Week 6 - Database Technologies Assignment

1)Explain the concept of constraints and differentiate between triggers with an real time PL/SQL statement.

1A) Constraints:

Constraints are the rules that we can apply on the type of data in a table. That is, we can specify the limit on the type of data that can be stored in a particular column in a table using constraints.

The available constraints in SQL are:

NOT NULL: This constraint tells that we cannot store a null value in a column. That is, if a column is specified as NOT NULL then we will not be able to store null in this particular column any more.
UNIQUE: This constraint when specified with a column, tells that all the values in the column must be unique. That is, the values in any row of a column must not be repeated.
PRIMARY KEY: A primary key is a field which can uniquely identify each row in a table. And this constraint is used to specify a field in a table as primary key.
FOREIGN KEY: A Foreign key is a field which can uniquely identify each row in a another table. And this constraint is used to specify a field as Foreign key.
CHECK: This constraint helps to validate the values of a column to meet a particular condition. That is, it helps to

ensure that the value stored in a column meets a specific condition.
DEFAULT: This constraint specifies a default value for the column when no value is specified by the user.

How to specify constraints?
We can specify constraints at the time of creating the table using CREATE TABLE statement. We can also specify the constraints after creating a table using ALTER TABLE statement.

Syntax:
Below is the syntax to create constraints using CREATE TABLE statement at the time of creating the table.

```
CREATE TABLE sample_table
(
column1 data_type(size) constraint_name,
column2 data_type(size) constraint_name,
column3 data_type(size) constraint_name,
....
);
```

sample_table: Name of the table to be created.
data_type: Type of data that can be stored in the field.
constraint_name: Name of the constraint. for example- NOT NULL, UNIQUE, PRIMARY KEY etc.

Let us see each of the constraint in detail.

1. NOT NULL –
If we specify a field in a table to be NOT NULL. Then the field will never accept null value. That is, you will be not allowed to

insert a new row in the table without specifying any value to this field.
For example, the below query creates a table Student with the fields ID and NAME as NOT NULL. That is, we are bound to specify values for these two fields every time we wish to insert a new row.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
ADDRESS varchar(20)
);
```

2. UNIQUE –
This constraint helps to uniquely identify each row in the table. i.e. for a particular column, all the rows should have unique values. We can have more than one UNIQUE columns in a table.
For example, the below query creates a table Student where the field ID is specified as UNIQUE. i.e, no two students can have the same ID. Unique constraint in detail.

```
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20)
);
```

3. PRIMARY KEY –
Primary Key is a field which uniquely identifies each row in the table. If a field in a table as primary key, then the field will not be able to contain NULL values as well as all the rows should have unique values for this field. So, in other words we can say that this is combination of NOT NULL and UNIQUE constraints. A table can have only one field as primary key. Below query will create a table named Student and specifies the field ID as primary key.

```
CREATE TABLE Student
(
ID int(6) NOT NULL UNIQUE,
NAME varchar(10),
ADDRESS varchar(20),
PRIMARY KEY(ID)
);
```

4. FOREIGN KEY –
Foreign Key is a field in a table which uniquely identifies each row of a another table. That is, this field points to primary key of another table. This usually creates a kind of link between the tables.

Consider the two tables as shown below:

Orders:

ORDER_ID	**ORDER_NUMBER**	**CUSTOMER_ID**
1	2253	3
2	3325	3
3	4521	2
4	8532	1

Customers :

CUSTOMER_ID	**NAME**	**ADDRESS**
1	RAMESH	DELHI
2	SURESH	NOIDA
3	DHARMESH	GURGAON

As we can see clearly that the field CUSTOMER_ID in Orders table is the primary key in Customers table, i.e. it uniquely identifies each row in the Customers table. Therefore, it is a Foreign Key in Orders table.

Syntax:

```
CREATE TABLE Orders
(
ORDER_ID int NOT NULL,
ORDER_NUMBER int NOT NULL,
C_ID int,
PRIMARY KEY (ORDER_ID),
FOREIGN KEY (CUSTOMER_ID) REFERENCES
Customers(CUSTOMER_ID)
)
```

(i) CHECK –

Using the CHECK constraint we can specify a condition for a field, which should be satisfied at the time of entering values for this field.
For example, the below query creates a table Student and specifies the condition for the field AGE as (AGE >= 18). That is, the user will not be allowed to enter any record in the table with AGE < 18. Check constraint in detail

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int NOT NULL CHECK (AGE >= 18)
);
```

(ii) DEFAULT –

This constraint is used to provide a default value for the fields. That is, if at the time of entering new records in the table if the user does not specify any value for these fields then the default value will be assigned to them.
For example, the below query will create a table named Student and specify the default value for the field AGE as 18.

```
CREATE TABLE Student
(
ID int(6) NOT NULL,
NAME varchar(10) NOT NULL,
AGE int DEFAULT 18
);
```

