SQL
data manipulation language

SQL Data Manipulation Language (DML)

- Primarily declarative query language
  Specify *what* you want to compute and not *how*
- Starting point: relational calculus
  aka first-order predicate logic
- With many additions, bells and whistles…
- Corresponding procedural language: relational algebra

- Will discuss relational calculus & relational algebra later

Running example: Movie database

Movie		
Title	Director	Actor

Schedule	
Theater	Title

SQL DML: Basic Form

- Syntax:

  select attribName$_1$, …, attribName$_n$
  from relationName$_1$, …, relationName$_n$
  where condition

- The WHERE clause is optional
- Notation <RelationName>.<AttributeName>

  When more than one relation of the FROM has an attribute named A, we refer to a specific A attribute as <RelationName>.A

SQL Query Examples

Find titles of currently playing movies

```
select Title
from Schedule
```

Find the titles of all movies by “Berto”

```
select Title
from Movie
where Director=“Berto”
```

Find the titles and the directors of all currently playing movies

```
select Movie.Title, Director
from Movie, Schedule
where Movie.Title = Schedule.Title
```

Basic form: Informal semantics

Syntax

SELECT $a_1, \ldots, a_n$
FROM $R_1, \ldots, R_m$
WHERE *condition*

Semantics

for each tuple t_1 in R_1
  for each tuple t_2 in R_2
    …….
      for each tuple t_m in R_m

        if condition(t_1,t_2, … ,t_m) then
          output in answer attributes
          a_1,…,a_n of t_1,…,t_m

Informal Semantics

Examples revisited

Syntax

```
SELECT Title
FROM Movie
WHERE Director= “Berto”
```

Semantics

```
for each tuple m in Movie
      if m(Director) = “Berto”
         then output m(Title)
```

Informal Semantics

Examples revisited

Syntax

```
SELECT Movie.Title, Director
FROM Movie, Schedule
WHERE Movie.Title=Schedule.Title
```

Semantics

```
for each tuple m in Movie
   for each tuple s in Schedule
      if m(title) = s(title)
         then output <m(Title),m(Director)>
```

Tuple variables

- "Name" relations in the FROM clause
  Needed when using same relation more than once in FROM clause

e.g. find actors who are also directors

Syntax

```
SELECT  t.Actor
FROM  Movie  t, Movie s
WHERE t.Actor = s.Director
```

Semantics

```
for each t in Movie
   for each s in Movie
      if  t(Actor) = s(Director)
        then output t(Actor)
```

Tuple Variables

Examples revisited

Syntax *(without tuple vars)*

```
SELECT Title
FROM Movie
WHERE Director= “Berto”
```

Syntax *(with tuple vars)*

```
SELECT m.Title
FROM Movie m
WHERE m.Director = “Berto”
```

Tuple Variables

Examples revisited

Syntax *(without tuple vars)*

```
SELECT Movie.Title, Director
FROM Movie, Schedule
WHERE Movie.Title=Schedule.Title
```

Syntax *(with tuple vars)*

```
SELECT m.Title,  m.Director
FROM Movie m, Schedule s
WHERE m.Title = s.Title
```

*

- Used to select all attributes
- Example:
  Retrieve all movie attributes of currently playing movies

```
select Movie.*
from Movie, Schedule
where Movie.Title=Schedule.Title
```

LIKE Keyword

- Used to express pattern matching conditions
- Syntax:
  <attr> **LIKE** <pattern>
- Examples:
  Retrieve all movies where the title starts with “Ta”

  select *
  from Movie
  where Title **LIKE** ‘Ta%’

  Forgot if “Polanski” is spelled with ‘i’ or ‘y’

  select *
  from Movie
  where Director **LIKE** ‘Polansk_’

DISTINCT Keyword

- Used to do duplicate elimination

  By default query results contain duplicates: Duplicate elimination has to be explicitly specified

- Syntax:

  select distinct …
  from …
  where …

- Examples:

  Retrieve distinct movie titles

  select distinct title
  from Movie

ORDER BY clause

- Used to order the display of tuples in the result
- Example:

  List all titles and actors of movies by Fellini, in alphabetical order of titles

  select Title, Actor
  from Movie
  where Director = 'Fellini'
  ORDER BY Title

- Can specify order for each attribute

  Through DESC for descending and ASC for ascending order. Ascending order is the default.

  e.g. **ORDER BY** Title **DESC**

AS Keyword

- Used to rename attributes in the result
- Example:

  Find titles of movies by Bertolucci, under attribute Berto-title:

  select title **AS** Berto-title
  from movie
  where director = ‘Bertolucci

Aggregate Functions

- These functions operate on the multiset of values of a column of a relation, and return *a single* value
- Functions:

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

Aggregate Function Examples

Find the average account balance at the La Jolla branch

```
select avg (balance)
from account
where branch_name = 'La Jolla'
```

Find the number of tuples in the customer relation

```
select count (*)
from customer
```

Find the number of depositors in the bank

```
select count (distinct customer_name)
from depositor
```

Aggregate Function Examples

Find the maximum salary, the minimum salary, and the average salary among all employees for the Company database

```
select max(salary), min(salary), avg(salary)
from employee
```

Ops! Some SQL implementations *may not allow more than one function* in the SELECT-clause!

