■ DRC

$\{<Y> \mid (X, Y, Z) \in Suppliers \wedge \exists P, Q, R((P, Q, R) \in Parts \wedge$
$R = \text{' red'} \wedge \exists I, J, K((I, J, K) \in Catalog \wedge J = P \wedge I = X)))\}$

■ SQL

```
SELEC[illegible] [illegible].sname
FR[illegible] Suppliers S, Parts P, Catalog C
WHERE P.color='red' AND C.pid=P.pid AND C.sid=S.sids;
```

[illegible] ■ RA

$\pi_{\text{sid}}(\pi_{\text{pid}}(\sigma_{\text{color = 'red' } \vee \text{ color = 'green'}}\, Parts) \bowtie catalog)$

■ TRC

$\{T \mid \exists T1 \in Catalog\ (\exists X \in Parts\ ((X.color = \text{'red'} \vee X.color = \text{'green'})$
$\wedge X.pid = T1.pid) \wedge T.sid = T1.sid)\}$

- DRC

  $\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \exists A, B, C\, (\langle A, B, C\rangle \in Parts$

  $\wedge\ (C = \text{'red'} \vee C = \text{'green'}) \wedge A = Y)\}$

- SQL

  SELECT C.sid
  FROM Catalog C, Parts P
  WHERE (P.color = 'red' OR P.Color = 'green')
  AND P.pid = C.pid;

3. - RA

   $\rho\ (R1, \pi_{sid}\, ((\pi_{pid}\, \sigma_{color = \text{'red'}}\, Parts) \bowtie Catalog))$

   $\rho\ (R2, \pi_{sid}\, \sigma_{address = \text{'221 Packer Street'}}\, Suppliers)$

   $R1 \cup R2;$

   - TRC

   $\{T \mid \exists T1 \in Catalog\, (\exists X \in Parts\, (X.color = \text{'red'} \wedge X.pid = T1.pid)$

   $\wedge\ T.sid = T1.sid)$

   $\vee \exists T2 \in Suppliers\, (T2.address = \text{'21 PackerStreet'} \wedge T.sid = T2.sid)\}$

   - DRC

   $\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \exists A, B, C(\langle A, B, C\rangle \in Parts$

   $\wedge\ C = \text{'red'} \wedge A = Y)$

   $\vee \exists P.Q(\langle X, P, Q\rangle \in suppliers \wedge Q = \text{'221 PackerStreet'})\}$

   - SQL

   SELECT S.sid
   FROM Suppliers S
   WHERE S.address = '221 Packer street'
   OR S.sid IN (SELECT C.sid
   FROM Parts P, Catalog C
   WHERE P.color = 'red' AND P.pid = C.pid);

4. - RA

   $\rho(R1, \pi_{sid}\, ((\pi_{pid}\, \sigma_{color = \text{'red'}}\, Parts) \bowtie Catalog))$

   $\rho\ (Rs2, \pi_{sid}\, ((\pi_{pid}\, \sigma_{color = \text{'green'}}\, Parts) \bowtie Catalog))$

   $R1 \cap R2$

- TRC

  $\{ \wedge \exists T2 \in \mathit{Catalog}\,(\exists X \in \mathit{Parts}\,(Y.color = \text{'green'} \wedge Y.pid = T2.pid)$

  $\wedge\, T2.sid = T1.sid) \wedge T.sid = T1.sid)\}$

- DRC

  $\{\langle X\rangle \mid \langle X, Y, Z\rangle \in \mathit{Catalog} \wedge \exists A, B, C(\langle A, B, C\rangle \in \mathit{Parts}$

  $\wedge\, C = \text{'red'} \wedge A = Y)$

  $\wedge\, \exists P, Q, R(\langle P, Q, R\rangle \in \mathit{Catalog} \wedge \exists E, F, G(\langle E, F, G\rangle \in \mathit{Parts}$

  $\wedge\, G = \text{'green'} \wedge E = Q) \wedge P = X)\}$

- SQL

```
SELECT C.sid
FROM Parts P, Catalog C
WHERE P.color = 'red' AND P.pid = C.pid
            AND EXISTS (SELECT P2.pid
                        FROM Parts P2, Catalog C2
                        WHERE P2.color = 'green' AND C2.sid = C.sid
                              AND P2.pid = C2.pid);
```