Difference between Constraints & Triggers

S.no	**Constraints**	**Triggers**
1	This is defined by the relationship between data elements.	These are the actions that are executed when a specific reaction occurs.
2	It is used for a column.	It is used for a table.
3	These are useful for database consistency.	These are used for logging and auditing.
4	It precedes triggers when getting fired.	First constraints get fired. Only then do triggers get fired.
5	Constraints affects all the rows i.e. the once that existed before and the ones that were newly added.	Trigger affects only those rows, which are added after it is enabled.
6	Trigger affects only those rows, which are added after it is enabled.	Triggers unlike constraints is capable of implementing high-end business rules that are complicated.

I have tables: Teams and Matches.

'Teams' stores the information on football teams and the one I'm particularly interested in is the Country field.

'Matches' stores the information on football matches that happen between the teams listed in 'Teams'. The field I'm interested in from this table is the Competition field. Depending on the type of competition: "All-England" or "All-Spain", etc. the teams involved should be from the appropriate countries.

I'm currently trying to write a Constraint Trigger to handle this but am being thrown a really cryptic error by SQL Developer "invalid and failed re-validation" upon running even if it compiles alright. Anyone care to help? Suggestions on doing things a better way would be great too. PL/SQL block below and I apologise if the following code makes any of you cry. I know I'm terrible at this

```
CREATE OR REPLACE TRIGGER matchcountry_BIR
BEFORE INSERT ON matches
FOR EACH ROW

DECLARE
invalidEng EXCEPTION;
invalidSpa EXCEPTION;
team1 teams.Country%type;
team2 teams.Country%type;
comp matches.Competition%type;

BEGIN
SELECT Country INTO team1 FROM teams WHERE TeamID =
:new.TeamID_A;
```

PL/SQL Constraint Trigger

```
SELECT Country INTO team2 FROM teams WHERE TeamID =
:new.TeamID_B;
comp:=:new.Competition;

IF (comp='All England') AND (team1!='England' AND
team2!='England') THEN
RAISE invalidEng;
END IF;

IF (comp='All Spain') AND (team1!='Spain' AND
team2!='Spain') THEN
RAISE invalidSpa;
END IF;

EXCEPTION
WHEN invalidEng THEN
RAISE_APPLICATION_ERROR(-20005, 'Countries are invalid for
a English competition.');
END;

EXCEPTION
WHEN invalidSpa THEN
RAISE_APPLICATION_ERROR(-20006, 'Countries are invalid for
a Spanish competition.');
END;
```

In case ,the problem lies with the tables, I've also included their Create statements. Again, any suggestions or criticisms on these are very welcome.

```
CREATE TABLE teams
(
  TeamID number(2) PRIMARY KEY,
  TeamName varchar2(50),
  Country varchar2(30),
  CHECK (Country='Spain' OR Country='England')
);

CREATE TABLE matches
(
  MatchID number(2) PRIMARY KEY,
  TeamID_A number(2),
  TeamID_B number(2),
  Goal_A number(2),
  Goal_B number(2),
  Competition varchar2(50),
  CONSTRAINT fk_TeamA FOREIGN KEY(TeamID_A)
REFERENCES teams,
  CONSTRAINT fk_TeamB FOREIGN KEY(TeamID_B)
REFERENCES teams,
  CHECK (Goal_A >= 0),
  CHECK (Goal_B >= 0),
  CHECK (Competition='Champions League' OR
Competition='Europa League' OR Competition='All England' OR
Competition='All Spain')
);
```

You had 3 mistakes:

&& instead of AND.
Accessing matches table instead of :new.competition.
Double EXCEPTION keyword.
Try below code:

```
-- Trigger will not let an insert on the Match table with invalid
countries for the competition
CREATE OR REPLACE TRIGGER matchcountry_BIR
BEFORE INSERT ON matches
FOR EACH ROW

DECLARE
  invalidEng EXCEPTION;
  invalidSpa EXCEPTION;
  team1 teams.Country%type;
  team2 teams.Country%type;
  comp matches.Competition%type;

BEGIN
  SELECT Country INTO team1 FROM teams WHERE TeamID =
:new.TeamID_A;
  SELECT Country INTO team2 FROM teams WHERE TeamID =
:new.TeamID_B;
  --SELECT Competition INTO comp FROM matches WHERE
MatchID = :new.MatchID;
  comp := :new.competition;

  IF (comp='All England') AND (team1!='England' AND
team2!='England') THEN
    RAISE invalidEng;
  END IF;
  IF (comp='All Spain') AND (team1!='Spain' AND
team2!='Spain') THEN
    RAISE invalidSpa;
  END IF;

EXCEPTION
  WHEN invalidEng THEN
    RAISE_APPLICATION_ERROR(-20005, 'Countries are invalid
for a English competition.');

  WHEN invalidSpa THEN
    RAISE_APPLICATION_ERROR(-20006, 'Countries are invalid
for a Spanish competition.');
END;
```

2) What are the other uses of Triggers?

