2 : Database Constraints and Triggers

IT3306 – Data Management Systems

Level II - Semester 3

Overview

- Relational Model Constraints
- Specifying Constraints in SQL
- Constraints in Databases as Assertions
- Specifying Actions in Databases as Triggers

Intended Learning Outcomes

- At the end of this lesson, you will be able to:
    - Understand what relational model constraints are
    - Identify constraints violations
    - Write constraints in SQL
    - Understand what assertions are
    - Define what a trigger is
    - Write trigger statements

List of Sub topics

List of Sub topics

Relational Model Constraints

- Categories of Constraints
- Domain Constraints
  - Key Constraints and Constraints on NULL Values
  - Entity Integrity and Referential Integrity
  - Other Types of Constraints
  - Handling Constraint Violations for Insert, Delete and Update Operations

Categories of Constraints

- In this section, we discuss what a constraint is and categories of constraints.
- Constraint is a condition that specifies restrictions on the database state.
- **Constraints maintain data integrity of a database**. That is, constraints ensure that the values entered to a database is accurate and valid.
- For an example, NIC column should hold unique values and then when you insert or update values to this column, duplicate values should not be allowed. Constraint defined on the NIC column should maintain this uniqueness with respect to the data manipulation operations. The above example highlights only one type of constraint. However, constraint types in relational model can be divided into three main categories.

Categories of Constraints cont.

- Three constraint categories are:
  - Inherent model-based constraints or **implicit constraints**
    - They are the constraints which are inherent in relational model itself.
  - Schema-based constraints or **explicit constraints**
    - These constraints apply when you use Data Definition Language (DDL) to specify a schema.
  - Application-based or semantic constraints or **business rules**
    - These are the constraints that cannot directly be defined in the schema and are enforced through application programs or triggers. E.g. The salary of an employee should not exceed the salary of the employee’s supervisor or the maximum number of hours that an employee can work on all projects per week is 40.

Activity

- Map each statement into the correct column.
    - Primary key (eno)
    - Salary of an employee should not be greater than his or her manager
    - Foreign Key (dept_no)
    - Employee’s NIC should not contain NULL values
    - No duplications for tuples

Implicit constraints	Explicit constraints	Business rules

Implicit or Inherent model-based constraints

These constraints are assumed to hold by the definition of the relational model (i.e., built into the system and not specified by a user).

- **Inherent constraints**
  - A relation consists of a certain number of simple attributes.
  - An attribute value is atomic
  - No duplicate tuples are allowed

Implicit or Inherent model-based constraints

- Each attribute value in a tuple should have an atomic value; that is, attribute value is not divisible into components within the relational model.
- Hence, composite and multivalued attributes cannot be naturally represented.
- This model is known as the flat relational model.
- The theory behind the relational model was developed based on the first normal form assumption.
- As a result, multivalued attributes should be represented by separate relations, and composite attributes are represented only by their simple component attributes in the basic relational model.

Explicit or Schema-based Constraints

- Explicit or schema-based constraints apply directly in the schemas by defining them in Data Definition Language(DDL).
- There are different types of schema-based constraints. They are:
  - Domain constraints
  - Key constraints
  - Constraints on NULLs
  - Entity integrity constraints
  - Referential integrity constraints.

Domain Constraints

- Domain constraints ensure that the data value entered for a particular column matches with the pre-defined data type of that column. The pre-defined data types are Integers, Real numbers, Characters, Booleans, Fixed-length strings, Variable-length strings, Date, time etc. In the below query, each column is created with a data type. When data is entered to each column, it only allows values of the defined data type.
- Following is the SQL to create *Student* table.

```
CREATE TABLE STUDENT(
        STU_NO          CHAR(05),
        STU_NAME        VARCHAR(35) ,
        STU_DOB         DATE,
        EXAM_FEE        INT,
        STU_ADDRESS  VARCHAR(35) ,
        PRIMARY KEY (STU_NO));
```

Domain Constraints

- Domains can also be described by a subrange of values from a data type or as an enumerated data type in which all possible values are explicitly listed. This is facilitated through CHECK constraint in SQL as illustrated below.

```
CREATE TABLE STUDENT(
    ENROLL_NO CHAR(05) NOT NULL ,
    STU_NAME VARCHAR (35),
    STU_ADDRESS VARCHAR (35),
    STU_AGE INT CHECK (STU_AGE >= 18),
    GENDER   VARCHAR(06)
       CHECK  (GENDER in ('Male', 'Female’)),
    PRIMARY KEY (ENROLL_NO)
);
```

Checks whether the gender is Male or Female.