5. ■ RA

  $(\pi_{\text{sid, pid}}\, \mathit{Catalog}) / (\pi_{\text{pid}}\, \mathit{Parts})$

- TRC

  $\{T \mid \exists T1 \in \mathit{Catalog}\,(\forall X \in \mathit{Parts}\,(\exists T2 \in \mathit{Catalog}$

  $(T2.pid = X.pid \wedge T2.sid = T1.sid)) \wedge T.sid = T1.sid)\}$

- DRC

  $\{\langle X\rangle \mid \langle X, Y, Z\rangle \in \mathit{Catalog} \wedge \forall \langle A, B, C\rangle \in \mathit{Parts}$

  $(\exists\langle P, Q\, R\rangle \in \mathit{Catalog}\,(Q = A \wedge P = X))\}$

- SQL

```
SELECT C.sid
FROM Catalog C
WHERE NOT EXISTS (SELECT P.pid
            FROM Parts P
            WHERE NOT EXISTS (SELECT C1.sid
                        FROM Catalog C1
                        WHERE C1.sid = C.sid
                              AND C1.pid = p.pid));
```

6. ■ RA

$(\pi_{sd,pid}\ Catalog) / (\pi_{pid}\ \sigma_{color='red'}\ Parts)$

■ TRC

$\{T \mid \exists T1 \in Catalog\ (\forall X \in Parts\ (X.color \neq 'red'$
$\vee \exists T2 \in Catalog\ (T2.pid = X.pid \wedge T2.sid = T1.sid))$
$\wedge T.sid = T1.sid)\}$

■ DRC

$\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \forall \langle A, B, C\rangle \in Parts$
$(C \neq 'red' \vee \exists\langle P\ Q, R\rangle \in Catalog\ (Q = A \wedge P = X))\}$

■ SQL

```
SELECT C.sid
FROM Catalog C
WHERE NOT EXISTS (SELECT P.pid
                  FROM Parts P
                  WHERE P.color = 'red'
                  AND (NOT EXISTS (SELECT C1.sid
                                   FROM Catalog C1
                                   WHERE C1.sid = C.sid AND
                                         C1.pid = p.pid)));
```

7. ■ RA

$(\pi_{sid,pid}\ Catalog) / (\pi_{pid}\ \sigma_{color='r'red' \vee color='green'}\ Parts)$

■ TRC

$\{T \mid \exists T1 \in Catalog\ (\forall X \in Parts\ ((X.color \neq 'red'$
$\wedge X.color \neq 'green') \vee \exists T2 \in Catalog$
$(T2.pid = X.pid \wedge T2.sid = T1.sid)) \wedge T.sid = T1.sid)\}$

■ DRC

$\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \forall \langle A, B, C\rangle \in Parts$
$((C \neq 'red' \wedge C \neq 'green') \wedge \exists\langle P, Q, R\rangle \in Catalog$
$(Q = A \wedge P = X))\}$

■ SQL

```
SELECT C.sid
FROM Catalog C
```

```
WHERE NOT EXISTS (SELECT P.pid
                  FROM Parts P
                  WHERE (P.color = 'red' OR P.color = 'green' )
                  AND (NOT EXISTS (SELECT C1.sid
                                   FROM Catalog C1
                                   WHERE C1.sid = C.sid AND
                                         C1.pid = P.pid)))
```

8. ■ RA

$\rho(R1, ((\pi_{sid,pid}\ Catalog) / (\pi_{pid}\ \sigma_{color=\text{“red'}}\ Parts)))$

$\rho(R2, ((\pi_{sid,pid}\ Catalog) /(\pi_{pid}\ \sigma_{color=\text{‘green'}}\ Parts)))$

$R1 \cup R2$

■ TRC

$\{T \mid \exists T1 \in Catalog\ ((\forall\ X \in Parts$

$(X..color \neq \text{‘red'} \vee \exists Y \in Catalog\ (Y.pid = X.pid \wedge Y.sid = T1.sid))$

