UNIT II

Overview of the SQL:

SQL is a language to operate databases; it includes database creation, deletion, fetching rows, modifying rows, etc. SQL is an **ANSI** (American National Standards Institute) standard language, but there are many different versions of the SQL language.

SQL:

SQL is Structured Query Language, which is a computer language for storing, manipulating and retrieving data stored in a relational database.

SQL is the standard language for Relational Database System. All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, and SQL Server use SQL as their standard database language.

SQL is widely popular because it offers the following advantages −

- Allows users to access data in the relational database management systems.
- Allows users to describe the data.
- Allows users to define the data in a database and manipulate that data.
- Allows embedding within other languages using SQL modules, libraries & pre-compilers.
- Allows users to create and drop databases and tables.
- Allows users to create view, stored procedure, functions in a database.
- Allows users to set permissions on tables, procedures and views.

A Brief History of SQL:

IBM developed the original version of SQL, originally called sequel, as part of the SYSTEM R PROJECT in the early 1970's.

The sequel language has evolved since then and its name has changed to SQL(structured query language).

In 1986, the American national standards institute (ANSI) and the international standards organization (ISO) published an SQL standard, called SQL-86.

ANSI published an extended standard for SQL, SQL-89.

The next version of the standard was SQL-92 standard, followed by SQL-1999.

The most recent version is SQL: 2003.

SQL language has several parts:

1. **Data Definition Language (DDL)**

   DDL refers to a language that consists of SQL commands for the creation of database objects like tables, views, indexes, etc. It also contains commands that are used to specify access rights to those database objects.

2. **Data Manipulation Language (DML)**

DML refers to a language that consists of SQL commands so as to perform data operations like insert, delete, update and retrieve in the database tables.

3. **Integrity**

Integrity constraint is a condition that ensures the correct insertion of the data and prevents unauthorized data access thereby preserving the consistency of data.

4. **View Definition**

View definition is included in SQL DDL commands for describing views.

5. **Transaction Control**

Transaction control commands are included in SQL for describing the starting and ending of transactions.

6. **Embedded SQL and Dynamic SQL**

Embedded SQL and dynamic SQL describes the way in which SQL statements are embedded in general-purpose programming languages like C, C++ and Java.

7. **Authorization**

Authorization commands are included in SQL DDL for describing access rights associated to relations and views.

SQL DATA DEFINITION:

Data Definition Language (DDL):

The Data Definition Language (DDL) creates, alters and deletes relations in a database. The DDL performs the following operations,

1. CREATE
2. ALTER
3. DROP.

1. **CREATE**

The tables (or relations) in SQL are created using the CREATE TABLE command.

Syntax:

CREATE TABLE table_name

(Column1_name data type [NULL | NOT NULL [WITH DEFAULT] | UNION],

(Column2_name datatype [NULL | NOT NULL [WITH DEFAULT] | UNION],

(columnN_name datatype[NULL | NOT NULL [WITH DEFAULT] | UNION], [Primary_key_definition],
Alternate_key_defin

itions],

[Foreign_key_defin

itions]);

- In the definition of CREATE TABLE command, column name is the name of the column, data type is the type of data that the column will hold. The keywords NULL and NOT NULL are optional. These keywords specify the

conditions that must be checked while inserting data into the table.

- If a column is specified as NULL then a NULL value will be inserted in that column if the user does not insert value for that column.
- If a column is NOT NULL, then the user must specify value for that column otherwise the system does not accept that record and returns an error message.
- With NOT NULL, two other keywords WITH DEFAULT or UNION can also be specified. Both the keywords are optional.
- If a column is NOT NULL WITH DEFAULT, then the system uses the default value for that column if the user does not specify the value.
- For example, the default value for numeric data type is 0 and for character data type is a space.
- If a column is specified as NOT NULL UNIQUE then the system does not allow the duplicate values for that column.
- The last three lines in the CREATE TABLE definition specify the primary key, alternate key and the foreign key for the new table.

Example:

The following statement creates a new table named customer,

```
CREATE Table Customer
(cid INTEGER NOT NULL,
cname  CHAR(20),
accno   INTEGER;
fname    CHAR(30) NULL,
amt       REAL NOT NULL WITH DEFAULT,
Primary key(cid));
```

2. **ALTER**

ALTER TABLE command is used to modify the structure of the table or view. With Alter command following modification can be done,

(a) Addition of column
(b) Deletion of column
(c) Addition of constraint
(d) Deletion of constraint
(e) Setting a default value for a column
(f) Unsetting a default value for a column.

Syntax:

```
ALTER TABLE Table name ADD [COLUMN] Column name data type;
ALTER TABLE Table name DROP [COLUMN] Column name ;
ALTER TABLE Table name ADD [CONSTRAINT] Constraint name;
ALTER TABLE Table name DROP [CONSTRAINT] Constraint name ;
```

ALTER TABLE Table name ALTER [COLUMN] SET DEFAULT Default option];

ALTER TABLE Table name [ALTER [COLUMN] DROP DEFAULT];

Example

- ALTER TABLE Employee ADD Project -Number INTEGER;

3. **DROP**

Destroying or deleting a table can be done with the help of DROP command.

Syntax:

DROP TABLE table- name;

Example:

DROP TABLE Department;

This command deletes the department table.

Data Manipulation:

Data manipulation is a method of manipulating data on database using various operations. It uses DML for this purpose.

DML:

The Data Manipulation Language (DML) uses the following commands to perform operations in databases,

1. SELECT command
2. INSERT command
3. UPDATE command
4. DELETE command.

1. **SELECT Command:**

This command is mostly used in SQL queries. It is considered as the basic SQL command which is used to retrieve the information or to select specific columns from the tables. Every SQL query starts with the SELECT keyword and is followed by the list of columns that forms the resulting relation.

Syntax:

The basic format of SQL query is,

SELECT select_list

FROM from_list

WHERE condition;

Example:

SELECT Ename, Eid

FROM Employee

WHERE Esal > 20000;

(a) FROM Clause

This clause specifies the names of the tables from which data is to be retrieved. FROM is followed by a set of range values which uniquely identifies a table, when its name occurs more than once.

(b) WHERE Clause

WHERE keyword is used to specify a condition. All the tuples which satisfy the condition are selected.

The various other clauses that can be used in the SELECT statement are Group BY, HAVING, ORDER BY etc.

(c) GROUP BY Clause

Group BY clause is used to group the data of a table in a single row or column based on the given grouping condition. It applies some aggregate functions like MAX, MIN, AVG, SUM and COUNT on groups.

(d) ORDER BY Clause

ORDER BY clause is used to retrieve the data in a particular order i.e., either ascending or descending order.

(i) Ascending Order

This is the default order that sorts the table in increasing order of the attribute specified in ORDER BY clause.

