CS245: Databases

SQL

Vijaya saradhi

Department of Computer Science and Engineering
Indian Institute of Technology Guwahati

View Definitions - 01

Virtual Tables

- Relations defined using `CREATE TABLE` statement
- They actually exist in the database
- They are persistent
- Relations defined using `CREATE TEMPORARY TABLE` statemet
- They exist till certain period
- That is SQL system stores tables in some physical organization
- There is another class of SQL relations called views

View Definitions - 02

Virtual Tables

- Views do not exist physically
- They are defined by an expression much like a query
- View in turn be queried as if they exist physically
- In some cases they can be modified
- That is perform `INSERT, UPDATE, DELETE` operations on views

Declaring Views

Syntax Elements

Simple form of view definition is:

- The keyword CREATE VIEW
- The name of the view
- They keyword AS
- A query Q

About Q

Q is the definition of the view

Declaring Views

Syntax Elements

Simple form of view definition is:

- The keyword CREATE VIEW
- The name of the view
- They keyword AS
- A query Q

Complete Syntax

```
CREATE VIEW [view-name] AS [Q];
```

Creating Views

Example - 01

```
Movie(title, year, length, inColor, studioName, producerC)
```

```
CREATE VIEW ParamountMovies AS
        SELECT  title, year
        FROM    Movie
        WHERE   studioName = 'Paramount';
```

Querying Veiws

Example - 02

List titles of movies released in 1979 by Paramount studio from the view ParamountMovies

```
SELECT  title
FROM    ParamountMovies
WHERE   year = 1979;
```

Querying Veiws

Example - 03 internal conversion

List titles of movies released in 1979 by Paramount studio from the view ParamountMovies

```
SELECT  title
FROM    Movie
WHERE   studioName='Paramount' and year = 1979;
```

Querying Views AND tables

Example - 04

Query both view and table

```
SELECT  DISTINCT starName
FROM    ParamountMovies, StarsIn
WHERE   title='Top␣Gun' and year = 1986;
```

Creating Views

Example - 05 - Renaming attributes

`Movie(title, year, length, inColor, studioName, producerC)`

```
CREATE VIEW ParamountMovies(movieTitle, yr) AS
        SELECT  title, year
        FROM    Movie
        WHERE   studioName = 'Paramount';
```

Modifying Views - 01

Example

- Two types of views are created
- Read only view
- Updatable view

Modifying Views - 02

Example

- Updatable view should include the primary key
- For example, the primary key for `Movie` table is: (title, year, startName)
- Created view has all the three attributes then modification is:

```
INSERT INTO ParamountMovies('Top␣Gun␣02', 2020, 'Mr.␣ABCD');
```

- The record is inserted into the base table that is Movie
- The attributes length, inColor, producer assumes default value or NULL