Aggregate Function Examples

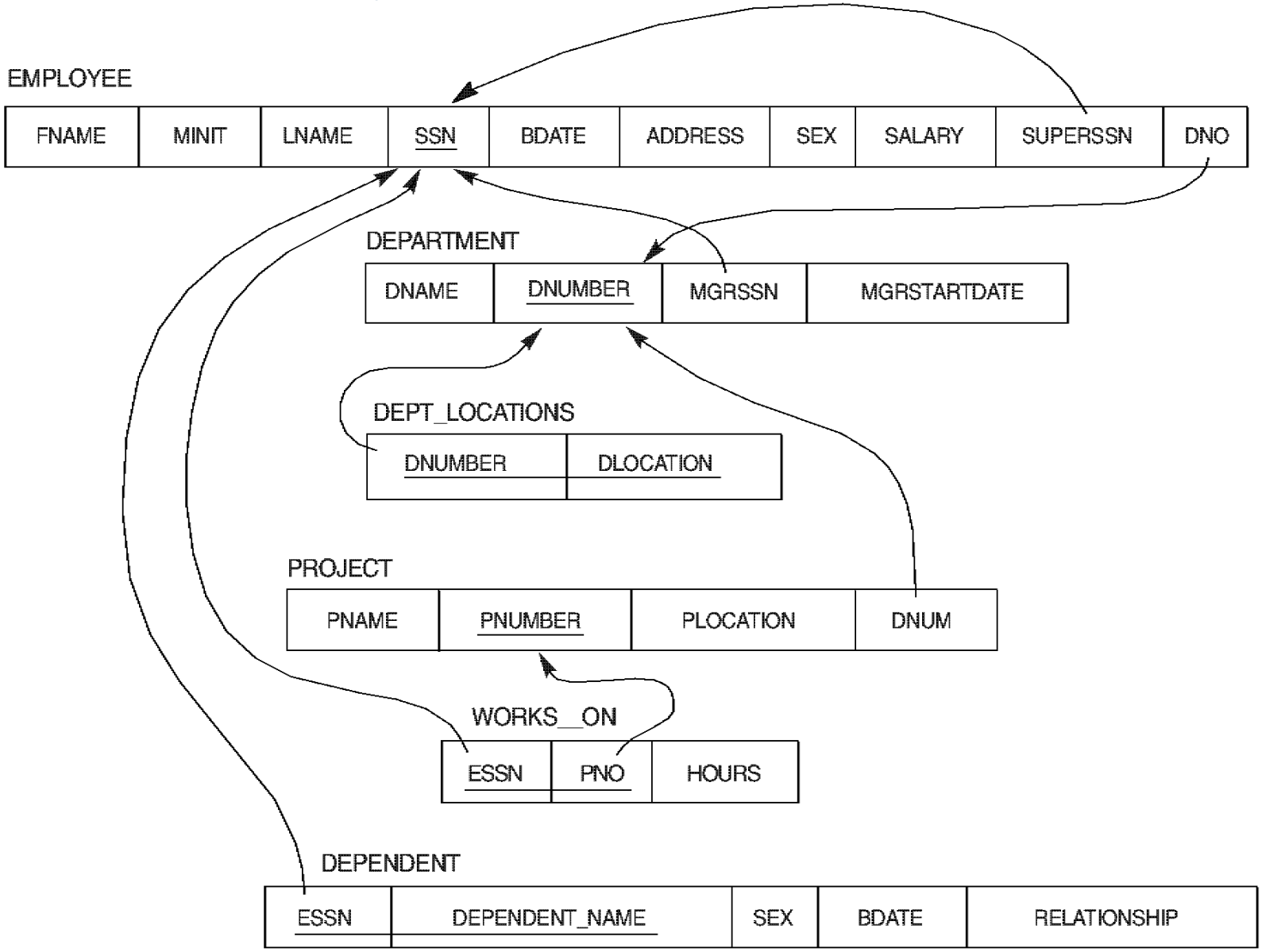
Find the maximum salary, the minimum salary, and the average salary among employees who work for the 'Research' department

```
select max(salary), min(salary), avg(salary)
from employee, department
where dno = dnumber and dname = ‘Research’
```

Note: The aggregate functions are applied to the relation consisting of all pairs of tuples from Employee and Department satisfying the condition in the WHERE clause

20

Reminder: Company schema



Grouping Example

Employee

Name	Dept	Salary
Joe	Toys	45
Nick	PCs	50
Jane	Toys	35
Maria	PCs	40

Find the average salary of all employees

```
select avg(Salary) AS AvgSal
from Employee
```

AvgSal
42.5

Find the average salary for each department

```
Select Dept, avg(Salary) AS AvgSal
from Employee
group by Dept
```

Dept	AvgSal
Toys	40
PCs	45

Grouping

- Allows to apply the aggregate functions
  to subgroups of tuples in a relation
- Each subgroup of tuples consists of the set of tuples that have *the same value* for the *grouping attribute(s)*
- The function is applied to each subgroup independently
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which *must also appear in the SELECT-clause*

Grouping

- For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
SELECT      DNO, COUNT (*) AS NUMEMP,  AVG (SALARY) AS AVGSAL
FROM        EMPLOYEE
GROUP BY    DNO
```

The EMPLOYEE tuples are divided into groups--each group having the same value for the grouping attribute DNO

The COUNT and AVG functions are applied to each such group of tuples separately

The SELECT-clause includes only the grouping attribute and the aggregate functions to be applied on each group of tuples

Grouping Example

- Example:

  For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
SELECT      PNUMBER, PNAME, COUNT (*)
FROM        PROJECT, WORKS_ON
WHERE       PNUMBER=PNO
GROUP BY    PNUMBER, PNAME
```

- Note:

  The grouping and functions are applied on pairs of tuples from PROJECT, WORKS_ON

Subtlety: suppose PNO and ESSN do not form a key for WORKS_ON
Problem: will get duplicate employees

Works_on	ESSN	PNO	HOURS	PROJECT	PNAME, PNUMBER	
	111-11-1111	001	20		Wiki	001
	111-11-1111	001	10		Geo	002
	22-22-2222	002	25			

Fix:

```
SELECT      PNUMBER, PNAME, COUNT (DISTINCT ESSN)
FROM        PROJECT, WORKS_ON
WHERE       PNUMBER=PNO
GROUP BY    PNUMBER, PNAME
```

HAVING Clause

- Sometimes we want to retrieve the values of aggregate functions for only those groups that satisfy certain conditions
- The HAVING-clause is used for specifying a selection condition on groups (rather than on individual tuples!)

