Lecture 2

Relational Database Language

Part 2

关系数据库语言

第2部分

Outline

- More SQL　更多SQL知识
- Subquery　子查询
- Group and Aggregation 分组与聚合
- Modification 更新

Outline

- More SQL　更多SQL知识
- Subquery　子查询
- Group and Aggregation 分组与聚合
- Modification 更新

Movie Database

Movies (title, year, length, genre, studioName, producerC#)

MovieStar (name, address, gender, birthdate)

StarsIn (movieTitle, movieYear, starName)

MovieExec (name, address, cert#, netWorth)

Studio (name, address, presC#)

SELECT-FROM-WHERE

```
SELECT *
FROM Movies
WHERE studioName='Disney' AND year=1990;
```

- FROM
  - Relation(s) to which the query refers
- WHERE
  - A condition
  - Tuples must satisfy the condition in order to match the query
- SELECT
  - Tells which attributes of the tuples matching the condition are produced as part of the answer

SQL Queries and Relational Algebra

SELECT L
FROM R
WHERE C

$$\Longleftrightarrow \quad \pi_L(\sigma_C(R))$$

– L is a list of expressions
– R is a relation
– C is a condition

Pattern Matching in SQL (Cont'd)

- Example

```
SELECT title
FROM Movies
WHERE  title LIKE 'Star ____';
```

  – We remember a movie "Star something",
  – And we remember that the something has four letters.
  – What could this movie be?

Pattern Matching in SQL (Cont'd)

- Example

```
SELECT title
FROM Movies
WHERE  title LIKE '%"s%';
```

  – Search for all movies with a possessive ('s) in their titles
  – Any title with 's as a substring will match the pattern

Pattern Matching in SQL (Cont'd)

- Escape characters in LIKE expressions

  s LIKE 'x%%x%' ESCAPE 'x'

  – What if the pattern we wish to use in a LIKE expression involves the character % or _ ?
  – SQL allows us to specify any one character we like as the escape character for a single pattern

  – Matches any string that begins and ends with the character %

Dates and Times 不同DBMS语法差异较大

- Date constant
  - DATE '1984-05-14'
- Time constant
  - TIME '15:00:02.5'
  - TIME '12:00:00-8:00'
    - Noon in Pacific Standard Time, eight hours behind GMT
- Combine dates and times
  - TIMESTAMP '1984-05-14 12:00:00'
- Compare dates and times using <

Null Values

- Null values NULL
  - Value unknown
    - An unknown birthdate
  - Value inapplicable
    - If we had a **spouse** attribute for the **MovieStar** relation, then an unmarried star might have NULL
  - Value withheld
    - An unlisted phone number might appear as NULL in the component for a **phone** attribute

Comparisons Involving NULL

- Two rules
  - Operate on a NULL and any value, using an arithmetic operator like * or +, the result is NULL
  - Compare a NULL and any value, using a comparison operator like = or >, the result is UNKNOWN
- NULL is *not* a constant
  - Cannot use NULL as an operand

Comparisons Involving NULL (Cont'd)

- Example
  - Let x have the value NULL
  - The value of $x + 3$ is also NULL
  - NULL + 3 is not a legal SQL expression

  - The value of $x = 3$ is UNKNOWN
  - NULL = 3 is not correct SQL

Comparisons Involving NULL (Cont'd)

- IS NULL
  - The correct way to ask if *x* has the value NULL

  x IS NULL
    - TRUE if *x* has the value NULL, FALSE otherwise

- IS NOT NULL

  x IS NOT NULL
    - TRUE unless the value of *x* is NULL

The Truth-Value UNKNOWN

- Three truth-values
  - TRUE as 1, FALSE as 0, and UNKNOWN as ½
  - The rules
    1. The AND of two truth-values is the minimum of those values
    2. The OR of two truth-values is the maximum of those values
    3. The negation of truth-value v is $1 - v$

The Truth-Value UNKNOWN (Cont'd)