(ii) Descending Order

This order sorts the table in decreasing order of attribute specified in ORDER BY clause.

Syntax

```
SELECT col1, col2, .....
FROM table_name
WHERE condition
ORDER BY[col1,col2,….., coln] ASC/DESC;
```

Example

```
SELECT Emp_name, Emp_salary, Emp_phone
FROM   Employee.
ORDER BY Emp_salary DESC;
```

(e) HAVING Clause

HAVING clause is similar to the WHERE clause. It is used to pose the conditions on groups. With WHERE clause, conditions are set on rows and with HAVING clause, conditions are set on groups.

Syntax

```
SELECT select_list FROM from_list
```

GROUP BY grouping_list ORDER BY ordering_list

HAVING condition;

Example

SELECT Ename, DNo, MAX(Esal) FROM Employee

GROUP BY Ename ORDER BY Esal DESC

HAVING MAX(Esal) > 6000;

INSERT Command

This command is used to add the information or column values to a row in a table.

Syntax:

INSERT INTO table_name(column1, column2, column3, ...) VALUES(value1, value2, value3,…..);

OR

INSERT INTO table_name VALUES(value1, value2, value3,…);

Example

INSERT INTO Employee (Eid, Ename, Esal, Age, Phone) VALUES('E-102', 'David', 28000, 35, 44004400);

OR

INSERT INTO Employee VALUES ('E-102', 'David', 28000, 35, 44004400);

From the above example, it is observed that at any time only a single row is inserted into the table. This is done based on the values specified for the columns. For instance, if the phone number of an employee is not known and a row is to be added in the table then the insertion can be done in the following two ways,

(i) INSERT INTO Employee(Eid, Ename, Esal, Age) VALUES('E-104', 'Bechkam', 30000, 32, NuLL);

(ii) INSERT INTO Employee VALuES('E-104', Bechkam', 30000, 32, NULL);

2. **UPDATE Command**

This command is used to change or modify the existing column values of a row in a table.

Syntax:

UPDATE table_name SET column_name = scalar_expression WHERE condition;

The rows that satisfy the condition of the WHERE clause is updated based on the assignments given in the SET clause.

All the rows in the table get updated whenever WHERE clause is not specified.

Example

UPDATE employee SET Eid = ‘E-120’ WHERE Age = 30;

In the above example, the Eid of an Employee whose age is 30 is set to ‘E-120’.

3. DELETE Command

This command is used to delete the existing column values of a row from the table.

Syntax :

DELETE FROM table_name WHERE condition;

All rows, which satisfy the condition of WHERE clause is deleted.

Example

DELETE FROM Employee WHERE Eid = ‘E-210’;

In the above example, all the students whose Eid is ‘E-210’ are deleted from the Employee table.

Whenever WHERE clause is not specified, all the rows in a table gets deleted. Its syntax is,

DELETE FROM table_name;

Example

DELETE FROM Employee;

In the above example, all rows from the Employee table get deleted.

BASIC STRUCTURE OF SQL QUERY:

Basic form of SQL Query:

The basic structure of an SQL query is,

SELECT [DISTINCT|ALL] Column-list

FROM TABLE-list

WHERE condition;

The above structure of an SQL query consists of following three clauses,

(i) SELECT

(ii) FROM

(iii) WHERE.

(i) SELECT Clause

SELECT clause is used to query data from a relation present in a database. The result after executing the SELECT statement is another relation which is derived from the relations specified. The resultant relation consists of list of columns that are desired to be in the result of a query.

SELECT clause can be followed by two optional keywords DISTINCT or ALL. If

the keyword DISTINCT is used along with the SELECT clause, then the resultant relation doesn't contain any duplicate rows. On the other hand, if keyword ALL is used, then it specifies that duplicate rows are eliminated from the resultant relation. Generally, asterisk symbol (*) is used to denote "all attributes" of a relation. This clause may even include arithmetic expressions that consists of arithmetic operations (+, –, ×, /) that are operating on constant or attributes of tuples.

(ii) FROM Clause

FROM clause specifies the list of relations that are to be scanned during the evaluation process so as to retrieve the desired data. It corresponds to the Cartesian product operation in a relational algebra. It is followed by table list that specifies the list of table names from which data is to be retrieved. This table list can in turn be followed by a range variable when the same table- name is used more than once in the table-list.

(iii) WHERE Clause

WHERE clause is used to specify a condition (s) based on which the resultant relation is derived. This relation contains only those rows that satisfy the given condition. SQL uses logical connectives as well as comparison operators in the WHERE clause. The operand of logical connectives are the expressions consisting of comparison operators.

Examples

Consider the following schema, Employee(Ename, Eid, Esal, Phone, Age, DNo) Department(Dname, DNo, Dept_Managerid) Project(PNo, Pname, Pduration, Plocation)

Query 1

Find distinct names of employee.

```
SELECT DISTINCT Ename
FROM Employee;
```

Query 2

Find all the employees who are working in department number 7.

```
SELECT *
FROM Employee E, Department D WHERE
D.DNo = 7;
```

Additional Basic operations:

String Functions:

String manipulation functions are the functions that are commonly used in the programming languages. These functions are very much helpful while creating a report using programming language. Some of the string manipulation functions include;

1. Concatenation

This function is used to concatenate / interconnect two different attribute values of

different columns to generate a single column value. This concatenation can be done by using '||' (which is supported in Oracle) or '+' (which is supported in both MS Access and SQL Server).

The syntax for concatenation is as follows,

attr_value || attr_value
attr_value + attr_value

Example

To retrieve all the employee names along with their Emp_id from 'Employee' table, consider the following query in Oracle,

```
SELECT Emp_name || Emp_id AS Identification.
FROM Employee;
```

In MS Access / SQL Server, the same query can also be written as,

```
SELECT Emp_name + Emp_id AS Identification
FROM Employee;
```

2. Upper/Lower

These functions are used to generate an attribute value in either uppercase or lowercase letters. Their syntax is as follows,

```
UPPER (attr_value)
LOWER(attr_value)
```

These functions are supported in both Oracle and SQL Server but not in MS Access.

Examples

(i) To retrieve all the employee_names in uppercase letters from the 'Employee' table, consider the following query:

```
SELECT UPPER(Emp_name)
FROM Employee;
```

(ii) To retrieve all the employee names in lowercase letters from the 'Employee' table, consider the following query:

```
SELECT LOWER(Emp_name)
FROM Employee;
```

3. Substring

This function is used to generate a part/substring of a given attribute value. This can be done by SUBSTR (which is supported by Oracle) or SUBSTRING (which is supported by SQL server). The function is not supported by MS Access and hence, the syntax is as follows,

```
SuBSTR(attr_value,strt_pos,attr_length)
SuBSTRING(attr_value, strt_pos, attr_length)
```

Example

To retrieve the first four characters of all the employee names from 'Employee' table, consider the following query in

Oracle:

SELECT Emp_name, SuBSTR(Emp_name, 1, 4) AS prefix

FROM Employee;

In SQL Server, the same query can also be written as:

SELECT Emp_name, SuBSTRING(Emp_name, 1, 4)AS prefix

FROM Employee;

4. Length

This function is used to generate the length of the characters in a given attribute value. This can be done by LENGTH (which is supported by Oracle) or LEN (which is supported by both MS Access and SQL Server).