$\vee \forall Z \in Parts\ (z.color \neq \text{‘green'} \vee \exists P \in Catalog$

$(P.pid = Z.pid \wedge P.sid = T1.sid))) \wedge T.sid = T1.sid)\}$

■ DRC

$\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge (\forall \langle A, B, C\rangle \in Partrs$

$(C \neq \text{‘red'} \vee \exists \langle P, Q, R\rangle \in catalog\ (Q = A \wedge P = X))$

$\vee \forall \langle U, V, W\rangle \in Parts\ (W \neq \text{‘green'} \vee \langle M, N, L\rangle \in Catalog$

$(N = U \wedge M = X)))\}$

■ SQL

```
SELECT C.sid
FROM Catalog C
WHERE (NOT EXISTS (SELECT P.pid
                   FROM Parts P
                   WHERE P.color = 'red' AND
                   (NOT EXISTS (SELECT (C1.sid
                                FROM Catalog C1
                                WHERE C1.sid = C.sid AND
                                OR     C1.pid = P.pid))))
                   (NOT EXISTS (SELECTE P1.pid
                                FROM parts.P1 WHERE P1.color = 'green'
                                AND (NOT EXISTS (SELECT C2.sid
                                        FROM C2.sid = C.sid AND
                                              C2.pid = P1))));
```

9. ■ RA

   $\rho(R1, Catalog)$

   $\rho(R2, Catalog)$

   $\pi_{R1.sid, R2.sid}(\sigma_{R1.pid = R2.pid \wedge R1.sid \neq R2.sid \wedge R1.cost > R2.cost}(R1 \times R2));$

   ■ TRC

   $\{T \mid \exists T1 \in Catalog (\exists T2 \in Catalog$
   $(T2.pid = T1.pid \wedge T2.sid \neq T1.sid$
   $\wedge T2.cost < T1.cost \wedge T.sid2 = T2.sid)$
   $\wedge T.sid1 = `T1.sid)\}$

   ■ DRC

   $\{\langle X, P\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \exists P, Q, R$
   $(\langle P, Q, R\rangle \in Catalog \wedge Q = Y \wedge P \neq X \wedge R < Z)\}$

   ■ SQL

```
SELECT C1.sid, C2.sid
FROM Catalog C1, Catalog C2
WHERE C1.pid = C2.pid AND C1.sid ≠ C2.sid
            AND c1.cost > C2.cost;
```

10. ■ RA

   $\rho(R1, catalog)$

   $\rho(R2, Catalog)$

   $\pi_{R1.pid}\sigma_{R1.pid = R2.pid \wedge R1.sid \neq R2.sid}(R1 \times R2)$

   ■ TRC

   $\{T \mid \exists T1 \in Catalog (\exists T2 \in Catalog$
   $(T2.pid = T1.pid \wedge T2.sid \neq T1.sid)$
   $\wedge T.pid = T1.pid)\}$

   ■ DRC

   $\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \exists A, B, C$
   $(\langle A, B, C\rangle \in Catalog \wedge B = Y \wedge A \neq X)\}$

■ SQL

```
SELECT C.pid
FROM Catalog C
WHERE EXISTS (SELECT C1.sid
              FROM Catalog C1
              WHERE C1.pid = C.pid AND C1.sid ≠ C.sid)
```

11. ■ RA

$\rho(R1, \pi_{\text{sid}}\, \sigma_{\text{sname='YosemiteSham'}}\, Suppliers)$

$\rho\ (R2, R1 \bowtie Catalog)$

$\rho\ (R3, R2)$

$\rho\ (R4(1 \rightarrow sid, 2 \rightarrow pid, 3 \rightarrow cost), \sigma_{\text{R3.cost<R2.cost}}\ (R3 \times R2))$

$\pi_{\text{pid}}\ (R2 - \pi_{\text{sid,pid,cost}}\, R4)$

■ TRC

$\{T \mid \exists T1 \in Catalog\ (\exists X \in Suppliers$

$(X.sname = \text{'YosemiteSham'} \wedge X.sid = T1.sid) \wedge \neg (\exists S \in Suppliers$