- Truth table for three-valued logic

x	*y*	*x* AND *y*	*x* OR *y*	NOT *x*
TRUE	TRUE	TRUE	TRUE	FALSE
TRUE	UNKNOWN	UNKNOWN	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE
UNKNOWN	TRUE	UNKNOWN	TRUE	UNKNOWN
UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN	UNKNOWN
UNKNOWN	FALSE	FALSE	UNKNOWN	UNKNOWN
FALSE	TRUE	FALSE	TRUE	TRUE
FALSE	UNKNOWN	FALSE	UNKNOWN	TRUE
FALSE	FALSE	FALSE	FALSE	TRUE

The Truth-Value UNKNOWN (Cont'd)

- Surprising behavior

  Movies (title, year, length, genre, studioName, producerC#)

```
SELECT *
FROM Movies
WHERE length <= 120 OR length > 120;
```

  - **Movies** tuples with NULL in the length component
  - length <= 120 OR length > 120 evaluate to UNKNOWN
  - Such a tuple is not returned as part of the answer to the query
  - "Find all the **Movies** tuples with non-NULL lengths"

Ordering the Output

- The ORDER BY clause

  ORDER BY <list of attributes>

  – The order is by default ascending
  – The keyword DESC for descending
  – The keyword ASC for ascending (unnecessary)

    - The ordering is performed on the result of the FROM, WHERE, and other clauses, just before we apply the SELECT clause

Ordering the Output (Cont'd)

- Example

```
SELECT *
FROM Movies
WHERE studioName='Disney' AND year=1990
ORDER BY length, title;
```

  - All the attributes of **Movies** are available at the time of sorting, even if they are not part of the SELECT clause

Products and Joins in SQL

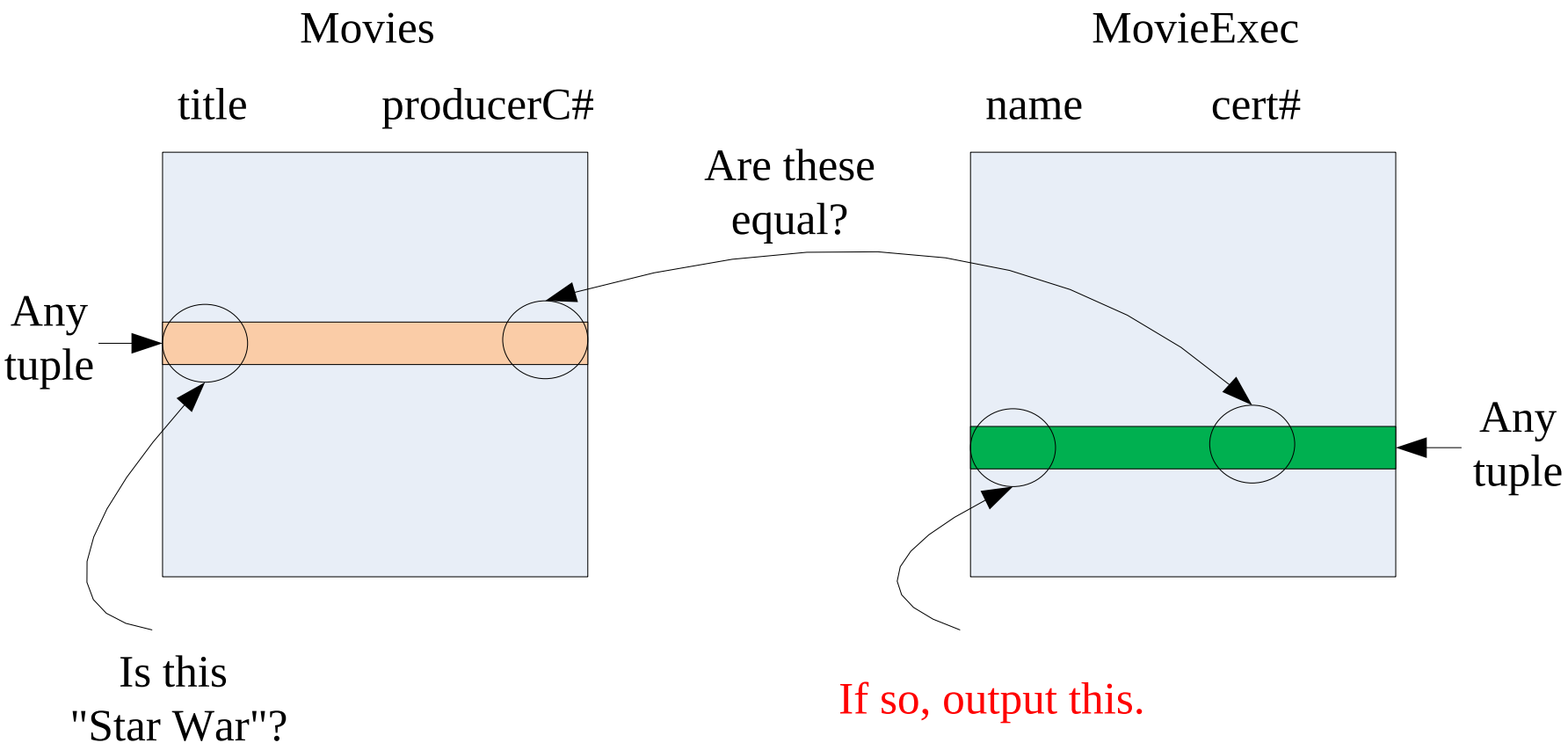
- Example
  - Want to know the name of the producer of *Star Wars*
  - Relations we need

