Managing Database Constraints and Triggers

OBJECTIVES

- **To be able to create and use SQL constraints**
- **To understand how referential integrity actions are implemented in SQL statements**
- **To understand how to create and use SQL triggers**

Managing Database Constraints

- Constraints can be defined within the **CREATE TABLE** statement
- Constraints can be added to the table after it is created using the **ALTER TABLE** statement.
- Five types of constraints:
  - PRIMARY KEY may not have null values
  - NULL/NOT NULL
  - UNIQUE may have null values
  - CHECK
  - FOREIGN KEY

Managing Database Constraints

- Primary Key Constraint
  - May be defined at the column level if one column

```
CREATE TABLE Persons (
    ID int PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int
);
```

  - Must be defined at the table level if more than one column

```
CREATE TABLE Persons (
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Phone char(10) NOT NULL,
    Age int,
    PRIMARY KEY (Lastname, Phone)
);
```

Managing Database Constraints

- Primary Key Constraint
  - Primary Key constraint implicitly includes NOT NULL and UNIQUE

- In order to make a **constraint modifiable**, you must give it a name

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT PK_Person PRIMARY KEY (ID,LastName)
);
```

Managing Database Constraints

- Then you can **modify the constraint**:

```
ALTER TABLE Persons
    DROP CONSTRAINT PK_Person;


ALTER TABLE Persons
    ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
```

Managing Database Constraints

- Unique Constraint
  - May be defined at the column level if one column

```
CREATE TABLE Persons (
    ID int PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Email varchar(255) UNIQUE,
    Age int);
```

  - Must be defined at the table level if more than one column

```
CREATE TABLE Persons (
    ID int PRIMARY KEY,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Phone char(10) NOT NULL,
    Age int,
    UNIQUE (Lastname, FirstName));
```

Managing Database Constraints

- Unique Constraint
- Give it a name

```
CREATE TABLE Persons
   (ID INT NOT NULL,
    LastName  VARCHAR(255) NOT NULL,
    FirstName VARCHAR(255) NOT NULL,
    Age       INT,
    CONSTRAINT PK_ID PRIMARY KEY (ID),
    CONSTRAINT UN_Name UNIQUE KEY (LastName,FirstName)
   );
```

Managing Database Constraints

- Check Constraint – applies a condition to a column
- Rules for conditions same as WHERE clause

```
CREATE TABLE Persons
   (ID        INT NOT NULL,
    LastName  VARCHAR(255) NOT NULL,
    FirstName VARCHAR(255) NOT NULL,
    Age       INT,
    CONSTRAINT CK_Age CHECK (Age > 18)
    );
```

Managing Database Constraints

```
CREATE TABLE t1
(CHECK (c1 <> c2),
c1 INT CHECK (c1 > 10),
c2 INT CHECK (c2 > 0),
c3 INT CHECK (c3 < 100),
CONSTRAINT c1_nonzero CHECK (c1<>0),CHECK (c1> c3)
);
```

Managing Database Constraints

- Foreign Key Constraint

```
CREATE TABLE t1
(c1_id int not null primary key,
c2 varchar(255) not null);

CREATE TABLE t2
(c3 int not null,
c1_id int not null primary key,
CONSTRAINT fk_id
FOREIGN KEY (c1_id) REFERENCES t1 (c1_id));
```

- **Can be added later**

```
ALTER TABLE table_name
ADD CONSTRAINT FK_name
FOREIGN KEY (columns)
REFERENCES parent_table(columns);
```

Managing Database Constraints

- Can be added later

```
DROP TABLES t1, t2;

        CREATE TABLE t1
        ( c1_id int not null primary key,
          c2 varchar(255) not null) ;

        CREATE TABLE t2
        (c3 int not null
          c1_id int not null primary key,
          CONSTRAINT FK_id FOREIGN KEY (c1_id) REFERENCES t1 (c1_id)) ;

        -- drop FK
        ALTER TABLE t2
        DROP FOREIGN KEY fk_id ;