$(S.sname = \text{'YosemiteSham'} \wedge \exists Z \in Catalog$

$(Z.sid = S.sid \wedge Z.cost > T1.cost))) \wedge T.pid = T1.pid)$

■ DRC

$\{\langle Y \rangle \mid \langle X, Y, Z \rangle \in Catalog \wedge \exists A, B, C$

$(\langle A, B, C \rangle \in Suppliers \wedge C = \text{'YosemiteSham'} \wedge A = X)$

$\wedge \neg (\exists P, Q, R(\langle P, Q, R \rangle \in Suppliers \wedge R = \text{'YosemiteSham'}$

$\wedge \in I, J, K(\langle I, J, K \rangle \in Catalog\ (I = P \wedge K > Z))))\}$

■ SQL

```
SELECT C.pid
FROM Catalog C, Suppliers S
WHERE S.sname = 'Yosemite Sham' AND C.sid = S.sid
              AND C.sot ≥ ALL (Select C2.cost
                               FROM Catalog C2, Suppliers S2
                               WHERE S2.sname = 'Yosemite Sham'
                                     AND C2.sid = S2.sid)
```

Q.34. *Consider the Supplier-Parts-Catalog schema from the previous question. State what the following queries compute :*

1. $\pi_{sname}(\pi_{sid}((\sigma_{color='red'}\, Parts) \bowtie (\sigma_{cost<100} Catalog)) \bowtie$ '*Suppliers*)

2. $\pi_{sname}(\pi_{sid}((\sigma_{color='red'}\, Parts) \bowtie (\sigma_{cost<100} Catalog) \bowtie Suppliers))$

3. $(\pi_{sname}((\sigma_{color='red'}\, Parts) \bowtie (\sigma_{cost<100}\, Catalog) \bowtie Suppliers)) \cap$

   $(\pi_{sname}((\sigma_{color='green'}\, Parts) \bowtie (\sigma_{cost<100} Catalog) \bowtie Suppliers))$

4. $(\pi_{sid}((\sigma_{color='red'}\, Parts) \bowtie (\sigma_{cost<100}\, Catalog) \bowtie Suppliers)) \cap$

   $(\pi_{sid}((\sigma_{color = 'green'}\, Parts) \bowtie (\sigma_{cost<100}\, Catalog) \bowtie Suppliers))$

5. $\pi_{sname}((\pi_{sid,sname}((\sigma_{color='red'}\, Parts) \bowtie (\sigma_{cost<100}\, Catalog) \bowtie Supplirs)) \cap$

   $(\pi_{sid,sname}((\sigma_{color='green'}\, Parts) \bowtie (\sigma_{cost<100}\, Catalog) \bowtie Suppliers)))$

Ans. The statments can be interpreted as :

1. Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars.
2. This Relational Algebra statement does not return anything because of the sequence of projection operators. Once, the sid is projected, it is the only field in the set. Therefore, projecting on sname will not return anything.
3. Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.
4. Find the Supplier *sid* of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.
5. Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.

Student (*snum* : integer, *sname* : string, *major* : string, *level* : string, *age* : integer)

Class (*name* : string, *meets_at* : string, *room* : string, *fid* : integer)

Enrolled (*snum* : integer, *cname* : string)

Faculty (*fid* : integer, *fname* : string, *deptid* : integer)

The meaning of these relations is straightforward; for example, Enrolled has one record per student-class pair such that the student is enrolled in the class.

Write the following queries in SQL. No duplicates should be printed in any of the answers.

1. Find the names of all Juniors (level = JR) who are enrolled in a class taught by I. Teach.
2. Find the age of the oldest student who is either a History major or enrolled in a course taught by I Teach.
3. Find the names of all classes that either meet in room R128 or have five or more students enrolled.