    Movies (title, year, length, genre, studioName, producerC#)

    MovieExec (name, address, cert#, netWorth)
  - The query

```
SELECT name
FROM Movies, MovieExec
WHERE title= 'Star Wars' AND producerC# = cert#;
```

Products and Joins in SQL (Cont'd)



- To pair every tuple of Movies with every tuple of MovieExec and test two conditions

Tuple Variables

- Tuple variables
  - Disambiguate more than one occurrences of the same relation
- Example
  - Want to know about two stars who share an address

```
SELECT Star1.name, Star2.name
FROM MovieStar Star1, MovieStar Star2
WHERE Star1.address = Star2.address
    AND Star1.name < Star2.name
```

Interpreting Multirelation Queries (Cont'd)

- Conversion to Relational Algebra
  - FROM clause → Cartesian product
  - WHERE clause → selection
  - SELECT clause → projection

Interpreting Multirelation Queries (Cont'd)

- Conversion to Relational Algebra
  - Example

```
SELECT Star1.name, Star2.name
FROM MovieStar Star1, MovieStar Star2
WHERE Star1.address = Star2.address
    AND Star1.name < Star2.name
```

$$\pi_{A1,A5}(\sigma_{A2=A6 \text{ AND } A1<A5}(\rho_{M(A1,A2,A3,A4)}(MovieStar) \times \rho_{N(A5,A6,A7,A8)}(MovieStar)))$$

Outline

- More SQL　更多SQL知识
- Subquery　子查询
- Group and Aggregation 分组与聚合
- Modification 更新

Subquery

- Subquery
  - A query that is part of another
  - Subqueries can have subqueries, and so on

- Subquery
  - Can return a single constant, used in WHERE
  - Can return relations, used in WHERE
  - Can appear in FROM, followed by a tuple variable

Subqueries that Produce Scalar Values

- Scalar
  - An **atomic value** that can appear as one component of a tuple
- Example

Movies (title, year, length, genre, studioName, producerC#)

MovieExec (name, address, cert#, netWorth)

```
SELECT name
FROM Movies, MovieExec
WHERE title= 'Star Wars' AND producerC# = cert#;
```

Subqueries that Produce Scalar Values: Example

- Example

```
SELECT name
FROM MovieExec
WHERE cert# =
        (SELECT producerC#
         FROM Movies
         WHERE title = 'Star Wars'
         );
```

Subqueries that Produce Scalar Values: Example

- Example

```
SELECT name
FROM MovieExec
WHERE cert# =
    (SELECT producerC#
     FROM Movies
     WHERE title = 'Star Wars'
     );
```