        -- add FK
        ALTER TABLE t2
        ADD CONSTRAINT fk_id FOREIGN KEY (c1_id) REFERENCES t1 (c1_id);
```

Managing Database Constraints

- Foreign Key Constraints:
  Maintaining Referential Integrity

- Prevents inserting a row into a child table where the parent key value is missing

- If an UPDATE or DELETE is done on the parent table
  - What to do to the child row?
    - SET NULL – sets the child value NULL
    - SET DEFAULT – sets the child value to the column default
    - CASCADE – Deletes/Updates the child
    - NO ACTION / RESTRICT – prevents action on parent

Managing Database Constraints

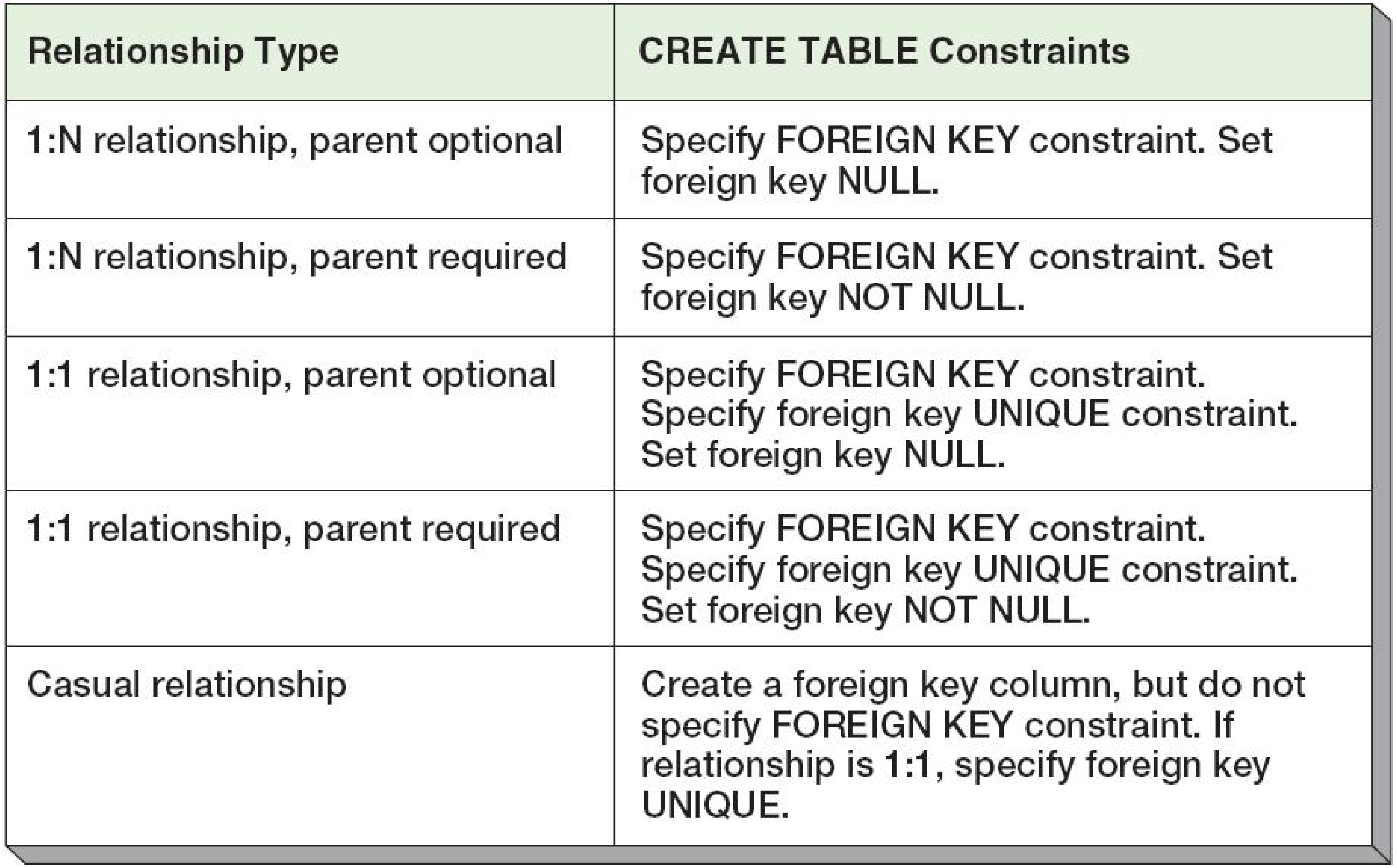
- Foreign Key Constraint
  - Maintaining RI