Domain Constraints

- UNIQUE Constraint enforces a column or set of columns to have unique values. Therefore, that specific column cannot contain duplicate values. When you define the PRIMARY KEY, then by default it maintains having unique values.

```
CREATE TABLE STUDENT(
    ENROLL_NO    CHAR(05)    NOT NULL,
    NIC  CHAR(10)   UNIQUE,
    STU_NAME  VARCHAR (35) NOT NULL,
    STU_AGE   INT  NOT NULL,
    STU_ADDRESS  VARCHAR (35),
    PRIMARY KEY (ENROLL_NO));
```

Domain Constraints

- The DEFAULT constraint is used when there is no value to insert as the column value of a table, instead it provides a default value.

```
CREATE TABLE STUDENT(
  ENROLL_NO CHAR(05) NOT NULL,
  STU_NAME VARCHAR (35) NOT NULL,
  STU_DoB DATE NOT NULL,
  EXAM_FEE INT DEFAULT 10000,
  STU_ADDRESS VARCHAR (35),
  PRIMARY KEY (ENROLL_NO)
);
```

Exam_Fee is set to 10000 which is the default value.

Activity

- Using SQL create UnderGrad_Student table for the given relation. Where DEFAULT Reg_course is ‘Computer Science’.

  UnderGrad_Student (<u>Sid, Enroll_Year</u>, Name, Address, Age, Reg_course)

Key Constraints

- No two tuples should have the same combination of values for their attributes. The value of a key attribute can be used to identify each tuple uniquely in the relation.
- This property is **time-invariant**.
- If a relation has more than one key, they are called candidate keys.
- In general, a candidate key with a fewer number of attributes is selected as the primary key.

```
CREATE TABLE Employee (
 Emp_id CHAR(05) PRIMARY KEY,
 Emp_name VARCHAR(55) NOT NULL,
 Hire_date DATE NOT NULL,
 NIC CHAR(10) NOT NULL,
 Salary DECIMAL (9,2)  NOT NULL );
```

Specifying Key and Referential Integrity Constraints: Primary Key Cont.

- Primary key of a relation can be a combination of more than one attribute which is known as a composite key. In this situation, you cannot state the primary key when declaring attributes. That is, the primary key has to be defined separately.

```
CREATE TABLE DEPENDENT(
        EMP_NO    CHAR(05),
        DEPENDENTREF   CHAR(05),
        DEPENDENT_NAME VARCHAR (35),
        AGE INT NOT NULL,
        PRIMARY KEY (EMP_NO, DEPENDENTREF));
```

Key Constraints

- In a relation, there are subset of attributes that when taken/considered together would enable unique identification of each tuple. This subset of attributes is known as a *superkey*.
- A *superkey* is a set of attributes that can be used to identify a tuple uniquely. A candidate key is a minimal set of attributes that require in identifying a tuple which is also known as a *minimal superkey*.
- It is a minimal superkey where you cannot remove any attribute to hold the uniqueness.

Key Constraints

Employee

Eid	Ename	Address	Salary	NIC	DoB
E1001	Amal	Kandy	200000	751234567V	1/1/1989
E1002	Sunil	Colombo	150000	772345678V	13/4/1980
E1003	Nimal	Matara	175000	822131412V	23/2/1975

- In the above relation, possible superkeys are: {Eid, Ename},

  {Eid, Ename, Address}

- In the above relation, possible candidate keys are: Eid and NIC

Entity Integrity Constraints

- The **entity integrity constraint** states that no primary key value can be NULL.
- This is because the primary key value is used to identify individual tuples uniquely in a relation.
- Having NULL values for the primary key implies that it is not possible to identify tuples uniquely in the database.

Activity

Match the correct answers.

• Constraint ensures that	allows values of the defined data type
• Domain constraint	subset of attributes
• Super key of a relation is	a schema-based constraints
• Referential key is	the values entered to a database is accurate and valid

NOT NULL Constraints

- Other than the Primary Key, there can be other attributes which cannot contain NULL values.
- For an example, if the columns Empname, Hire_date, NIC and salary cannot contain NULL values, when the table is created, you can state it as follows.

```
CREATE TABLE Employee (
  Emp_id CHAR(05) PRIMARY KEY,
  Emp_name VARCHAR(55) NOT NULL,
  Hire_date DATE NOT NULL,
  NIC CHAR(10) NOT NULL,
  Salary DECIMAL (9,2)  NOT NULL );
```

NOT NULL Constraints

- NOT NULL constraint makes sure that a column does not hold NULL value. When we cannot give a value for a particular column while inserting a record into a table, default value taken by it is NULL. By specifying NOT NULL constraint, we ensure that a particular column(s) does (do) not contain NULL values.

```
CREATE TABLE STUDENT(
    ENROLL_NO    CHAR(05)  NOT NULL,
    STU_NAME VARCHAR (35)  NOT NULL,
    STU_DoB DATE,
    EXAM_FEE INT,
    STU_ADDRESS VARCHAR (35) ,
    PRIMARY KEY (ENROLL_NO));
```

As per the SQL, ENROLL_NO and STU_NAME cannot contain NULL Values

Candidate Keys

- If a relation has more than one key, they are called **candidate keys**.
- In general, we select a candidate key with a fewer number of attributes as the primary key. The remaining candidate keys are nominated as unique keys.
- When an attribute contains UNIQUE values and a NOT NULL constraint, then NOT NULL and UNIQUE combination would make that attribute to be a candidate key.