HAVING Clause

- Example:

  Find the names of all branches where the average account balance is more than $1,200

```
select    branch_name, avg (balance)
from      account
group by  branch_name
HAVING    avg(balance) > 1200
```

- Condition in HAVING clause can use:
  - Values of attributes in group-by clause
  - Aggregate functions on the other attributes

HAVING Clause

- Example:

  For each project on which more than two employees work , retrieve the project number, project name, and the number of employees who work on that project.

```
select      pnumber, pname, count(*)
from        project, works_on
where       pnumber=pno
group by    pnumber, pname
HAVING      count (*) > 2
```

- Note:

  Predicates in the having clause are applied after the formation of groups whereas predicates in the where clause are applied before forming groups

HAVING Clause

- Example:

  For each movie having more than 100 actors, find the number of theaters showing the movie

  select m.Title, **count**(distinct s.Theater) **as** number
  from Schedule s, Movie m
  where s.Title = m.Title
  group by m.Title
  having count(**distinct** m.Actor) > 100

- Note:

  Aggregate is taken over pairs <s,m> with same Title

SQL Queries: Nesting

- The WHERE clause can contain predicates of the form
  attr/value IN <SQL query>
  attr/value NOT IN <SQL query>
- Semantics:
  The IN predicate is satisfied if the attr or value appears in the result of the nested <SQL query>
- Examples:
  Find directors of current movies

  SELECT director **FROM** Movie
  WHERE title **IN**

  (**SELECT** title
  FROM schedule)

  The nested query finds currently playing movies

Nesting Example

- Example:
  Find actors playing in some movie by Bertolucci

  SELECT actor
  FROM Movie
  WHERE title **IN**

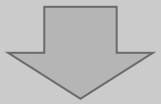
  (**SELECT** title
  FROM Movie
  WHERE director = “Bertolucci”)

- Note:
  The nested query finds the titles of movies by Bertolucci

Nesting Example

- Example:
  In this case we can eliminate nesting:

```
SELECT actor
FROM Movie
WHERE title IN
        (SELECT title
        FROM Movie
        WHERE director = "Bertolucci")
```



```
SELECT m1. actor
FROM Movie m1, Movie m2
WHERE  m1.title = m2.title AND
        m2.director = "Bertolucci"
```

Question

- Can we always eliminate nesting?

  Queries involving nesting but no negation can always be unnested in contrast to queries with nesting and negation

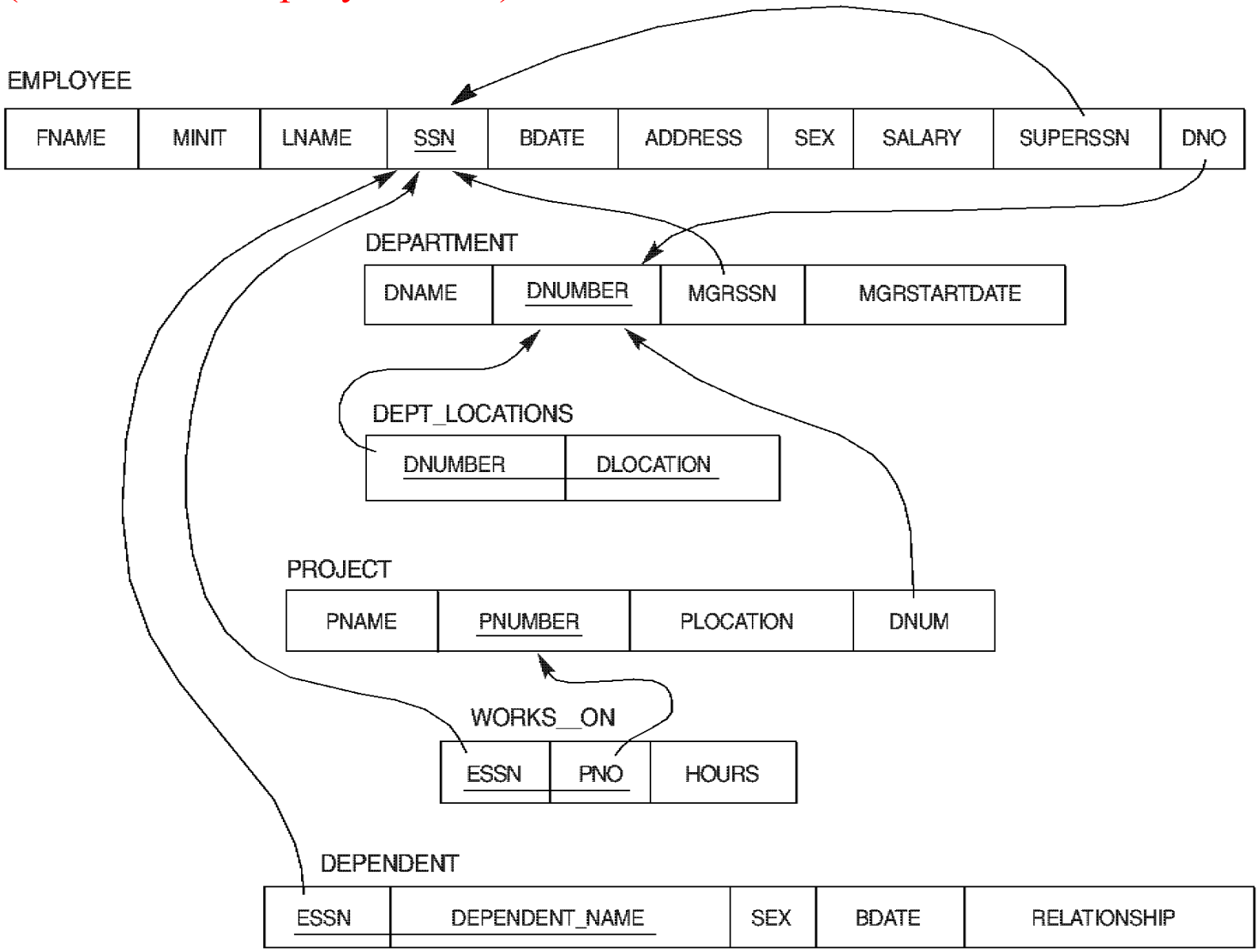
Correlated Nested Queries

- If a condition in the WHERE-clause of a *nested query* references an attribute of a relation declared in the *outer query* , the two queries are said to be correlated
- The result of a correlated nested query may be different for each tuple (or combination of tuples) of the relation(s) the outer query
- Example:

