Database Management Systems

Vijaya Saradhi

IIT Guwahati

Tue, 25th Feb 2020
(Friday Time Table)

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

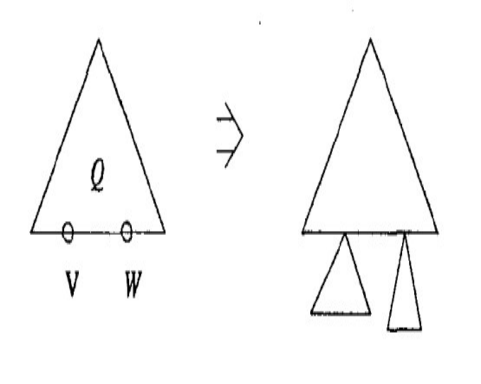
Modifying Views - 03

Example

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

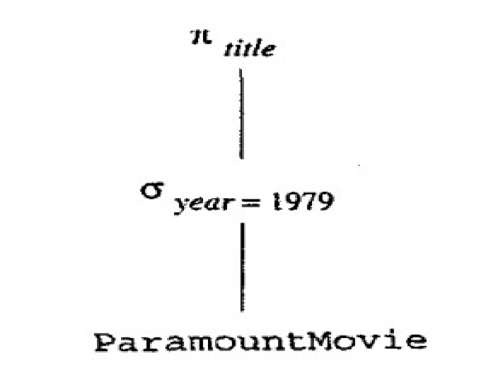
Interpreting Queries Involving Views

Interpretation - 01



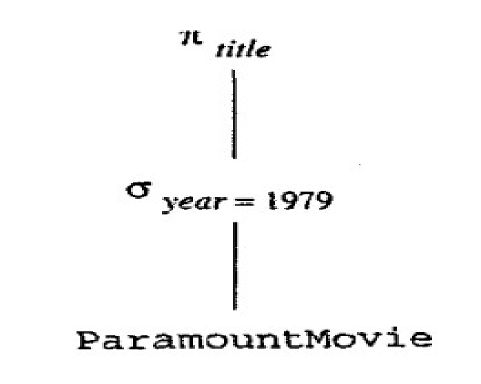
Interpreting Queries Involving Views

Interpretation - 02



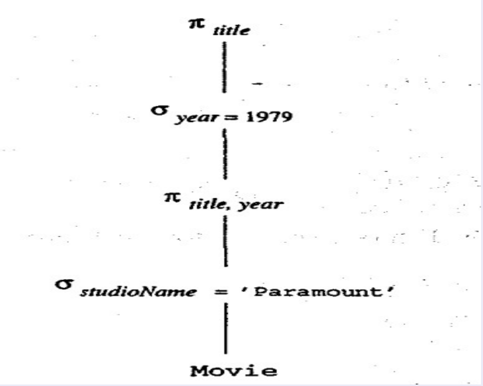
Interpreting Queries Involving Views

Interpretation - 03



Interpreting Queries Involving Views

Interpretation - 04



Types of VIEWS

Types

- Single-table projection and restrictions
- Calculated columns
- Translated 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

Translated columns

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
    ='red';
```

Dropping VIEWS

Droping

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

Active Databases - Assertions

Assertions

- SQL standard proposes a simple form of assertion that allows to enforce any condition
- The form of assertion is:
  - The keyword CREATE ASSERTION
  - The name of the assertion
  - The keyword CHECK
  - A parenthesized condition

```
CREATE ASSERTION assertion_name CHECK (
specified_condition)
```

- The specified_condition must be true when the assertion is created and must alway remain true

Active Databases - Assertions

Assertions - Example

```
CREATE TABLE Sailors(sid INT, sname CHAR(10), rating INT, age
     FLOAT,
    PRIMARY KEY(sid),
    CHECK(rating >= 1 AND rating <= 10)
    CHECK(
     ((SELECT COUNT(sid) FROM Sailors) +
        (SELECT COUNT(bid) FROM Boats)) < 100
    )
)
```

- The CHECK condition is performed only on Sailors table
- The condition is always true when Sailors table has 0 rows and Boats table have 101 rows
- This makes the constraint to be redundant

Active Databases - Assertions

Assertions - Example