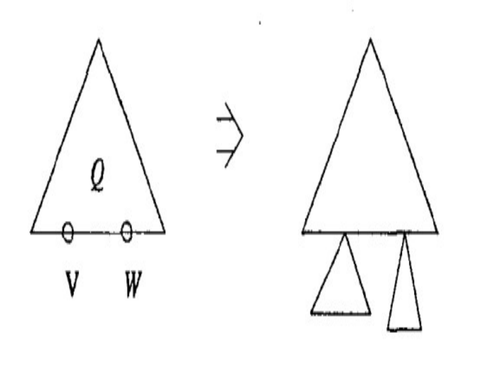
Modifying Views - 03

Example

```
DELETE
FROM      ParamountMovies
WHERE     title LIKE '%Trek%';
```

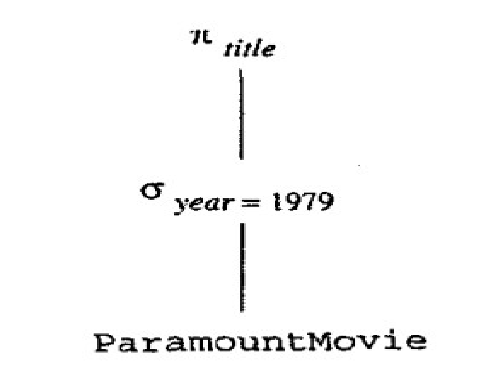
Interpreting Queries Involving Views

Interpretation - 01



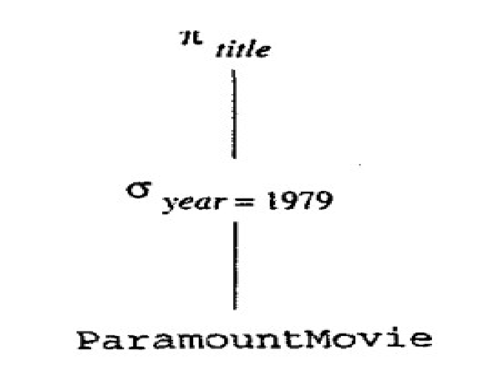
Interpreting Queries Involving Views

Interpretation - 02



Interpreting Queries Involving Views

Interpretation - 03



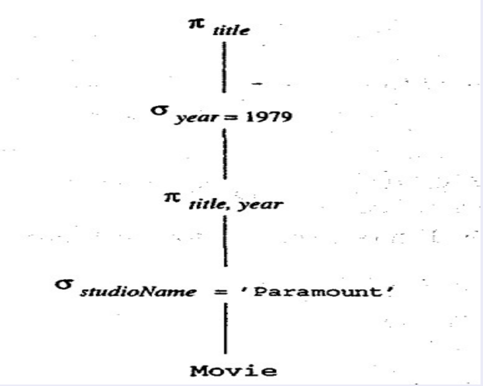
π_{title}

$\sigma_{year = 1979}$

ParamountMovie

Interpreting Queries Involving Views

Interpretation - 04



Types of VIEWS

Types

- Single-table projection and restrictions
- Calculated columns
- Grouped views
- Union-ed views
- Joins in views
- Nested views

Types of VIEWS

Calculated columns

Personnel(emp_id, salary, commision, $\cdots$)

```
CREATE VIEW Payroll AS
          SELECT emp_id, (salary + COALESCE(commission), 0.00)
          FROM     Personnel;
```

COALESCE returns a non-null value in the given list

Types of VIEWS

Another example

T1(a11, a12); T2(a21, a22);

```
CREATE VIEW temp_view AS
        SELECT T1.a21, T2.a22
        FROM   T1, T2
        WHERE  T1.a11 = T2.a21;
```

Types of VIEWS

Grouped Views

```
CREATE VIEW BigSales AS
        SELECT state_code, MAX(sales_amount)
        FROM    Sales
        GROUP BY state_code;
```

Types of VIEWS

UNION-ed Views

```
CREATE VIEW UnionView AS
 (SELECT   *
FROM      T1
WHERE     a11 = 1)
          UNION
 (SELECT   *
FROM      T2
WHERE     a21 = 2)
```

Types of VIEWS

Nested Views

```
CREATE VIEW all_boats AS SELECT * FROM boats;
CREATE VIEW red_boats AS SELECT * from all_boats where bcolor='re
```

Dropping VIEWS

Droping

```
DROP VIEW red_boats;
DROP VIEW all_boats;
```

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; Work on the data from within the server rather than transferring data across network
- Can be used to implement Common routines accessible from multiple applications
- They can be executed either within 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

```
DELIMITER //
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;//
DELIMITER ;
```

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

```
DELIMITER //
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;//
DELIMITER ;
```

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

```
DELIMITER //
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;//
DELIMITER ;
```

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

```
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 = 'CS245';
```

Built-in Functions

Date and time functions

```
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

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

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

```
DELIMITER //
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;//
DELIMITER ;
```

Nested Blocks

Nested block structure - Example

```
DELIMITER //
CREATE PROCEDURE f2()
BEGIN
  DECLARE outer_variable INT DEFAULT 10;
  BEGIN
    DECLARE  inner_variable INT DEFAULT 20;
    SET inner_variable = 22;
  END;
  SET outer_variable = 12;
  SELECT inner_variable, 'This␣statement␣causes␣an␣error';

END; //
DELIMITER ;
```

Nested Blocks - Overriding variables

Nested block structure - Example