  Retrieve the name of each employee who has a dependent with the same first name as the employee

```
SELECT   E.FNAME, E.LNAME
FROM     EMPLOYEE  E
WHERE    E.SSN  IN
         (SELECT ESSN
         FROM     DEPENDENT
         WHERE    ESSN=E.SSN
          AND     E.FNAME=DEPENDENT_NAME)
```

(Reminder: company schema)



Correlated Nested Queries

- Correlated queries using just the = or IN comparison operators can still be unnested:

  e.g., the previous query can be unnested as follows:

```
SELECT   E.FNAME, E.LNAME
FROM     EMPLOYEE  E, DEPENDENT  D
WHERE    E.SSN=D.ESSN    AND
         E.FNAME=D.DEPENDENT_NAME
```

- Use of NOT IN tests increases expressive power!

37

Simple use of NOT IN

- Example:
  Find all movies in which Hitchcock does not act

  SELECT title **FROM** Movie
  WHERE title **NOT IN**

  (**SELECT** title **FROM** Movie
  WHERE actor = 'Hitchcock')

38

Simple use of NOT IN

- Example:
  Find all movies that are not currently playing

```
SELECT  title FROM Movie
WHERE title NOT IN
        (SELECT title FROM Schedule)
```

Why can't this be flattened?

Hand-waving "proof":

- Basic queries with no nesting are monotonic:

  The answer never decreases when the database increases
  DB1 $\subseteq$ DB2 implies Query(DB1) $\subseteq$ Query(DB2)

- But queries using NOT IN are not monotonic:

  e.g., **SELECT** title **FROM** Movie
  WHERE title **NOT IN**
  (**SELECT** title **FROM** Schedule)

  If Schedule increases, the answer might decrease

Recall

Semantics of basic queries

Syntax

SELECT $a_1, \ldots, a_n$
FROM $R_1, \ldots, R_m$
WHERE *condition*

Semantics

for each tuple t_1 in R_1
  for each tuple t_2 in R_2
    …….
      for each tuple t_m in R_m

        if condition(t_1,t_2, … ,t_m) then
          output in answer attributes
          a_1,…,a_n of t_1,…,t_m

This is monotonic if condition has no nested queries

More complex use of NOT IN

- Example:
  Find the names of employees with the maximum salary

```
SELECT  name FROM Employee
WHERE salary NOT IN
        (SELECT  e.salary
        FROM Employee e, Employee f
        WHERE   e.salary < f.salary)
```

Intuition: salary is maximum if it is not among salaries e.salary lower than some f.salary

More complex use of NOT IN

- Example:
  Find actors playing in every movie by “Berto”

```
SELECT Actor FROM Movie
WHERE Actor NOT IN
        (SELECT m1.Actor
        FROM Movie m1, Movie m2,
        WHERE m2.Director=“Berto”
        AND m1.Actor NOT IN
                (SELECT Actor
                FROM Movie
                WHERE Title=m2.Title))
```

The shaded query finds actors for which there is some movie by “Berto” in which they do not act

More complex use of NOT IN

- Example:
  Find actors playing in every movie by “Berto”

  SQL’s way of saying this:

  find the actors for which
  there is no movie by
  Bertolucci in which
  they do not act

  OR equivalently:

  find the actors not among
  the actors for which there is
  some movie by Bertolucci
  in which they do not act

EXISTS

- Another construct used with nesting
- Syntax:

  SELECT …
  FROM …
  WHERE EXISTS (<query>)

- Semantics:

  EXISTS(<query>) is true iff the result of <query> is non-empty

  NOT EXISTS(<query>) is true iff the result of <query> is empty

Example of EXISTS

- Example:
  Find titles of currently playing movies directed by Berto

```
SELECT s.title
FROM  schedule s
WHERE EXISTS   (SELECT * FROM movie
                WHERE movie.title = s.title AND
                      movie.director = 'Berto' )
```

Example of EXISTS

- Example (Boolean Predicate):
  Everybody likes UCSD

NOT EXISTS

(**SELECT** * **FROM** PERSON
WHERE NOT EXISTS

(**SELECT** * **FROM** LIKES
WHERE PERSON.name = LIKES.name
AND school= 'UCSD'

Example of EXISTS

- Example:
  Find the actors playing in every movie by Berto

```
SELECT a.actor FROM movie a
WHERE NOT EXISTS
        (SELECT * FROM movie m
        WHERE m.director = 'Berto' AND
                NOT EXISTS
                        (SELECT *
                        FROM movie t
                        WHERE m.title = t.title
                        AND t.actor = a.actor))
```

Union, Intersection & Difference

- Union:
  <SQL Query 1> **UNION** <SQL Query 1>
- Intersection:
  <SQL Query 1> **INTERSECT** <SQL Query 1>
- Difference:
  <SQL Query 1> **EXCEPT** <SQL Query 1>

Union, Intersection & Difference

- Example:
  Find all actors or directors

  (**SELECT** Actor **AS** Name
  FROM Movie)

  UNION

  (**SELECT** Director **AS** Name
  FROM Movie)

Union, Intersection & Difference

- Example:
  Find all actors who are not directors

  (**SELECT** Actor **AS** Name
  FROM Movie)

  EXCEPT

  (**SELECT** Director **AS** Name
  FROM Movie)

Natural Join

- Combines tuples from two tables by matching on common attributes

movie	title	director	actor
	Tango	Berto	Brando
	Sky	Berto	Winger
	Psycho	Hitchcock	Perkins

schedule	theater	title
	Hillcrest	Tango
	Paloma	Tango
	Paloma	Bambi
	Ken	Psycho

movie **natural join** schedule	title	director	actor	theater
	Tango	Berto	Brando	Hillcrest
	Tango	Berto	Brando	Paloma
	Psycho	Hitchcock	Perkins	Ken

Natural Join

- Example:
  Find the directors of all movies showing in Hillcrest

  select director
  from movie **natural join** schedule
  where theater = 'Hillcrest'

