Savtribai Phule Pune University

Semester - IV (Information Technology)

Database Management System

Unit III

Chapter - 4 : Introduction to SQL

Q.1 Explain role of SQL with example. **(4 Marks)**

Ans. :

Role of SQL

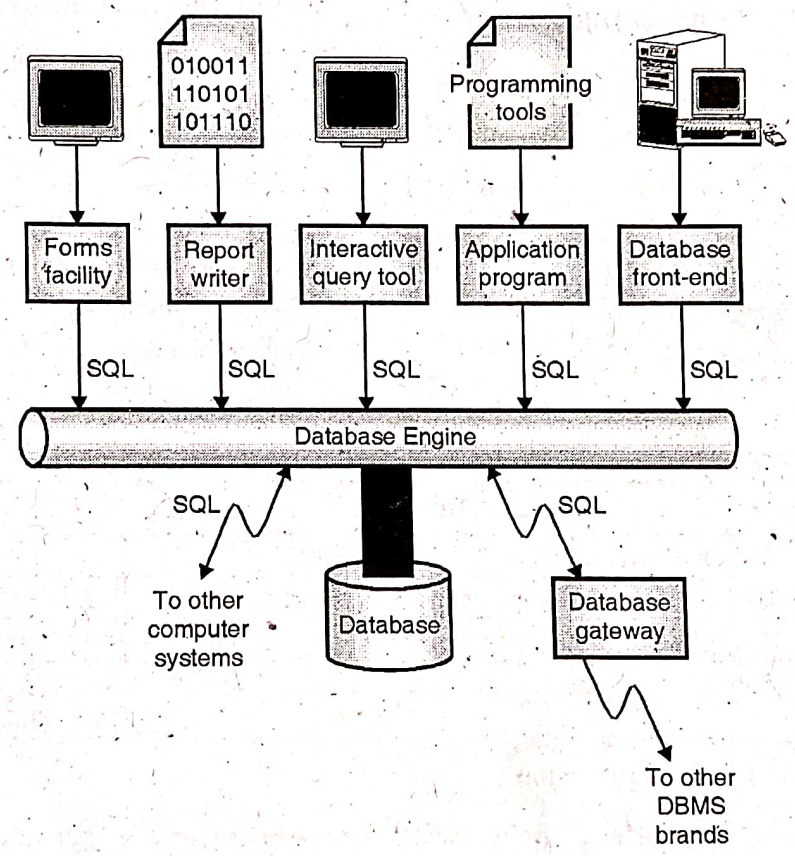


Fig. 4.1 : Role of SQL in DBMS

- SQL is an interactive query language which can be used to retrieve data from database.
- SQL is a database programming language which can be used along with programming language to access data from database.
- SQL is a database administration language which can be used to monitor and control data access by various users.
- SQL can be used as an Internet data access language.

Q.2 What are the characteristics and Advantages of SQL ? **(4 Marks)**

Ans. : Characteristics and Advantages of SQL

SQL is both an easy-to-understand language and a comprehensive tool for managing data. Some of the major features of SQL are listed as follows :

1. Microsoft commitment (SQL Server, ODBC, and ADO)
2. Internet database access
3. Java integration (JDBC)
4. Vendor independence
5. Portability across computer systems
6. SQL standards
7. IBM endorsement and commitment (DB2)
8. Relational foundation
9. High-level, English-like structure
10. Interactive, ad hoc queries
11. Programmatic database access
12. Multiple views of data
13. Complete database language
14. Dynamic data definition
15. Client/server architecture
16. Enterprise application support
17. Extensibility and object technology
18. Industry infrastructure

Q.3 Discuss various MYSQL data types.

SPPU - Dec. 17, 6 Marks

Ans. : MySQL Data Types

1. Built-in Data Types

- The basic data types available with SQL standard are as enlisted below, all data types may not be supported by SQL server or Oracle.
- Data types include numeric, character string, bit string, Boolean and datetime.

1. Numeric Data Types

This datatype is used to store a number values that can be decimal or floating point values.

(a) Integer number of various size

- These types of system are used to store natural numbers which are not having any decimal values.

  Example: 111, 23 etc.
- As per size of number we can use following types of integers.

  (i) INTEGER (p) (ii) INTEGER or INT

  (iii) SMALLINT (iv) BIGINT

Table 4.1

Sr. No.	Data type	Abbrivation	Description	Range
1.	INTEGER	INT	Integer numerical with precision 10.	– 2,147,483,648 to 2,147,483,647
2.	SMALL INTEGER	SMALLINT	Integer numerical with precision 5.	– 32768 to 32767
3.	BIG INTEGER	BIGINT	Integer numerical with precision 19.	– 9, 223, 372, 036, 854, 775, 808 to 9, 223, 372, 036, 854, 775, 807

(b) Floating point numbers of various precision

- This system used for storing decimal numbers which may be of greater size than integers.

  Example : 11.2, 12.3 etc.
- As per size of floating number we can use following types of numbers.

  (i) FLOAT or REAL (ii) DOUBLE PRECISION

Table 4.2

Sr.No.	Data Type	Abbrivation	Description	Range
1.	FLOAT (p)	FLOAT (p)	Approximate numerical with mantissa precision p.	$1 \le p \le 45$ Zero or absolute value 10^{-999} to 10^{+999}
2.	REAL	REAL	Approximate numerical with mantissa precision 7.	Zero or absolute value 10^{-38} to 10^{+38}

(c) Formatted numbers

This system used for storing some special numbers which may be of greater size than integers and floating point numbers.

Example :

1.12342 (Numeric(1,5)), 12.234 (Numeric(2,3)) etc.

(i) DECIMAL or DEC(i, j)

(ii) NUMERIC(i, j)

Where i = Precision = Total number of digits after decimal point

j = Scale = Total number of digits after decimal point.

Table 4.3 : (default value is 0)

Data Type	Description	Range
DECIMAL(p, s)	Exact numerical with precision p and scale s.	$1 \le p \le 45$ $0 \le s \le p$
NUMERIC(p, s)	Exact numerical with precision p and scale s.	$1 \le p \le 45$ $0 \le s \le p$

2. Character String Data Type

- This data type is used to store a character string which is combination of some alpha bates and enclosed in single quotation marks.

Example : 'Mahesh','abc' etc.