Index No. is a candidate key being Unique and Not Null.

```
CREATE TABLE STUDENT(
    ENROLL_NO   CHAR(05)  NOT NULL,
    INDEX_NO  CHAR(05)  NOT NULL UNIQUE,
    NIC    CHAR(10)  UNIQUE ,
    STU_NAME  VARCHAR (35) NOT NULL,
    STU_DOB  DATE  NOT NULL,
    STU_ADDRESS  VARCHAR (35),
    PRIMARY KEY (ENROLL_NO));
```

Activity

- Employee table is comprised of following attributes.
    - Empid, Salary, NIC, Name, Contact_No, Gender
- The above attributes contain below constraints.
    - Check whether the Salary is greater than 20000
    - Check whether the Gender is Male or Female
    - NIC is a candidate key
    - Empid is the Primary Key of the relation
    - Name and Contact_No cannot contain NULL values
- Write a SQL statement to create Employee table adhering to the above constraints.

Specifying Names to Constraints

- Name to a constraint is given to identify a constraint uniquely in the system. Therefore, the constraint name should be unique.
- When you want to remove a constraint from a relation, you can drop the constraint. However, giving a constraint name is optional.
- Following is the syntax for specifying a name to a constraint.

CONSTRAINT <constraint name> <constraint type>

Examples of Specifying Names to Constraints: CHECK

```
CREATE TABLE STUDENT (
    ENROLL_NO    CHAR(05)    NOTNULL,
    STU_NAME   VARCHAR(50)   NOT NULL,
    STU_DOB   DATE    NOT NULL,
    STU_ADDRESS  VARCHAR(35),
    GENDER    VARCHAR(06),
    PRIMARY KEY (ENROLL_NO),
    CONSTRAINT UG_Students
            CHECK  (GENDER in ('Male', 'Female'))
     );
```

Examples of Specifying Names to Constraints: PRIMARY KEY

```
CREATE TABLE STUDENT (
        STDID       CHAR(05),
        NAME        VARCHAR(20),
        ADDRESS     VARCHAR(35),
        DoB         DATE,
        CONSTRAINT PK_STDID PRIMARY KEY (STDID)
        );
```

Activity

- Using constraint names
  - Add default value to Reg_course as ‘Computer Science’.
  - Add check constraint to Age where Age is between 19 and 26.

  for the table UnderGrad_Student (**Sid**, Name, Address, Age, Reg_course)

Referential Integrity

- Tables in a database are normally not independent and there are links between tables.
- Referential constraints are introduced to maintain referential integrity of data that is linked across tables.
- A referential constraint is defined for a specific column (called a foreign key) when a table is defined.
- The table in which a referential constraint and a foreign key are defined is called the *referencing table*.
- The table that is referenced from a referencing table with a foreign key is called the *referenced table*.
- The primary key that is referenced by the foreign key must be pre-defined in the referenced table.

Referential Integrity

- Referential integrity allows the consistency of values across related tables. Referential integrity between the following two tables is defined when the foreign Key (stdid) of the StudentMarks table is created.

StudentMarks (Referencing Table)		
stdid	**course_id**	**grade**
53666	CS100	C
53667	IT101	B
53668	IS102	A
53669	CS103	B

Students (Referenced Table)		
student_id	**name**	**age**
53666	Amal	18
53667	Shiva	18
53668	Saman	19
53669	Fathima	20

- Referential integrity constraint verifies the values of the stdid column against the corresponding student_id column in the Students table. This ensures that anyone entered a stdid value into the StudentMarks table is an existing value in the Students table:

Referential Integrity

- In enforcing referential integrity constraint, the foreign key should be in the same domain as the Primary Key.
- Foreign key is not allowed to have NULL if it is part of the primary key of the referencing table as illustrated in the previous example. However, there can be situations where foreign key could contain NULL values.

Employee (Referencing Table)

<u>Empid</u>	**ename**	**address**	**did**
E1001	Amal Silva	Kandy	002
E1002	Shiva Kumar	Colombo	001
E1003	Fathima Siyam	Ampara	NULL

Department (Referenced Table)