- Question:
  Can we write this in a different way?

  select director
  from movie, schedule
  where movie.title = schedule.title **and** theater = 'Hillcrest'

- Note:
  More variations of joins available in SQL…

Nested Queries: Existential and Universal Quantification

- *A op* ANY <*nested query*> is satisfied if ***there is*** a value *X* in the result of the <*nested query*> and the condition *A op X* is satisfied
  ANY aka SOME
- *A op* ALL <*nested query*> is satisfied if ***for every*** value *X* in the result of the <*nested query*> the condition *A op X* is satisfied

Nested Queries: Existential & Universal Quantification

- Example:
  Find directors of currently playing movies

```
SELECT Director
FROM Movie
WHERE Title = ANY
    SELECT Title
    FROM Schedule
```

- Example:
  Find the employees with the highest salary

```
SELECT Name
FROM Employee
WHERE Salary >= ALL
    SELECT Salary
    FROM Employee
```

Nested Queries in FROM Clause

- SQL allows nested queries in the FROM clause
- Example:
  Find directors of movies showing in Hillcrest

```
select  m.director
from movie m,
                    (select title from schedule
                     where theater = 'Hillcrest') t
where m.title = t.title
```

- Note:
  This is syntactic sugar and can be eliminated

Null values in SQL

- Testing if an attribute is null:
  A **is null**, A **is not null**
- Example:
  Find all employees with unknown phone number

  select name **from** employee
  where phone **is null**
- Arithmetic operations involving any null return null
  e.g., if Salary is null, then Salary + 1 evaluates to null
- Comparisons involving null return unknown new truth value
  e.g., if Salary is null, then Salary = 0 evaluates to unknown

Null values in SQL

- Boolean operations must now handle 3 truth values:
  true, false, unknown
- Boolean expressions involving unknown are evaluated using the following truth tables

AND		
true	unknown	unknown
false	unknown	false
unknown	unknown	unknown

NOT	
unknown	unknown

OR		
true	unknown	true
false	unknown	unknown
unknown	unknown	unknown

- WHERE clause conditions evaluating to unknown are treated as false

Null values: Examples

Movie	title	director	actor
	Tango	Berto	Brando
	Psycho	Hitch	Perkins
	Bambi	null	null

Select title
Where dir = 'Hitch'

title
Psycho

Select title
Where dir **<>** 'Hitch'

title
Tango
Bambi

A: yes
B: no

title
Tango

B

Null values: Examples

Movie	title	director	actor
	Tango	Berto	Brando
	Psycho	Hitch	Perkins
	Bambi	null	null

Select title
Where dir = 'null'

title
Bambi

A: yes
B: no

Select title
Where dir **is null**

title
Bambi

Anomalies of null semantics

if Salary is null, then:

-- Salary > 0 evaluates to unknown even if the domain is restricted to positive integers in the schema definition

-- Consider the queries

```
select name from employee
where Salary <= 100 OR Salary > 100
```

and

```
select name from employee
```

Are these equivalent? A: yes B: no

These are not equivalent if some salaries are null

62

Null Values and Aggregates

- Total all loan amounts

  select sum (*amount*)
  from *loan*

  Above statement ignores null amounts

  Result is *null* if there is no non-null amount

- All aggregate operations except **count(*)** ignore tuples with null values on the aggregated attributes.

Suppose R has a single attribute A. Are these equivalent?

select count(*) from R

select count(A) from R

A: ~~yes~~ B: no

Null Values and Group-By

- Null group-by attributes are treated like any other value

R	A	B
	2	3
	2	5
	Null	0
	Null	1
	Null	2

```
SELECT A, COUNT(B) AS C
FROM R
GROUP BY A
```

	A	C
	2	2
	Null	3

64

Creating nulls with Outer Joins

- Idea: To avoid losing tuples in natural joins, pad with null values
- P <outer join> Q
- natural left outer join:
  keep all tuples from left relation (P)
- natural right outer join:
  keep all tuples from right relation (Q)
- natural full outer join:
  keep all tuples from both relations

Creating nulls with Outer Joins

- Combines tuples from two tables by matching on common attributes

movie	title	director	actor
	Tango	Berto	Brando
	Sky	Berto	Winger
	Psycho	Hitchcock	Hopkins

schedule	theater	title
	Hillcrest	Tango
	Paloma	Tango
	Paloma	Bambi
	Ken	Psycho

movie **natural left outer join** schedule

title	director	actor	theater
Tango	Berto	Brando	Hillcrest
Tango	Berto	Brando	Paloma
Psycho	Hitchcock	Hopkins	Ken
Sky	Berto	Winger	*null*

(Inner) Natural Join

- Combines tuples from two tables by matching on common attributes

movie	title	director	actor
	Tango	Berto	Brando
	Sky	Berto	Winger
	Psycho	Hitchcock	Perkins

schedule	theater	title
	Hillcrest	Tango
	Paloma	Tango
	Paloma	Bambi
	Ken	Psycho

movie **natural join** schedule	title	director	actor	theater
	Tango	Berto	Brando	Hillcrest
	Tango	Berto	Brando	Paloma
	Psycho	Hitchcock	Perkins	Ken

Creating nulls with Outer Joins

- Combines tuples from two tables by matching on common attributes

movie	title	director	actor
	Tango	Berto	Brando
	Sky	Berto	Winger
	Psycho	Hitchcock	Hopkins

schedule	theater	title
	Hillcrest	Tango
	Paloma	Tango
	Paloma	Bambi
	Ken	Psycho

movie **natural left outer join** schedule	title	director	actor	theater
	Tango	Berto	Brando	Hillcrest
	Tango	Berto	Brando	Paloma
	Psycho	Hitchcock	Hopkins	Ken
	Sky	Berto	Winger	*null*

