10. EMBEDDED SQL

- Embedded SQL is a method of inserting inline SQL statements or queries into the code of a programming language, which is known as a host language.
- This is the simplest approach to embed SQL statements directly into the source code files that will be used to create an application. This technique is referred to as embedded SQL programming.
- Structure of embedded SQL defines step by step process of establishing a connection with DB and executing the code in the DB within the high level language.
- High level programming language compilers cannot interpret SQL statements.
- Hence source code files containing embedded SQL statements must be preprocessed before compiling.
- Thus each SQL statement coded in a high level programming language source code file must be prefixed with the keywords EXEC SQL and terminated with either a semicolon or the keyword END_EXEC.

Connection to Database:

- This is the first step while writing a query in high level languages. First connection to the DB that we are accessing needs to be established.
- This can be done using the keyword CONNECT. But it has to precede with _EXEC SQLto indicate that it is a SQL statement.

```
EXEC SQL CONNECT db_name;

EXEC SQL CONNECT HR_USER; //connects to DB HR_USER
```

- Once connection is established with DB, we can perform DB transactions.

Host variables

- Database manager cannot work directly with high level programming language variables.
- Instead, it must be special variables known as host variables to move data between an application and a database.

Two types of host variables.

- Input host variables: Transfer data to database.
- Output host variable :Receives data from database

Host variables are ordinary programming language., They must be defined within a special section known as declare section.

```
EXEC SQL BEGIN DECLARE SECTION;
```

int STD_ID;

char STD_NAME [15];

char ADDRESS[20];

EXEC SQL END DECLARE SECTION;

- Each host variable must be assigned a unique name even though declared in different declaration section.

The following code is a simple embedded SQL program, written in C.

- The program prompts the user for an order number, retrieves the customer number, salesperson, and status of the order, and displays the retrieved information on the screen.

```
main() {
 EXEC SQL BEGIN DECLARE SECTION;
 int OrderID, CustID;
 char SalesPerson[10], Status[6];
 EXEC SQL END DECLARE SECTION;

 printf ("Enter order number: ");
 scanf ("%d", &OrderID);

 EXEC SQL SELECT CustID, SalesPerson, Status FROM
Orders WHERE OrderID = :OrderID INTO :CustID,
:SalesPerson, :Status;

 printf ("Customer number: %d \n", CustID);
 printf ("Salesperson: %s \n", SalesPerson);
 printf ("Status: %s \n", Status);
}
```

12. DYNAMIC SQL

- **Static or Embedded SQL** are SQL statements in an application that do not change at runtime and, therefore, can be hard-coded into the application. Dynamic SQL is SQL statements that are constructed at runtime; for example, the application may allow users to enter their own queries.
- **Dynamic SQL** is a programming technique that enables you to build SQL statements dynamically at runtime. You can create more general purpose, flexible applications by using dynamic SQL

Since query needs to be prepared at run time, in addition to the structures discussed in embedded SQL, we have three more clauses in dynamic SQL. These are mainly used to build the query and execute them at run time.

PREPARE

Since dynamic SQL builds a query at run time, as a first step we need to capture all the inputs from the user. It will be stored in a string variable. Depending on the inputs received from the user, string variable is appended with inputs and SQL keywords. These SQL like string statements are then converted into SQL query. This is done by using PREPARE statement.

EXECUTE

This statement is used to compile and execute the SQL statements prepared in DB.

```
EXEC SQL EXECUTE sql_query;
```

EXECUTE IMMEDIATE

This statement is used to prepare SQL statement as well as execute the SQL statements in DB. It performs the task of PREPARE and EXECUTE in a single line.

```
EXEC SQL EXECUTE IMMEDIATE :sql_stmt;
```

Example

```
#include stdio.h
#include conio.h
int main(){
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
int STD_ID;
char *STD_NAME;
int  CLASS_ID;
char *sql_stmt;
char *sql_query;
EXEC SQL END DECLARE SECTION;
EXEC WHENEVER NOT FOUND GOTO error_msg1;
EXEC WHENEVER SQLERROR GOTO error_msg2;
```

```
printf("Enter the Student name:");

scanf("%s", STD_Name);

printf("Enter the Class ID:");

scanf("%d", &CLASS_ID);

sql_stmt = "SELECT STD_ID FROM STUDENT ";

if (strcmp(STD_NAME, ' ') != 0)

{

sql_stmt = sql_stmt || " WHERE STD_NAME = :STD_NAME";

}

else if (CLASS_ID > 0)

{

sql_stmt = sql_stmt || " WHERE CLASS_ID = :CLASS_ID";

}

EXEC SQL PREPARE sql_queryFROM :sql_stmt;

EXEC SQL EXECUTE sql_query;

printf("STUDENT ID:%d", STD_ID);

exit(0);
```

Output

Assume the table

STUDENT

STD_ID	**STD_NAME**	**CLASS_ID**	**CITY**	**CONTACT_NO**
CS001	ARUN	101	NAGERCOIL	XXX
CS002	ASHA	102	VALLIYOR	XXX
CS025	MAHESH	202	TIRUNELVELI	XX

Enter the Student name : ASHA

(STD_NAME = ASHA)

Enter the Class ID: 202

(**CLASS_ID=202**)

(If give one valid string as student name the query will be constructed like this:)

SELECT STD_ID FROM STUDENT WHERE STD_NAME = :STD_NAME

And the output will be

CS002

Else

Enter the Student name : PRIYA

(**STD_NAME = PRIYA**)

If CLASS_ID >0 the query will be framed like

SELECT STD_ID FROM STUDENT WHERE CLASS_ID = :CLASS_ID

And the output will be

CS025