(a) Fixed length : CHAR (n), Where n = number of characters

Example : If 'abc' is stored in char (10) will be stored as 'abc '. (abc padded with 7 blank spaces)

(b) Varying length: VARCHAR (n) Where n = maximum number of characters

Example : If 'abc' is stored in VARCHAR (10) will be stored as 'abc' (no blank spaces)

Table 4.4

Data Type	Description	Range
CHAR(n)	Character string of fixed length n.	$1 \le n \le 15000$
VARCHAR(n)	Variable length character string of maximum length n.	$1 \le n \le 15000$

3. Date Time Data Type

(a) Date

- The DATE data type has ten positions, and its components are YEAR, MONTH and DAY in form YYYY-MM-DD.
- The length is 10.
- Generally not supported by SQL server.(Supported by DB2)
- **Example :** Date '2009-01-01' (as 'YYYY-MM-DD')

(b) Time

- The TIME data type has at least eight positions, and its components are HOUR, MINUTES and SECOND in form HH:MM:SS [.sF] where, F is the fractional part of the SECOND value.
- Generally not supported by SQL server.
- If a second's precision is not specified, *s* defaults to 0. The length is 8 (or 9 + *s*, if $s > 0$).
- **Example :** Time '11:16:59' (as 'HH:MM:SS')

(c) Timestamp / datetime

- The TIMESTAMP data type includes both date and time fields, plus a minimum of six positions and for decimal fractions of second and optional with TIMEZONE Qualifier.
- Represented using the fields YEAR, MONTH, DAY, HOUR, MINUTE and SECOND in the format YYYY-MM-DD HH:MM:SS[.sF] where, F is the fractional part of the SECOND value.
- If a second precision is not specified, *s* defaults to 6. The length is 26 (or 19, if s = 0 or 20+*s*, if $s > 0$).
- **Example :** Time Stamp '2009:01:01 11:16:59 648302'

(as 'YYYY-MM-DD HH:MM:SS TIMEZONE')

CurrentTimeStamp : Local date and time without time zone.

(d) Interval

This specifies an interval a relative value that can be used to increment or decrement an absolute value of date, time or timestamp.

INTERVAL YEAR TO MONTH Datatype.

Example 1 :

'21-5' Year(2) To Month indicates an interval of 21 years and 5 months.

'21-5' Year(2) indicates an interval of 21 Years.

'5' Month(2) indicates an interval of 5 month.

INTERVAL DAY TO SECOND Datatype.

Example 2:

'5 03:15:20' Day(2) To Second indicates an interval of 5 days,3 hours,15 minutes and 20 seconds.

Generally not supported by SQL server but supported by Oracle.

2. Literals

- Literals are explicit representations of data in for of some expressions.
- Literals are used in SQL statements to represent data that does not change.

1. **Character Literal :** The character data stored in character type of data is character literal.

   Char chr='test';

2. **Numeric Literal :** The numeric data stored in character type of data is character literal.

   Number(10,2) num_1=1002;

3. **Boolean Literals :** The character Boolean data is stored.

   Boolean bln='True';

Q.4 Explain DDL commands. **(4 Marks)**

Ans. : DDL Commands

- To create database schema and database objects like table, view, trigger we need to use **Data Definition Language (DDL).**
- DDL statements are used to build and modify the structure of your tables and other objects in the database.
- The set of DDL commands are as follows :

1. **CREATE Statement :** To create Database objects
2. **ALTER Statement :** To modify structure of database objects
3. **DROP Statement :** To remove database objects
4. **RENAME Statement :** To Rename Database objects
5. **TRUNCATE Statement :** To empty the database table

- When you execute a DDL statement, it takes effect immediately, as it is **Autocommited** into database. Hence no rollback operation (Undo) can be performed with these set of commands.
- Database objects are any data structure created in database.

Example: Table, View, Sequence etc.

Chapter – 5 : Tables

Q.1 Explain CREATE command with example. **(2 Marks)**

Ans. : Creating Tables Using CREATE Command

- To create database object like table, database, view etc. we use Data Definition Language (DDL).
- DDL statements are used to build and modify the structure of your tables and other objects in the database.
- CREATE statement is used to create any database object.

Syntax :

```
CREATE TABLE <Table_Name>
      (  Column_1 datatype,
         Column_2 datatype,
         ...
         Column_n datatype
      );
```

Example :

```
CREATE TABLE Employee
      (  Eid varchar2(10),
      First_Name char (50),
      Last_Name char (50),
      Address char (50),
      City chars (50),
      Country char (25),
      Birth_Date date
      );
```

- To view the structure of table created in database;

```
DESCRIBE Employee;
```

Q.2 Explain ALTER command with example. **(2 Marks)**

Ans. : Modifying Table Using ALTER Command

- Once table is created in database, we may require to change structure of database object. This can be done with help of ALTER command.
- The alter table statement may be used as you have seen to specify primary and foreign key constraints, as well as to make other modifications to the table structure.
- Key constraints may also be specified in the CREATE TABLE statement.

Syntax :

```
ALTER TABLE <Table_Name>
ADD Column_1 datatype ;
OR
ALTER TABLE <Table_Name>
Modify Column_1 New_datatype;
OR
ALTER TABLE <Table_Name>
DROP Column_1;
```

Example :

```
ALTER TABLE Employee
ADD Address Varchar2 (100);
```

To view the changed structure of table

```
DESCRIBE TABLE Employee;
```

Q.3 Describe DROP TABLE command of SQL with both the options CASCADE and RESTRICT.

SPPU - Dec. 18, Oct.19, 2 Marks

Ans. : Removing Table using DROP Command

- This command can be used to remove database objects from our DBMS.
- DROP command removes data from your database.

Syntax :

```
DROP TABLE <Table_Name>;
```

Example :

If we want to permanently remove the Employee table that we created, we'd use the following command.

```
DROP TABLE Employee;
```

Similarly, the command below would be used to remove the entire employees database.

```
DROP DATABASE employees ;
```

Q.4 Explain the purpose of foreign key. **(4 Marks)**

Ans. : Foreign Key

- A value appearing in one relation (table) for a given set of attributes also appears for another set of attributes in another relation (table). This is called **referential integrity.**
- The referential integrity constraint is specified between two tables to maintain the consistency among tuples in the two tables.
- The tuple in one relation refers only to an existing tuple in another relation.

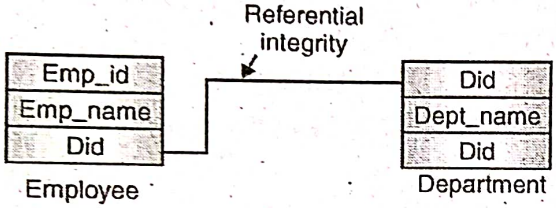


Fig. 5.1 : Referential integrity

Emp Table		
Emp_Id	**Emp_name**	**Did**
1	Sachin	20
2	Suhas	10
3	Jay	20
4	Om	10

Department Table	
Did	**Dept_name**
10	HR
20	TIS
30	L&D

- In the above example 'Emp' table has 'Did' as foreign key reference this is called as referential integrity.
- Here we are forcing database to check the 'Did' value key from the 'department' table while inserting any value of 'Emp' table in Did column if there is no value existing in department table of that 'Did' then we cannot insert that value in 'Emp' table. This helps to maintain data consistency.

Example :

For above shown relation we can write table as follows,

```
Create table Emp ( Emp_id integer,
    Emp_name varchar (100) not null,
    Did as integer references department(did))
Create table Department ( Did integer,
    Dept_name varchar (100) not null,
    Primary key (did));
```

Q.5 Write a note on DML. **(4 Marks)**

Ans. : DML

- DML is set of commands used to,
  - Insert data into table
  - Delete data from table
  - Update data of table

(1) INSERT Statement

Insert statement used to add records to the existing table.