```
CREATE OR REPLACE TABLE Persons
  (ID                  INT PRIMARY KEY,
   LastName            VARCHAR(255) NOT NULL,
   FirstName           VARCHAR(255) NOT NULL,
   DepartmentNumber INT,
   Age                 INT,
   CONSTRAINT FK_Department (DepartmentNumber)
         REFERENCES Department (DeptID)
               ON DELETE SET NULL
               ON UPDATE CASCADE
  );
```

Managing Database Constraints

Can Foreign Keys Be NULL?

- Depends on the Business Rules depicted in your data model



Relationship Type	CREATE TABLE Constraints
1:N relationship, parent optional	Specify FOREIGN KEY constraint. Set foreign key NULL.
1:N relationship, parent required	Specify FOREIGN KEY constraint. Set foreign key NOT NULL.
1:1 relationship, parent optional	Specify FOREIGN KEY constraint. Specify foreign key UNIQUE constraint. Set foreign key NULL.
1:1 relationship, parent required	Specify FOREIGN KEY constraint. Specify foreign key UNIQUE constraint. Set foreign key NOT NULL.
Casual relationship	Create a foreign key column, but do not specify FOREIGN KEY constraint. If relationship is 1:1, specify foreign key UNIQUE.

Managing Database Triggers

- **Trigger**
  - A piece of code
  - Associated with a TABLE
  - Associated with an EVENT
  - The CODE fires when the EVENT happens

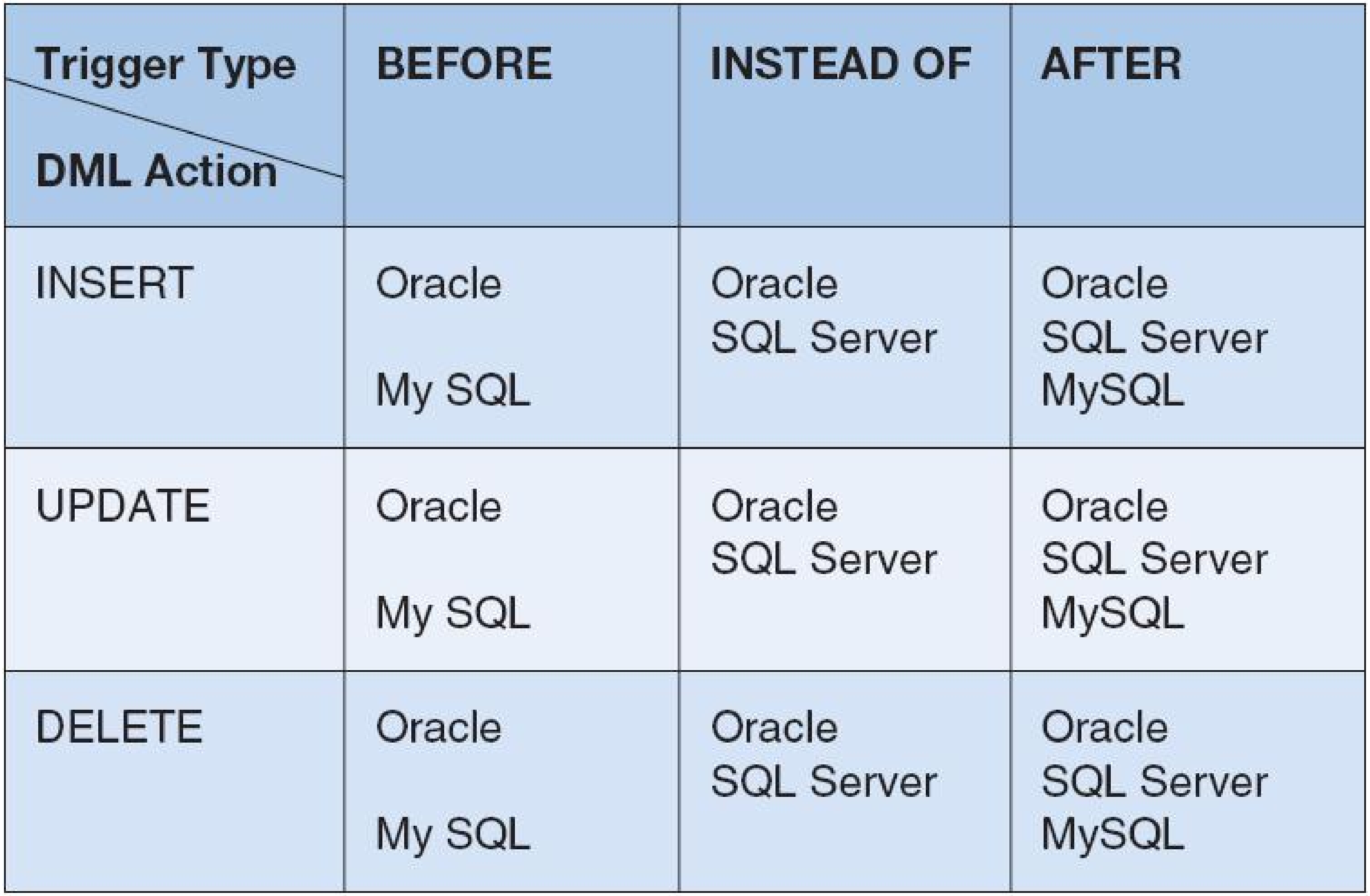
- **Why use triggers?**

  - Enforces business rules
  - Moves code from an application program to the database
  - Performance improvement – all work done on server

Managing Database Triggers

- Trigger event
  - Insert, Update, Delete
- Trigger timer
  - Before, After, (Instead Of)

Managing Database Triggers



Trigger Type / DML Action	BEFORE	INSTEAD OF	AFTER
INSERT	Oracle My SQL	Oracle SQL Server	Oracle SQL Server MySQL
UPDATE	Oracle My SQL	Oracle SQL Server	Oracle SQL Server MySQL
DELETE	Oracle My SQL	Oracle SQL Server	Oracle SQL Server MySQL

Managing Database Triggers

- Special Features

  When a trigger is fired, the DBMS supplies:

  – OLD and NEW values for the update

  – NEW values for inserts

  – OLD values for deletions

Allows you to reference either the OLD or NEW value of a column within the code of the trigger.

Managing Database Triggers

- Example

  Archival of data before a delete.

  Certain records are deleted from the nwOrders table when the OrderDate becomes aged.

  For this example, we will delete orders with an order date before 2013-08-01.

  Note that when I DROP and CREATE nworders, the trigger is also dropped.

Managing Database Triggers

CREATE TRIGGER statement:

