Database Management Systems

Vijaya Saradhi

IIT Guwahati

Fri, 06^{th} Mar 2020

Preserving FDs

Central Idea

- In addition to lossless decomposition, dependency preserving property must be satisified by decomposition
- The decmpositions statisfy all the FDs that are satisfied by the original relation
- Reason FDs satisified by a relation define integrity constraints that the relation needs to meet

Preserving FDs

Example

- Let R(X, Y, Z) satisfies the FDs: {XY $\rightarrow$ Z, Z $\rightarrow$ X }
- Let R be decomposed into two relations R_1(Y, Z) and R_2(Z, X)
- R_1 and R_2 are lossless decompositions; that is $R = R_1 \bowtie R_2$
- However, $R_1 \bowtie R_2$ do not preserve the FD XY $\rightarrow$ Z

R_1		R_2	
Y	Z	Z	X
y_1	z_1	z_1	x_1
y_1	z_2	z_2	x_1

XY $\rightarrow$ Z does not satisfy for $R_1 \bowtie R_2$

$R_1 \bowtie R_2$		
X	Y	Z
x_1	y_1	z_1
x_1	y_1	z_2

Projection of set of Dependencies

Onto set of attributes & algorithm

- Let r(R) has been recomposed into $(R_1, R_2, \cdots, R_k)$
- Let F be the set of FDs satisfied by r
- Define projection of F onto Z $\pi_Z(F)$ as

$$\pi_Z(F) = \{X \to Y \subset F | XY \subset Z\}$$

Projection of set of Dependencies

Onto set of attributes & algorithm

- To compute $\pi_Z(F)$, consider the proper subsets of Z that appear as determinant of an FD
- For each element in the proper subset of Z Do
  - Calcluate X^+
  - For P $\in X^+$ that satisfy
    - P $\subset$ Z
    - P $\subset X^+$
    - P $\not\subset$ X

  include X $\rightarrow$ P in $\pi_Z(F)$

Projection of set of Dependencies

Example

- Let r(X, Y, W, K, Q) and F = {X $\rightarrow$ K, Y $\rightarrow$ Q, KQ $\rightarrow$ W}
- Compute: $\pi_{\{X,Y,W\}}(F)$. That is Z = {X, Y, W}
- Note that only {X, Y} appear on LHS in F

Projection of set of Dependencies

Onto set of attributes & algorithm

- To compute $\pi_Z(F)$, consider the proper subsets of Z that appear as determinant of an FD
- For each element in the proper subset of Z Do
  - Calcluate X^+
  - For P $\in X^+$ that satisfy
    - P $\subset$ Z
    - P $\subset$ X^+
    - P $\not\subset$ X

  include X $\rightarrow$ P in $\pi_Z(F)$

Example

- Subset of {X, Y, W} that appear as determinant of an FD are: {X, Y}
- Both {X, Y} appear in LHS of F
- Proper subsets of {X, Y} are: {X}, {Y}, {X, Y}
- Consider {X}; Compute $\{X\}^+$
- $X^+ = \{X, K\}$
  - is X $\subset$ {X, Y, W}? Yes
  - is X $\subset$ X^+ = {X ,Y}; Yes
  - is X $\not\subset$ X; No

do not include X $\rightarrow$ X in $\pi_Z(F)$

Projection of set of Dependencies

Onto set of attributes & algorithm

- To compute $\pi_Z(F)$, consider the proper subsets of Z that appear as determinant of an FD
- For each element in the proper subset of Z Do
  - Calcluate X^+
  - For P $\in X^+$ that satisfy
    - P $\subset$ Z
    - P $\subset$ X^+
    - P $\not\subset$ X

  include X $\rightarrow$ P in $\pi_Z(F)$

Example

- Subset of {X, Y, W} that appear as determinant of an FD are: {X, Y}
- Both {X, Y} appear in LHS of F
- Proper subsets of {X, Y} are: {X}, {Y}, {X, Y}
- Consider {X}; Compute $\{X\}^+$
- $X^+ = \{X, K\}$
  - is K $\subset$ {X, Y, W}? No
  - is K $\subset$ X^+ = {X ,Y}; No
  - is K $\not\subset$ X Yes

  do not include X $\rightarrow$ K in $\pi_Z(F)$

Projection of set of Dependencies

Onto set of attributes & algorithm

- To compute $\pi_Z(F)$, consider the proper subsets of Z that appear as determinant of an FD
- For each element in the proper subset of Z Do
  - Calcluate X^+
  - For P $\in X^+$ that satisfy
    - P $\subset$ Z
    - P $\subset$ X^+
    - P $\not\subset$ X