<u>Dept_id</u>	**dname**
001	HR
002	Finance
003	Research
004	Marketing

Since Employee (did) is not a part of the PK, it could be allowed to have NULL

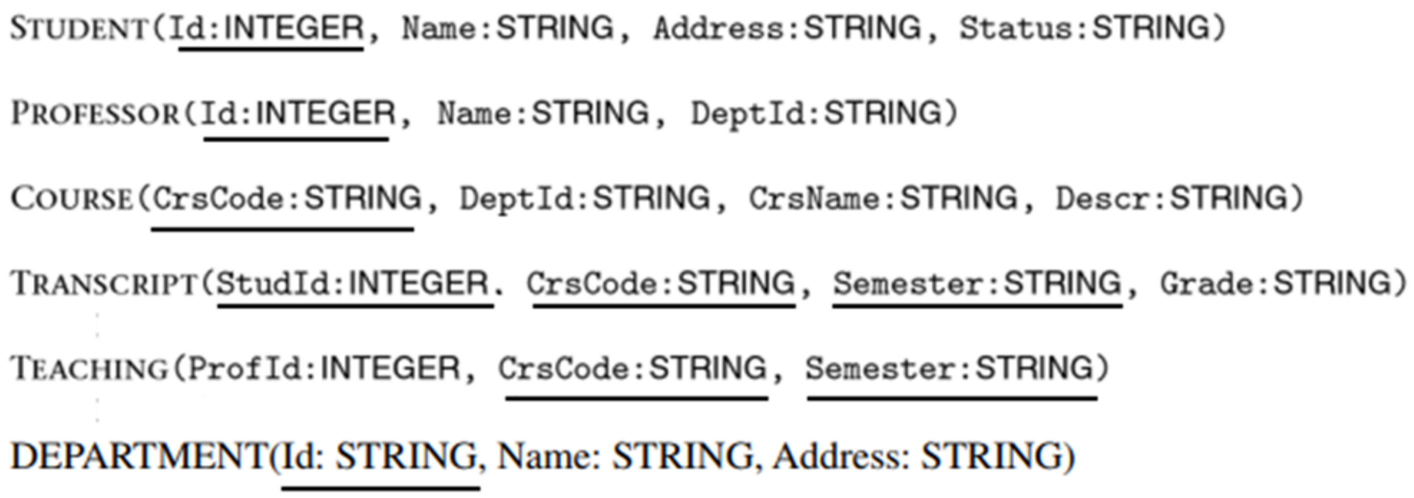
Referential Integrity

- Referential integrity is declared in the table definition using foreign key constraint as given below.

```
CREATE TABLE STUDENTMARKS (
    STDID          CHAR(05),
    COURSE_ID      CHAR(05),
    GRADE          CHAR(01),
    PRIMARY KEY(STDID, COURSE_ID),
    CONSTRAINT FK_GRADE
     FOREIGN KEY(STDID) REFERENCES STUDENTS(STUDENT_ID)
    CONSTRAINT FK_COURSE
     FOREIGN KEY(COURSE_ID) REFERENCES COURSE(COURSE_ID)
   );
```

Activity

- Identify which relations can be enforced with referential integrity. constraints.



STUDENT(Id:INTEGER, Name:STRING, Address:STRING, Status:STRING)

PROFESSOR(Id:INTEGER, Name:STRING, DeptId:STRING)

COURSE(CrsCode:STRING, DeptId:STRING, CrsName:STRING, Descr:STRING)

TRANSCRIPT(StudId:INTEGER, CrsCode:STRING, Semester:STRING, Grade:STRING)

TEACHING(ProfId:INTEGER, CrsCode:STRING, Semester:STRING)

DEPARTMENT(Id: STRING, Name: STRING, Address: STRING)

Relation	Yes/No
Student	
Professor	
Course	
Transcript	
Teaching	
Department	

Activity

- Following two tables illustrate rows that a user tries to enter. Identify what would happen when each of the tuple is inserted into the two tables given below. Assume that the course table already has the rows relevant to CS100, CS101 and CS103 courses.

Students(<u>student_id CHAR(05)</u>, name VARCHAR(50), age INT)

StudentMarks(<u>stdid CHAR(05),course_id CHAR(05),</u>grade CHAR)

Relation	Tuple	Violation
Students (primary key(student_id))	(53666, 'Amal', 18)	
	(53667, 'Anne', 'eighteen')	
	(53668, 'Saman', 19)	
StudentMarks (primary key (stdid, course_id))	(53666, 'CS100', 'C')	
	(53669, 'CS101', 'B')	
	(NULL, 'CS103', 'B')	

Referential Constraint Actions

- Referential constraint actions define alternate processing options for the referencing table in the event a referenced row is deleted, or referenced columns are updated when there are existing matching rows.
- Referential actions are specified as given below with an update operation (ON UPDATE) or a delete operation (ON DELETE), or both, in any sequence:

  CASCADE,
  SET NULL,
  SET DEFAUT,
  SET NULL or RESTRICT.

- The ON UPDATE and ON DELETE have the following syntax to enforce the referential actions:

ON UPDATE {CASCADE | SET NULL | RESTRICT | NO ACTION}

or

ON DELETE {CASCADE | SET NULL | RESTRICT | NO ACTION}

Referential Constraint Actions: ON UPDATE CASCADE

- A foreign key with **UPDATE CASCADE** means that if value of the primary key of the parent/referenced table is changed, the corresponding value of foreign key in the child/referencing table is also changed.

Foreign Key → stdid

StudentMarks (Referencing Table)

stdid	Course_id	grade
53666	CS100	C
53667	CS101	B
53668	IT102	A
~~53669~~ 53670	CS103	B
~~53669~~ 53670	CS100	A

Students (Referenced Table)

student_id	name	age
53666	Amal	18
53667	Shiva	18
53668	Saman	19
~~53669~~ 53670	Fathima	20

SQL code is executed to update 53669 to 53670 in referenced table

Reference Constraints are checked and If there is a FK whose value is 53669, it is updated to 53670 in referencing table.