Syntax :

```
INSERT INTO  <Table_Name>|(Column1, . . . ,
             ColumnN)
VALUES       (column1. . . , columnN);
```

Example :

```
/* To add data in employee table*/

INSERT INTO Employee
VALUES (1001, 'Mahesh');

INSERT INTO Employee
VALUES (NULL, 'Jayendra');

INSERTINTO Employee (Name,Eid)
VALUES ('Sachin' , 1002);

INSERTINTO Employee (Name,Eid)
VALUES ('Suhas' , NULL);
```

To check inserted rows, write following query.

```
/*To Print all data in employee table*/
SELECT  *
FROM Employee;
```

Output :

Eid	Name
1001	Mahesh
1002	Sachin
NULL	Jayendra
NULL	Suhas

(2) DELETE Statement

Delete statement is used to delete some or all records from the existing table.

Syntax :

```
DELETE
FROM    <Table_Name>
WHERE   Condition;
```

Example :

```
DELETE
FROM    Employee
WHERE   Eid IS NULL;
```

- If we omits the WHERE condition then all rows will get deleted.
- To check inserted rows write following query.

```
SELECT  *
FROM    Employee;
```

Output :

Eid	Name
1001	Mahesh
1002	Sachin

(3) UPDATE Statement

Insert statement used to modify the records present in existing table.

Syntax :

```
UPDATE  <Table_Name>
SET     column1=value
WHERE   condition;
```

Example :

```
UPDATE  Employee
SET     Eid=1003
WHERE   name = 'suhas';
```

- If we commit the WHERE condition then all rows will get updated with given value.
- To check inserted rows write following query.

```
SELECT  *
FROM    Employee;
```

Output :

Eid	Name
1001	Mahesh
1002	Sachin
1003	Suhas
NULL	Jayendra

Chapter – 6 : Views

Q.1 Explain the concept of view along with its operations.

SPPU - Dec. 19, 4 Marks

Ans. :

1. Definition

- A view is defined as a database object that allows us to create a virtual table in the database whose contents are defined by a query or taken from one or more tables.
- View is defined to hide complexity of query from user.

2. Base table

- The table on which view is defined is called as Base table.

3. View – as a window of entire table

- Instead of showing entire table to a user we can show a glimpsed of table to the user which is required for him

Example :

Consider a student table contains following columns,

STUDENT (Stud_Id, Stud_Name, Std, Div, Addr, Sports, Fees, Cultural_Activity);

Now for a sports teacher requires only sports related data of students so we can create view called as Stud_Sports_View for teacher as below which will only depicts sports data of student to sports teacher.

Stud_Sports_View (Stud_Id, Stud_Name, Sports)

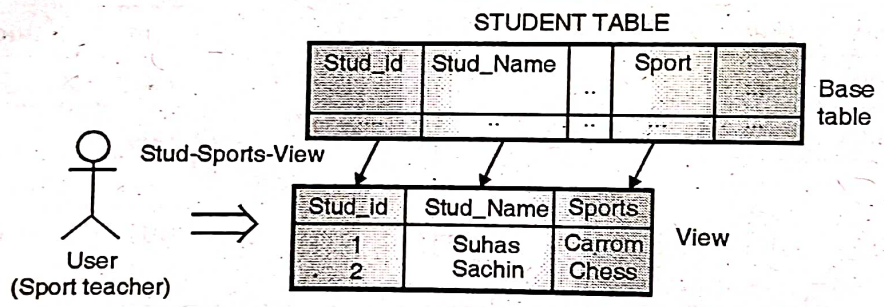


Fig. 6.1 : Overview of view

4. Types of views

(a) Simple view (b) Complex View

(a) Simple view

- The views which are based on only one table called as Simple view.
- Allow to perform DML (Data Manipulation Language) operations with some restrictions.
- Query defining simple view cannot have any join or grouping condition.

(b) Complex view

- The views which are based on more than one table called as complex view.
- Do not allow DML operations to be performed.
- Query defining complex view can have join or grouping condition.

5. Working of views

- When we call view in SQL query it refers to database and finds definition of views which is already stored in database.

- Then the DBMS convert this call of view into equal request on the base tables of the view and carries out the operations written in view definition and returns result set to query from which view is called.

Q.2 What are advantages of views ? (4 Marks)

Ans. : Advantages of Views

(a) Security

- View can restrict user from accessing all data.
- In case of view only data that is given in view is accessible to user. So all data of base table is not accessible to user which will give you security of information.
- **Example :** sports teacher can see data related to sports only and view preventing him from manipulating data pertaining to fees of students.

(b) Hides complexity

- The view may be result of very complex query. Hence instead of writing such complicated query again and again we can store such result to a view and access it whenever we want to access.
- So by writing query we can hide the complexity of original query.

(c) Dynamic nature

- View definition remains unaffected although there is any change in structure of a table.
- This dynamic nature does not hold true in case if base table is dropped or the column selected by view is altered.
- **Example :** If view is made on two tables, selecting two columns from first table and two columns from second table if we add one more column to first table does not cause any change on view.

(d) Does not allows direct access to the tables of data dictionary

- This act like functionality of safeguard to data stored in the data dictionary.
- By this way user cannot change data dictionary to damage database.
- Views can helps to make data in data dictionary easily comprehensible and helpful.

(e) Data Integrity

- If data is accessed through a view, the DBMS can automatically check the data to check for specified integrity constraints.

Q. 3 Write short note on SQL Indexes ? (6 Marks)

Ans. : SQL Indexes

1. Introduction

- An index can be created in a table to access data in database more quickly and efficiently.
- MySQL uses indexes to quickly search rows with specific column values as specified in query. Without an index, MySQL engine need to scan the entire table to locate the relevant rows. As table size increases the search speed will be reduced further.
- A database index is a data structure which will improves the speed of operations in a table. Indexes can be created using one or more columns

2. Need of Indexing

- Sometime while fetching data one or more columns in table are more repeating frequently in a WHERE clause of query then indexes on such columns can improve performance and speed of data retrieval.
- The index is generally required on column contains a wide range of values or a large number of null values.

3. Working of Index

- An index will make use of some data structure like B-Tree which will improves the speed of data retrieval on a table, it may include some additional write operations and storage for maintaining it.
- When we create any column of a table with a primary key or unique key, MySQL will automatically create an index named as PRIMARY.
- This type of index is also called the clustered index
- The index creation must consider all columns used in SQL queries and create one or more indexes on such columns.
- Practically, indexes are type of table, which keep key field and a pointer to each record into the table.
- The INSERT and UPDATE statements will take more time on tables due to updates required in index information which creates extra burden on system, whereas the SELECT statements will become fast on such tables.