  include X $\rightarrow$ P in $\pi_Z(F)$

Example

- Subset of {X, Y, W} that appear as determinant of an FD are: {X, Y}
- Both {X, Y} appear in LHS of F
- Proper subsets of {X, Y} are: {X}, {Y}, {X, Y}
- Consider {Y}; Compute $\{Y\}^+$
- $Y^+ = \{Y, Q\}$
  - is Y $\subset$ {X, Y, W}? Yes
  - is Y $\subset$ Y^+ = {X ,Y}; Yes
  - is Y $\not\subset$ Y

  do not include Y $\rightarrow$ Y in $\pi_Z(F)$; No

Projection of set of Dependencies

Onto set of attributes & algorithm

- To compute $\pi_Z(F)$, consider the proper subsets of Z that appear as determinant of an FD
- For each element in the proper subset of Z Do
  - Calculate X^+
  - For P $\in X^+$ that satisfy
    - P $\subset$ Z
    - P $\subset X^+$
    - P $\not\subset$ X

  include X $\rightarrow$ P in $\pi_Z(F)$

Example

- Subset of {X, Y, W} that appear as determinant of an FD are: {X, Y}
- Both {X, Y} appear in LHS of F
- Proper subsets of {X, Y} are: {X}, {Y}, {X, Y}
- Consider {Y}; Compute $\{Y\}^+$
- $Y^+ = \{Y, Q\}$
  - is Q $\subset$ {X, Y, W}? No
  - is Q $\subset Y^+$ = {X ,Y}; No
  - is Q $\not\subset Y$; Yes

  do not include Y $\rightarrow$ Q in $\pi_Z(F)$; Yes

Projection of set of Dependencies

Onto set of attributes & algorithm

- To compute $\pi_Z(F)$, consider the proper subsets of Z that appear as determinant of an FD
- For each element in the proper subset of Z Do
  - Calcluate X^+
  - For P $\in X^+$ that satisfy
    - P $\subset$ Z
    - P $\subset X^+$
    - P $\not\subset$ X

  include X $\rightarrow$ P in $\pi_Z(F)$

Example

- Subset of {X, Y, W} that appear as determinant of an FD are: {X, Y}
- Both {X, Y} appear in LHS of F
- Proper subsets of {X, Y} are: {X}, Y, {X, Y}
- Consider {X, Y}; Compute $\{X, Y\}^+$
- $\{X, Y\}^+ = \{X, Y, K, Q, W\}$
- $\{X, Y\} \rightarrow$ X is a trivial dependency. Exclude

Projection of set of Dependencies

Onto set of attributes & algorithm

- To compute $\pi_Z(F)$, consider the proper subsets of Z that appear as determinant of an FD
- For each element in the proper subset of Z Do
  - Calcluate X^+
  - For P $\in X^+$ that satisfy
    - P $\subset$ Z
    - P $\subset$ X^+
    - P $\not\subset$ X

  include X $\rightarrow$ P in $\pi_Z(F)$

Example

- Subset of {X, Y, W} that appear as determinant of an FD are: {X, Y}
- Both {X, Y} appear in LHS of F
- Proper subsets of {X, Y} are: {X}, Y, {X, Y}
- Consider {X, Y}; Compute $\{X, Y\}^+$
- $\{X, Y\}^+ = \{X, Y, K, Q, W\}$
- $\{X, Y\} \rightarrow Y$ is a trivial dependency. Exclude

Projection of set of Dependencies

Onto set of attributes & algorithm

- To compute $\pi_Z(F)$, consider the proper subsets of Z that appear as determinant of an FD
- For each element in the proper subset of Z Do
  - Calculate X^+
  - For P $\in X^+$ that satisfy
    - P $\subset$ Z
    - P $\subset$ X^+
    - P $\not\subset$ X

  include X $\rightarrow$ P in $\pi_Z(F)$

Example

- Subset of {X, Y, W} that appear as determinant of an FD are: {X, Y}
- Both {X, Y} appear in LHS of F
- Proper subsets of {X, Y} are: {X}, Y, {X, Y}
- Consider {X, Y}; Compute $\{X, Y\}^+$
- $\{X, Y\}^+ = \{X, Y, K, Q, W\}$
  - is K $\subset$ {X, Y, W}? No
  - is K $\subset$ $\{X, Y\}^+$ = {X ,Y, K, Q, W}; Yes
  - is K $\not\subset$ $\{X, Y\}$; Yes

  do not include {X, Y} $\rightarrow$ K in $\pi_Z(F)$; No

Projection of set of Dependencies

Onto set of attributes & algorithm

- To compute $\pi_Z(F)$, consider the proper subsets of Z that appear as determinant of an FD
- For each element in the proper subset of Z Do
  - Calculate X^+
  - For P $\in X^+$ that satisfy
    - P $\subset$ Z
    - P $\subset$ X^+
    - P $\not\subset$ X