The result is a unary relation with only one tuple, i.e., a scalar

A nested subquery

Subqueries that Produce Scalar Values: Example

- Example

```
SELECT name
FROM MovieExec
WHERE cert# =
        12345
```

The result is a unary relation with only one tuple, i.e., a scalar

A nested subquery

Subqueries that Produce Scalar Values: Example

- Example

```
SELECT name
FROM MovieExec
WHERE cert# =
    12345
```

The result is a unary relation with only one tuple, i.e., a scalar

A nested subquery

name
George Lucas

Conditions Involving Relations

- SQL operators
  - Apply to a relation *R*
  - Produce a Boolean result
  - *R* must be expressed as a subquery

1. EXISTS *R*

   is a condition that is true if and only if *R* is not empty

Conditions Involving Relations

- SQL operators
  - Apply to a relation *R*
  - Produce a Boolean result
  - *R* must be expressed as a subquery

2. *s* IN *R*

   is true if and only if *s* is equal to one of the values in *R*. Assume *R* is a unary relation

Conditions Involving Relations

- SQL operators
  - Apply to a relation *R*
  - Produce a Boolean result
  - *R* must be expressed as a subquery

3. *s* >ALL *R*

   is true if and only if *s* is greater than every value in unary relation *R*.

   s <ALL *R*, *s* <=ALL *R*, *s* >=ALL *R*, *s* =ALL *R*,
   s <> ALL *R* is the same as *s* NOT IN *R*

Conditions Involving Relations

- SQL operators
  - Apply to a relation *R*
  - Produce a Boolean result
  - *R* must be expressed as a subquery

4. *s* >ANY *R*

   is true if and only if *s* is greater than at least one value in unary relation *R*.

   s <ANY *R*, *s* <=ANY *R*, *s* >=ANY *R*, *s* <>ANY *R*,
   s = ANY *R* is the same as *s* IN *R*

Conditions Involving Relations

- SQL operators
  - EXISTS, ALL and ANY can be negated by putting NOT in front of the entire expression.

NOT EXISTS is true iff *R* is empty

NOT *s* >= ALL *R* is true iff *s* is not the maximum value in *R*

NOT *s* > ANY *R* is true iff *s* is the minimum value in *R*

ANY或ALL谓词

- 等价关系
  - ANY和ALL谓词、聚合函数、IN谓词

	=	<>	<	<=	>	>=
ANY	IN	--	<MAX	<=MAX	>MIN	>=MIN
ALL	--	NOT IN	<MIN	<=MIN	>MAX	>=MAX

Conditions Involving Tuples

- Example

```
SELECT name
FROM MovieExec
WHERE cert# IN
        (SELECT producerC#
         FROM Movies
         WHERE (title, year) IN
                (SELECT movieTitle, movieYear
                 FROM StarsIn
                 WHERE starName = 'Harrison Ford'
                )
         );
```

Conditions Involving Tuples

- Example

Main query

Subquery

A third subquery

```
SELECT name
FROM MovieExec
WHERE cert# IN
    (SELECT producerC#
     FROM Movies
     WHERE (title, year) IN
         (SELECT movieTitle, movieYear
          FROM StarsIn
          WHERE starName = 'Harrison Ford'
         )
    );
```

Conditions Involving Tuples

- Example

```
SELECT name
FROM MovieExec
WHERE cert# IN
        (SELECT producerC#
         FROM Movies
         WHERE (title, year) IN
```

title	*year*
Star Wars	1977
Raiders of the Lost Ark	1981
The Fugitive	1993
…	…

```
        );
```

Conditions Involving Tuples

- Example

```
SELECT name
FROM MovieExec
WHERE cert# IN
);
```

producerC#
12345
23456
34567
…

title	*year*
Star Wars	1977
Raiders of the Lost Ark	1981
The Fugitive	1993
…	…

Conditions Involving Tuples (Cont'd)

- Example
  - A single select-from-where expression

```
SELECT name
FROM MovieExec, Movies, StarsIn
WHERE cert# = producerC# AND
      title = movieTitle AND
      year = movieYear AND
      starName = 'Harrison Ford';
```

Correlated Subqueries

- Correlated subquery　　相关子查询
  - Requires the subquery to be evaluated many times
    - Once for each assignment of a value to some term in the subquery that comes from a tuple variable outside the subquery

Correlated Subqueries: Example

- Example
  - Find the titles that have been used for two or more movies

```
SELECT title
FROM Movies Old
WHERE year < ANY
      (SELECT year
       FROM Movies
       WHERE title = Old.title
      );
```

A movie made twice will be listed once, a movie made three times will be listed twice, and so on.