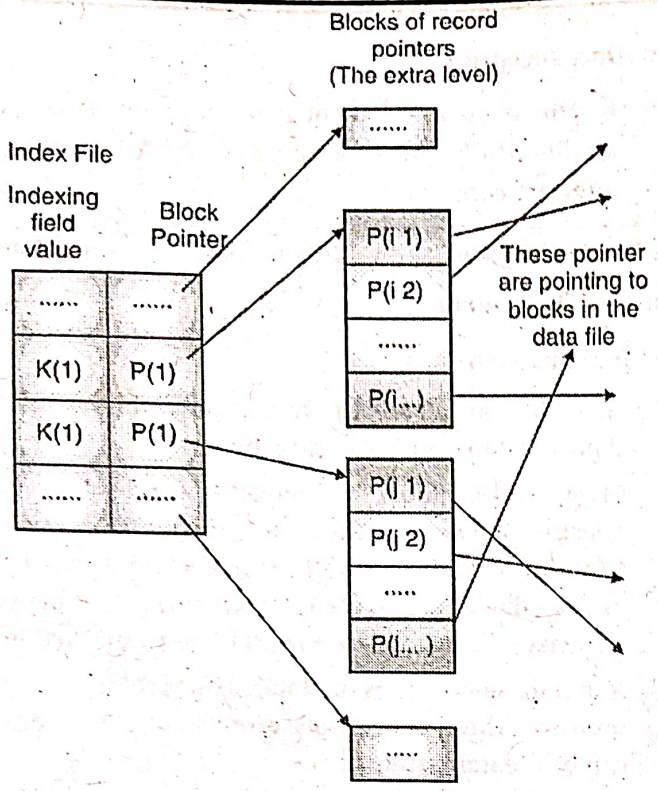


Fig. 6.2

- The PRIMARY index is stored together with the data in the same table. The clustered index will maintain the order of rows in the table.
- The indexes other than the PRIMARY index are called secondary indexes or also known as **non-clustered indexes.**
- The indexing concept may not be useful if table is very small or there is no condition is applied on column in query also is table is updated very frequently.

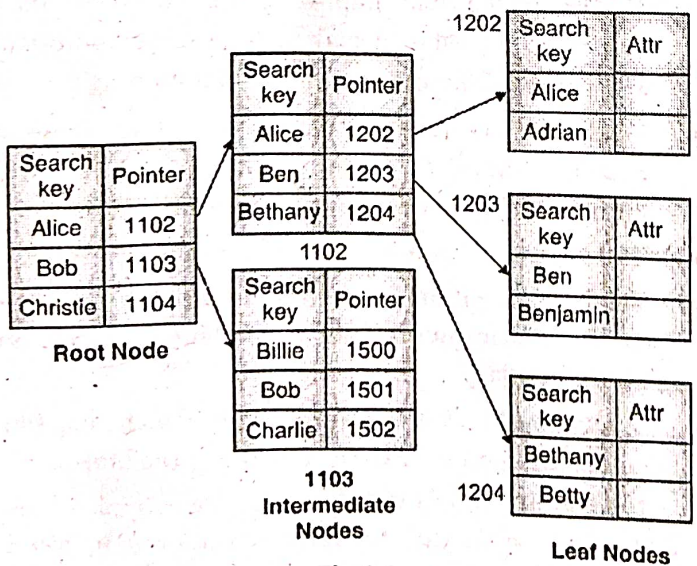


Fig.6.3

4. Performance enhancement by indexing

- The query optimizer may use indexes to locate data at faster speed and without having to scan every row in a table for a given query.
- The index may not be useful if table is very small or there is no condition is applied on column in query also is table is updated frequently.

5. Types of Simple Indexes

(i) Automatic

A unique index is created automatically when you define a PRIMARY KEY or UNIQUE constraint in a CREATE TABLE statement.

(ii) Manual

Users can create non unique indexes on columns to speed up access to the rows.

Syntax (simple Index)

```
CREATE INDEX index
ON table (column[, column]...);
```

Example of Simple Index

- Create index on LAST_NAME column in the EMPLOYEES table.

```
CREATE INDEX emp_last_name_idx
ON employees(last_name);
Index created.
CREATE TABLE Employee
(    Name INT PRIMARY KEY,
     ID INT NOT NULL,
     DEPT INT NOT NULL,
     GROUP VARCHAR(10),
     INDEX (DEPT, GROUP);
);
Table created.
```

- Create Function based index on LAST_NAME column in the EMPLOYEES table.

```
CREATE INDEX emp_last_name_idx
ON employees (UPPER(last_name));
Index created.
```

6. Removing Index

- Drop above created index.

```
DROP INDEX upper_last_name_idx;
Index dropped.
```

Chapter - 7 : SQL Queries

Q.1 Explain 'ORDER BY' clause. **(4 Marks)**

Ans. : Order BY Clause :

(1) Introduction

- The ORDER BY keyword is used to sort the result-set by a specified column.
- The ORDER BY keyword sorts the records in ascending order by default.
- If you want to sort the records in a descending order, you can use the DESC keyword.

(2) Syntax

```
SELECT    column_name
FROM      table_name
ORDER BY  column_name ASC|DESC;
```

Example :

- The "Faculty" table

Table 7.1 : Faculty

Faculty_code	Faculty_Name	DOB	Subject	Hours
100	Yogesh	17/07/64	DSA	16
101	Amit	24/12/72	MIS	16
102	Omprakash	03/02/80	PWRC	8
103	Nitin	28/11/66	DT	10
104	Mahesh	01/01/86	DT	10

- Now we want to select all the Faculties from the table above, however, we want to sort the Faculty by their Faculty_Name.
- We use the following SELECT statement :

```
SELECT    *
FROM      Faculty
ORDER BY  Faculty_Name;
```

- In second query ORDER BY 2 indicates order by second column of table.
- The result-set will look like this. (Table 7.2)

Table 7.2

Faculty_code	Faculty_Name	DOB	Subject	Hours
101	Amit	24/12/72	MIS	16
104	Mahesh	01/01/86	DT	10
103	Nitin	28/11/66	DT	10
102	Omprakash	03/02/80	PWRC	8
100	Yogesh	17/07/64	DSA	16

Example :

- Now we want to select all the Faculties from the table above, however, we want to sort the Faculties descending by their Faculty_Name.
- We use the following SELECT statement:

```
SELECT    *
FROM      Faculty
ORDER BY  Faculty_Name DESC;
```

- The result-set will look like (Table 7.2.3)

Table 7.3

Faculty_code	Faculty_Name	DOB	Subject	Hours
100	Yogesh	17/07/64	DSA	16
102	Omprakash	03/02/80	PWRC	8
103	Nitin	28/11/66	DT	10
104	Mahesh	01/01/86	DT	10
101	Amit	24/12/72	MIS	16

Q.2 Describe SET Operations. **(4 Marks)**

Ans. : SET Operations

- The results of two queries can be combined using the set operations union, intersection, and difference,

```
Query_1 UNION      [ALL] Query_2
Query_1 INTERSECT  [ALL] Query_2
Query_1 EXCEPT     [ALL] Query_2
```

- In order to calculate the union, intersection, or difference of two queries, the two queries must be "union compatible", which means that they both

return the same number of columns, and that the corresponding columns have compatible data types.

1. UNION

- Union effectively appends the result of Query_2 to the result of Query_1.
- Furthermore, it eliminates all duplicate rows, in the sense of **DISTINCT**, unless ALL is specified.

DISTINCT : Duplicate row shown only once.

ALL : Duplicate row shown only once.