Creating nulls with Outer Joins

- Combines tuples from two tables by matching on common attributes

movie	title	director	actor
	Tango	Berto	Brando
	Sky	Berto	Winger
	Psycho	Hitchcock	Hopkins

schedule	theater	title
	Hillcrest	Tango
	Paloma	Tango
	Paloma	Bambi
	Ken	Psycho

movie **natural right outer join** schedule

title	director	actor	theater
Tango	Berto	Brando	Hillcrest
Tango	Berto	Brando	Paloma
Psycho	Hitchcock	Hopkins	Ken
Bambi	*null*	*null*	Paloma

Creating nulls with Outer Joins

- Combines tuples from two tables by matching on common attributes

movie	title	director	actor
	Tango	Berto	Brando
	Sky	Berto	Winger
	Psycho	Hitchcock	Hopkins

schedule	theater	title
	Hillcrest	Tango
	Paloma	Tango
	Paloma	Bambi
	Ken	Psycho

movie **natural full outer join** schedule

title	director	actor	theater
Tango	Berto	Brando	Hillcrest
Tango	Berto	Brando	Paloma
Psycho	Hitchcock	Hopkins	Ken
Bambi	*null*	*null*	Paloma
Sky	Berto	Winger	*null*

Outer Join Example

- Example:
  Find theaters showing only movies by Berto

```
select theater from schedule
where theater not in
      (select theater
       from schedule natural left outer join
            (select title, director from movie where director = 'Berto')
       where director is null)
```

Movie	title	director	actor
	Tango	Berto	Brando
	Sky	Berto	Winger
	Psycho	Hitchcock	Hopkins

schedule	theater	title
	Hillcrest	Tango
	Paloma	Tango
	Paloma	Psycho

Outer Join Example

- Example:
  Find theaters showing only movies by Berto

```
select theater from schedule
where theater not in
      (select theater
       from schedule natural left outer join
            (select title, director from movie where director = 'Berto')
       where director is null)
```

```
select title, director from movie where director = 'Berto'
```

title	director
Tango	Berto
Sky	Berto

Outer Join Example

- Example:
  Find theaters showing only movies by Berto

```
select theater from schedule
where theater not in
      (select theater
       from schedule natural left outer join
            (select title, director from movie where director = 'Berto')
       where director is null)
```

```
select title, director from movie where director = 'Berto'
```

title	director
Tango	Berto
Sky	Berto

Outer Join Example

- Example:
  Find theaters showing only movies by Berto

```
select theater from schedule
where theater not in
      (select theater
       from schedule natural left outer join
            (select title, director from movie where director = 'Berto')
       where director is null)
```

title	director
Tango	Berto
Sky	Berto

schedule	theater	title
	Hillcrest	Tango
	Paloma	Tango
	Paloma	Psycho

Outer Join Example

- Example:
  Find theaters showing only movies by Berto

```
select theater from schedule
where theater not in
      (select theater
       from schedule natural left outer join
            (select title, director from movie where director = 'Berto')
       where director is null)
```

```
schedule natural left outer join  (select title, director from movie
                                   where director = 'Berto')
```

title	director
Tango	Berto
Sky	Berto

schedule	theater	title
	Hillcrest	Tango
	Paloma	Tango
	Paloma	Psycho

Outer Join Example

- Example:
  Find theaters showing only movies by Berto

```
select theater from schedule
where theater not in
      (select theater
       from schedule natural left outer join
            (select title, director from movie where director = 'Berto')
       where director is null)
```

```
schedule natural left outer join  (select title, director from movie
                                   where director = 'Berto')
```

theater	title	director
Hillcrest	Tango	Berto
Paloma	Tango	Berto
Paloma	Psycho	null

Summary of basic SQL Queries

- A query in SQL can consist of up to six clauses, but only the first two, SELECT and FROM, are mandatory.
- The clauses are specified in the following order:

```
SELECT <attribute list>
FROM   <table list>
[WHERE      <condition>]
[GROUP BY <grouping attribute(s)>]
[HAVING     <group condition>]
[ORDER BY <attribute list>]
```

Summary of basic SQL Queries

- The SELECT-clause lists the attributes or functions to be retrieved
- The FROM-clause specifies all relations (or aliases) needed in the query but not those needed in nested queries
- The WHERE-clause specifies the conditions for selection of tuples from the relations specified in the FROM-clause
- GROUP BY specifies grouping attributes
- HAVING specifies a condition for selection of groups
- ORDER BY specifies an order for displaying the result of a query
- A query is evaluated by first applying the WHERE-clause, then GROUP BY and HAVING, and finally the SELECT-clause

SQL Update Language

- Insertions
- Updates
- Deletions

SQL Update Language
Insertions

- Insert tuples
  INSERT INTO R **VALUES** (v1,…,vk);

  e.g. **INSERT INTO** Movie
  VALUES (“Matchpoint”, “Allen”, “Allen”)

- Some values may be left NULL

  e.g. **INSERT INTO** Movie(Title,Director)
  VALUES (“Matchpoint”, “Allen”)

- Can use results of queries for insertion
  INSERT INTO R **SELECT** … **FROM** … **WHERE**

  e.g. **INSERT INTO** BertoMovie
  SELECT * **FROM** Movie
  WHERE Director = “Berto”

SQL Update Language
Deletions

- Delete every tuple that satisfies <cond>
  DELETE FROM R WHERE <cond>

  e.g. Delete all movies that are not currently playing

  DELETE FROM Movie
  WHERE Title **NOT IN SELECT** Title
  FROM Schedule

SQL Update Language

Updates

- Update values of tuples

  Basic form: Update every tuple that satisfies <cond> in the way specified by the SET clause

  UPDATE R
  SET A1=<exp1>, …, Ak=<expk>
  WHERE <cond>

  e.g. Change all “Berto” entries to “Bertolucci”

  UPDATE Movie
  SET Director=“Bertolucci”
  WHERE Director=“Berto”

  e.g. Increase all salaries in the toys dept by 10%

  UPDATE Employee
  SET Salary = 1.1 * Salary
  WHERE Dept = “Toys”

Views, Assertions & Triggers

- Views

  are a mechanism for customizing the database; also used for creating temporary virtual tables