Referential Constraint Actions: ON UPDATE CASCADE

UPDATE CASCADE is specified as given below

- In **StudentMarks** Table

```
CONSTRAINT Student_ID_FK
FOREIGN KEY (stdid)
REFERENCES Students (student_id)
                    ON UPDATE CASCADE ;


UPDATE Students SET student_id = 53670
WHERE student _id = 53669;
```

Updating a student_id will result in changing it in the StudentMarks table.

Referential Constraint Actions: ON DELETE CASCADE

- A foreign key with ON DELETE CASCADE means if value of primary key of the parent/referenced table is deleted, the corresponding value of foreign key in the child/referencing table is also deleted.

Foreign Key

StudentMarks (Referencing Table)

stdid	Course_id	grade
53666	100	C
53667	101	B
53668	102	A
~~53669~~	~~103~~	~~B~~

Students (Referenced Table)

student_id	name	age
53666	Amal	18
53667	Shiva	18
53668	Saman	19
~~53669~~	~~Fathima~~	~~20~~

Reference Constraints are checked and If there is a FK whose value is 53669, then it will be deleted too.

SQL code is executed to delete 53669

Referential Constraint Actions: ON DELETE CASCADE

When **DELETE CASCADE** is specified

- In **StudentMarks** Table

  CONSTRAINT Student_ID_FK

  FOREIGN KEY (stdid) REFERENCES Students (student_id)

  ON DELETE CASCADE;

  DELETE FROM Students

  WHERE student _id = 53669;

Deleting a student_id will result in deleting it in the StudentMarks table.

Referential Constraint Actions: NO ACTION/ RESTRICT

- NO ACTION/RESTRICT is the default behavior of returning an error in attempting to delete or update a row in the referenced table with matching rows in the referencing table.
- This action can also be explicitly defined as:

  ON DELETE RESTRICT (or ON DELETE NO ACTION)

  ON UPDATE RESTRICT (or ON UPDATE NO ACTION)

Referential Constraint Actions: ON DELETE RESTRICT

- The following example illustrates how NO ACTION/RESTRICT prevents deleting a row of the parent/referenced if there are rows with the matching foreign key in the child/referencing table.

StudentMarks (Referencing Table)

Foreign Key → stdid

stdid	Course_id	grade
53666	100	C
53667	101	B
53668	102	A
53669	103	B

Students (Referenced Table)

student_id	name	age
53666	Amal	18
53667	Shiva	18
53668	Saman	19
53669	Fathima	20

Reference Constraints are checked and If there is a FK whose value is 53669, then you can't perform the delete action

SQL code is executed to delete 53669

Referential Constraint Actions: ON DELETE RESTRICT

When **DELETE RESTRICT** is specified

- In **StudentMarks** Table

```
CONSTRAINT Student_ID_FK
FOREIGN KEY (stdid) REFERENCES Students (student_id) ON DELETE RESTRICT;


DELETE  FROM Students
WHERE student _id = 53669;
```

Deleting a student_id will not be performed.

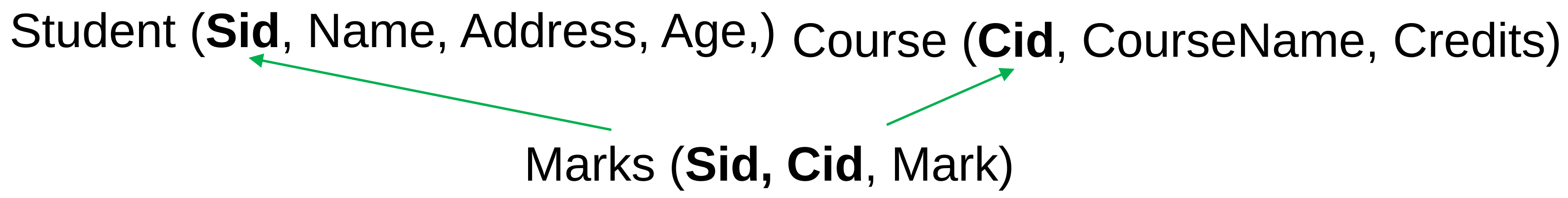
Referential Constraint Actions: SET DEFAULT/ SET NULL

- The other two actions are SET DEFAULT and SET NULL.
- With these actions, when you update or delete a value in the referenced table you can set a default value or null for the referencing value.

```
CONSTRAINT Student_ID_FK
FOREIGN KEY (stdid) REFERENCES Students (student_id)
ON DELETE SET NULL ON UPDATE SET DEFAULT
```

Referential Constraint Actions

- Given below is an example where referential actions for both delete and update operations are used together.