Example :

```
SELECT    *
FROM      stud ;
```

Id	Name	Semester	Diploma	Gender
1	Fred	2	Bio	M
3	Tom	1	Bio	M
4	John	3	Phy	M
2	Lisa	2	Bio	F

(4 Rows Selected)

```
SELECT    *
FROM      stud_extern ;
```

Id	Name	Semester	Diploma	Gender
3	Tom	1	Bio	M
4	John	3	Phy	M
5	Simon	4	Inf	F

(3 Rows Selected)

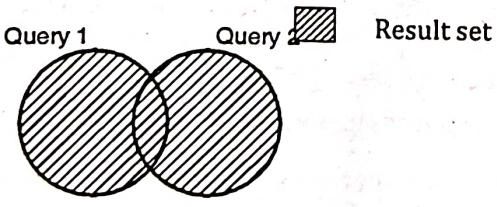


Fig. 7.1 : Union operation

```
SELECT    *
FROM      stud
UNION
SELECT    *
FROM      stud_extern ;
```

Id	Name	Semester	Diploma	Gender
1	Fred	2	Bio	M
2	Lisa	2	bio	F
3	Tom	1	Bio	M
4	John	3	Phy	M
5	Simon	4	Inf	F

(5 Rows Selected)

2. INTERSECT

- Returns all rows that are both in the result of Query_1 and in the result of Query_2. Duplicate rows are eliminated unless **ALL** is specified.

Example :

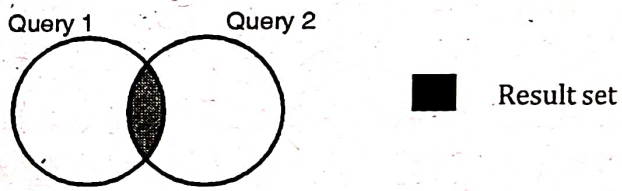


Fig. 7.2 : Intersect operation

```
SELECT    *
FROM      stud
INTERSECT
SELECT    *
FROM      stud_extern;
```

Id	Name	Semester	Diploma	Gender
3	Tom	1	Bio	M
4	John	3	Phy	M

(2 Rows Selected)

3 EXCEPT

- Returns all rows that are in the result of query1 but not in the result of query2.
- Again, duplicates are eliminated unless **ALL** is specified.

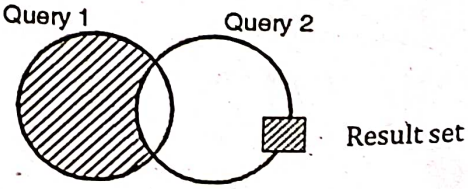


Fig. 7.3 : Except Operation

```
SELECT    *
FROM      stud
EXCEPT
SELECT    *
FROM      stud_extern ;
```

Id	Name	Semester	Diploma	Gender
1	Fred	2	Bio	M
2	Lisa	2	bio	F

(2 Rows Selected)

4 Nesting SET Operations

Set operations can be nested and chained like below Example

```
Query1 UNION query2 UNION query3
```

This really says

```
(Query1 UNION query2) UNION query3
```

Q.3 Write a note on various aggregate function. **(4 Marks)**

Ans. : Aggregate Functions

- Sometimes for decision making we need summarize data from table like average, sum, minimum etc.
- SQL provides various aggregate functions which can summarize data of given table. The function operates on the table data produces a single output.
- Such queries are generally used for producing reports and summary forms in an application.

Types of Aggregate Functions

(a) COUNT ([DISTINCT] C) : The number of (unique) values in the column C.

(b) SUM ([DISTINCT] C) : The sum of all (unique) values in the column C.

(c) AVG ([DISTINCT] C) : The average of all (unique) values in the column C.

(d) MIN (C) : The minimum value in the column C.

(e) MAX (C) : The maximum value in the column C.

Example :

Table 7.4 : Exam_Marks

Sid	SName	Marks
1	Mahesh	90
2	Suhas	80
3	Jyendra	89
4	Sachin	99
5	Vishal	88
6	Payal	90

(A) COUNT ()

- This function is used to calculate number of rows (or records) in a table selected by query.
- COUNT returns the number of rows in the table when the column value is not NULL.
- Column in the query must be numeric.

Example :

Find total number of students in above Table 7.4.

```
SELECT  Count(Sid) as Count
FROM    Exam_Marks;
```

COUNT
6

Let us consider above table modified as below, (Some duplicate rows are present in Table 7.5).

Table 7.5

Sid	SName	Marks
1	Mahesh	90
1	Mahesh	90
2	Suhas	80
3	Jyendra	89
3	Jyendra	89
4	Sachin	99
5	Vishal	88
6	Payal	90

In this case Query 1 results to wrong result as below,

```
SELECT  Count(Sid) as Count
FROM    Exam_Marks;
```

COUNT
8

So we can modify query as given below.

Example :

Find total number of diffrent students in above Table 7.5.

```
SELECT  Count(Distinct Sid) as Count
FROM    Exam_Marks ;
```

COUNT
6

(B) SUM ()

- This function is used to calculate sum of column values in a table selected by query.
- Column in the query must be numeric.
- Value of the sum must be within the range of that data type.

Example :

Find total of marks scored by all students.

```
SELECT  SUM(Marks) as Sum
FROM    Exam_Marks;
```

SUM
446

(C) AVG ()

- This function is used to calculate Average of column values in a table selected by query.
- This function first calculates sum of column and then divide by total number of rows.
- AVG returns the average of all the values in the specified column.
- Column in the query must be numeric.

Example :

Find avarage marks of students.

```
SELECT  AVG(Marks) as AVG
FROM    Exam_Marks;
```

AVG
89.33

(D) MIN()

- This function is used to find minimum value out of column values in a table selected by query.
- Column in the query need not be numeric data type.

Example :

Find minimum marks scored by students.

```
SELECT  MIN(Marks) as Min
FROM    Exam_Marks;
```

MIN
80

(E) MAX()

- This function is used to find maximum value out of column values in a table selected by query.
- Column in the query need not be numeric data type.

Example :

Find maximum marks scored by students.

```
SELECT  MAX(Marks) as Max
FROM    Exam_Marks
```

MAX
80

Q.4 Explain various types of outer join operations with example. **SPPU - Dec. 17, 6 Marks**

Ans. : Outer Join

- In an inner join or in case of a simple join, the resultant table contains only the combinations of rows that satisfy the join conditions. Rows that do not satisfy the join conditions are discarded. Outer join, joins two table although there is no match between two joining tables.
- An outer join makes one of the tables dominant. Such table is called the outer table and other table is called as subordinate table.
- In an outer join, the resultant table contains the combinations of rows from dominant table that satisfy the join conditions and also rows that do not having matching rows in the subordinate table.
- The rows from the dominant table that do not have matching rows in the subordinate table contain NULL values in the columns selected from the subordinate table.
- The syntax for performing an outer join in SQL is DBMS dependent.

Definition

- The Join that can be used to see rows of table that do not meet the join condition along with rows that satisfies join condition is called as **outer join.**

Following two types of syntax are used generally.