  include X $\rightarrow$ P in $\pi_Z(F)$

Example

- Subset of {X, Y, W} that appear as determinant of an FD are: {X, Y}
- Both {X, Y} appear in LHS of F
- Proper subsets of {X, Y} are: {X}, Y, {X, Y}
- Consider {X, Y}; Compute $\{X, Y\}^+$
- $\{X, Y\}^+ = \{X, Y, K, Q, W\}$
  - is Q $\subset$ {X, Y, W}? No
  - is Q $\subset$ $\{X, Y\}^+$ = {X ,Y, K, Q, W}; Yes
  - is Q $\not\subset$ $\{X, Y\}$; Yes

  do not include {X, Y} $\rightarrow$ Q in $\pi_Z(F)$; No

Projection of set of Dependencies

Onto set of attributes & algorithm

- To compute $\pi_Z(F)$, consider the proper subsets of Z that appear as determinant of an FD
- For each element in the proper subset of Z Do
  - Calcluate X^+
  - For P $\in X^+$ that satisfy
    - P $\subset$ Z
    - P $\subset X^+$
    - P $\not\subset$ X

  include X $\rightarrow$ P in $\pi_Z(F)$

Example

- Subset of {X, Y, W} that appear as determinant of an FD are: {X, Y}
- Both {X, Y} appear in LHS of F
- Proper subsets of {X, Y} are: {X}, Y, {X, Y}
- Consider {X, Y}; Compute $\{X, Y\}^+$
- $\{X, Y\}^+ = \{X, Y, K, Q, W\}$
  - is W $\subset$ {X, Y, W}? Yes
  - is W $\subset$ $\{X, Y\}^+$ = {X ,Y, K, Q, W}; Yes
  - is W $\not\subset$ $\{X, Y\}$; Yes

  include {X, Y} $\rightarrow$ W in $\pi_Z(F)$; Yes

Dependency Preservation Testing

Method

- Given r(R), a decomposition $(R_1, R_2, \cdots, R_k)$
- A set F of FDs satisfied by r(R)
- Compute $\pi_{R_i}(F)$
- G $= \cup_{i=1}^{k} \pi_{R_i}(F)$
- Is G = F? if yes, then $(R_1, R_2, \cdots, R_k)$ is dependency preserving decomposition

Active Databases

Constructs

- Triggers: a series of actions associated with INSERT, UPDATE or DELETE queries and performed whenever these queries are involved
- Assertions: a boolean valued SQL expression that must be true at all times
- Events: Time based actions as opposed to query based

Active Databases - Triggers

Triggers

- Triggers also known as event-condition-action rules or ECA rules
- Triggers are involved only when certain conditions specified by the database programmer occur
- Trigger tests a specified condition. If the condition does not hold then nothing else associated with the trigger happens
- If the condition is satisfied then associated action is performed

Active Databases - Triggers

Triggers

- Has all the power of assertions
- Easier to implement
- Programmer specifies when they should be invoked
- Every trigger must be associated with a table
- Triggers are invoked automatically
- Triggers cannot be called directly
- Are part of transactions and can ROLLBACK transactions

Active Databases - Triggers

Triggers

- Cascade changes through related tables in database
- Enforce complex data integrity than a CHECK constraint
- Define custom error messages
- Compare before and after states of data under modification
- Triggers can be
  - Created
  - Altered
  - Dropped

Active Databases - Triggers

Triggers

- The action may be executed either before or after the triggering event
- Action can refer to old and new values of tuples that were inserted, deleted or updated
- Condition may be specified using WHEN clause
- Programmer has an option of specifying that the action is performed either:
  - Once for each modified tuple OR
  - Once for all the tuples that are changed in the database operation

Active Databases - Triggers

Triggers

- Invoke certain operations upon specified action on a table
- Action could be: insert a tuple into a table
- Action could be: delete a row from a table
- Action could be: update a row from in a table
- Performed operation can be on the table itself
- Performed Operation can be on other tables and/or databases

Trigger - Example - 01

Totaling amount

account (acct_num INT, amount FLOAT)

Sum Keep track of how much amount is deposited (irrespective of account number)

Insert The above operation should be performed for deposites only (not withdraw)

Before Sum opertaion should be performed even before the tuple (acct_num, amount) is inserted into the `account` table

Trigger - Example - 01

```
CREATE TABLE account(acct_num INT, amount FLOAT);

-- Create a global variable @sum
SET @sum = 0;

CREATE TRIGGER insert_sum
BEFORE INSERT
ON account
FOR EACH ROW
    SET @sum = @sum + NEW.amount;
```

Trigger - Example - 01

Trigger Action

```
CREATE TRIGGER insert_sum
BEFORE INSERT
ON account
FOR EACH ROW
    SET @sum = @sum + NEW.amount;
```

Trigger Events

```
INSERT INTO account VALUES (137, 14.98);
INSERT INTO account VALUES (141,1937.50);
INSERT INTO account VALUES (97,-100.00);

SELECT @sum AS "Total amount inserted';
```