Student (**Sid**, Name, Address, Age,) Course (**Cid**, CourseName, Credits)

Marks (**Sid, Cid**, Mark)

```
CREATE TABLE Marks (
    Sid       CHAR(05),
    Cid       CHAR(05),
    Mark      CHAR(01),
    PRIMARY KEY (Sid, Cid),
    FOREIGN KEY Sid REFERENCES Student (Sid)
    ON DELETE SET NULL  ON UPDATE SET NULL,
    FOREIGN KEY Cid REFERENCES Course(Cid)
    ON UPDATE CASCADE ON DELETE RESTRICT);
```

Activity

Create tables for the given schema enforcing suitable referential integrity constraints and referential actions where necessary.

UnderGrad_Student (**Sid, Cid**, Name, Address, Age, Reg_course)

Course (**Cid**, CourseName, Credits)

Insert, Delete and Update Operations Dealing with Constraint Violations

- When the following data manipulation operations are performed, it is important to maintain the integrity of the data.

  - **Insert** : insert one or more rows to a relation
  - **Delete** : delete an existing row in a table
  - **Update** : modify values of an existing row

- That is, performing these operations should not violate constraints of a relation.

Insert Operations Dealing with Constraint Violations

- Insert operation adds values with new tuples to a table. When we perform this operation, there are four types of constraints that could get violated. They are:
  - **Entity integrity violation** – Insert a row with NULL for the primary key
  - **Key constraint violation** – Insert a new tuple with an existing primary key
  - **Referential integrity violation** – Insert a new tuple but the referenced tuple does not exist
  - **Domain constraint violation** – Insert a tuple with inappropriate data type
- If an insert violates a constraint then by default it rejects the insertion.

Delete Operations Dealing with Constraint Violations

- Performing a delete operation violates referential integrity. If you delete the primary key of a relation that is referenced by another tuple, then it violates referential integrity.
- How can you prevent this violation?
    - Restrict – Reject performing delete operation
    - Cascade – delete tuples that refer the primary key
    - Set NULL or Set DEFAULT – when you delete a primary key, set NULL or DEFAULT value to the referencing tuples.

Update Operations Dealing with Constraint Violations

- Update operations change values of attributes. Update does not have any impact on attributes which are not primary key or foreign key. Also, when updating a value it could violate the domain if the data type of the value is inappropriate. Hence the possible violations are:
  - *Primary key violation* – Update the primary key value of a tuple by giving an existing value
  - *Referential integrity violation* – Update the primary key value of a tuple without updating that value in a referencing table. Or, Update foreign key value without checking whether that value exists in the referenced table.

Activity

- Match the correct answers.

Entity integrity violation	by default it rejects the insertion
If an insert violates a constraint	would prevent referential integrity violation
Set NULL or Set DEFAULT	can violate integrity of the data
Performing modification operations	May insert a row with NULL for the primary key

Activity

- StudentMarks (Primary key (stdid, course_id)) and Student (primary key (stdid)) tables have 3 tuples for each. You want to perform operations given in the bottom table. Identify whether you can perform the following operations for the given relations.

StudentMarks		
<u>stdid</u>	**<u>Course_id</u>**	**grade**
53666	CS100	C
53667	CS101	B
53668	CS103	B

Students		
<u>Student_id</u>	**name**	**age**
53666	Amal	18
53667	Anne	18
53668	Saman	19

Operation	Action
Insert <53669, CS104, 'A'> into StudentMarks *Assume that the course table already has the rows relevant to CS104.*	
Insert <53669, 'Nimal', 21> into Students	
Insert <NULL, 'Sunil', 19> into Students	
Update Students Set age = 'Twenty' Where Student_id=53666	

Drop Table

- Drop table is deleting tables. Two types of actions can be taken when dropping tables.
  - RESTRICT – if there is a constraint (FK / View) then do not drop the table
  - CASCADE - drop all the other constraints & views that refer the table

DROP TABLE Employee [RESTRICT|CASCADE]

ADD or REMOVE Constraints

- Drop the primary key constraint of a table

  - ALTER TABLE Employees

  DROP CONSTRAINT PK_Employees;

  Or for some DBMS the following statement is possible

  - ALTER TABLE Student Drop PRIMARY KEY;

- Drop a unique, foreign key, or check constraint

  - ALTER TABLE Employee Drop Constraint FK_EmpDept;

- Add a new constraint

  - ALTER TABLE PassStudents ADD Constraint avg_Marks CHECK (marks >= 50);

Other Types of Constraints

- Previous constraints discussed in this lesson were included in the Data Definition Language (DDL). The other types of constraints are semantic integrity constraints.
- If you need to set a constraint that The salary of an employee should not exceed the salary of the employee’s supervisor, then it cannot be directly done by DDL. You need to do it through application program.
- For this purpose, we use triggers and assertions in SQL. In the later part of the lesson, we discuss how to create triggers.