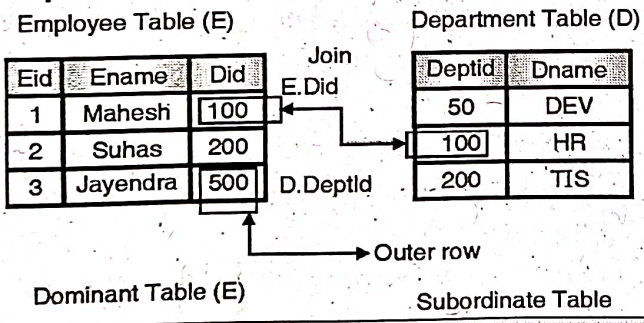
SQL 3 syntax

```
SELECT          Column_List
FROM            Table1
[LEFT/RIGHT/FULL] OUTER JOIN Table2
                ON (JOIN_Condition);
```

(A) Left Outer Join

- In the syntax of a left outer join, the dominant table of the outer join appears to the left of the keyword 'outer join'.
- A left outer join returns all of the rows for which the join condition is true and, in addition, returns all other rows from the dominant table and displays the corresponding values from the subordinate table as NULL.

Example :



Eid	Ename	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	500

Deptid	Dname
50	DEV
100	HR
200	TIS

```
SELECT  E.Eid [Eid],
        E.Did [Did],
        D.Dname [Dname]
FROM    Employee E
        LEFT OUTER JOIN
        Department D
ON      E.Did = D.DeptId;
```

Eid	Did	Dname
1	100	HR
2	200	TIS
3	500	NULL

- The above table returns rows in employees table those having or not having any department. The query below will returns only rows in which employees do not have any department. In this query, the database server applies the filter in the WHERE clause after it performs the outer join on Did column of the Employee and Department tables.

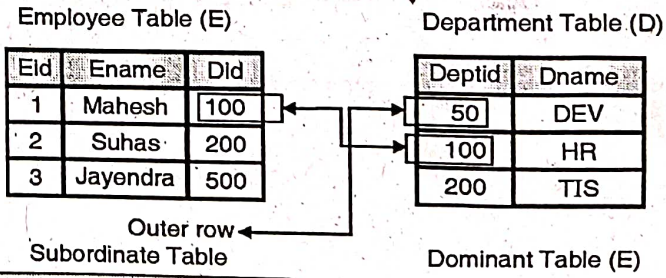
```
SELECT  E.Eid [Eid],
        E.Did [Did],
        D.Dname [DName]
FROM    Employee E
        LEFT OUTER JOIN Department D
ON      E.Did = D. DeptId
WHERE   D.DeptId IS NULL;
```

Eid	Did	DName
3	500	NULL

(B) Right Outer Join

- In the syntax of a right outer join, the dominant table of the outer join appears to the right of the keyword 'outer join'.
- A right outer join returns all of the rows for which the join condition is true and also includes all other rows from the dominant table and displays the corresponding values from the subordinate table as NULL.

Example :



Eid	Ename	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	500

Deptid	Dname
50	DEV
100	HR
200	TIS

```
SELECT  E.Eid [Eid],
        E.Did [Did],
        D.Dname [Dname]
FROM    Employee E
        RIGHT OUTER JOIN Department D
ON      E.Did = D. DeptId;
```

Eid	Did	Dname
1	100	HR
2	200	TIS
NULL	50	DEV

- Above table returns all rows from the dominant table Department and, as necessary, displays the corresponding values from the subordinate table Employee as NULL.

(C) Full Outer Join

- In full outer join both the table acts as dominant tables alternatively.
- A full outer join returns all of the rows for which the join condition is true and also includes,
  (a) All other rows from the right table and displays the corresponding values from the left table as NULL.
  (b) All other rows from the left table and displays the corresponding values from the right table as NULL.

Example :

Employee Table (E)

Eid	Ename	Did
1	Mahesh	100
2	Suhas	200
3	Jayendra	500

Department Table (D)

DeptId	Dname
50	DEV
100	HR
200	TIS

Outer row ← → Outer row

Dominant Table Dominant Table

```
SELECT  E.Eid [Eid],
        E.Did [Did],
        D.Did [Did],
        D.Dname [Dname]
FROM    Employee E
        FULL OUTER JOIN
        Department D
ON      E.Did = D. DeptId;
```

Eid	Did	Dname
1	100	HR
2	200	TIS
3	500	NULL
NULL	50	DEV

Q.5 What do you mean by correlated subquery? **SPPU - May 18, 4 Marks**

Ans. :

Correlated Sub Queries

- In case of simple subquery, first subquery will be solved then using result of subquery, outer query will be solved
- Simple subquery is evaluated once for each query.
- If the previous request is changed slightly, the subquery in the **HAVING** clause becomes a correlated subquery.
- Correlated subquery is evaluated once for each resultant row in query.
- In case of correlated subquery, first row of main query is found then for this result of main query we solve subquery.
- Correlated sub queries processes data in row-by-row pattern. Each subquery is executed once for every row of the outer query.
- In case of nested queries, we can refer to the table in the FROM clause of the outer query in the inner query using correlated sub-queries.
- We can use a correlated subquery in the HAVING clause.

Syntax :

```
SELECT  column1, column2, ...
FROM    table1 [ref]
WHERE   column1 operator ( SELECT colum1,
column2
                    FROM   table2
                    WHERE  expr1 = ref.expr2);
```

The subquery references a column from a table in the main query.

Example :

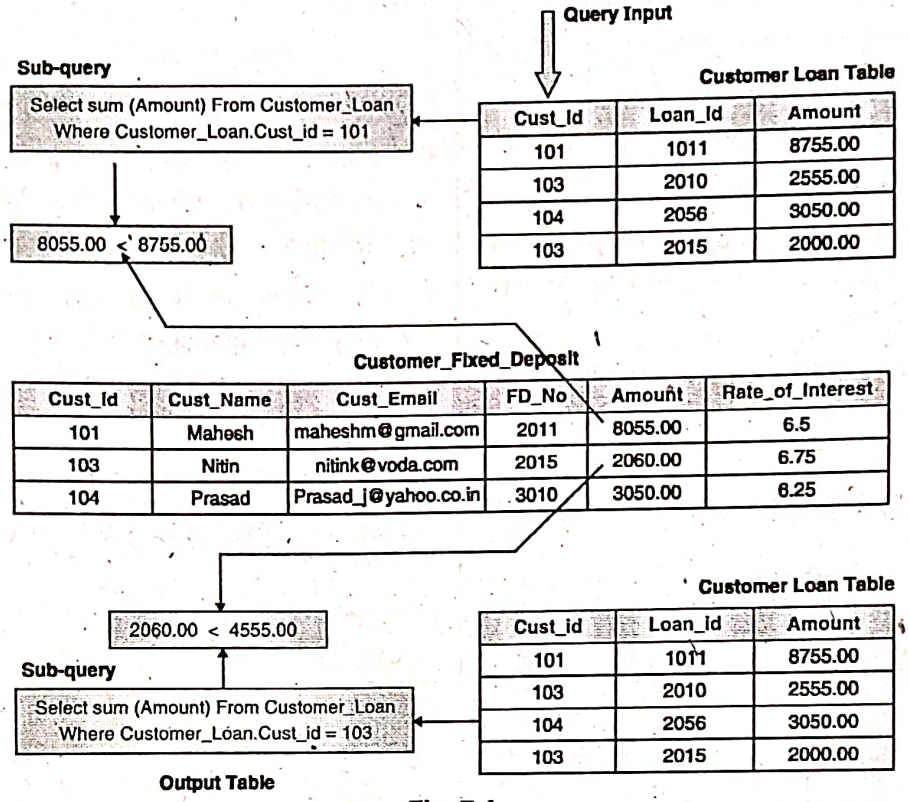


Fig. 7.4

The inner query is executed separately for each row of the outer query. i.e. in correlated sub-queries, SQL performs a sub-query over and over again – once for each row of the main query.