```
CREATE TRIGGER trigger_name
          trigger_time  trigger_event
ON table_name
FOR EACH ROW

BEGIN
………
……….
END;
```

Managing Database Triggers

- The nwOrdersArchive table:

```
DROP TABLE IF EXISTS 'nwordersarchive';
CREATE TABLE 'nwOrdersArchive' (
  'OrderID'     int(11) NOT NULL,
  'CustomerID'  varchar(5) DEFAULT NULL,
  'EmployeeID'  int(11) DEFAULT NULL,
  'OrderDate'   date DEFAULT NULL,
  'ArchiveDate' date DEFAULT NULL,
  PRIMARY KEY ('OrderID')
) CHARACTER SET utf8 COLLATE utf8_general_ci;
```

Managing Database Triggers

- The Trigger

```
DELIMITER $$
CREATE TRIGGER before_order_delete
    BEFORE DELETE
ON nwOrders
    FOR EACH ROW
BEGIN
    INSERT INTO nwOrdersArchive
        SET OrderID = OLD.OrderID,
            CustomerID = OLD.CustomerID,
            EmployeeID = OLD.EmployeeID,
            OrderDate = OLD.OrderDate,
            ArchiveDate = NOW();
END$$
DELIMITER ;
```

Managing Database Triggers

- The Delete query

```
DELETE FROM nworders
    WHERE orderdate < '2013-08-01';
```

Managing Database Triggers

Example:

```
USE northwinds;

CREATE TABLE Original_T1
(
	Column_C1 INT,
	Column_C2 VARCHAR(15),
	Column_C3 DATE
);

INSERT INTO Original_T1
VALUES (1, 'A', '2019/01/01'),
	(2, 'B', '2019/01/02')
;

select * from Original_T1;
```

Managing Database Triggers

```
CREATE TABLE Trigger_T1
(
      Column_C1 INT,
      Column_C2 VARCHAR(15),
      Column_C3 DATE
);

-- create trigger for archiving old data

CREATE TRIGGER Before_Trigger
            BEFORE UPDATE ON Original_T1
      FOR EACH ROW
            INSERT INTO Trigger_T1
            SET
            Column_C1 = OLD.Column_C1,
            Column_C2 = OLD.Column_C2,
            Column_C3 = curdate()
;
```

Managing Database Triggers

```
SHOW TRIGGERS;

SELECT * FROM Original_T1;

SELECT * FROM Trigger_T1;

UPDATE Original_T1
       SET Column_C2 = 'X'
       WHERE Column_C1 = 1;

SELECT * FROM Original_T1;

SELECT * FROM Trigger_T1;

DROP TRIGGER Before_Trigger;

DROP TABLE Original_T1, Trigger_T1;
```