Example

Explanation

- `CREATE TRIGGER` will create a trigger with the name `insert_sum`
- The trigger will not get executed immediately
- Condition for invoking trigger is: When a `INSERT` operation is performed on table `account`
- Statements in trigger gets executed even before the row is written into the account table

Names and meanings

- Before the statement `INSERT INTO account VALUES (137, 14.98);` there are no rows in the table
- Attributes/colums in a new row to be inserted are referred with NEW
- NEW.acct_num refers to 137
- NEW.amount refers to 14.98
- Rows that are already present in the `account` table are referred with OLD
- The statement SET @sum = @sum + NEW.amount; gets executed before row is inserted into account table

Trigger - Example - 02

Assumption

- Assume existance of table: `account(acc_num, amount)`
- updated amount must alway be between 0 and 100
- If the updated amount is more than 100, clamp to 100
- It the updated amount is less than 100, clamp to 0

```
DELIMITER //
CREATE TRIGGER update\_check
BEFORE UPDATE ON account
FOR EACH ROW
BEGIN
    IF NEW.amount < 0 THEN
        SET NEW.amount = 0;
    ELSEIF NEW.amount > 100 THEN
        SET NEW.amount = 100;
    END IF
END; //
DELIMITER ;
```

Trigger - Example - 03

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL PRIMARY KEY(a3));
CREATE TABLE test4(a4 INT NOT NULL PRIMARY KEY(a4), b4 INT DEFAULT 0);
```

Trigger - Example - 03

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
  END;
|

DELIMITER ;
```

Trigger - Example - 03

```
INSERT INTO test3 values (1), (2), (3), (4), (5), (6), (7), (8), (9), (10);
INSERT INTO test4 values (1, 0), (2, 0), (3, 0), (4, 0), (5, 0), (6, 0), (7,
        0), (8, 0), (9, 0), (10, 0);
```

Database state

test1	test2	test3	test4	
a1	a2	a3	a4	b4
		1	1	0
		2	2	0
		3	3	0
		4	4	0
		5	5	0
		6	6	0
		7	7	0
		8	8	0
		9	9	0
		10	10	0

Trigger - Example - 03 (a)

```
INSERT INTO test1 VALUES (1);
```

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
    test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
     NEW.a1;
  END;
|

DELIMITER ;
```

Database state

Trigger - Example - 03 (a)

```
INSERT INTO test1 VALUES (1);
```

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
    test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
     NEW.a1;
  END;
|

DELIMITER ;
```

Database state

test1	test2	test3	test4	
a1	a2	a3	a4	b4
1	1	~~1~~	1	1
		2	2	0
		3	3	0
		4	4	0
		5	5	0
		6	6	0
		7	7	0
		8	8	0
		9	9	0
		10	10	0

Trigger - Example - 03 (a)

```
INSERT INTO test1 VALUES (3);
```

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
    test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
     NEW.a1;
  END;
|

DELIMITER ;
```

Database state

Trigger - Example - 03 (a)

```
INSERT INTO test1 VALUES (3);
```

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
      test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
      NEW.a1;
  END;
|

DELIMITER ;
```

Database state

test1	test2	test3	test4	
a1	a2	a3	a4	b4
1	1	~~1~~	1	1
3	3	2	2	0
		~~3~~	3	1
		4	4	0
		5	5	0
		6	6	0
		7	7	0
		8	8	0
		9	9	0
		10	10	0

Trigger - Example - 03 (a)

```
INSERT INTO test1 VALUES (1);
```

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
    test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
     NEW.a1;
  END;
|

DELIMITER ;
```

Database state

Trigger - Example - 03 (a)

```sql
INSERT INTO test1 VALUES (1);
```

```sql
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
    test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
      NEW.a1;
  END;
|

DELIMITER ;
```

Database state

test1	test2	test3	test4	
a1	a2	a3	a4	b4
1	1	~~1~~	1	2
3	3	2	2	0
1	1	~~3~~	3	1
		4	4	0
		5	5	0
		6	6	0
		7	7	0
		8	8	0
		9	9	0
		10	10	0

Trigger - Example - 03 (a)

```
INSERT INTO test1 VALUES (7);
```

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
    test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
     NEW.a1;
  END;
|

DELIMITER ;
```

Database state

Trigger - Example - 03 (a)

```
INSERT INTO test1 VALUES (7);
```

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
    test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
      NEW.a1;
  END;
|

DELIMITER ;
```

Database state

test1	test2	test3	test4	
a1	a2	a3	a4	b4
1	1	~~1~~	1	2
3	3	2	2	0
1	1	~~3~~	3	1
7	7	4	4	0
		5	5	0
		6	6	0
		~~7~~	7	1
		8	8	0
		9	9	0
		10	10	0

Trigger - Example - 03 (a)

```
INSERT INTO test1 VALUES (1);
```

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
    test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
     NEW.a1;
  END;
|

DELIMITER ;
```