Example :

1. Find customers with fixed deposit less than the sum of all their loans.

```
SELECT    Cust_id, Cust_Name
FROM      Customer_Fixed_Deposit  [CFD]
WHERE     Amount < ( SELECT  sum(Amount)
          FROM    Customer_Loan [CL]
          WHERE  [CL].Cust_id = [CFD].Cust_id );
```

Find all employees who earn less than the average salary in own department.

```
SELECT   Faculty_Id, department
FROM  employees outer
WHERE   salary < ( SELECT  AVG(salary)
          FROM    employees
          WHERE  department = outer.department);
```

2. Find out the employees who have at least one person reporting to them.

```
SELECT     employee_id, last_name
FROM     employees e
WHERE    EXISTS (SELECT  'X'
                 FROM    employee
                 WHERE   manager_Id = e.employee_Id);
```

TRUE - Print row
FALSE - Do not print that row

- Inner query checks values of 'manager_id' . is matched with the any value in 'employee_id' column of employee table (of outer query).
- To check there is any employee working for respective manager.
- If result set of inner query is nonempty depicts that manager having some employees working for him. So, print result on screen.
- The EXISTS operator ensures that the search should not continue further to optimize performance of query.
- To find out the employees who have no one which is reporting to them.

```
SELECT  employee_id, last_name
FROM    employees e
WHERE   NOT EXISTS (SELECT  'X'
                    FROM    employees
                    WHERE   manager_id = e.employee_id);
```

- Inner query checks values of 'manager_id' is matched with the any value in 'employee_id' column of employee table (of outer query).
- To check there is any employee working for respective manager.
- If result set of inner query is empty depicts that manager having no employees working for him. So, print result on screen.

EMPLOYEE_ID	LAST_NAME	MANAGER_ID	SALARY
141	Rajs	142	3500

Chapter 8 : Stored Procedures

Q. 1 What are triggers? Explain trigger with example. **SPPU – Dec. 19, 8 Marks**

Ans. : Trigger

- A trigger is a procedure that is automatically invoked by the DBMS in response to specific alteration to the database or a table in database.
- Triggers are stored in database as a simple database object.
- A database that has a set of associated triggers is called an active database.
- A database trigger enables DBA (Database Administrators) to create additional relationships between separate databases.

1. Components of Trigger (E-C-A model) – How to Apply Trigger ?

- Trigger is used as ECA Model : Event-Control-Action Model.
- ECA will explain how to use trigger in applicable scenario.
- **Event (E)** - SQL statement that causes the trigger to fire (or activate). This event may be insert, update or delete operation database table.
- **Condition (C)** - A condition that must be satisfied for execution of trigger.
- **Action (A)** - This is code or statement that execute when triggering condition is satisfied and trigger is activated on database table.

2. Trigger Syntax

```
CREATE [OR REPLACE] TRIGGER <Trigger_Name>
[<ENABLE | DISABLE>]
<BEFORE | AFTER>
<INSERT | UPDATE | DELETE>
ON <Table_Name>
   [FOR EACH ROW]
DECLARE
   <Variable_Definitions>;
BEGIN
   <Trigger_Code>;
END;
```

Table 8.1 : Trigger parameter

OR REPLACE	If trigger is already present then drop and recreate the trigger
<Trigger_Name>	Name of trigger to be created.
BEFORE	Indicates that trigger is to be fired before the triggering event occurs.
AFTER	Indicates that trigger is to be fired after the triggering event occurs.
INSERT	Indicates that trigger is to be fired whenever insert statement adds a row to table.
UPDATE	Indicates that trigger is to be fired whenever Update statement modifies a row in a table.
DELETE	Indicates that trigger is to be fired whenever delete statement removes a row from table.
FOR EACH ROW	Trigger will be fired only once for each row.
WHEN	Contains condition that must be satisfied to execute trigger.
<trigger_code>	Code to be executed whenever triggering event occurs

3 Trigger Types

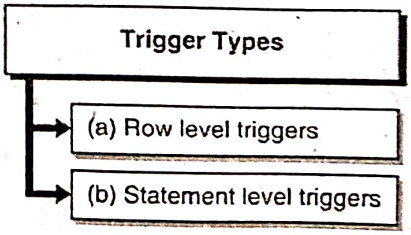


Fig. 8.1 : Trigger types

(a) Row level triggers

- A row level trigger is fired each time the table is affected by the triggering statement.

- **Example :**, if an UPDATE statement changes multiple rows in a table, a row trigger is fired once for each row affected by the UPDATE statement.
- If a triggering statement do not affect any row then a row trigger will not run only.
- If FOR EACH ROW clause is written that means trigger is row level trigger

(b) Statement level triggers

- A **statement level trigger** is fired only once on behalf of the triggering statement, irrespective of the number of rows in the table that are affected by the triggering statement
- This trigger executes once even if no rows are affected.
- **Example :** if a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only one time.
- This is Default type, when FOR EACH ROW clause is not written in trigger that means trigger is statement level trigger

4 Trigger Example

- Creating a trigger on employee table whenever new employee added a comment is written to EmpLog Table.
- Let us write a trigger and study its effect.

Example :

```
SQL> CREATE OR REPLACE TRIGGER AutoRecruit
 2   AFTER INSERT ON EMP
 3   FOR EACH ROW
 4   BEGIN
 5       Insert into EmpLog values ('Employee
Inserted');
 6   END;
 7   /
Trigger created.
SQL> INSERT INTO EMP
2 VALUES
3 (1,'Mahesh','Manager','1-JAN-1986',3000,null,10);
1 row created.
SQL> SELECT * FROM EmpLog;
STATUS
----------
Employee Inserted;
```

Consider another example, whenever there comes a new student add him to CS (Computer Science).

Example :

```
SQL>  CREATE TRIGGER CSAutoRecruit
   AFTER INSERT ON Student
   FOR EACH ROW
   BEGIN
   INSERT INTO Take VALUES (111, 'CS');
   END;
```

5 Trigger Operations

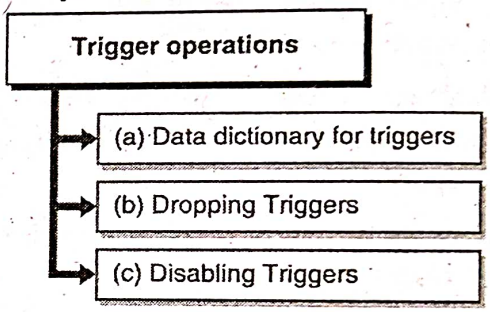


Fig. 8.2 : Triggers operations

(a) Data dictionary for triggers

Once triggers are created their definitions can be viewed by selecting it from system tables as shown below :

Syntax :

```
MySQL>   Select   *
   From   User_Triggers
   Where  Trigger_Name = '<Trigger_Name>';
```

This statement will give you all properties of trigger including trigger code as well.