  - For each such tuple, we ask in the subquery whether there is a movie with the same title and a greater year

Subqueries in FROM Clauses

- Example

```
SELECT name
FROM MovieExec, (SELECT producerC#
                 FROM Movies, StarsIn
                 WHERE title = movieTitle AND
                       year = movieYear AND
                       starName = 'Harrison Ford'
                 ) Prod
WHERE cert# = Prod.producerC#;
```

Subqueries in FROM Clauses

- Example

Subquery

```
SELECT name
FROM MovieExec, (SELECT producerC#
                 FROM Movies, StarsIn
                WHERE title = movieTitle AND
                      year = movieYear AND
                      starName = 'Harrison Ford'
                ) Prod
WHERE cert# = Prod.producerC#;
```

Outline

- More SQL　更多SQL知识
- Subquery　子查询
- Group and Aggregation 分组与聚合
- Modification 更新

Full-Relation Operations

- Eliminating Duplicates: DISTINCT
- Grouping
- Aggregation
- HAVING Clauses

Grouping and Aggregation

- Grouping
  - Partition the tuples of a relation into "groups",
  - based on the values of tuples in one or more attributes

- Aggregation
  - Aggregate certain other columns of the relation by applying "aggregation" operators to those columns
  - **The aggregation is done separately for each group**

Aggregation Operators

- Five aggregation operators in SQL

SUM	Sum of a column with numerical values
AVG	Average of a column with numerical values
MIN	Smallest value of a column with numerical values
MAX	Largest value of a column with numerical values
COUNT	Number of (not necessarily distinct) values in a column

Aggregation Operators

- Five aggregation operators in SQL
  - NULL value

遇到空值

- 处理
  - COUNT(*)
- 忽略
  - COUNT、SUM、AVG、MAX、MIN

Aggregation Operators: Example

- Example
  - Finds the average net worth of all movie executives

MovieExec (name, address, cert#, netWorth)

```
SELECT AVG(netWorth)
FROM MovieExec;
```

Aggregation Operators: Example

- Example
  - Counts the number of tuples in the StarsIn

StarsIn (movieTitle, movieYear, starName)

```
SELECT COUNT(*)
FROM StarsIn;
```

  - The similar query

```
SELECT COUNT(starName)
FROM StarsIn;
```

Do these two queries have the same result?

Aggregation Operators: Example

- Example
  - Counts the number of tuples in the StarsIn

StarsIn (movieTitle, movieYear, starName)

```
SELECT COUNT(*)
FROM StarsIn;
```

Duplicate values are eliminated before we count.

  - The similar query

```
SELECT COUNT(DISTINCT starName)
FROM StarsIn;
```

Grouping

- The GROUP BY clause
  - Followed by a list of grouping attributes
  - Has tuples grouped according to their values in the grouping attributes
  - Whatever aggregation operators are used in the SELECT clause are applied only within groups

Grouping: Example

- Example

  Movies (title, year, length, genre, studioName, producerC#)

  – Finds the sum of the lengths of all movies for each studio

```
SELECT studioName, SUM(length)
FROM Movies
GROUP BY studioName;
```

Grouping: Example

- Example

```
SELECT studioName, SUM(length)
FROM Movies
GROUP BY studioName;
```

				studioName	
				Disney **Disney** **Disney**	
				MGM **MGM**	
				o **o**	

Grouping: The SELECT Clause

```
SELECT studioName, SUM(length)
FROM Movies
GROUP BY studioName;
```

- Two kinds of terms in the SELECT clause
  - Aggregations
    - An aggregate operator is applied to an attribute
    - These terms are evaluated on a per-group basis
  - Attributes in the GROUP BY clause
    - Only those attributes may appear unaggregated in the SELECT clause

Grouping: The SELECT Clause

```
SELECT studioName, SUM(length)
FROM Movies
GROUP BY studioName;
```

- Two kinds of terms in the SELECT clause

规律

- 出现在SELECT后面的列，
  - 要么是GROUP BY后面的分组列，
  - 要么是对其他列应用聚合函数

Grouping (Cont'd)