2A) Triggers are programs that are available in the memory, with unique names made up of SQL queries which we need to fire on our database on and off.

Triggers can be made to insert, update and delete statements in SQL. We have two types of triggers:

1. Row Level Triggers

In a row-level trigger, the changes are put on all the rows on which the insert, delete, or update transaction is performed.

If we have 1000 rows in a database and a delete operation is being run on them, then the trigger would also run 1000 times automatically. It is accessible in MySQL.

2. Statement Level Triggers

In the statement-level triggers, the operation is under execution only once no matter how many rows are involved in the operation. These triggers are not accessible by MySQL.

Uses of Triggers in SQL

Some of the prominent advantages of triggers are as follows:

1. Helps us to automate the data alterations.

2. Allows us to reuse the queries once written.

3. Provides a method to check the data integrity of the database.

4. Helps us to detect errors on the database level.

5. Allows easy auditing of data.Create an audit trail of activity in the database. For example, you can track updates to the orders table by updating corroborating information to an audit table.

6. Implement a business rule. For example, you can determine when an order exceeds a customer's credit limit and display a message to that effect.

7. Derive additional data that is not available within a table or within the database. For example, when an update occurs to the quantity column of the items table, you can calculate the corresponding adjustment to the total_price column.

8. Enforce referential integrity. When you delete a customer, for example, you can use a trigger to delete corresponding rows that have the same customer number in the orders table.

3)Write the syntax for creating a trigger.

3A)Syntax:

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]
```

Explanation of syntax:

create trigger [trigger_name]: Creates or replaces an existing trigger with the trigger_name.

[before | after]: This specifies when the trigger will be executed.

{insert | update | delete}: This specifies the DML operation.
on
[table_name]: This specifies the name of the table associated with the trigger.

[for each row]: This specifies a row-level trigger, i.e., the trigger will be executed for each row being affected.

[trigger_body]: This provides the operation to be performed as trigger is fired

BEFORE and AFTER of Trigger:
BEFORE triggers run the trigger action before the triggering statement is run.
AFTER triggers run the trigger action after the triggering statement is run.

Example:
Given Student Report Database, in which student marks assessment is recorded. In such schema, create a trigger so that the total and average of specified marks is automatically inserted whenever a record is insert.

Here, as trigger will invoke before record is inserted so, BEFORE Tag can be used.

```
mysql> desc Student;
+-------+-------------+------+-----+---------+----------------+
| Field | Type        | Null | Key | Default | Extra          |
+-------+-------------+------+-----+---------+----------------+
| tid   | int(4)      | NO   | PRI | NULL    | auto_increment |
| name  | varchar(30) | YES  |     | NULL    |                |
| subj1 | int(2)      | YES  |     | NULL    |                |
| subj2 | int(2)      | YES  |     | NULL    |                |
| subj3 | int(2)      | YES  |     | NULL    |                |
| total | int(3)      | YES  |     | NULL    |                |
| per   | int(3)      | YES  |     | NULL    |                |
+-------+-------------+------+-----+---------+----------------+
7 rows in set (0.00 sec)
```

SQL Trigger to problem statement:

```
create trigger stud_marks
before INSERT
on
Student
for each row
set Student.total = Student.subj1 + Student.subj2 +
Student.subj3, Student.per = Student.total * 60 / 100;
```

Above SQL statement will create a trigger in the student database in which whenever subjects marks are entered, before inserting this data into the database, trigger will compute those two values and insert with the entered values. i.e.,

```
mysql> insert into Student values(0, "ABCDE", 20, 20, 20, 0,
0);
Query OK, 1 row affected (0.09 sec)

mysql> select * from Student;
+-----+-------+-------+-------+-------+-------+------+
| tid | name  | subj1 | subj2 | subj3 | total | per  |
+-----+-------+-------+-------+-------+-------+------+
| 100 | ABCDE |    20 |    20 |    20 |    60 |   36 |
+-----+-------+-------+-------+-------+-------+------+
1 row in set (0.00 sec)
```

In this way trigger can be creates and executed in the databases.

4)Create student, department t and college table and execute the below

Create a trigger for auto CGPA
Stop rows inserting marks more than 100 marks
How many triggers are possible for a table?
Update percentage after each semester, using a update trigger

4A)

CREATE TABLE Student (StudentID INTEGER NOT NULL, DepartmentID INTEGER, CollegeID INTEGER, CGPA DECIMAL(3,2), PRIMARY KEY (StudentID)); CREATE TABLE D