(b) Dropping Triggers

To remove trigger from database we use command DROP

Syntax :

```
MySQL>Drop trigger <Trigger_Name>;
```

(c) Disabling Triggers

To deactivate trigger temporarily this can be activated again by enabling it.

Syntax :

```
MySQL> Alter trigger <Trigger_Name> {disable | enable};
```

6 Trigger Advantages \ Uses of Trigger

1. Triggers are useful for enforcing referential integrity, which preserves the defined relationships between tables when you add, update, or delete the rows in those tables. Make sure that a column is filled with default information.
2. After finding that the new information is inconsistent with the database, raise an error that will cause the

7 Trigger Disadvantages

A trigger **hampers the performance of system** as database operations will go on slower due to triggering action.

1. Restrictions on triggers

- You cannot modify the same table on which triggering event is written.
- You cannot modify a table which is connected to the triggering table by primary key, foreign key relation.

2. Mutating table errors

- This happens when the trigger is querying or modifying table whose modification activates the trigger, or a table that might need to be updated because of a foreign key constraint with a CASCADE policy.
- This problem is called as **mutating table problem.**
- Error : ORA-04091: table name is mutating, trigger/function may not see it.

❑❑❑

Chapter - 9 : Programmatic SQL

Q.1 Explain embedded SQL

SPPU - Dec. 17, Dec.19, 4 Marks

Ans. : Embedded (Static) SQL

1. Introduction

- Embedded SQL is a programming technique that enables you to build SQL statements dynamically at runtime.
- With help of Embedded SQL we can put a SQL statement inside a variable and execute that statement.
- Embedded SQL is a code that is generated programmatically (in part or fully) by your program before it is executed.
- Embedded SQL is a very flexible and powerful tool.

2. Working with various languages

- Embedded SQL is generally build on the client and executed on server machine.
- Embedded SQL in C programming is given As Below,
- If SQL statements are to be embedded within the C program, then there is a need for a mechanism to pass values between the C program environment and the SQL statements that communicate with the Oracle database server.
- Special type of variables like host variables are defined in the embedded program for this purpose.
- We can use the SELECT Statement in embedded SQL of C language as,

```
EXEC SQL begin declare section;
    int sno;
    varchar sname[50];
    varchar city[50];
    char mobile[10];
EXEC SQL end declare section;
```

- We can use select into statement which can be used if it is guaranteed that the query returns exactly one or zero rows.

```
scanf("%d",&cno);
EXEC SQL  SELECT cnameinto :cname
          FROM customers
          WHERE cno= :cno;
printf("%d",cname);
....
```

3. Advantages

- These results in SQL code created are based on the user input. We can generate the query from this input to specify what we want to as output and in what format.
- Query can be changed just by changing input parameters.
- Facilitates the automatic generation and execution of program statements.
- Embedded SQL statements can change from one execution to the next.
- Embedded SQL statements can be written by people with comparatively less programming experience, because the program does most of the actual generation of the code.
- Embedded SQL statements can be built interactively with input from users having little or no knowledge of SQL.

4. Disadvantages

- Embedded SQL statements can be entered by the application programmer or it can be generated by the program itself, they are not embedded in the source program.
- As it works in changing data requirements so it hampers performance of a system.

Q.2 Explain dynamic SQL

SPPU - Dec. 17, Dec.19, 4 Marks

Ans. : Interactive (Dynamic) SQL

Introduction

- The Dynamic SQL statements are used in situations where the SQL statements to be executed in an application program are not known at compile time. These statements are built spontaneously or dynamically as and when program is executing.
- Such, SQL mechanism is called as Dynamic SQL.
- We can use dynamic SQL to accomplish tasks such as adding where clauses to a search based on fields filled on a front end form or to create tables with varying names.

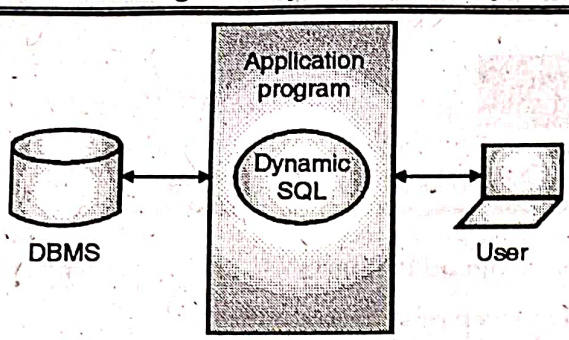


Fig. 9.1 : Working of dynamic SQL

Working with Various Languages

- Embedded SQL is generally build on the client and executed on server machine.
- **Execute Immediate Statement**
- EXEC SQL execute immediate (host-var);
- **Prepare Statement**
- EXEC SQL prepare s from :sql_stmt;
- **Execute Statement**

  EXEC SQL execute s using :var1, ..., :varn;
- **Embedded SQL in ASP programming,**

```
Dim sql
sql = "Select * FROM Employee WHERE EmpID = "
sql = sql&request.querystring("EmpID")
<!-  - Comment : Accepting Employee ID from user --
>
set results = objConn.execute(sql)
```

- The above example if you try to pass query table name as a parameter it will not be allowed in SQL server so we can use some other logic for passing table name in query, as follows

```
Create Procedure TableSelect
@TableNameVarchar(100)
AS
Declare @SQL VarChar(1000)
SELECT @SQL = 'SELECT * FROM '
SELECT @SQL = @SQL + @TableName
- - Comment: Build a SQL query as a string then
execute it
Exec ( @SQL)
- - Comment: Execute constructed SQL query as a
string
```

- Generally stored procedure can not cache the execution plan for this dynamic query. So, for complex queries performance may be hampered.
- The advantage is, of course, that you are able to achieve flexibility in your code that you can not get with standard SQL.

Advantages

- These results in SQL code created are based on the user input. We can generate the query from this input to specify what we want to as output and in what format.
- Query can be changed just by changing input parameters.
- Facilitates the automatic generation and execution of program statements.
- Embedded SQL statements can change from one execution to the next.
- Embedded SQL statements can be written by people with comparatively less programming experience, because the program does most of the actual generation of the code.
- Embedded SQL statements can be built interactively with input from users having little or no knowledge of SQL.

Disadvantages

- Not pre-parsed
- Static SQL is parsed at compile time of SQL statement and then any future executions can use the parsed version which is much more efficient. This is not possible with Dynamic SQL.
- Cannot be checked at compile time
- Dynamic SQL is valid or not we cannot determine it till you execute it. This may be results in a lot of complicated de-bugging issues.
- No dependency check
- Suppose your complex SQL query refers to a column or table that no longer exists. If the query is written as static SQL, then as soon as the column/table was removed the package will also be removed. But if it's dynamic SQL, then you will not know there's a problem until the query is actually executed.

Applications

Sometimes budget doesn't provide sufficient funding to pay for the database work or the staff isn't trained enough in writing Stored Procedures. The project is put into such development.