

Triggers

Course: INFO6210 Data Management and Database Design

Week: 11

Instructor: Mutsalklisana

Trigger Procedure

- Triggers are similar to stored procedures in that they are self-contained units of SQL code.
 - You don't explicitly call a trigger, as you do with a stored procedure
 - Automatically trigger invokes when a predefined event occurs
- Triggers exist in most DBMS to manage and monitor tables during insert, update or delete

Trigger Procedure

- Purpose of Triggers:
 - Log changes on records
 - Validation
 - Create backup or duplicate information (data/object)

Trigger Procedure

- **CREATE TRIGGER Syntax:**

CREATE TRIGGER <trigger name>

{BEFORE | AFTER}

{INSERT | UPDATE | DELETE}

ON <table name>

FOR EACH ROW

<Triggered SQL statement>

Trigger Name

Time

Event

Trigger Procedure

- How many times should the trigger body execute when the triggering event takes place?
 - **Per Statement**: the trigger body executes once for the triggering event (This is the default)
 - **For each row**: the trigger body executes once for each row affected by the triggering event
- When the trigger can be fired?
 - Relative to the execution of an SQL statement (BEFORE or AFTER or instead of it)
 - Exactly in a situation depending on specific system resources (e.g., signal from system clock)

Trigger Procedure

- Most RDMSs generally support three types of triggers:
 - INSERT
 - UPDATE
 - DELETE
- Regardless of the type of trigger, define it on a specific table, and when a related event occurs, the trigger is fired

Trigger Event

- INSERT:
 - If define an INSERT trigger on a table, the trigger is fired when data is inserted in the table
- UPDATE:
 - If define an UPDATE trigger on a table, the trigger is fired when the table is updated
- DELETE:
 - If define an DELETE trigger on a table, the trigger is fired when data is deleted from the table

Trigger Example:

- Suppose that you are setting up a database to support a CD retail business
 - The DB includes two tables: the CDs table, which includes a list of CDs currently sold by the business, and the CDSPast table, which includes those CDs that have been sold in the past but are no longer carried
- Trigger can be created so that whenever a CD is deleted from the CDs table, it is automatically added to the CDSPast table
 - When the trigger is created, assign it to the CDs table
 - Trigger includes an INSERT statement that adds the deleted CD to the CDSPast table when a DELETE event occurs on the CDs table

Trigger - Delimiter

- Creating a New Delimiter
 - We can specify a new delimiter instead of semi-colon (;)
 - We will most likely need a new Delimiter in creating triggers
 - For this class, we will use // as the new delimiter
 - Syntax: **DELIMITER** <New Delimiter>

Trigger - Delimiter

- Example of Creating a New Delimiter

Cmd> **DELIMITER //**

Cmd> **DELIMITER <**

Cmd> **DELIMITER >>**

Cmd> **DELIMITER <<**

- Again, for this class, we will use `//` as the new delimiter

Trigger - Delimiter

- Example of Delimiter Structure

```
DELIMITER //  
CREATE TRIGGER tblEnrollment_bi  
  BEFORE INSERT ON tblEnrollment  
  FOR EACH ROW  
  BEGIN  
    SET @new = NEW.IDNo;  
    SET @old = OLD.IDNo;  
  END; //
```

Trigger - Variables

- Variables are placeholders that holds a certain data
- Variable Name Syntax:
 - **@<variable name>**
- Example of variables
 - @X
 - @Test
 - From the previous example,
 - @new
 - @old

Variable Assignment Statement

- `<variables> = <value>`
- Example of variable assignment
 - `@X = 5;` # @X gets the value of 5
 - `@Test = 10;` # @Test gets the value of 10
 - `@Y = @X;` # @Y gets the value of @X
 - `@Z = @Y + 7;` # @Z gets the value of @Y + 7
 - `@X = @Y + 7;` # @X gets the value of @Y + 7
- What is X?

Assignment Format

- Left variable always gets the right value

➤ @X = 5



Correct Assignment

➤ 5 = @X



Wrong Assignment Format

New and Old Columns Update

- OLD.<Column Name>
 - Value of the column before it was updated
- NEW.<Column Name>
 - Value of the column after it was updated

Example – Cont'd

```
DELIMITER //  
CREATE TRIGGER tblEnrollment_bi  
BEFORE INSERT ON tblEnrollment  
FOR EACH ROW  
BEGIN  
    SET @new = NEW.IDNo;  
    SET @old = OLD.IDNo;  
END; //
```

- @new will get the value of column IDNo after it was updated
- @old will get the value of column IDNo before it was updated

