Assignment 1

Prepared by: Absalat DAWIT (ATR/8266/08)

Addis Ababa Institute of Technology

Database Administration

**Title:** Trigger

**Description:**

A database trigger is special [stored procedure](http://www.essentialsql.com/what-is-a-stored-procedure/) that is run when specific actions occur within a database.  Most triggers are defined to run when changes are made to a table’s data.  Triggers can be defined to run instead of or after DML (Data Manipulation Language) actions such as INSERT, UPDATE, and DELETE.

**Implementation:**

An example of Trigger implementation can be:

CREATE TRIGGER [student].[name] ON [student].[Order]

AFTER INSERT AS

BEGIN

DELARE @count int;

SET @count -- @rowcount;

IF @count - 0

RETURN;

BEGIN TRY

INSERT INTO [student].[Grade](

[Name]

,[ID]

,[Courses])

SELECT

Inserted.[ID]

FROM inserted;

END TRY

**Code Explanation:**

1. *The CREATE Statement* – It defines which table is associated with the trigger. In addition, this statement is used to specify when the trigger executes (e.g. after insert).
2. *The actual program -* This program runs whenever one or more rows are inserted into the Order table.
3. *Special database objects* – Triggers use specially defined databases objects such as INSERTED, or DELETED to access records affected by the database action.
4. The trigger is using the INSERTED object to gain access to the newly created rows. The INSERT statement is used to table those rows and add them to a history table.

**Title:** Table

**Description:**

Table is a database object used to create a table in database.

**Implementation:**

An example of Table implementation can be:

CREATE TABLE Student

(name varchar(20),

ID int(5)

Department varchar(20));

**Code Explanation:**

This above code will create a table named student with name, ID and department as it’s entity with the specified datatype and size.

**Title:** View

**Description:**

View is a database object used to create a view in database. A view a logical view of the table that has been created.

**Implementation:**

An example of View implementation can be:

CREATE VIEW grade

AS SELECT student\_id ID\_NUMBER, last\_name NAME,

Grade\*4 CGPA

FROM student

WHERE department = ‘ITSC’;

**Code Explanation:**

A view doesn’t contain a data of its own. It acts like a window from which the data in the table can be viewed from. The above code stores a SELECT statement.

**Title:** Sequence

**Description:**

Sequence is a database object used to create a sequence in database. Sequence is a user created database object that can be shared between multiple user to generate a unique integer. It’s usually used to create a primary key value, which unique for each row.

**Implementation:**

An example of Sequence implementation can be:

CREATE SEQUENCE Student\_ID

INCREMENT BY 1;

START WITH 99;

MAXVALUE 1000;

NOCACHE

NOCYCLE;

**Code Explanation:**

This above code will create a sequence, the will be on the student id, with it being incremented by one. Starting at 99 and having a maximum value of 100.

**Title:** Index

**Description:**

Index is a database object used to indexes a table in database. An Oracle index is used to speed up the retrieval of rows by using a pointer. Indexes can be created automatically or explicitly. Index helps for a whole table scan no to happen every time.

**Implementation:**

An example of Index implementation can be:

CREATE INDEX Student\_name

ON student(last\_name);

**Code Explanation:**

An index provides direct and fast access to rows in a table. Its purpose is to reduce the necessity of disk I/O by using an indexed path to locate data quickly. Indexes are logically and physically independent of the table they index. This means that they can be created or dropped at any time and have no effect on the base tables or other indexes.

**Title:** Cursor

**Description:** A database cursor is an object used to pinpoint records in a database. It shows you the specific record in a database that is being worked upon.

**Implementation:**

An example of Cursor implementation can be:

DECLARE @studentid as INT;

DECLARE @firstName as varchar(50),

@lastName as varchar(50);

DECLARE @stcursor as CURSOR;

SET @stcursor = CURSOR FOR

SELECT studentid,

FirstName,

LastName

FROM  Student.Student

OPEN @stcursor

FETCH NEXT FROM @stcursor INTO @studentid,

@firstName,

@lastName

WHILE @@FETCH\_STATUS = 0

BEGIN

PRINT cast(@studentid as VARCHAR (50))

+ ' - ' + @firstName

+ ' ' + @lastName;

FETCH NEXT FROM @stcursor INTO @studentid,

@firstName,

@lastName

END

CLOSE @stcursor; DEALLOCATE @stcursor;

**Code Explanation:**

We use cursor to iterate or move from one row to the next and updating rows as we go.  If we encounter an error, try something else, or skip the operation.  The difference is, that when you use cursors, you can act on each row. The print statement sends output to the SQL Server Management Studio Message Window.

**Title:** Package

**Description:**

Packages provide a method of encapsulating related procedures, functions, and associated cursors and variables together as a unit in the database.

**Implementation:**

An example of Package implementation can be:

CREATE PACKAGE BODY bank\_transactions AS

new\_status CHAR(20);

PROCEDURE do\_journal\_entry (acct NUMBER,

kind CHAR) IS

BEGIN

INSERT INTO journal

VALUES (acct, kind, sysdate);

IF kind = 'D' THEN

new\_status := 'Debit applied';

ELSIF kind = 'C' THEN

new\_status := 'Credit applied';

ELSE

new\_status := 'New account';

END IF;

END do\_journal\_entry;

**Code Explanation:**

While packages allow the database administrator or application developer to organize similar routines, they also offer increased functionality and database performance.