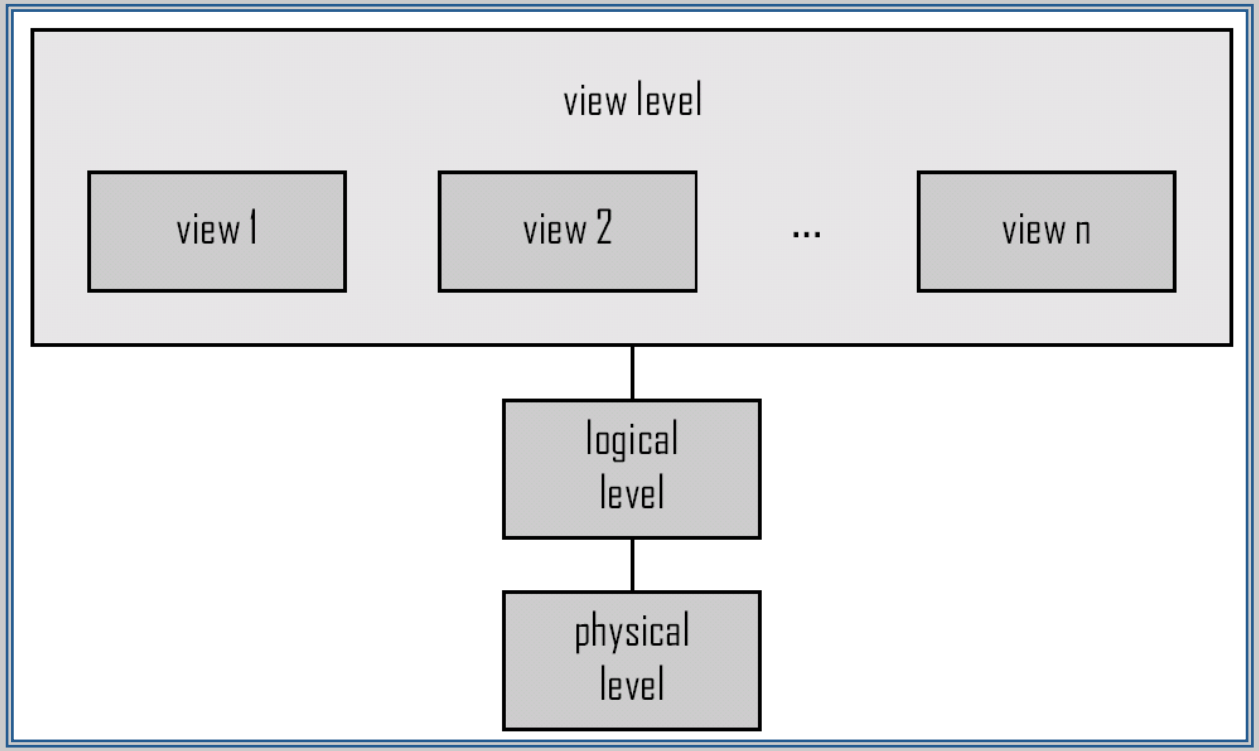
- Assertions

  provide a means to specify additional constraints

- Triggers

  are a special kind of assertions; they define actions to be taken when certain conditions occur

Basic DBMS Architecture



Views

- In some cases, it is not desirable for all users to see the entire logical model (i.e, all the actual relations stored in the database)

  e.g., Consider a person who needs to know customers' loan numbers but has no need to see the loan amounts. This person should see a relation described, in SQL, by

  (**select** customer_name, loan_number
  from customer c, borrower b
  where c.customer_id = b.customer_id)

- A view provides a mechanism to hide or restructure data for certain users.
- Any relation that is not in the database schema but is made visible to a user as a "virtual relation" is called a view.

Bank Relational Schema

- branch = (branch_name, branch_city, assets)
- loan = (loan_number, branch_name, amount)
- account = (account_number, branch_name , balance)
- borrower = (customer_id, loan_number)
- depositor = (customer_id, account_number)
- customer = (customer_id, customer_name)

View Definition

- Syntax

  create view V **as** <query expression>

  where V is the view name and <query expression> is any legal SQL query. A list of attribute names for V is optional.

- Notes
  - Once a view is defined, the view name can be used in queries
  - Only limited updates can be applied to the view (more later)
  - View definition is not the same as creating a new relation by evaluating the query expression: the view contents changes automatically when the database is updated

View Examples

- View:
  A view consisting of bank branches and all their customers

  create view all_customers **as**

  (**select** branch_name, customer_id
  from depositor d, account a
  where d.account_number = a.account_number)

  union

  (**select** branch_name, customer_id
  from borrower b, loan l
  where b.loan_number = l.loan_number)

- Query:
  Find all customers of the La Jolla branch

  select customer_id
  from all_customers
  where branch_name = 'La Jolla'

Views defined using other views

- One view may be used in the expression defining another view
- A view relation V_1 is said to *depend directly* on a view relation V_2 if V_2 is used in the expression defining V_1
- A view relation V_1 is said to *depend on* view relation V_2 if either V_1 depends directly to V_2 or there is a path of dependencies from V_1 to V_2
- A view relation V is said to be *recursive* if it depends on itself → will discuss later…

Views can simplify complex queries

- Example:
  Find actors playing in every movie by “Berto”

```
SELECT Actor FROM Movie
WHERE Actor NOT IN
    (SELECT m1.Actor
    FROM Movie m1, Movie m2,
    WHERE m2.Director=“Berto”
    AND m1.Actor NOT IN
            (SELECT Actor
            FROM Movie
            WHERE Title=m2.Title))
```

The shaded query finds actors NOT playing is some movie by “Berto”

Views can simplify complex queries

- Same query using views:

```
CREATE VIEW Berto-Movies AS
SELECT title FROM Movie
WHERE director = "Bertolucci"
```

```
CREATE VIEW Not-All-Berto AS
SELECT m.actor FROM Movies m, Berto-Movies
WHERE  Berto-Movies.title NOT IN
                (SELECT title FROM Movies
                WHERE actor = m.actor)
```

```
SELECT actor FROM Movies
WHERE actor NOT IN
      (SELECT * FROM Not-All-Berto)
```