Database state

Trigger - Example - 03 (a)

```sql
INSERT INTO test1 VALUES (1);
```

```sql
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
    test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
      NEW.a1;
  END;
|

DELIMITER ;
```

Database state

test1	test2	test3	test4	
a1	a2	a3	a4	b4
1	1	~~1~~	1	3
3	3	2	2	0
1	1	~~3~~	3	1
7	7	4	4	0
1	1	5	5	0
		6	6	0
		7	7	1
		8	8	0
		9	9	0
		10	10	0

Trigger - Example - 03 (a)

```
INSERT INTO test1 VALUES (8);
```

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
    test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
      NEW.a1;
  END;
|

DELIMITER ;
```

Database state

Trigger - Example - 03 (a)

```
INSERT INTO test1 VALUES (8);
```

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
    test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
      NEW.a1;
  END;
|

DELIMITER ;
```

Database state

test1	test2	test3	test4	
a1	a2	a3	a4	b4
1	1	~~1~~	1	3
3	3	2	2	0
1	1	~~3~~	3	1
7	7	4	4	0
1	1	5	5	0
8	8	6	6	0
		7	7	1
		~~8~~	8	1
		9	9	0
		10	10	0

Trigger - Example - 03 (a)

```
INSERT INTO test1 VALUES (4);
```

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
    test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
     NEW.a1;
  END;
|

DELIMITER ;
```

Database state

Trigger - Example - 03 (a)

```
INSERT INTO test1 VALUES (4);
```

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
    test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
      NEW.a1;
  END;
|

DELIMITER ;
```

Database state

test1	test2	test3	test4	
a1	a2	a3	a4	b4
1	1	~~1~~	1	3
3	3	2	2	0
1	1	~~3~~	3	1
7	7	4	4	1
1	1	5	5	0
8	8	6	6	0
4	4	7	7	1
		8	8	1
		9	9	0
		10	10	0

Trigger - Example - 03 (a)

```
INSERT INTO test1 VALUES (4);
```

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
    test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
     NEW.a1;
  END;
|

DELIMITER ;
```

Database state

Trigger - Example - 03 (a)

```
INSERT INTO test1 VALUES (4);
```

```
DELIMITER |

CREATE TRIGGER testref BEFORE INSERT ON
      test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 =
      NEW.a1;
  END;
|

DELIMITER ;
```

Database state

test1	test2	test3	test4	
a1	a2	a3	a4	b4
1	1	~~1~~	1	3
3	3	2	2	0
1	1	~~3~~	3	1
7	7	~~4~~	4	2
1	1	5	5	0
8	8	6	6	0
4	4	7	7	1
4	4	8	8	1
		9	9	0
		10	10	0

Multiple Triggers On Same Table

Multiple Triggers

- Multiple triggers can be placed on a single table
- Source of multiple triggers are due to the way a trigger is created

```
CREATE TRIGGER trigger_name
{BEFORE | AFTER} {INSERT | DELETE | UPDATE }
ON table_name
{FOLLOWS | PRECEDES}
```

- When multiple triggers exists on same table, they must be ordered
- The ordering is specified at the time of creation
- $Trigger_1 \rightarrow Trigger_2 \rightarrow Trigger_3 \cdots$
- $Trigger_2$ follows $Trigger_1$
- $Trigger_3$ follows $Trigger_2$ and so on

Multiple Triggers On Same Table

Example

```
CREATE TABLE T2 (
    id INT,
    productCode VARCHAR(15) NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    updated_at TIMESTAMP NOT NULL
            DEFAULT CURRENT_TIMESTAMP
            ON UPDATE CURRENT_TIMESTAMP,
    PRIMARY KEY (id),
    FOREIGN KEY (productCode)
        REFERENCES T1 (productCode)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

Multiple Triggers On Same Table

Example

```
DELIMITER |

CREATE TRIGGER before_products_update
   BEFORE UPDATE ON T1
   FOR EACH ROW
BEGIN
     IF OLD.msrp <> NEW.msrp THEN
         INSERT INTO T2(product_code, price)
         VALUES(old.productCode, old.msrp);
     END IF;
END|

DELIMITER ;
```

Multiple Triggers On Same Table

Example

```sql
SELECT
    productCode,
    msrp
FROM
    T1
WHERE
    productCode = 'S12_1099';
```

productCode	msrp
S12_1099	194.57

Multiple Triggers On Same Table

Example

```
UPDATE T1
SET msrp = 200
WHERE productCode = 'S12_1099';
```

T2			
id	productCode	price	updated_at
1	S12_1099	194.57	2019-09-08 09:07:02

Multiple Triggers On Same Table

Example

```
CREATE TABLE T3 (
    id INT,
    productCode VARCHAR(15) DEFAULT NULL,
    updatedAt TIMESTAMP NOT NULL
            DEFAULT CURRENT_TIMESTAMP
        ON UPDATE CURRENT_TIMESTAMP,
    updatedBy VARCHAR(30) NOT NULL,
    PRIMARY KEY (id),
    FOREIGN KEY (productCode)
        REFERENCES T1 (productCode)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
```