4. Find the names of all students who are enrolled in two classes that meet at the same time.
5. Find the names of faculty members who teach in every room in which some class is taught.
6. Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five.
7. For each level, print the level and the average age of students for that level.
8. For all levels except JR, print the level and the average age of students for that level.
9. For each faculty member that has taught classes only in room R128, print the faculty member's name and the total number of classes she or he has taught.
10. Find the names of students enrolled in the maximum number of classes.
11. Find the names of students not enrolled in any class.
12. For each age value that appears in Students, find the level value that appears most often. For example, if there are more FR level students aged 18 than SR, JR, or SO students aged 18, you should print the pair (18, FR).

1.
```
SELECT DISTINCT S.Sname
FROM Studnet S, Class C, Enrolled E, Faculty F
WHERE S.snum = E.snum AND E.cname = C.name AND C.fid = F.fid AND
            F.fname = 'I.Teach' AND S.level = 'JR';
```

2.
```
SELEFT MAX (S.age)
FROM Student S
WHERE (S.major = 'History')
            OR S.snum IN (SELECT E.snum
                          FROM Class C, Enrolled E, Faculty F
                          WHERE E.cname = C.name AND C.fid = F.fid
                               AND F.fname = 'I.Teach');
```

3.
```
SELECT C.name
FROM Class C
WHERE C.room = 'R128'
            OR C.name IN (SELECT E.cname
                          FROM Enrolled E
                          GROUP BY E.cname
                          HAVING COUNT (*) > = 5);
```

4.
```
SELECT DISTINCT S.sname
FROM Studnet S
WHERE S.snum IN (SELECT E1.snum
            FROM Enrolled E1, Enrolled E2, Class C1, Class C2
            WHERE E1.snum = E2.snum AND E1.cname < > E2.cname
```

```
                    AND E1.cname = C1.name
                    AND E2.cname = C2.name AND C1.meets_at = C2.meets_at);
```

5.
```
SELECT DISTINCT F.fname
FROM Faculty F
WHERE NOT EXISTS ((SELECT *
                    FROM Class C)
                    EXCEPT
                    (SELECT C1.room
                    FROM Class C1
                    WHERE C1.fid = F.fid));
```

6.
```
SELECT DISTINCT F.fname
FROM Faculty F
WHERE 5 > (SELECT COUNT (E.snum)
                    FROM Class C, Enrolled E
                    WHERE C.name = E.cname
                    AND C.fid = F.fid);
```

7.
```
SELECT S.level, AVG(S.age)
FROM Student S
GROUP BY S.level;
```

8.
```
SELECT S.level, AVG(S.age)
FROM Student S
WHERE S.level < > 'JR'
GROUP BY S.level;
```

9.
```
SELECT F.fname, COUNT(*) AS CourseCount
FROM Faculty F, Class C
WHERE F.fid = C.fid
GROUP BY F.fid, F.fname
HAVING EVERY (C.room = 'R128');
```

10.
```
SELECT DISTINCT S.sname
FROM Student S
WHERE S.snum IN (SELECT E.snum
FROM Enrolled E
```

```
        GROUP BY E.snum
                        HAVING COUNT (*) > = ALL (SELECT COUNT (*)
                                        FROM Enrolled E2
                                        GROUP BY E2.snum))

11.     SELECT DISTINCT S.sname
        FROM Studnet S
        WHERE S.snum NOT IN (SELECT E.snum FROM Enrolled E)

12.     SELECT S.age, S.level
        FROM Student S
        GROUP BY S.age, S.level,
        HAVING S.level IN (SELECT S1.level
                        FROM Student S1
                        WHERE S1.age = S.age
                        GROUP BY S1.level, S1.age
                        HAVING COUNT (*) > = ALL (SELECT COUNT (*)
                                        FROM Student S2
                                        WHERE S1.age = S2.age
                                        GROUP BY S2.level, S2.age))
```

Q. 35. The following relations keep track of airline flight information :

Flights (*flno* : integer, *from* : string, *to* : string, *distance* : integer, *departs* : time, *arrives* : time, *price* : real)

Aircraft (*aid* : integer, *aname* : string, *cruisingrange* : integer)

Certified (*eid* : integer, *aid* : integer)

Employees (*eid* : integer, *ename* : string, *salary* : integer)

1. Find the names of aircraft such that all pilots certified to operate them have salaries more than $80,000.
2. For each pilot who is certified for more than three aircraft, find the *eid* and the maximum *cruisingrange* of the aircraft for which she or he is certified.
3. Find the names of pilots whose *salary* is less than the price of the cheapest route from Los Angeles to Honolulu.
4. For all aircraft with *cruisingrange* over 1000 miles, find the name of the aircraft and the average salary of all pilots certified for this aircraft.
5. Find the names of pilots certified for some Being aircraft.
6. Find the *aids* of all aircraft that can be used on routes from Los Angles to Chicago.
7. Identify the routes that can be piloted by every pilot who makes more than $100,000.