The syntax is as follows,

LENGTH(attr_value)

LEN(attr_value)

Example

To retrieve the length of all employee names from 'Employee' table, consider the following query in Oracle,

SELECT Emp_name, LENGTH(Emp_name) AS LENGTHOFNAME

FROM Employee;

In MS Access / SQL Server, the same query can also be written as:

SELECT Emp_name, LEN(Emp_name) AS LENGTHOFNAME

FROM Employee;

5. Collation

It is a mechanism that is used to compare the string characters to determine which characters are smaller (ASCII code) than the other characters in a particular string.

In addition to this, SQL provides another operator LIKE operator to perform pattern matching. It is of the form, Scalar expression LIKE literal [Escape character]

Where,

Scalar expression = String value

Literal = '–' single character

= '%' zero or more character sequence.

EXAMPLE:

Consider the following table.

Eid	Ename	DNo	Esal	Age	Phone
101	John	2	35000	50	24578912
107	Henry	7	22000	25	55809192
97	David	5	30000	41	23535135
108	Sam	1	25000	32	24532121
102	Henry	2	22000	35	24578290
120	Smith	4	20000	20	56408489

To understand the collation, consider the following query,

Query:

List the names of the employees whose name start with 'H' and have 'r' as the third character.

Solution:

```
SELECT  E.ename  AS
name,  E.esal  as  Salary
FROM    Employee E
WHERE   E.ename LIKE 'H-r%';
```

This will result in a relation consisting of names of all the employees whose name start with H and third character is 'r'.

The answer in this case is,

Name	Salary
Harry	18000

SET OPERATIONS:

Set operators merge the result set of two different queries into a single result set. Before, using the set operators, it is necessary to ensure that every select statement must have same number of columns and every column belongs to same data type family. Following are the three commonly used set operators in SQL.

(iv) UNION

(v) INTERSECT

(vi) EXCEPT.

(i) UNION

uNION operator combines two or more result sets into single result set. Though UNION operator in SQL, have same functionality as join operation in relational algebra, the former generates a single result set by merging multiple result sets, whereas the latter extends the row horizontally. There are two forms of UNION operator,

(a) UNION [DISTINCT]

(b) UNION ALL.

(a) UNION [DISTINCT]

This operator returns a result set that doesn't contain any duplicate rows.

Syntax

```
SELECT column1, column2, column3.
FROM Table1
UNION [DISTINCT]
SELECT column1, column2, column3.
FROM Table2
```

Example

```
SELECT E.ename, D.DNo, D.PNo
FROM Employee E, Department D
WHERE E.eid = D. Dept_managerid AND D.PNo = 44 AND D.Dlocation = 'Hyderabad'
UNION DISTINCT
SELECT E1.ename, D1.DNo, D1.PNo
FROM Employee E1, Department D1,
WHERE E1.eid = D1.Dept_managerid AND D1.PNo = 44 AND D1.Dlocation = 'Bombay';
```

The above statement lists the unique names of the employees who are working on project number 44 and project location is either Hyderabad or Bombay.

(b) UNION ALL

This UNION operator returns the result set that contains duplicate rows.

Syntax

```
SELECT Column1, Column2, Column3.  FROM Table1
UNION ALL
SELECT Column1, Column2, Column3.  FROM Table2
```

Example

```
SELECT E.ename, D.DNo, D.Dlocation
FROM Employee E, Department D
WHERE E.eid = D.Dept_managerid AND D.PNo = 44 AND D.Dlocation = 'Hyderabad' UNION ALL
SELECT E1.ename, D1.DNo, D1.PNo
FROM Employee E1, Department D1
WHERE E1.eid = D1.Dept_managerid AND D1.PNo = 44 AND
```

D.Dlocation = ‘Bombay’;

The above statement lists all the names of employee (including duplicate names) who are working on project number 44 and project location is either Hyderabad or Bombay.

(ii) INTERSECT

INTERSECT operator combines the result sets of two different queries into a single result set that contains only the columns which are common in both relations. INTERSECT operator in SQL is similar to the inner join operator in relational algebra, but the difference is that the former finds the common columns vertically whereas the latter finds it horizontally. The following are the two forms of INTERSECT operator,

(a) INTERSECT [DISTINCT]

(b) INTERSECT ALL.

(a) INTERSECT [DISTINCT]

This operator removes duplicate rows from the final result set.

Syntax

SELECT column1, column2, column3,.

FROM Table1

INTERSECT [DISTINCT]

SELECT column1, column2, column3,.

FROM Table2

Example

SELECT E.ename FROM Employee E WHERE E.DNo

= 4 INTERSECT

SELECT E1.ename FROM Employee E1 WHERE

E1.DNo = 6;

The above statement lists the unique names of employees who are working for department 4 and 6.

(b) INTERSECT ALL

This operator doesn’t remove duplicate rows from the final result set.

Syntax

SELECT column1, column2, column3. FROM Table 1

INTERSECT ALL

SELECT column1, column2, column3. FROM Table 2

Example

SELECT E.ename FROM Employee E WHERE E.DNo

= 4

INTERSECT ALL

SELECT E1.ename FROM Employee E1 WHERE

E1.DNo = 6;

The above statement lists all the names of employees (including duplicate names) who are working for department 4 and 6.

(iii) EXCEPT

EXCEPT operator combines the multiple result set of two different queries into single result set that contain the unmatched rows present in the result set of first relation, but not in the result set of second relation. This operator is similar to "Outer join" operation but the difference is that the EXCEPT finds the unmatched rows vertically whereas the outer join finds the unmatched rows horizontally. The two forms of EXCEPT operator are,

(a) EXCEPT [DISTINCT]

(b) EXCEPT ALL.

(a) EXCEPT [DISTINCT]

This operator returns a result set that doesn't contain any duplicates.

Syntax

SELECT column1, column2, column3.

FROM Table1

ExCEPT [DISTINCT]

SELECT column1, column2, column3.

FROM Table2

Example

SELECT D.Dname FROM Department D WHERE D.PNo

= 44 INTERSECT

SELECT D1.Dname FROM Department D1 WHERE D1.PNo = 55;

The above statement lists the unique names of department that are controlling the project number 44 not project number 55.