Multiple Triggers On Same Table

Example

- Table T1 has one trigger on BEFORE UPDATE to insert some content into T2
- We now set another trigger on BEFORE UPDATE on T1 to insert some content into T3

Multiple Triggers On Same Table

Example

```
DELIMITER |

CREATE TRIGGER before_products_update_log_user
   BEFORE UPDATE ON T1
   FOR EACH ROW
   FOLLOWS before_products_update
BEGIN
    IF OLD.msrp <> NEW.msrp THEN
    INSERT INTO
            T3(productCode,updatedBy)
        VALUES
            (OLD.productCode,USER());
    END IF;
END|

DELIMITER ;
```

Multiple Triggers On Same Table

Example

```
UPDATE
    T1
SET
    msrp = 220
WHERE
    productCode = 'S12_1099';
```

T2

id	productCode	price	updated_at
1	S12_1099	194.57	2019-09-08 09:07:02
2	S12_1099	200.00	2019-09-08 09:10:32

Multiple Triggers On Same Table

Example

```
UPDATE
    T1
SET
    msrp = 220
WHERE
    productCode = 'S12_1099';
```

T3		
productCode	UpdatedAt	UpdatedBy
S12_1099	2019-09-08 09:10:32	root@localhost

System Information

Obtaining All Triggers

```
SHOW TRIGGERS
FROM classicmodels
WHERE `table` = 'T1';
```

TRIGGERS

Trigger	Event	Table	Statement	Timing
before_products_update	UPDATE	T1	BEGIN IF old.msrp ...	BEFORE
before_products_update_log_user	UPDATE	T1	BEGIN IF OLD.msrp ...	BEFORE

System Information

Action Order

```sql
SELECT
    trigger_name,
    action_order
FROM
    information_schema.triggers
WHERE
    trigger_schema = 'classicmodels'
ORDER BY
    event_object_table ,
    action_timing ,
    event_manipulation;
```

information_schema	
TRIGGER_NAME	ACTION_ORDER
before_products_update	1
before_products_update_log_user	2

Nested Triggers

Example

1. Place a trigger on table T1 with some action (say when a row gets inserted)
2. Place a trigger on table T2 with action that on insert, invoke a trigger to update table T3
3. When a row gets inserted into T1, it invokes first trigger
4. Invocation of first triggers causes invocation of second trigger

Recursive Triggers

Types

Direct recursion Occurs when a trigger fires and performs an action that causes the same trigger to fire again

Indirect recursion Occurs when a trigger fires and performs an action that causes a trigger on another table to fire... that causes original trigger to fire again

Considerations for Using Triggers

Considerations

- Constraints are proactive
- Triggers are reactive
- Constraints are checked before triggers
- Multiple triggers can be placed for an action
- Each trigger must be sequenced

Introduction

MySQL Stored Programs

- Stored programs is a generic term used for stored procedure, stored functions and triggers
- Without stored programs database system cannot claim full compliance with variety of standards including ANSI/ISO standars
- These standards describe how a DBMS should execute stored programs.
- Judicial use of stored programs lead to greater database security and integriety
- Improve overall application performance
- Improve maintainability

What is Stored Program?

What is it anyway?

- A computer program
- A series of instructions associated with a name
- The source code and any compiled version of the stored program are held within database server's system tables
- Program is executed within the memory address of database server

What is Stored Program?

Stored Procedures

Invocation	A generic program unit that is executed on request
Parameters	Accepts multiple input and output parameters

What is Stored Program?

Stored Procedures

Invocation	A generic program unit that is executed on request
Parameters	Accepts multiple input and output parameters

Stored Functions

Similar	to stored procedures
Constraint	Execution results in the return of single value
Invocation	Can be used within standard SQL statements
Extend SQL	Use of functions in SQL statements amount to extending SQL functionality

What is Stored Program?

Stored Procedures

Invocation	A generic program unit that is executed on request
Parameters	Accepts multiple input and output parameters

Stored Functions

Similar	to stored procedures
Constraint	Execution results in the return of single value
Invocation	Can be used within standard SQL statements
Extend SQL	Use of functions in SQL statements amount to extending SQL functionality

Triggers

Invocation	Activated in response to an activity within the database
DML	In particular when INSERT, UPDATE or DELETE statements are used

Why use Stored Programs?

Why another language?

- Developers have multitude of programming languages from which to choose
- Many of these are not database languages
- The code written in these languages does not reside in or managed by database server
- Stored programs offer many advantages. These are

Why use Stored Programs?