Sample Codes for Practice

- The following codes you can try and see what happens:

```
(Cmd)> Select * from @old, @new; //
```

```
(Cmd)> UPDATE tblEnrollment
```

```
> SET IDNo = 55
```

```
> WHERE EnrollmentNo = 1; //
```

```
(Cmd)> SELECT * from @old, @new; //
```

Practice Exercise

Create a **trigger** that logs the following information to a table `tblEnrollment_LOG` every time a record is updated in `tblEnrollment`

- User who modified the record
- Date when the record was modified
- The value of `IDNo` before the record was updated
- The value of `IDNo` after the record was updated

* `tblEnrollment_LOG(User, DateModified, Old_IDNo, New_IDNo)`

Practice Exercise - Solution

Once done, it should look like this:

```
CREATE TRIGGER tr_tblEnrollment
AFTER UPDATE ON tblEnrollment
FOR EACH ROW
BEGIN
    INSERT INTO tblEnrollment_LOG
    VALUE (user(), sysdate(), OLD.IDNo, NEW.IDNo);
END; //
```

TRIGGER – DROP Statement

Syntax:

DROP TRIGGER trigger_name [,...n]

Applications for Database Triggers

- **Auditing Table Operations**
 - Each time a table is accessed auditing information is recorded against it
- **Tracking Record Value Changes**
 - Each time a record value is changed the previous value is recorded
- **Protecting Database Referential Integrity**
(If foreign key points to changing records)
 - Referential integrity must be maintained

Applications for Database Triggers

- **Maintenance of Semantic Integrity**
 - E.g., when the factory is closed, all employees should become unemployed
- **Storing Derive Data**
 - E.g., the number of items in the trolley should correspond to the current session selection
- **Security Access Control**
 - E.g., checking user privileges when accessing sensitive information

Ex: Counting Statement Execution

```
SQL>CREATE OR REPLACE TRIGGER audit_emp
 2 AFTER DELETE ON emp
 3 FOR EACH ROW
 4 BEGIN
 5     UPDATE audit_table SET del = del + 1
 6     WHERE user_name = USER
 7     AND table_name = 'EMP' ;
 7 END ;
 8 /
```

Whenever an employee record is deleted from database, counter is an audit table registering the number of deleted rows for current user in system variable USER is incremented

Views

Course: INFO6210 Data Management and Database Design

Week: 11

Instructor: Mutsalklisana

SQL Views

- A view is a virtual table based on the result of SQL statement, allow users to do the followings:
 - Structure data in way that users or classes of users find natural or intuitive
 - Restrict access to the data such that a user can see and (sometimes) modify exactly what they need and no more
 - Summarize data from various tables which can be used to generate reports

SQL Views

- A view is a virtual table based on the result of SQL statement
- A view contains rows and columns, just like a real table
- Fields in a view are fields from one or more real tables in the database
- Function such as WHERE & JOIN statements can be added a view statement and present the data as if the data were coming from one single table

SQL Views

- CREATE VIEW Syntax:

```
CREATE VIEW view_name AS  
SELECT column_name(s)  
FROM table_name  
WHERE condition
```

- Note: A view always shows up-to-date data! The database engine recreates the data, using the view's SQL statement, every time a user queries a view.

Benefits of VIEW

A view provides several benefits:

- Views can hide complexity
 - If you have a query that requires joining several tables, or has complex logic or calculation, you can code all that logic into a view, then select from the view just like you would a table
- Views can be used as a security mechanism
 - A view can select certain columns and/or rows from a table, and permission set on the view instead of the underlying tables. This allow surfacing only the data that a user needs to see

Benefits of VIEW- Cont'd

- Views can simplify supporting legacy code
 - If you need to refactor a table that would break a lot of code, you can replace the table with a view of the same name
 - The view provides the exact same schema as the original table, while the actual schema has changed
 - This keep the legacy code that references the table from breaking, allowing you to change the legacy code at your leisure



Other Benefits of VIEW

- Add'l benefits:
 - Reducing redundancy in writing queries
 - Establishing a standard for relating entities
 - Providing opportunities to evaluate and maximize performance for complex calculations and joins (e.g., indexing on Schema bound views in MSSQL)
 - Making data more accessible and intuitive to team members and non-developers

SQL CREATE VIEW Examples

- The View “Current Product List” lists *all active products (products that are not discontinued)* from the “Products” table

```
CREATE VIEW [Current Product List] AS  
SELECT ProductID, ProductName  
FROM Products  
WHERE Discontinued=No
```

- View above can be query as follows:

```
SELECT * FROM [Current Product List]
```

CREATE VIEW Examples

- The View “Products Above Average Price” lists all products with a unit price higher than the average unit price from the “Products” table

```
CREATE VIEW [Products Above Average Price] AS  
SELECT ProductName, UnitPrice  
FROM Products  
WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Products)
```