(b) EXCEPT ALL

This operator returns a result set that contains duplicate rows.

Syntax

SELECT column1, column2, column3,.FROM Table1

ExCEPT ALL

SELECT column1, column2, column3.FROM Table2

Example

```
SELECT D.Dname FROM     Department D WHERE D.PNo = 44

EXCEPT ALL
SELECT D1.Dname FROM    Department D1 WHERE D1.PNo = 55;
```

The above statement lists all the departments (including duplicate departments) that are controlling the project number 44 not project number 55.

Null Values:

Null values are those values which are assigned to an attribute if its value is unknown or is not applicable. These values are used by DBMS when the user does not know the type of information to be entered for a particular field. When the user doesn't enter any value for a field then uppercase DBMS assigns a NULL value to that field.

A NULL value does not represent a zero or spaces it just means the absence of value for that field and this value can be inserted later.

Example

In the employee table it is not necessary that all the employees should have a phone number, so for the employer who does not have a phone number, a NULL value can be assigned. When a tuple (124, 'Kelly', 6, NULL, 26, NULL) is inserted in employee table then for the attributes e.sal and phone NULL values are assigned. Assignment of NULL value simply indicates that the value is not known or is inapplicable for that attribute. A field which is not declared as NOT NULL can have NULL values. NULL values are used to deal with the exceptional data or the data that is not complete.

Comparison using NULL Values :

It is difficult to perform comparison of valid values with NULL values if two valued Logic - TRUE or FALSE is used. Therefore to avoid this issues three valued logic - TRUE, FALSE or UNKNOWN must be used with NULL value. For example, if a NULL value is entered for salary attribute of Kelly and if a condition that queries to list all the employees whose salary >20000 is specified.

Then the evaluation of this condition for Kelly is UNKNOWN because salary is a NULL value for Kelly.

Similarly, for other comparison operators (>, <, =, <>), the evaluation of the condition will always be unknown.

SQL provides IS NULL, is NOT NULL comparison operators to compare the NULL values i.e., to test whether a value is NULL or not.

Aggregate Functions (Operations):

Basically, the aggregate operators take the entire column of data as its argument and generate a result set that summarizes the column.

Types of Aggregate Functions (Operations) :

The following are different types of aggregate functions

- COUNT
- SUM
- MAX and MIN
- AVG.

(i) COUNT

COUNT followed by a column name returns the number of non-null values (tuples) in the specified column. If DISTINCT keyword is used, then this function returns only the count value of unique tuples, otherwise, it will return count values of all tuples including duplicates.

Syntax 1:

```
SELECT COUNT([DISTINCT] column_name)
FROM   Table_name
 [WHERE condition]
```

This syntax returns the number of non-null rows that satisfy the given criteria. The parameter specified within the COUNT function can either be a column_name (like COUNT (Emp_name)) or an expression (like COUNT (Emp_salary + 1000)).

Example

```
SELECT COUNT(DISTINCT Emp_name)
FROM Employee
```

This query counts the number of different employee names from Employee table.

Syntax 2

```
SELECT  COUNT(*)
FROM table_name
[WHERE condition]
```

This syntax returns the total number of rows (including both null and non-null rows) that satisfy the given condition.

```
SELECT  COUNT(*)
FROM   Employee
WHERE Emp_salary > 10000 ;
```

This query returns the number of employees whose salary is greater than 10000 from Employee_table

(ii) **SUM**

SUM followed by a column name or an expression computes the sum of all the

attribute values specified in the given column condition or expression. If DISTINCT keyword is used, then this function returns the sum of all unique values in the column, otherwise, it will return the sum of all values (including duplicates) in the column.

Syntax

```
SELECT SUM([DISTINCT] column_name)
FROM   table_name [WHERE condition] ;
```

Example

```
SELECT SUM(Emp_salary) FROM  Employee.
```

The above query returns the sum of salaries of all the employees.

(iii) MAX and MIN

MAx followed by a column name returns the maximum valued tuple from all the tuples present in the specified column.

Syntax

```
SELECT MAX(column_name)
FROM Table_name [WHERE condition]
```

Example

```
SELECT MAx(Emp_salary) FROM Employee ;
```

The above query returns the maximum salary from the employee table.

MIN followed by a column name returns the minimum valued tuple from all the tuples present in the specified column.

Syntax

```
SELECT MIN(column_name)
FROM Table_name [WHERE condition] ;
```

Example

```
SELECT MIN(Emp_salary) FROM Employee ;
```

This query returns the minimum salary from the Employee table.

(iv) AVG

AVG followed by a column name computes the average of all the attribute values present in the specified column. If DISTINCT keyword is used then it will return average of distinct values, otherwise, the function returns the average of all values including duplicate.

Syntax

```
SELECT AVG(column_name)
FROM Table_name [WHERE condition]
```

Example

SELECT AVG(Emp_salary) FROM Employee

The above query returns the average of employee salary's from employee table.

Like MAx, MIN functions, AVG aggregate function can also be applied to numeric function so as to return only a single average value.

Nested Queries

A query inside a query is called as nested query. Inner query is called as sub query. Sub query is usually present in WHERE or HAVING clause. In other words the main query that contains the sub query is called as nested query.

Example

Find the names of employees who are working in department number 6.

Solution

SELECT E.Ename FROM Employee E WHERE E.Eid IN

(SELECT D.Dept_Managerid FROM Department D WHERE

D.DNo = 6);

The execution of this nested subquery is done in following manner,

The inner query is executed first,

SELECT D.Dept_Managerid FROM Department D WHERE D.DNo = 6

This query retrieves the Dept_Managerid of all the employees whose DNo = 6.

(i) Next, the DBMS checks the presence of resultant value of Dept_Managerid in employee table. If, it exist then it displays the result for that id (i.e., the Ename).

The main query that contains the sub queries in called as outer query.

The IN keyword can also be replaced with NOT IN keyword. In this case, it checks for the tuples which are not present in the given relation. Example: Find the employee who are not working in dept number 6,

SELECT E.Ename FROM Employee E

WHERE E.Eid NOT IN (SELECT D.Dept_Managerid FROM Department D WHERE D.DNo = 6);

Views:

A view is a virtual (logical) table that provides the user with a logical view of data. It is basically an Oracle object that is based on a SELECT query, which consists of columns, aliases and aggregate functions from an underlying table(s). These tables on which the view depends are referred to as base tables. The main purpose of defining view is to restrict the user in viewing all the columns from a table (i.e., view allows the user to view only few columns). View does not contain any data, instead it derives the required data from the base tables.

The user can think of view as any other normal relation and can perform all the

operations that can be performed on relations. Views are dynamic in nature i.e., the changes/modifications made to the views are reflected back to the original table and vice-versa.