Advantages of Stored Programs

- Can lead to more secure database
- Offer mechanism to abstract data access routines in turn improve the maintainability of code as data structures evolve
- Reduces network traffic as then work on the data from within the server rather than transfering data across network
- Can be used to implement Common routines accessibile from multiple applications
- They can be executed either witin the database server
- Database-centric logic can be isolated in stored programs

Language Fundamentals

Variables

Declaration DECLARE `variable_name` datatype;

Example `DECLARE first_var INT;`

Value `first_var` is initialized with $\perp$ (NULL)

Example `DECLARE first_var INT DEFAULT 0;`

Value `first_var` is initialized with value 0

Language Fundamentals

Variables

Declaration DECLARE `variable_name` datatype;

Example `DECLARE first_var INT;`

Value `first_var` is initialized with $\perp$ (NULL)

Example `DECLARE first_var INT DEFAULT 0;`

Value `first_var` is initialized with value 0

More examples

- `DECLARE var1 INT DEFAULT -20000;`
- `DECLARE var2 FLOAT DEFAULT 1.8e-8;`
- `DECLARE var3 DOUBLE DEFAULT 2e45;`
- `DECLARE var4 DATE DEFAULT '1999-12-31';`

Assigning Values to Variables

Example - 1

```
SET variable_name = expression;
SET var1 = 10;
```

Example - 2

```
SET variable_name = expression;
SET var2 = 10.0001;
```

Example - 3

```
SET variable_name = expression;
SET var4 = '2018-11-12';
```

Parameters

Procedures and Functions

Are variables that can be passed into or out of the stored program

Three types exists

- **IN** Value must be specified by calling program. Modifications within stored program cannot be accessed from calling program
- **OUT** Modifications within stored program can be accessed from calling program.
- **INOUT** AN INOUT parameter acts both as IN and as an OUT parameter

Parameter - IN

Example

```
CREATE PROCEDURE demoIN(IN var1 INT)
BEGIN
    -- See the value of IN parameter
    SELECT var1;

    -- Modify
    SET var1 = 2;

    -- See the value of IN parameter
    SELECT var1;
END;
```

Execution

```
mysql> SET @myvar = 1;
mysql> CALL demoIN(@myvar);
mysql> SELECT @myvar;
```

- First line initializes @myvar variable
- Second line calls the stored procedure `demoIN`
- Withing `demoIN` var1 is read containing value 1
- Withing `demoIN` var1 is modified to value 2
- Third line read the variable @myvar which is 1

Parameter - OUT

Example

```sql
CREATE PROCEDURE demoOUT(OUT var1 INT)
BEGIN
    -- See the value of OUT parameter
    SELECT var1;

    -- Modify
    SET var1 = 2;

    -- See the value of OUT parameter
    SELECT var1;
END;
```

Execution

```
mysql> SET @myvar = 1;
mysql> CALL demoOUT(@myvar);
mysql> SELECT @myvar;
```

- First line initializes @myvar variable
- Second line calls the stored procedure `demoOUT`
- Withing `demoOUT` var1 is read containing value NULL (irrespective of its initialization outside procedure)
- Withing `demoOUT` var1 is modified to value 2
- Third line read the variable @myvar which is 2

Parameter - INOUT

Example

```
CREATE PROCEDURE demoINOUT(INOUT var1 INT)
BEGIN
    -- See the value of INOUT parameter
    SELECT var1;

    -- Modify
    SET var1 = 2;

    -- See the value of INOUT parameter
    SELECT var1;
END;
```

Execution

```
mysql> SET @myvar = 1;
mysql> CALL demoINOUT(@myvar);
mysql> SELECT @myvar;
```

- First line initializes @myvar variable
- Second line calls the stored procedure `demoINOUT`
- Withing `demoINOUT` var1 is read containing value 1
- Withing `demoINOUT` var1 is modified to value 2
- Third line read the variable @myvar which is 2

Built-in Functions

Categories

String functions Perform string manipulation; concatenation of two strings, obtaining substring etc

Mathematical functions Example: trigonometric functions, random number functions, logarithms etc

Date and time functions add or subtract time intervals from dates; find difference between two dates etc

Miscellaneous functions every thing not easily categorized in the above three groupings; encryption functions etc

Built-in Functions

String functions

```
SELECT roll_number, CONCAT(sur_name, " ",
first_name, " ", last_name) as full_name
FROM    Student
WHERE   Dept = 'EEE';
```

Built-in Functions

Mathematical functions

```
SELECT roll_number, ABS(quiz1_marks)
FROM   Student
WHERE  Dept = 'BSBE';
```

Built-in Functions

Mathematical functions

```sql
SELECT  roll_number, ROUND(SPI, 2)
FROM    Student
WHERE   Dept = 'EEE';
```

Built-in Functions

Date and time functions

```
SELECT roll_number, DAYNAME(held_on)
FROM   Attendance
WHERE  cid = 'CS441M';
```