**Title:** Procedure

**Description:**

A **procedure** is a group of PL/SQL statements that you can call by name. Stored procedures offer advantages in the areas of development, integrity, security, performance, and memory allocation.

**Implementation:**

An example of Procedure implementation can be:

CREATE PROCEDURE remove\_emp (employee\_id NUMBER) AS

*tot\_emps NUMBER;*

*BEGIN*

*DELETE FROM employees*

*WHERE employees.employee\_id = remove\_emp.employee\_id;*

*tot\_emps := tot\_emps - 1;*

*END*;

**Code Explanation:**

The remove\_emp procedure removes a specified employee. When you call the procedure, you must specify the employee\_id of the employee to be removed.

The procedure uses a DELETE statement to remove from the employees table the row of employee\_id.

**Title:** Function

**Description:**

A **stored function** (also called a **user function** or **user-defined function**) is a set of PL/SQL statements you can call by name. Stored functions are very similar to procedures, except that a function returns a value to the environment in which it is called. User functions can be used as part of a SQL expression.

**Implementation:**

An example of Function implementation can be:

CREATE FUNCTION get\_bal(acc\_no IN NUMBER)

RETURN NUMBER

IS acc\_bal NUMBER(11,2);

BEGIN

SELECT order\_total

INTO acc\_bal

FROM orders

WHERE customer\_id = acc\_no;

RETURN(acc\_bal);

END;

**Code Explanation:**

The get\_bal function returns the balance of a specified account.

When you call the function, you must specify the argument acc\_no. The function returns the account balance. The function uses a SELECT statement to select the balance column from the row identified by the argument acc\_no in the orders table. The function uses a RETURN statement to return this value to the environment in which the function is called.

**Title:** Database Link

**Description:**

A **database link** is a schema object in one database that enables you to access objects on another database. The databases doesn’t need to an oracle Database system.

**Implementation:**

An example of Database Link implementation can be:

CREATE DATABASE LINK remote.us.oracle.com

CONNECT TO CURRENT\_USER

USING 'remote';

**Code Explanation:**

The above statement defines a current-user database link to the remote database. The user who issues this statement must be a global user registered with the LDAP directory service.

**Title:** Objects

**Description:**

Oracle Database recognizes objects that are associated with a particular schema and objects that are not associated with a particular schema. Examples of Schema Objects are: Database Links, triggers, indexes, tables, views, packages, sequences etc. Examples of Nonschema Objects are: Users, roles, profiles, etc.

**Title:** Primary Key

**Description:**

The PRIMARY KEY constraint uniquely identifies each record in a table. Primary keys must contain UNIQUE values, and cannot contain NULL values. A table can have only one primary key, which may consist of single or multiple fields.

**Implementation:**

An example of Primary Key implementation can be:

CREATE TABLE student (

ID int NOT NULL PRIMARY KEY,

LastName varchar(10),

FirstName varchar(23),

Age int);

**Code Explanation:**

This will set the student ID as it’s uniquely identifier as it’s set as a primary key.

**Title:** Unique Key

**Description:**

A unique key is a set of one or more than one fields/columns of a table that uniquely identify a record in a database table. You can say that it is little like primary key but it can accept only one null value and it cannot have duplicate values. The unique key and primary key both provide a guarantee for uniqueness for a column or a set of columns. There is an automatically defined unique key constraint within a primary key constraint. There may be many unique key constraints for one table, but only one PRIMARY KEY constraint for one table.

**Implementation:**

An example of Unique Key implementation can be:

CREATE TABLE student (

ID int NOT NULL UNIQUE,

LastName varchar(10),

FirstName varchar(23),

Age int);

**Code Explanation:**

This will set the student ID as it’s uniquely identifier.

**Title:** NOT NULL

**Description:**

The NOT NULL constraint enforces a column to NOT accept NULL values. This enforces a field to always contain a value, which means that you cannot insert a new record, or update a record without adding a value to this field.

**Implementation:**

An example of NOT NULL implementation can be:

CREATE TABLE student (

ID int NOT NULL ,

LastName varchar(10),

FirstName varchar(23),

Age int);

**Code Explanation:**

This will set that student ID can’t be a null value.

**Title:** Check

**Description:**

Checks the integrity of a database.

**Implementation:**

An example of Check implementation can be:

CHECK DATABASE [--skip-graph] [-v]

**Code Explanation:**

[-- skip - graph] will skip the check of the graph. It check the integrity of the database.

**Title:** Foreign Key

**Description:**

A FOREIGN KEY is a key used to link two tables together. A FOREIGN KEY is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table. The table containing the foreign key is called the child table, and the table containing the candidate key is called the referenced or parent table.

**Implementation:**

An example of Foreign Key implementation can be:

CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    OrderNumber int NOT NULL,  
    PersonID int,  
    PRIMARY KEY (OrderID),  
    FOREIGN KEY (PersonID) REFERENCES Persons(PersonID)  
);

**Code Explanation:**

It creates a foreign key on the “person ID” column when the “orders” table is created.

**Relational Database Vs. Object Oriented Database:**

The difference between relational database and object oriented database is that the relational data base stores data in the form of tables which contains rows and columns. Every column in the table has its specific name and every row of the table has its own primary key. While in the object oriented database the data is stored in the form of objects. In the object oriented data, the data is stored along with its actions that processes or reads the existing data.