8. Print the *enames* of pilots who can operate planes with *cruisingrange* greater than 30000 miles but are not certified on any Boeing aircraft.
9. A customer wants to travel from Madison to New York with no more than two changes of flight. List the choice of departure times from Madison if the customer wants to arrive in New york by 6 p.m.
10. Computer the difference between the average salary of a pilot and the average salary of all employees (including pilots).
11. Print the name and salary of every nonpilot whose salary is more than the average salary for pilots.
12. Print the names of employees who are certified only on aircrafts with cruising range longer than 10000 miles.
13. Print the names of employees who are certified only on aircrafts with cruising range longer than 1000 miles, but on at least two such aircraft.
14. Print the names of employees who are certified only on aircrafts with cruising rang longer than 1000 miles and who are certified on some Boeing aircraft.

Ans. The answer are given below :

1.
```
SELECT DISTINCT A.aname
FROM Aircraft A
WHERE A.Aid IN (SELECT C.aid
                FROM Certified C, Employees E
                WHERE C.eid = E.eid AND
                NOT EXISTS (SELECT *
                            FROM Employees E1
                            WHERE E1.eid = E.eid AND E1.salary < 80000));
```

2.
```
SELECT C.eid, MAX (A.cruisingrange)
FROM Certified C, Aircraft A
WHERE C.aid = A.aid
GROUP BY C.eid
HAVING COUNT (*) > 3;
```

3.
```
SELECT DISTINCT E.ename
FROM Employees E
WHERE E.salary < (SELECT MIN (F.price)
                  FROM Flights F
                  WHERE F.from = 'Los Angeles' AND F.to = 'Honolulu')
```

4. Observe that *aid* is the key for Aircraft, but the question asks for aircraft names; we deal with this complication by using an intermediate relation Temp :

```
SELECT Temp.name, Temp.AvgSalary
FROM (SELECT A.aid, A.aname AS name, AVG (E.salary) AS AvgSalary
        FROM Aircraft A, Certified C, Employees E
        WHERE A.aid = C.aid AND
                     C.eid = E.eid AND A.cruisingrange > 1000
        GROUP BY A.aid, A.aname) As Temp;

5.      SELECT DISTINCT E.ename
        FROM Employees E, Certified C, Aircraft A
        WHERE E.eid = C.eid AND
                     C.aid = A.aid AND
                     A.aname LIKE 'Boeing%';

6.      SELECT a.aid
        FROM Aircraft A
        WHERE A.cruisingrange > (SELECT MIN (F.distance)
                     FROM Flights F
                     WHERE F.from = 'Los Angeles' AND F.to = 'Chicago');

7.      SELECT DISTINCT F.from, F.to
        FROM Flights F
        WHERE NOT EXISTS (SELECT *
                     FROM Employees E
                     WHERE E.salary > 100000
                     AND
                     NOT EXISTS (SELECT *
                                  FROM Aircraft A, Certified C
                                  WHERE A.cruisingrange > F.distance
                                  AND E.eid = C.eid
                                  AND A.aid = C.aid))

8.      SELECT DISTINCT E.ename
        FROM Employees E
        WHERE E.eid IN ((SELECT C.eid
                     FROM Certified C
                     WHERE EXISTS (SELECT A.aid
                                  FROM Aircraft A
                                  WHERE A.aid = C.aid
                                  AND A.cruisingrange > 3000)
```

```
                    AND
                    NOT EXISTS (SELECT A1.aid
                                FROM Aircraft A1
                                WHERE A1.aid = C.aid
                                AND A1.aname LIKE 'Boeing%'))
```