Creation of Views

The following SQL statements are used to create a view.

CREATE VIEW (view name) (field1, field2, , field n)

AS SELECT (attribute1, attribute2, , attribute n)

FROM Table1, table2, , table n

WHERE <condition>;

Example

Consider the creation of a view for employee table.

This view gives the information about the project located at New Jersy.

CREATE VIEW Project_info (Name, Id, DNo, Pnumber, PName) AS

SELECT E.Ename, E.Eid, E.DNo, P.PNo, P.PName

FROM Employee E, PROJECT P

WHERE E.Eid = P.Eid ANDPLocation = 'New Jersy';

The view Project_info contains the following fields, Name (of employee who are working on project), Id, DNo, PNo,PName.

Id	Name	DNo	Pnumber	PName
12345263	Green	7	51	Softdrinks
12345262	Jack	7	71	Product B

Advantages of Views

1. Views provides data security – as views contain only a part of data, the data of the base table is protected from the unauthorized users as they can access only the part of data not all the data.
2. Different users can view same data from different per- spective in different ways at the same time.
3. Views can be used to hide complex queries such as the queries which involve "joins" operation. The user can issue simple queries for the views and RDBMS will take care of the complicated process.
4. Views can also be used to include extra/additional in- formation.

TRANSACTIONS:

A transaction is defined as a logical unit of work which alters the database by performing commit or rollback operations. The modifications done to a table are permanent if commit operation is performed. A transaction is initiated automatically when a SQL statement is executed and ends when DDL statement like COMMIT or ROLLBACK is executed explicitly.

1. COMMIT

COMMIT is a command used to permanently save the changes made to the table in the database. However, when a database is opened and the power shuts down before executing the COMMIT command, then the changes made to that database get lost and the actual/original table will be retained. Thus, the changes like adding, deleting and updating/modifying a row can be permanently made in the database.

Syntax: COMMIT [operation];

2. ROLLBACK

ROLLBACK is similar to the COMMIT command. Basically, it is used to restore the database to its original state/ condition.

Syntax: ROLLBACK;

Both COMMIT and ROLLBACK commands ensures update integrity requirements in a database. And they are used only with the Data Manipulation Commands, which can perform addition, deletion or updation of a particular row.

Integrity Constraints:

Integrity constraint is a condition that ensures the correct insertion of the data and prevents unauthorized data access thereby preserving the consistency of data.

1. NOT NULL Constraint

The insertion of NULL values for a field can be restricted by specifying the NOT NULL constraint within the respective field definitives. This implies that the field cannot take null values. For the primary key field it is mandatory to include NOT NULL.

It is constraint as these fields are not allowed to consider null values. Basically, the NOT NULL constraints implicitly specifies for every primary key field.

Example

```
CREATE TABLE Student (Sid INT NOT NuLL,Sname CHAR(10) NOT NULL,

Project VARCHAR(15), Class INT,PRIMARY KEY(eid));
```

In the above example, Sid is the primary key hence it must be unique and it should be NOT NULL. Project field indicates the project taken up by the student. This field can take NULL values as it is possible that a student is not interested in taking any project/not yet assigned a project. However, if this field is not allowed to take NULL values, then the NOT NULL constraint must be specified within the field.

2. Attribute-Based Check Constraints

The attribute-based CHECK constraint is considered as a part of attribute definition. It is activated upon the insertion performed on corresponding table or updation of attribute.

It is checked when a tuple is assigned a new value for that attribute, the assignment of new value could be performed using update or insert operation. And if the new value violates the constraint, the system rejects the modification. The CHECK constraint applied on attribute is checked only when,

(i) The value of the attribute is modified

(ii) The new value to the tuple is inserted.

The CHECK constraint performs changes to the values belonging to single attribute in each tuple. It can apply any condition used in WHERE clause. Additionally, the CHECK statement enforces sub query so as to specify other attributes belonging to same or other relations.

Example

Student_Report varchar(4) CHECK (Report IN ('Pass', 'Fail'))

3. Tuple-Based Check Constraints

Constraints that are applied on a single table are called 'table constraints' and can be applied on the table with the help of check option. Constraint is a condition that must be satisfied by a table and check option is used to check that condition. For example, if we want to make sure that the entire employee's salary must be greater than 15000 but less than 40000 is an employee relation, we can insert this condition at the time of creation of employee table. We may write as,

```
CREATE TABLE Employee(eid INT NOT NuLL, ename CHAR(20) NOT NuLL,
        DNo INT NOT NuLL, esal DECIMAL(10,2),eage INT NOT NuLL,
        Phone CHAR(18), PRIMARY KEY(eid),
        FOREIGN KEY(dNo) REFERENCES Department (DNo), CHECK (esal >
        = 15000 AND esal < = 40000))
```

4. Referential Integrity

Referential integrity refers to a rule in which all values related to a specified attribute in a given relation must also be present in some other relation.

(i) Key Constraints

SQL DATATYPES

The data types supported by SQL are as follows,

1. Numeric

This data type is used for storing data values, which are of type 'number'. It has several different formats like,

(i) Number (L, D)

In this, 'L' denotes the number of digits that can be included and 'D' denotes the decimal place that is to be specified. For example, Number (6, 3) specifies that the numbers that can be stored with three decimal places and may have upto six digits and also a sign before the decimal. Such type of data include the following examples,

812579.111

+2231.001

(ii) Integer

It stores only the integer values i.e., it does not include decimal values. It is abbreviated as INT.

(iii) Small Int

It also stores only the integer values but only upto a limited value i.e., upto six digits.

(iv) Decimal (L, D)

Its specification is similar to Number (L, D). But here, the storage lengths must be greater like, Decimal (10, 3), Decimal
(8, 2), etc.

2. Character

This data type is used for storing data values that are of type characters. It has several different formats like,

(i) Char (L)

It can store a fixed-length of characters i.e., from 1 to 255 characters. For instance, char(18) specifies that it can store upto
18 characters. And, if the strings used only certain characters i.e., 9 characters then the remaining spaces are left unused.

(ii) Varchar (L) or Varchar2(L)

It can store a variable-length character data. For instance, varchar (16) store upto 16 characters. But in this, the data types doesn't leave any unused space. Consequently, the RDBMS i.e., Oracle automatically converts varchar to varchar2.

3. Date

This data type is used for storing the date values in the Julian format. It can include year, month and day values.

4. Time

This data type is used for storing the data that contain hour, minute and second values.

5. Timestamp

This data type is used for storing the data that contain year, month, day, hour, minute and second values.

6. Float

This data type is used for storing the data that contain decimal values.

7. Real

This data type is used for storing the data that contains a single-precision floating point number.

8. Double Precision

This data type is used for storing the data that contains a double-precision floating point number.