Constraints in Databases as Assertions

- An assertion is a condition that allows entering valid values to the tables. It gets enforced on any number of other tables, or any number of other rows in the same table.
- Assertions differ from CHECK constraint since Assertions are defined on schema level. CHECK constraints relate to one single row only and hence are defined on a column or a table level.
- Enforcing assertions are complex in nature. Therefore, most of the DBMSs do not support it.

```
CREATE ASSERTION <Constraint Name>
                        CHECK <Search Condition>
```

Example

Employee (eid, ename, address, Salary, Ssn, Dno)

Department (Dnumber, dname, Mgr_ssn)

Example, you want to add a constraint where "*No employee should have a salary greater than their manager*". Therefore:

```
CREATE ASSERTION SALARY_CONSTRAINT
    CHECK ( NOT EXISTS (
        SELECT *
        FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
        WHERE E.Salary>M.Salary AND E.Dno = D.Dnumber
                                AND D.Mgr_ssn = M.Ssn ) );
```

Constraint Name

Conditions

Activity

- Write an Assertion to the following.
  - No student can be graduated if his/her average Mark for all courses is less than 30.

    Marks (**Sid, Cid**, Mark)

Activity

- Write an Assertion to ensure that every mortgage customer who has a mortgage should keep a minimum of Rs. 500 in their bank account.

Bank_Account (**<u>Account_no</u>**, BranchID, OwnerName, OwnerNIC, OwnerAddress, mortgageID, balance)

Loan_Details(**<u>mortgageID</u>**,amount,interest_rate, NoOfYears)

Specifying Actions in Databases as Triggers

- Introduction to Triggers and Create Trigger Statement
- Active Databases and Triggers

Active Databases

- Active database is where a database can perform actions based on events and conditions that take place in the system.
- ECA model or event-condition-action model is used to state active database rules.
- Active databases perform active rules to notify conditions. E.g. indicate when pressure level of a pipe is exceeding the danger level.
- It can also to apply integrity constraints to evaluate business rules.
- Active rules maintain derived data when values of tuples change.
- TRIGGERS are used to define automatic actions that need to be executed when events and conditions happen.

Introduction to Triggers

- A trigger is a statement that the system executes automatically as a side effect of a modification to the database.
- A trigger has an event that causes the trigger to be checked and a condition that must be satisfied for trigger execution to proceed.
- A trigger has an action that has to be taken when the trigger executes.
- The following example illustrates the reason for using Triggers
    - A production organisation maintains a warehouse. It maintains a minimum inventory level. When the inventory level goes down it is possible for a trigger to alert the manager with respect to the reorder level.

Design A Trigger

- To design a trigger, you require to identify;
  - When should the trigger be executed?
    - **Event** – a cause to check the trigger
    - **Condition** – logic to be satisfied to execute the trigger
    - **Action** - The action that should be taken when

      the trigger is executed
- Model of a trigger also known as event- condition - action model

Event-Condition-Action model

- Event-Condition-Action model comprises of
  - on Event
    - Event occurs in database

      E.g. insertion of new row, deletion of row
  - if Condition
    - Condition is checked

      E.g. is salary < 10000 ? Has student passed?
  - then Action
    - Actions are executed if condition is satisfied

      E.g. give a salary increment, congratulate student

Types of Triggers

- Row Triggers
  - Trigger is fired once for each row in a transaction.
  - If an update statement modifies multiple tuples of a table, a row trigger is fired once for each tuple which are affected by the update query. If there is no tuple affected by the query, then the row trigger is not executed.
  - Have access to *:new* *(new values)* and *:old* (old values).

    E.g. update the total salary of a department when the salary value of an employee tuple is changed.

Types of Triggers

- Statement Triggers
  - Trigger is fired once for each transaction.
  - If a DELETE statement deletes several rows from a table, a statement-level DELETE trigger is fired only once, regardless of how many rows are deleted from the table.
  - Does not have access to *:new* and *:old* values.

  E.g. Delete a row relevant to an employee who has completed the contract period.

Tigger Timing

- Before Trigger
    - Runs before any change is made to the database.

      E.g. Before withdrawing money account balance is required to be checked.
- After Trigger
    - Runs after changes are made to the database

      E.g. After withdrawing money update the account balance
- Instead of
    - INSTEAD OF triggers are used to modify views that cannot be modified directly through UPDATE, INSERT, and DELETE statements.

Syntax for Specifying a Trigger

```
CREATE [OR REPLACE ] TRIGGER trigger_name
{BEFORE | AFTER | INSTEAD OF }
{INSERT [OR] | UPDATE [OR] | DELETE}
[OF col_name]
ON table_name
[REFERENCING OLD AS o NEW AS n]
[FOR EACH ROW | STATEMENT ]
WHEN (condition)
DECLARE
   Declaration-statements
BEGIN
   Executable-statements
EXCEPTION
   Exception-handling-statements
END;
```

Trigger Time

Trigger Event

Type of Trigger

Trigger Action

Example

When inserting values for the Employee table ensure whether the salary is >= 20000.

Employee(Empid, Ename, Salary, Dno).