9.
```
SELECT F.departs
FROM Flights F
WHERE F.flno IN ((SELECT F0.flno
          FROM Flights F0
          WHERE F0.from = 'Madison' AND F0.to = 'New York'
                    AND F0.arrives < '18:00')
          UNION
          (SELECT F0.flno
          FROM Flights F0, Flights F1, Flgiths F2
          WHERE F0.from = 'Madison'
                    AND F0.to = F1.from
                    AND F1.to = F2.from
                    AND F2.to = 'New York'
                    AND F0.to < > 'New York'
                    AND F1.to < > 'New York'
                    AND F1.departs >F0.arrives
                    AND F2.departs > F1.arrives
                    AND F2.arrives < '18:00'));
```

10.
```
SELECT Temp1.avg - Temp2.avg
FROM (SELECT AVG (E.salary) AS avg
          FROM Employees E
          WHERE E.eid IN (SELECT DISTINCT C.eid
                    FROM Certified C)) AS Temp1,
          (SELECT AVG (E1.salary) AS avg
          FROM Employees E1) AS Temp2;
```

11.
```
SELECT E.ename, E.salary
FROM Employees E
WHERE E.eid NOT IN (SELECT DISTINCT C.eid
          FROM Certified C)
AND E.salary > (SELECT AVG E1.salary)
```

```
                 FROM Employees E1
                 WHERE E1.eid IN
                          (SELECT DISTINCT C1.eid
                          FROM Certified C1))
```

```
12.     SELECT E.ename
        FROM Employees E, Certified C, Aircraft A
        WHERE C.aid = A.aid AND E.eid = C.eid
        GROUP BY E.eid, E.ename
        HAVING EVERY (A.cruisingrange > 1000)
```

```
13.     SELECT       E.ename
        FROM         Employees E, Certified C, Aircraft A
        WHERE        C.aid = A.aid AND E.eid = C.eid
        GROUP        BY E.eid, E.ename
        HAVING       EVERY (A.crusingrange > 1000) AND COUNT (*) > 1
```

```
14.     SELECT       E.ename
        FROM         Employees E, Certified C, Aircraft A
        WHERE        C.aid = A.aid AND E.eid = C.eid
        GROUP        BY E.eid, E.ename
        HAVING       EVERY (A.cruisingrange > 1000) AND ANY (A.name = 'Boeing');
```

Review Questions

1. In the context of a relational model, discuss each of the following concepts.
   (a) Relation (b) Attributes (c) Tuple
   (d) Cardinality (e) Domain
2. Discuss the various types of keys that are used in relational model.
3. What do you mean by relational algebra? Define all the operators of relational algebra.
4. What do you mean by structure of relational model of database system? Explain the significance of domain and keys in the relational model.
5. What is relational algebra? What is its use? List relational operators.
6. What do you mean by relational calculus? What are the types of relational calculus?
7. What is difference between JOIN & OUTER JOIN operator?
8. Describe the SELECT operation. What does it accomplish?
9. Describe the PROJECT operation. What does it accomplish?
10. Describe the JOIN operation. What does it accomplish.
11. What is query language? What are its advantages?
12. What is structured query language? What are its advantages and disadvantages?
13. Explain the syntax of SQL for executing query.
14. What are SQL operators? Explain it.
15. Write short notes on the following :
    (i) Data Manipulation Language (DML).
    (ii) Data Definition Language (DDL).
    (iii) Transaction Control Statement (TCS).
    (iv) Data Control Language (DCL).
16. How do we create lable? Views and index using SQL commands?
17. Explain briefly the following :
    (i) Database cursor (ii) Database trigger
    (iii) Database packages
18. Explain a sequence.
19. What are safe expressions in tuple relational calculus? Explain with example.
20. Explain trigger and assertion with suitable example. Write the assertion for the following statement : "Every loan has at least one customer who maintains an account with minimum balance of Rs. 1000 in banking system?"
21. What is trigger? Explain the advantages of triggers with suitable example.
22. Write a short notes on subqueries, aggregate functions, joins, indexes, sequences.

□□□