9. Interval

This data type is used for storing the data that specifies a particular period of time. It has the following two formats,

(i) year-month Interval

It stores the data that contains a year, month or both values.

(ii) Day-time Interval
It stores the data that contains a day, minute or second values.

Authorization

Authorization is a technique that helps in identifying the resources which can be accessed by an authenticated user. Whereas, View is another way of ensuring data security. It puts restriction on the users such that they are not allowed to view the entire data rather allows them to view only the part of data for which they have been authorized.

For doing this, GRANT and REVOKE privileges are provided by SQL.

(i) GRANT

This privilege enables the user to perform modifications on relation using SELECT, UPDATE, DELETE, INSERT. It is used for controlling access to various DBMS resources, database objects and roles.

Syntax

GRANT<privilege list> ON TABLE table name to user name.

Example

GRANT UPDATE (std_city) ON TABLE STUDENT
TO John.

Here, John was granted an authorization of updating a single attribute 'std_city' in the relation STUDENT.

(ii) REVOKE

This privilege disables the granted privileges defined on SELECT, DELETE, INSERT, UPDATE.

Syntax

REVOKE <privilege list> ON TABLE tablename FROM username.

Example

REVOKE UPDATE (std_city) ON TABLE STUDENT FROM John

John was revoked from the authorization of 'update'. That is the ability to modify the attribute std_city which is available in STUDENT relation is not possible.

Types of View Access

The different access authorizations that can be applied to a view include,

(a) Read Authorization

It allows the user to read the data available in the database. These users are not allowed to modify the data in the database.

(b) Insert Authorization

It allows the users to insert new data into the database. But these users are not allowed to update the data which is already available in the database.

(c) Update Authorization

It allows the users to insert new data and to update existing data. But, it does not allow the user to delete the content available in the database.

(d) Delete Authorization

It allows the users to create new data, to update existing data and often delete the data from the database.

Accessing SQL From a Programming Language:

SQL provides a powerful declarative query language. Writing queries in SQL is usually much easier than coding the same queries in a general-purpose programming language. However, a database programmer must have access to a general-purpose programming language for at least two reasons:

1. Not all queries can be expressed in SQL, since SQL does not provide the full expressive power of a general-purpose language. That is, there exist queries that can be expressed in a language such as C, Java, or COBOL that cannot be expressed in SQL. To write such queries, we can embed SQL within a more powerful language.

2. Non declarative actions—such as printing a report, interacting with a user, or sending the results of a query to a graphical user interface —cannot be done from within SQL. Applications usually have several components, and querying or updating data is only one component; other components are written in general- purpose programming languages. For an integrated application, there must be a means to combine SQL with a general-purpose programming language.

There are two approaches to accessing SQL from a general-purpose programming language:

• Dynamic SQL: A general-purpose program can connect to and communicate with a database server using a collection of functions (for procedural languages) or methods (for object-oriented languages). Dynamic SQL allows the program to construct an SQL queries as character string at runtime, submit the query,andthen retrieve the result into program variables a tuple at a time. The dynamic SQL component of SQL allows programs to construct and submit SQL queries at runtime.

• Embedded SQL: Like dynamic SQL , embedded SQL provides a means by which a program can interact with a database server. However, under embedded SQL, the SQL statements are identified at compile time using a pre-processor. The pre-processor submits the SQL statements to the database system for precompilation and optimization; then it replaces the SQL statements in the application program with appropriate code and function calls before invoking the programming-language compiler.

JDBC (java database connectivity):

The JDBC standard defines an application program interface (API) that Java programs can use to connect to database servers.

```
public static void JDBCexample(String userid, String passwd)
{
try
{
Class.forName ("oracle.jdbc.driver.OracleDriver");
Connection conn = DriverManager.getConnection("jdbc:oracle:thin:@db.yale.edu:1521:univdb",userid, passwd);
Statement stmt = conn.createStatement();
try
{
stmt.executeUpdate( "insert into instructor values('77987', 'Kim', 'Physics', 98000)");
}
catch ( SQLException sqle)
{ System.out.println("Could not insert tuple. " + sqle);
}
ResultSet rset = stmt.executeQuery( "select dept name, avg (salary) "+" from instructor "+" group by dept name");
while (rset.next())
{ System.out.println(rset.getString("dept name") + " " +rset.getFloat(2));
}
stmt.close(); conn.close();
}
catch (Exception sqle)
{
System.out.println("Exception : " + sqle);
 }
 }
```

The above code shows an example Java program that uses the JDBC interface. It illustrates how connections are opened, how statements are executed and results processed, and how connections are closed.

The Java program must import java.sql.* , which contains the interface definitions for the functionality provided by JDBC .

1. Connecting to the Database

2. Shipping SQL Statements to the Database System

3. Retrieving the Result of a Query

4. Prepared Statements 5. Callable Statements

6. Metadata Features

ODBC: The Open Database Connectivity (ODBC) standard defines an API that applications can use to open a connection with a database, send queries and updates, and get back results. Applications such as graphical user interfaces, statistics packages, and spreadsheets can make use of the same ODBC API to connect to any database server that supports ODBC. Each database system supporting ODBC provides a library that must be linked with the client program. When the client program makes an ODBC API call, the code in the library communicates with the server to carry out the requested action, and fetch results.

Functions and procedures:

Procedures:

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

- At the schema level
- Inside a package
- Inside a PL/SQL block
- At the schema level, subprogram is a standalone subprogram. It is created with the CREATE PROCEDURE or the CREATE FUNCTION statement. It is stored in the database and can be deleted with the DROP PROCEDURE or DROP FUNCTION statement.
- A subprogram created inside a package is a packaged subprogram. It is stored in the database and can be deleted only when the package is deleted with the DROP PACKAGE statement.

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

  (i)Functions − these subprograms return a single value; mainly used to compute and return a value. (ii) Procedures − these subprograms do not return a value directly; mainly used to perform an action.

Each PL/SQL subprogram has a name, and may also have a parameter list.

Parameter Modes in PL/SQL Subprograms

1. **IN:** An IN parameter lets you pass a value to the subprogram. It is a read- only parameter. Inside the subprogram, an IN parameter acts like a 1 constant. It cannot be assigned a value. You can pass a constant, literal, initialized variable, or expression as an IN parameter. You can also initialize it to a default value; however, in that case, it is omitted from the subprogram call. It is the default mode of parameter passing. Parameters are passed by reference.
2. **OUT** An OUT parameter returns a value to the calling program. Inside the 2 subprogram, an OUT parameter acts like a variable. You can change its value and reference the value after assigning it. The actual parameter must be variable and it is passed by value.
3. **IN OUT:** An IN OUT parameter passes an initial value to a subprogram and returns an updated value to the caller. It can be assigned a value and the value can be read.