- No aggregations

```
SELECT studioName
FROM Movies
GROUP BY studioName;
```

  – Has the same effect as

```
SELECT DISTINCT studioName
FROM Movies;
```

HAVING Clauses

- WHERE clause
  - Restrict the tuples prior to grouping
    - Only wanted the total length of movies for producers with a net worth of more than $10,000,000

```
SELECT name, SUM(length)
FROM MovieExec, Movies
WHERE producerC# = cert# AND netWorth > 10000000
GROUP BY name;
```

HAVING Clauses (Cont'd)

- HAVING clause
  - We want to choose groups **based on some aggregate property** of the group itself

GROUP BY <*grouping attributes*>

HAVING <*condition about the group*>

HAVING Clauses (Cont'd)

- Example
  - Print the total film length for only those producers who made at least one film prior to 1930

```
SELECT name, SUM(length)
FROM MovieExec, Movies
WHERE producerC# = cert#
GROUP BY name
HAVING MIN(year) < 1930
```

The resulting query would remove from the grouped relation all those groups in which every tuple had a year component 1930 or higher.

HAVING Clauses (Cont'd)

- Two rules about HAVING clauses
  1. An aggregation in a HAVING clause applies only to the tuples of the group being tested
  2. Any attribute of relations in the FROM clause may be aggregated in the HAVING clause, but only those attributes that are in the GROUP BY list may appear **unaggregated** in the HAVING clause
     - The same rule as for the SELECT clause

Outline

- More SQL　更多SQL知识
- Subquery　子查询
- Group and Aggregation 分组与聚合
- Modification 更新

Database Modifications

- Three types
  1. Insert tuples into a relation
  2. Delete certain tuples from a relation
  3. Update values of certain components of certain existing tuples

Insertion

- Basic form

  INSERT INTO $R(A_1, \ldots, A_n)$ VALUES $(v_1, \ldots, v_n)$;

  – A tuple is created using the value v_i for attribute A_i, for $i = 1, 2, \ldots, n$

    - If the list of attributes does not include all attributes of the relation R, then the tuple created has default values for all missing attributes

Insertion: Example

- Example

```
INSERT INTO StarsIn(movieTitle, movieYear, starName)
VALUES('The Maltese Falcon', 1942, 'Sydney Greenstreet');
```

  – If we provide values for all attributes of the relation, then we may omit the list of attributes that follows the relation name.

```
INSERT INTO StarsIn
VALUES('The Maltese Falcon', 1942, 'Sydney Greenstreet');
```

Insertion: Subquery

- Example
  - Add to the relation **Studio** all movie studios that are mentioned in the relation **Movies**, but do not appear in **Studio**

```
INSERT INTO Studio(name)
    SELECT DISTINCT studioName
    FROM Movies
    WHERE studioName NOT IN
        (SELECT name
         FROM Studio);
```

Deletion

- Form

  DELETE FROM *R* WHERE <*condition*>;

  – Every tuple satisfying the condition will be deleted from relation *R*

Deletion: Example

- Example

  StarsIn (movieTitle, movieYear, starName)

  – Delete the fact that Sydney Greenstreet was a star in *The Maltese Falcon*

```
DELETE FROM StarsIn
WHERE movieTitle = 'The Maltese Falcon' AND
        movieYear = 1942 AND
        starName = 'Sydney Greenstreet';
```

Deletion: Example

- Example

  StarsIn (movieTitle, movieYear, starName)

  – Delete all tuples in the relation **StarsIn**
    - Make the relation **StarsIn** empty

  DELETE FROM StarsIn;

Updates

- Updates in SQL
  - One or more tuples that already exist in the database have some of their components changed
- Form

  UPDATE *R* SET <*new-value assignments*>

  WHERE <*condition*>;

Updates: Example

- Example
  - Modify the relation

  MovieExec (name, address, cert#, netWorth)

    - by attaching the title Pres. In front of the name of every movie executive who is the president of a studio

```
UPDATE MovieExec
SET name = 'Pres. ' || name
WHERE cert# IN (SELECT presC# FROM Studio);
```

…The End of This Lecture…