```
DELIMITER //
CREATE PROCEDURE f3()
BEGIN
  DECLARE outer_variable INT DEFAULT 10;
  SET outer_variable = 27;
  BEGIN
    SET outer_variable = 57;
  END;
  SELECT outer_variable, 'This_statement_causes_overwriting_on_27_with_57';
END;//
DELIMITER ;
```

Nested Blocks

Nested block structure - Example

Changes made to an overloaded variable in an inner block are not visible outside the block

```
DELIMITER //
CREATE PROCEDURE f4()
BEGIN
  DECLARE my_variable varchar(20);
  SET my_variable='This value was set in the OUTER block';

  BEGIN
    DECLARE  my_variable varchar(20);
    SET my_variable='This value was set in the INNER block';
  END;

  SELECT my_variable, 'Can''t see changes made in the INNER block';
  SELECT 'As the scope of INNER BLOCK is ended';
END;//
DELIMITER ;
```

LEAVE statement

Exiting nested blocks

```
DELIMITER //
CREATE PROCEDURE f5()
   outer_block: BEGIN

     DECLARE l_status INT;
     SET l_status=1;

     inner_block: BEGIN
       IF (l_status = 1)
       THEN
         LEAVE inner_block;
       END IF
       SELECT 'This_statement_will_never_be_executed';
     END inner_block;
   SELECT 'End_of_program';
END outer_block;//
DELIMITER ;
```

Conditional Control

Conditional Statement - IF

```
DELIMITER //
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; //
DELIMITER ;


mysql> SELECT customer_name, s_AND_d(sale_id, sale_value) FROM Customer;
```

Conditional Control

Conditional Statement - IF

```
DELIMITER //
CREATE PROCEDURE f6(IN cpi FLOAT)
BEGIN
  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
    WHERE  Dept <> 'BSBE' AND Dept <> 'EEE';

        END IF;
END; //
DELIMITER ;
```

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

```
DELIMITER //
CREATE PROCEDURE f7(IN sale_value FLOAT, IN customer_status
  ENUM(‘PLATINUM’,␣‘GOLD’, ‘SILVER’,␣‘BRONZE’), IN sale_id INT)
BEGIN
  DECLARE dummy INT DEFAULT -1;
  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
    CALL free_shipping(sale_id);
    CALL apply_discount(sale_id, 5);
  WHEN (sale_value > 200 ) THEN
    CALL free_shipping(sale_id);
  ELSE
    SET dummy = 0;
END CASE; //
DELIMITER ;
```

Iterative Processing with Loops

- LOOP statement
- REPEAT ... UNTIL
- WHILE

LOOP

Syntax

```
[label:] LOOP
  statements
END LOOP [label];
```

LOOP

Example

```
DELIMITER //
CREATE PROCEDURE f7()
BEGIN
  DECLARE i INT DEFAULT 1;
  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';
END; //
DELIMITER ;
```

REPEAT ... UNTIL

Syntax

```
[label:] REPEAT
  statements
UNTIL expression
END REPEAT [label]
```

REPEAT ... UNTIL

Example

```
DELIMITER //
CREATE PROCEDURE f8()
BEGIN
  DECLARE i INT DEFAULT 1;
  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;
END; //
DELIMITER;
```

WHILE Loop

Syntax

```
[label:] WHILE expression
DO
   statements
END WHILE [label]
```

WHILE Statement

Example

```
DELIMITER //
CREATE PROCEDURE f9()
BEGIN
  DECLARE i INT DEFAULT 1;
  SET i = 1;
  loop1: WHILE i <= 10 DO

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

  END WHILE loop1;
END; //
DELIMITER;
```

Nested loops

Example

```
DELIMITER //
CREATE PROCEDURE f10()
BEGIN
  DECLARE i INT DEFAULT 1;
  DECLARE j INT DEFAULT 1;
  outer_loop: LOOP
    SET j = 1;
    inner_loop: LOOP
      SELECT CONCAT(i, "␣times␣", j, "␣is␣", i * j);
      SET j = j + 1;
      IF j > 12
      THEN
        LEAVE inner_loop;
      END IF;
    END LOOP inner_loop;

    SET i = i + 1;
    IF i > 12
    THEN
      LEAVE outer_loop;
    END IF;
  END outer_loop;
END; //
DELIMITER ;
```

Stored Procedure

Example

```
DELIMITER //
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; //
DELIMITER ;
```