- View above can be query as follows:

```
SELECT * FROM [Products Above Average Price]
```


CREATE VIEW Examples

- The View “Category Sales For 1997” calculates the total sale for each category in 1997 (Note: this view also selects its data from another view called “Product Sales for 1997”)

```
CREATE VIEW [Category Sales For 1997] AS  
SELECT DISTINCT CategoryName, Sum(ProductSales) AS CategorySales  
FROM [Product Sales for 1997]  
GROUP BY CategoryName
```

- View above can be query as follows:

```
SELECT * FROM [Category Sales For 1997]
```

- A Condition can be added to the query:

```
SELECT * FROM [Category Sales For 1997]  
WHERE CategoryName='Beverages'
```

CREATE VIEW Examples

- Consider the CUSTOMERS table with records as follows:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

CREATE VIEW Examples

- Following SQL codes create a view from CUSTOMERS table to display only customer name and age:

```
SQL> CREATE VIEW CUSTOMERS_VIEW AS  
      SELECT name, age  
      FROM CUSTOMERS;
```

CREATE VIEW Examples

- Now, you can query CUSTOMERS_VIEW in similar way as you would query an actual table

```
SQL> SELECT * FROM CUSTOMERS_VIEW;
```

- The result is as follows:

name	age
Ramesh	32
Khilan	25
kaushik	23
Chaitali	25
Hardik	27
Komal	22
Muffy	24

VIEW WITH CHECK OPTION

- The WITH CHECK OPTION is a option to ensure that all UPDATE and INSERTs satisfy the condition(s) in the view condition; if not satisfy, the UPDATE or INSERT returns an error

```
CREATE VIEW CUSTOMERS_VIEW AS  
SELECT name, age  
FROM CUSTOMERS  
WHERE age IS NOT NULL  
WITH CHECK OPTION;
```

- WITH CHECK OPTION denies the entry of any NULL value in the view's age column

Updating a VIEW in SQL

- View can be updated by using the following syntax:

SQL CREATE OR REPLACE VIEW Syntax

```
CREATE OR REPLACE VIEW view_name AS  
SELECT column_name(s)  
FROM table_name  
WHERE condition
```

Limitation on Updating a VIEW

- SELECT clause may not contain the followings:
 - keyword DISTINCT, summary functions, set functions, set operator, an ORDER BY clause
- FROM clause may not contain multiple tables
- WHERE clause may not contain subqueries
- Query may not contain GROUP BY or HAVING
- Calculated columns may not be updated
- All NOT NULL columns from base table must be included in view in order for the INSERT query to function

Updating a VIEW in SQL Example

- From previous example of “Current Product List”

```
CREATE VIEW [Current Product List] AS  
SELECT ProductID, ProductName  
FROM Products  
WHERE Discontinued=No
```

- Let’s add “Category” column to the view:

```
CREATE VIEW [Curent Product List] AS  
SELECT ProductID, ProductName, Category  
FROM Products  
WHERE Discontinued=No
```


Updating a VIEW in SQL Example

- From previous example of “CUSTOMERS_VIEW”, the code below update the age of Ramesh from 32 to 35:

```
SQL> UPDATE CUSTOMERS_VIEW  
SET AGE = 35  
WHERE name='Ramesh';
```

- That would update the base table CUSTOMERS and same would reflect in the view. Using SELECT statement would produce the following result:

```
SQL> SELECT * FROM CUSTOMERS_VIEW;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

Inserting Rows into a VIEW

- Rows of data can be inserted into a view;
Same rules that apply to UPDATE command also apply to the INSERT command

Deleting Rows into a VIEW

- Rows of data can be deleted from a view; Same rules that apply to UPDATE command also apply to the DELETE command

- Example of deleting a record having AGE=22

```
SQL> DELETE FROM CUSTOMERS_VIEW  
WHERE age = 22;
```

- This would delete a row from the base table CUSTOMERS and also reflect the view itself; Using SELECT statement would produce the following result:

```
SQL> SELECT * FROM CUSTOMERS_VIEW;
```

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	35	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
7	Muffy	24	Indore	10000.00

Dropping a VIEW in SQL

- A view can be deleted with the DROP VIEW command
- DROP VIEW Syntax:

`DROP VIEW view_name`

- For example:

`DROP VIEW CUSTOMERS_VIEW`

References

- <http://www.w3schools.com/sql/>
- <http://www.mysqltutorial.org/sql-triggers.aspx>
- <http://www.mssqltips.com/sqlservertutorial/2911/working-with-triggers/>
- <http://stackoverflow.com/questions/1278521/why-do-you-create-a-view-in-a-database>
- http://www.w3schools.com/sql/sql_view.asp