Creating a Procedure:

A procedure is created with the CREATE OR REPLACE PROCEDURE statement.

Syntax:

CREATE [OR REPLACE] PROCEDURE

Procedure-name [(parameter-name [IN | OUT | IN OUT] type [, ...])]

{IS | AS}

BEGIN

< Procedure-body >

END procedure-name;

Where,

Procedure-name specifies the name of the procedure.

[OR REPLACE] option allows the modification of an existing procedure.

The optional parameter list contains name, mode and types of the parameters.

IN represents the value that will be passed from outside and

OUT represents the parameter that will be used to return a value outside of the procedure.

Procedure-body contains the executable part.

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

Example

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

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

Executing a Standalone Procedure

A standalone procedure can be called in two ways −

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

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

```
EXECUTE greetings;
```

The above call will display −

```
Hello World

PL/SQL procedure successfully completed.
```

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

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

The above call will display − Hello World

/

Deleting a Standalone Procedure:

A standalone procedure is deleted with the DROP PROCEDURE statement.

Syntax:

DROP PROCEDURE procedure-name;

Example:

DROP PROCEDURE greetings;

IN & OUT Mode Example 1

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

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

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

```
Minimum of (23, 45) : 23

PL/SQL procedure successfully completed.
```

Functions:

A function is same as a procedure except that it returns a value. It returns a value.

Creating a Function :

Standalone function is created using the CREATE FUNCTION statement.

Syntax:

CREATE [OR REPLACE] FUNCTION

function_name [(parameter_name [IN | OUT | IN OUT] type [, ...])] RETURN return_datatype

{IS | AS}

BEGIN

< function_body >

END [function_name];

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

[OR REPLACE] option allows the modification of an existing function.

The optional parameter list contains name, mode and types of the parameters.

IN represents the value that will be passed from outside and

OUT represents the parameter that will be used to return a value outside of the procedure.

The function must contain a return statement.

The RETURN clause specifies the data type you are going to return from the function.

Function-body contains the executable part.

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

```
DECLARE a number;
                    b number;
                     c number;
FUNCTION findMax(x IN number, y IN number)
 RETURN number
IS z number;
BEGIN
 IF x > y THEN
 z:= x;
ELSE
Z:= y;
END IF;
RETURN z;
 END;
BEGIN
a:= 23;
b:= 45;
c := findMax(a, b);
dbms_output.put_line(' Maximum of (23,45): ' || c);
END;
```

/

When the above code is executed at the SQL prompt, it produces the following result − Maximum of (23,45): 45

PL/SQL procedure successfully completed.

Triggers:

A trigger can be defined as a program that is executedby DBMS whenever updations are specified on the database tables. More precisely, it is like an event which occurs whenever a change is done to the tables or columns of the tables. Only DBA can specify triggers.

The general format of the trigger includes the following,

(a) Event

(b) Condition

(c) Action.

(a) Event: Event describes the modifications done to the database which lead to the activation of trigger. The following fall under the category of events.

(a) Inserting, updating, deleting columns of the tables or rows of tables may activate the trigger.

(b) Creating, altering or dropping any database object may also lead to activation of triggers.

(c) An error message or user log-on or log-off may also activate the trigger.

(b) Condition: Conditions are used to specify whether the particular action must be performed or not. If the condition is evaluated to true then the respective action is taken otherwise the action is rejected.

(c) Action: Action specifies the action to be taken when the corresponding event occurs and the condition evaluates to true. An action is a collection of SQL statements that are executed as a part of trigger activation.

It is possible to activate the trigger before the event or after the event.

For example, consider the following trigger description wherein the trigger “emp-count” counts the number of employees inserted.

CREATE TRIGGER emp-count BEFORE INSERT ON EMPLOYEE /* Event */

```
DECLARE
      Count INTEGER
BEGIN
      Count : =Count( );
END
```

/* Action */

This example clearly explains that the trigger must be activated before each insertion into the employee table and the action to be taken whenever an insertion is made.

This was an example of activating a trigger before the event.

Now, consider this example, which shows the trigger activation after the event.

CREATE TRIGGER count-emp AFTER INSERT ON EMPLOYEE /* Event */

```
WHEN (new.salary < 20000)
FOR EACH ROW
```

/* Condition */

```
BEGIN
      Count : = Count + 1;
END
```

/* Action */

There are two types of triggers. They are as follows,

(a) Row-level triggers

(b) Statement-level triggers.

(a) Row-Level Triggers: Row-level triggers are the triggers that run for each row of the table that is affected by the event of triggers.

(b) Statement-Level Triggers: Statement-level triggers are the triggers that are executed only once for multiple rows which are affected by trigger action.

In the example described previously, the emp-count is a statement-level trigger as it is activated for multiple rows of the tables whereas count-emp is a row-level trigger because it is activated for each row of the table where salary is greater than 20000.

Creation of Triggers :

The triggers are created using the CREATE OR REPLACE TRIGGER statement.

Syntax:

CREATE OR REPLACE TRIGGER

[BEFORE/AFTER][DELETE/INSERT/UPDATE OF column]ON tableName

[FOR EACH ROW]

[DECLARE]

[variableName datatype [: = initial value]]

BEGIN

PL/SQL instructions;

END;

Example

Consider the STUD_RESULTS table as shown below,

STUD_RESULTS

STUD_ ID SUBJECT MARKS RESULT

A statement level trigger can be created on such a table, which can get executed implicitly whenever a new row is inserted into it or whenever its ‘marks’ attribute is updated. Its code is given below,

CREATE OR REPLACE TRIGGER TRIG_RESULTS

AFTER INSERT OR UPDATE OF MARKS ON STUD_RESULTS

BEGIN

UPDATE STUD_RESULTS

SET RESULT = ‘PASS’

WHERE MARKS > = 35;

END;

/

Trigger created.

Deletion of Triggers: As triggers are associated with data tables, deletion of data tables also deletes its corresponding trigger objects. However, specific triggers of a table can be deleted independently without deleting the table, using the DROP TRIGGER command. Its

Syntax : DROP TRIGGER trigger Name.

Recursive Queries:

Recursive queries are the type of queries that refer to themselves. These queries can be used for querying hierarchical data for expressing transitive closure.

Example

Consider the following table.

Course_id (Crs_id)	Prereq_id (Prereq_id)
CSE-100	CSE-100
CSE-200	CSE-100
IT-462	IT-400
IT-500	IT-400
EE-281	PHY-100

Figure (1): The Prereq Relation

From the above table it can be inferred that, prereq is an instance of relation which gives the information about different courses that are provided by the university along with its associated prereq_id. Suppose, if CSE-300 is a prerequisite for CSE-400 and CSE-200 is a prerequisite for CSE-300 and CSE-100 is a pre requisite for CSE-200 then CSE-300, CSE-200 and CSE-100 are all prerequisites for CSE-400.