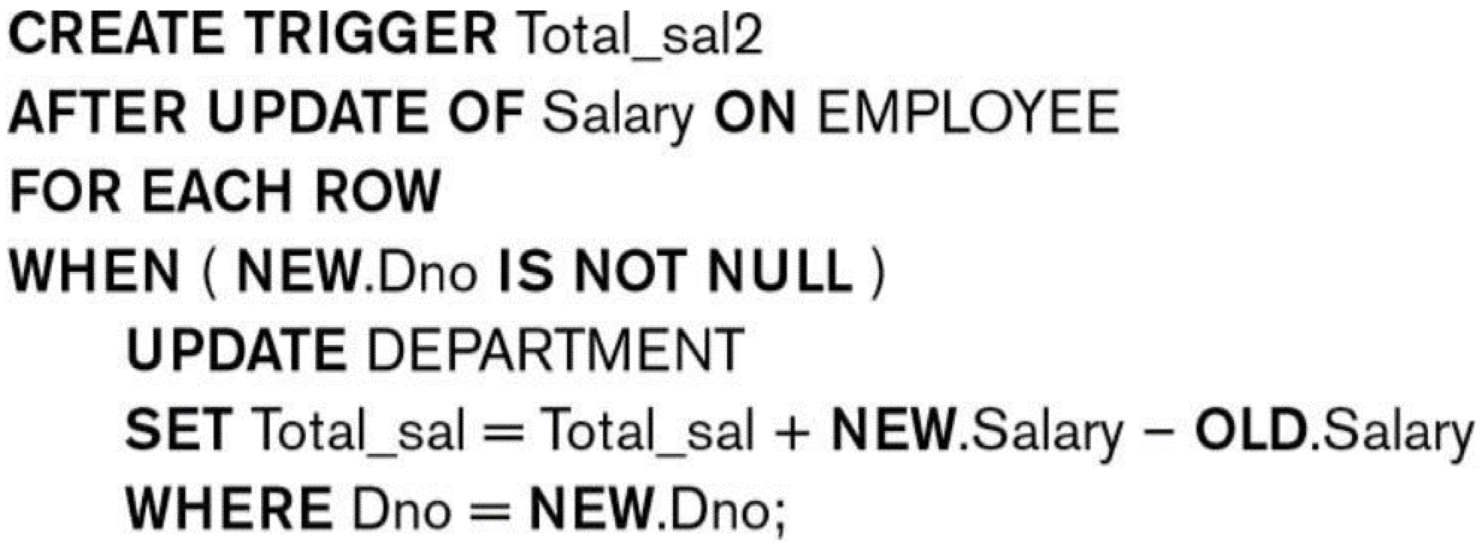
```
CREATE TRIGGER check_salary
     BEFORE INSERT ON Employee
     FOR EACH ROW
          BEGIN
               IF (:new.Salary < 20000)
          THEN
               PRINT 'Wrong Salary';
          END IF;
 END;
```

OLD and NEW references

- OLD references old values for DML statements. NEW references new values for DML operations. Delete, Update, Insert are DML operations.
- OLD and NEW references are not available for table-level triggers, and it can be used for record-level triggers.
- Table-level triggers cannot fire a trigger for each row of a table. Record-level or row-level triggers can fire a trigger for each row of a table.

Example of OLD and NEW references

- Following trigger is an example of updating total salary (delete old salary and add new salary of employees) of a particular department



```
CREATE TRIGGER Total_sal2
AFTER UPDATE OF Salary ON EMPLOYEE
FOR EACH ROW
WHEN ( NEW.Dno IS NOT NULL )
        UPDATE DEPARTMENT
        SET Total_sal = Total_sal + NEW.Salary – OLD.Salary
        WHERE Dno = NEW.Dno;
```

(Elmasri and Navathe, 2015)

Activity

```
CREATE TRIGGER derive_commission_trg
BEFORE UPDATE OF salary ON Employee
FOR EACH ROW
WHEN (new.job = 'Salesman')
        BEGIN
        :new.comm := :old.comm * (:new.salary/:old.salary);
        END;
```

In the above query:

- What is the event?
- What is the Condition?
- What is the action?

Activity

- Create a trigger that accepts insertion into the student table and checks the GPA. If the GPA of the inserted student is greater than 3.3, or less than or equal to 3.6, that student will be automatically applying for Computer Science stream. Otherwise, the student has to apply for Software Engineering stream (Stream is Computer Science, Biology etc; enrollment is number of students enrolled for the college, decision is yes or no in getting selected for a stream)

  Student(sID, sName, GPA)

  Apply(sID, cName, stream, decision)

  College(cName, state, enrollment)

Activity

Create a trigger that simulates the behavior of cascaded delete when there is a referential integrity constraint from the student ID in the Apply table, to the student ID in the student table. Make it activate on delete cascade when there is a deletion from the Student table.

College(<u>cName</u>, state, enrollment);
Student(<u>sID</u>, sName, GPA);
Apply(<u>sID, cName</u>, major, decision);

Summary

Relational Model Constraints	• Categories of Constraints • Domain Constraints
Specifying Constraints in SQL	• Specifying Key and Referential Integrity Constraints • Specifying Constraints on Tuples Using CHECK • Specifying Names to Constraints
Constraints in Databases as Assertions	• Introduction to Assertions
Specifying Actions in Databases as Triggers	• Introduction to Triggers and Create Trigger Statement • Active Databases and Triggers