Built-in Functions

Date and time functions

```sql
SELECT DATE_ADD('2018-05-01',INTERVAL 1 DAY);
-- '2018-05-02'

SELECT DATE_SUB('2018-05-01',INTERVAL 1 YEAR);
-- '2017-05-01'

SELECT DATE_ADD('2020-12-31 23:59:59', INTERVAL 1
SECOND);
-- '2021-01-01 00:00:00'

SELECT DATE_ADD('2018-12-31 23:59:59', INTERVAL 1
DAY);
-- '2019-01-01 23:59:59'
```

Blocks, Conditional statements

Block structure of stored programs

- Stored program consists of one or more blocks
- Each block commences with a BEGIN statement and terminate by an END
- Blocks are useful for defining variables within a block
- Variable within a block are not visible outside the block

Blocks

Block structure

- Various types of declarations can appear in a block
- Order in which these can occur is as follows
- Variable and condition declarations (errors)
- Cursor declarations
- Handler declarations
- Program code
- Violation of this order results in error

Blocks

Block structure - order

```
[label:] BEGIN
    variable declarations
    condition declarations
    cursor declarations
    handler declarations

    program code
END [label];
```

Blocks

Block structure - Example

```
CREATE PROCEDURE f1()
BEGIN
    DECLARE var1 INT DEFAULT 10;
END;
```

Nested Blocks

Nested block structures

- Some instances needed nested block structures
- Blocks that are defined within an enclosing block
- Variables defined within a block are not available outside the block
- However the variables are visible to blocks that are declared within the block

Nested Blocks

Nested block structure - Example

```
CREATE PROCEDURE f1()
BEGIN
    DECLARE outer_variable INT DEFAULT 10;
    BEGIN
        DECLARE inner_variable INT DEFAULT
20;
        SET inner_variable = 22;
    END;

    SET outer_variable = 12;

END;
```

Conditional Control

Conditional Statement - IF

```
            CREATE FUNCTION s_AND_d(IN sale_id INT, IN
    sale_value FLOAT)
            BEGIN
                IF( sale_value > 200 )
                THEN
                    CALL apply_free_shipping(sale_id);

                    IF ( sale_vale > 500 )
                    THEN
                        CALL apply_discount(sale_id, 20);
                    END IF;

                END IF;

            END;
```

Conditional Control

Conditional Statement - IF

```
IF( cpi > 7.0 )
THEN

    SELECT roll_number, full_name
    FROM   Student
    WHERE  Dept = 'EEE';

ELSE IF ( cpi BETWEEN 5.0 AND 7.0 )
THEN

    SELECT roll_number, full_name
    FROM   Student
    WHERE  Dept = 'BSBE';

ELSE

    SELECT roll_number, full_name
    FROM   Student
```

Conditional Control

Conditional Statement - CASE

Functionally equivalent to IF - ELSE IF - ELSE - END block

```
CASE
    WHEN condition THEN
        statements
    [WHEN condition THEN
        statements]
    [ELSE
        statements]
END CASE;
```

Conditional Control

Conditional Statement - CASE

```
            CASE
                WHEN (sale_value > 200 AND customer_status =
    'PLATINUM' ) THEN
                    CALL free_shipping(sale_id);
                    CALL apply_discount(sale_id, 20);

                WHEN (sale_value > 200 AND customer_status =
    'GOLD' ) THEN
                    CALL free_shipping(sale_id);
                    CALL apply_discount(sale_id, 15);

                WHEN (sale_value > 200 AND customer_status =
    'SILVER' ) THEN
                    CALL free_shipping(sale_id);
                    CALL apply_discount(sale_id, 10);

                WHEN (sale_value > 200 AND customer_status =
    'BRONZE' ) THEN
```

Iterative Processing with Loops

- LOOP statement
- REPEAT ... UNTIL
- WHILE

LOOP

Example

```
SET i = 1;
myloop: LOOP
    SET i = i + 1;
    IF i = 10
    THEN
        LEAVE myloop;
    END IF;
END LOOP myloop;
SELECT 'I can count 10';
```

REPEAT ... UNTIL

Example

```
SET i = 0;
loop1: REPEAT

    SET i = i + 1;

    IF MOD(i, 2) <> 0
    THEN
        SELECT CONCAT(i, " is an ODD number");
    END IF;

UNTIL i >= 10;

END REPEAT loop1;
```

WHILE Statement

Example

```
SET i = 1;
loop1: WHILE i <= 10 DO

    IF MOD(i, 2) <> 0
    THNE
        SELECT CONCAT(i, " is ODD number");
    END IF;
    SET i = i + 1;

    END WHILE loop1;
```

Stored Procedure

Example

```
CREATE PROCEDURE simple_sqls()
BEGIN
    DECLARE i INT DEFAULT 1;

    DROP TABLE IF EXISTS test_table;
    CREATE TABLE test_table(id INT, some_data
 CHAR(30), PRIMARY KEY (id));

    WHILE ( i <= 10 )
    DO
        INSERT INTO test_table(i, CONCAT("
record", i));
        SET i = i + 1;
    END WHILE;

END;
```