```
CREATE ASSERTION smallClub
CHECK(
 ((SELECT COUNT(sid) FROM Sailors) +
    (SELECT COUNT(bid) FROM Boats)) < 100
)
```

- ASSERTION can span multiple tables
- specified conditions are checked for true at the end of the query
- When found to be FALSE, the query is rejected

Decomposition - Lossy/Loss-less?

Introduction

- Let R be a relation
- Let R be decomposed into two relations R_1 and R_2
- Let n be the number of tuples in R
- Decomposition should be performed in such a way that complete R with n tuples can be recovered using R_1 and R_2
- If we can recover original relation we say the decomposition is lossless
- Otherwise the decomposition is lossy

Decomposition - Lossy/Loss-less?

Example - 01

R		
X	Y	Z
x_1	y_1	z_1
x_2	y_2	z_2
x_3	y_2	z_3
x_4	y_3	z_4

R_1		R_2	
X	Y	Y	Z
x_1	y_1	y_1	z_1
x_2	y_2	y_2	z_2
x_3	y_2	y_2	z_3
x_4	y_3	y_3	z_4

Decomposition - Lossy/Loss-less?

Example - 01

$R_1 \bowtie R_2$		
X	Y	Z
x_1	y_1	z_1
x_2	y_2	z_2
x_2	y_2	z_3
x_3	y_2	z_2
x_3	y_2	z_3
x_4	y_3	z_4

Decomposition - Lossy/Loss-less?

Example - 02

R		
A	B	C
a	b	c

Example - 02

- Let the FD B $\rightarrow$ C exists on R;
- Let R be decomposed into R_1(A, B) and R_2(B, C)
- The relatoin is not in BCNF ($\{$B$\}$ is not the super key)
- If there is another FD: A $\rightarrow$ B then there is transitive dependency and $\{$A$\}$ will be the key
- If no other FD exists, then $\{$A, B$\}$ would be the key
- B is still not the superkey!

Decomposition - Lossy/Loss-less?

Example - 02

R_1		R_2	
A	B	B	C
a	b	b	c

Decomposition - Lossy/Loss-less?

Example - 02

R_1		R_2	
A	B	B	C
a	b	b	c

- $R_1 \bowtie R_2$ will yield original R
- The decomposition is lossless

Decomposition - Lossy/Loss-less?

Example - 03

If R contains the following tuples

R		
A	B	C
a	b	c
a	b	e

Example - 03

R_1		R_2	
A	B	B	C
a	b	b	c
a	b	b	e

Decomposition - Lossy/Loss-less?

Example - 03

If R contains the following tuples

R		
A	B	C
a	b	c
a	b	e

Example - 03

$R_1 \bowtie R_2$		
A	B	C
a	b	c
a	b	e
a	b	c
a	b	e

Decomposition - Lossy/Loss-less?

Example - 03

If R contains the following tuples

R		
A	B	C
a	b	c
a	b	e

Example - 03

- However, R cannot contain the tuple (a, b, e)
- As the FD: B $\rightarrow$ C is in place
- That is c = e
- When relations are decomposed according to FDs, then original relation can be recovered

Functional Dependency

Definition

Let $\mathbf{R} = \{R_1, R_2, R_3, \cdots R_p\}$ be a set of relation schemas over $\mathbf{U}$. A relation r($\mathbf{U}$) satisfies join dependency $*[R_1, R_2, \cdots, R_p]$ if r decomposes losslessly onto $R_1, R_2, \cdots, R_p$

$$r = \pi_{R_1}(r) \bowtie \pi_{R_2}(r) \bowtie \cdots \bowtie \pi_{R_p}(r)$$

Functional Dependency

Example

R		
A	B	C
a_1	b_1	c_1
a_1	b_2	c_2
a_3	b_3	c_3
a_4	b_3	c_4
a_5	b_5	c_5
a_6	b_6	c_5

R $= R_1 \bowtie R_2 \bowtie R_3$; *[AB, AC, BC]

R_1	
A	B
a_1	b_1
a_1	b_2
a_3	b_3
a_4	b_3
a_5	b_5
a_6	b_6

R_2	
A	C
a_1	c_1
a_1	c_2
a_3	c_3
a_4	c_4
a_5	c_5
a_6	c_5

R_3	
B	C
b_1	c_1
b_2	c_2
b_3	c_3
b_3	c_4
b_5	c_5
b_6	c_5

Trivial FD

Definition

A JD $*[R_1, R_2, \cdots, R_p]$ over R is trivial if it is satisfied by every relation r(R)

Project-Join Normal Form (PJNF)

Definition (5NF)

Let R be a relation scheme and let F be a set of FDs and JDs over R. R is in PJNF if every JD is trivial or R_i is a superkey for R.

Project-Join Normal Form (PJNF)

Example

- Let F = {*[ABCD, CDE, BDI], *[AB, BCD, AD], A $\rightarrow$ BCDE, BC $\rightarrow$ AI }
- R = A B C D E I
- R is not in PJNF with respect to F because of *[ABCD, CDE, BDI]
- Let $R_1 = ABCD$; $R_2 = CDE$; and $R_3 = BDI$
- The JD *[AB, BCD, AD]: each set of attributes is a superkey for R_1 due to FDs {A $\rightarrow$ BCDE, BC $\rightarrow$ AI }
- The FDs are either trivial or have keys as left sides

Preserving FDs

Introduction

- For a relation *r* to be recoverable from its projects its decomposition must be lossless
- In addition, decomposition should satisfy dependency preservation
- That is the decompositions satisfy all the FDs that are satisfied by the original relation
- Any decomposition that does not preserve the dependencies of the original relations imposes burden on RDBMS

Preserving FDs

Example

Let $r(X, Y, Z)$ satisfies FDs: $\{$XY $\rightarrow$ Z, Z $\rightarrow$ X$\}$. Let r(X, Y, Z) be decomposed into $R_1(YZ)$ and $R_2(ZX)$.

Preserving FDs

Introduction

R_1	
Y	Z
y_1	z_1
y_1	z_2

R_2	
Z	X
z_1	x_1
z_2	x_1

Join

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

R_2 satisfies Z $\rightarrow$ X; but $R_1 \bowtie R_2$ does not satisfy XY $\rightarrow$ Z