Transitive Closure:

Transitive closure can make use of iteration in order to write queries so as to find the direct prerequisites. The iteration process is performed till it finds a new prerequisite.

Example:

As we know that a bicycle can have many subparts like pedals, wheels and so on. These sub parts are again having other subparts like spins, rims, tires etc. So, these type of hierarchies can make use of transitive closure to identify each and every part available in a bicycle.

Let us create a function find all prereq (Crs_id) which has only one parameter (i.e., course id). This function is used for computing the set of prerequisites which are direct and indirect to that course and then returns these set of prerequisites. Here, this procedure makes use of three temporary tables which are as follows,

1. Crs-prereq Temporary Table

It is used for storing the set of tuples which are required to be returned.

2. New-crs-prereq Temporary Table

It is used for storing the courses which are identified in the prior iteration.

3. Tem-temporary Table

It is used for storing data temporarily during the manipulation of the set of courses.

The above tables can be created by using create temporary table command. These tables are present till the end of the transaction and then they are deleted (dropped). After creating the

temporary tables, the procedure inserts all prerequisites which are direct to that course (crs_id) into new_crs_prereq temporary table prior to the repeat loop. This repeat loop initially inserts all courses into the new_crs_prereq temporary table to crs_prereq temporary table. Then, computation of all prerequisites of courses which are available in new_crs_prereq temporary table are performed by excluding those courses which already identified and prerequisites of crs_id are stored in temporary table (tem). After completing the execution, the content of new_crs_prereq is replaced by tem table. Hence, if no direct (new) prerequisites are not found then, repeat loop is terminated. This scenario is depicted in the below figure.

Create function findall prereq (crs_id varchar(9))

return table

```
Begin
Create temporary table crs_prereq (crs_id varchar(9))
Create temporary table new_crs_prereq (crs_id varchar(9))
Create temporary table tem (crs_id varchar(9))
Insert into new_crs_prereq
        select prereq_id
        from prereq
        where    course_id=crs_id;
repeat
        insert into crs_prereq
        select course_id
        from new_crs_prereq;
        insert into tem
              (select prereq.course_id
               from new_crs_prereq,prereq
               where new_crs_prereq.course_id=prereq.prereq_id)
except (select course_id from crs_prereq);
delete from new_crs_prereq;
insert into new_crs_prereq
       select * from tem;
delete from tem;
until not exists (select * from new_crs_prereq)
end repeat;
return table crs_prereq;
end
```

Figure (2): Finding All the Prequisties of a Course

Recursion in SQL: A recursion can be defined as a powerful technique designed to ease the designing and creation of algorithms. Primarily, the recursion is the self reference.

Recursion: It can also be defined as the process of calling a subprogram which performs the same

task. Each of these recursive calls generates a new instance of variables, cursors or any other item. Recursion can be used for programs which require a set of statements to be repeated number of times to meet a specific condition. Moreover, it can also be used to simplify the functionality of a program. Recursion in SQL is defined by using 'with recursive' clause. Here temporary view (view) can be stated in terms of itself

and with clause is used for defining this temporary view. This temporary view is used within the query in which it is defined. The word recursive is used to define that the view is recursive. Hence, this scenario can be depicted in the below function.

```
With recursive crs_prereq (crs_id, prereq_id)
as (
select crs_id, prereq_id
from prereq
Union
select prereq.prereq_id, crs_prereq.course_id
from prereq, crs_prereq
where prereq.course_id=crs_prereq.prereq_id
)
```

OLAP: OLAP refers to online analytical processing. It comprises of set of standards that are responsible for providing dimensional structure for supporting decision system. The main purpose of OLAP is to perform data analysis and to access data on-line. It provides user-friendly interface for evaluating data interactively. OLAP are tools which are dependent on multidimensional database conception. It enables highly developed users to evaluate the data using complicated and sophisticated views. They assume that data is arranged in multidimensional model, which is backed-up by relational databases. OLAP system consists of more complicated query outcomes than transactional database system. This analytical processing is performed on data warehouses which involve analysis of actual data.

The main objective of OLAP system is to assist adhoc querying required to support decision support system. It is a type of software technology that allows system analysts, manager to understand the data using fast, interactive technologies.

Properties of OLAP System:

The four main characteristics of OLAP are as follows,

(i) It support multidimensional data analysis

(ii) It provides advanced database support

(iii) It provides easy-to-use end user interface, user-friendly interface

(iv)It supports client/server architecture.

OLAP Operations:

Data in multidimensional model is arranged at different level of granularity of dimensions defined by concept hierarchies. These hierarchies specifies mapping from a specialized low-level concept to more generalized high level concepts. Multiple concept hierarchies are not directly expressed within database schema. Because of this granularity, users are provided with a benefit of viewing data from different point of views. In order to materialize these views, numerous OLAP data cube operations

are executed. These operations allow flexible interactive querying and data analysis.

The different OLAP operations are as follows,

1. Drilling operation

2. Slicing operation

3. Dicing operation

4. Rotation operation.

1. Drilling Operation

There are two kinds of drilling operations,

(i) Drill-up operation

(ii) Drill-down operation.

(i) Drill-up Operation

This operation browses or traverses from higher level generalized data to lower level specialized data within same concept hierarchy. It aggregate different data cubes by moving up the concept hierarchy for covering a dimension or by performing dimension reduction. In dimension reduction, at least one dimension is discarded from the given data cube. This

operation enables a user to ask a query that moves up an aggregation hierarchy. Roll-up operation is performed in ascending order.

(ii) Drill-down Operation

This operation is opposite of drill-up operation. It traverses from lower-level specialized data to higher-level generalized data. It can be performed either by moving down the concept hierarchy for covering a dimension or by introducing extra dimensions which adds or appends more detailed fact information to the given data. Drill-down operation is performed in descending order.

Apart from drill-up, drill-down operations there are other drilling operations that can be performed on a data cube.

Drill Across

It executes queries that involve at least two fact tables. It does this by navigating from a dimension in one concept hierarchy to different dimension in different concept hierarchy.

Drill Through

It starts from lower-level of data cube and move down to its back-end relation table. This is done by using relational SQL operations.

2. Slicing Operation

When slice operation is executed on the central data cube, a sub cube is resulted by performing selection on a single dimension of a given data cube. It views the sub cubes to get more specific information.

3. Dicing Operation

This selects a sub cube and analyzes it from different views. It performs selection on two or more dimensions. Dicing is typically performed so as to reduce the size of data cube to one or more dimension.

4. Rotation Operation

This is a visualization operation that is responsible for rotating the axes of data to different coordinate axes so as to represent data with different perspectives.