Another syntax: WITH clause

```
WITH Berto-Movies AS
SELECT title FROM Movie
WHERE director = "Bertolucci"
```

```
WITH Not-All-Berto AS
SELECT m.actor FROM Movies m, Berto-Movies
WHERE  Berto-Movies.title NOT IN
                (SELECT title FROM Movies
                WHERE actor = m.actor)
```

```
SELECT actor FROM Movies
WHERE actor NOT IN
      (SELECT * FROM Not-All-Berto)
```

Note: Berto-Movies and Not-All-Berto are temporary tables, not views

Efficient view implementation

- Materialized views:
  Physically create and maintain a view table

  Assumption: other queries on the view will follow

  Concerns: maintaining correspondence between the base table and the view when the base table is updated

  Strategy: incremental update

94

Efficient view implementation

- Virtual views:

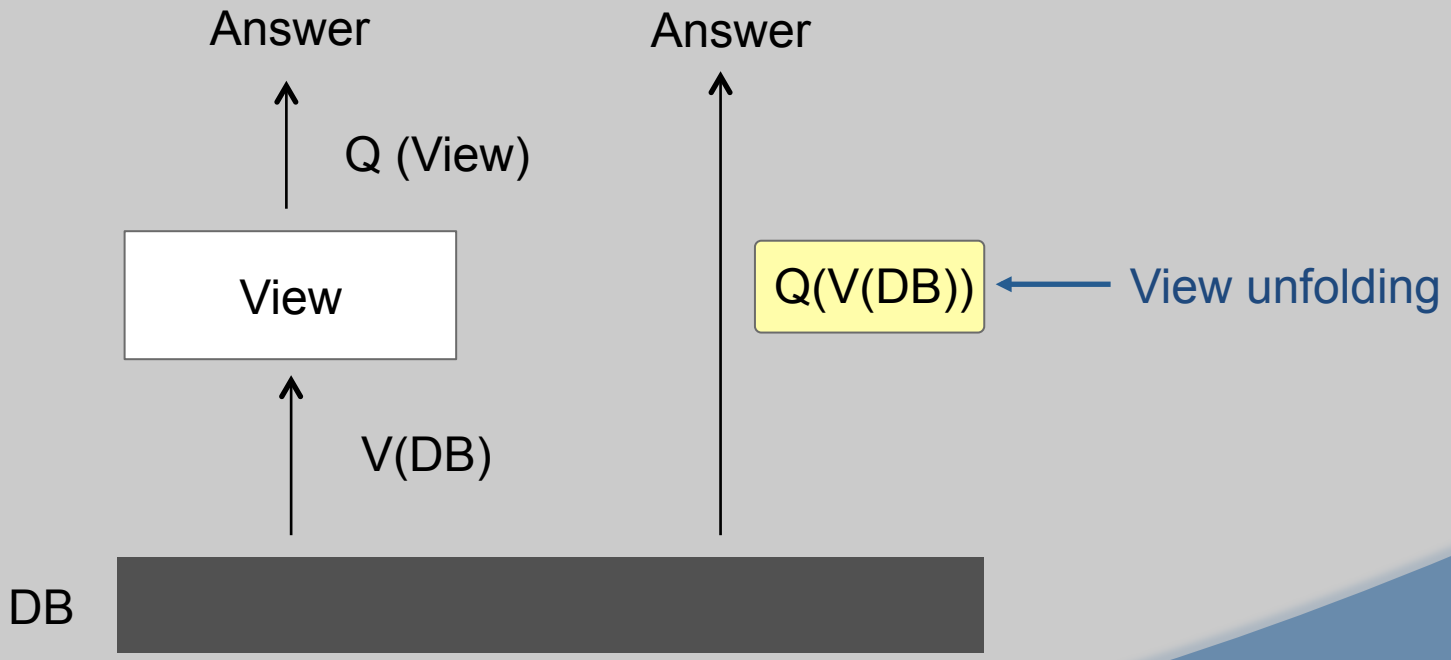
  Never physically created: Answer queries on the view by reformulating it as a query on the underlying base tables (by replacing the views by their definitions)

  Disadvantage: Inefficient for views defined via complex queries (especially if additional queries are to be applied to the view within a short time period)

  Advantage: No need to maintain correspondence with base tables

Query answering in the presence of virtual views

- View unfolding



Example of view unfolding:

```
CREATE VIEW Berto-Movies  AS
 SELECT title FROM Movie WHERE director = "Berto" ;
```

View

```
SELECT theater FROM schedule WHERE title IN
        (SELECT * FROM Berto-Movies)
```

Query

```
SELECT theater FROM schedule WHERE title IN
        (SELECT title FROM Movie WHERE director = "Berto" )
```

Example of View Unfolding

Database:

Patient	pid	hospital	docid

Doctor	docid	docname

View *(Scripps doctors)*:

```
create view ScrippsDoc as
select  d1.* from Doctor d1, Patient p1
where p.hospital = 'Scripps' and p.docid = d.docid
```

View *(Scripps patients)*:

```
create view ScrippsPatient as
select  p2.* from Patient p2
where hospital = 'Scripps'
```

Scripps Query (using views):

```
select p.pid, d.docname
from ScrippsPatient p, ScrippsDoc d
where p.docid = d.docid
```

Example of View Unfolding

query using view

```
select p.pid, d.docname
from ScrippsPatient p, ScrippsDoc d
where p.docid = d.docid
```

view1

```
create view ScrippsDoc as
select  d1.* from Doctor d1, Patient p1
where p1.hospital = 'Scripps' and p1.docid = d1.docid
```

view2

```
create view ScrippsPatient as
select  p2.* from Patient p2
where p2.hospital = 'Scripps'
```

result of view unfolding

```
select p.pid, d.docname
from Patient p, Doctor d, Patient p1
where p.docid = d.docid and p.hospital = 'Scripps'
and p1.hospital = 'Scripps' and p1.docid = d.docid
```

View Updates

- Example

  Consider a view of all loan data in the *loan* relation, hiding the *amount* attribute

  create view branch_loan **as**
  select branch_name, loan_number
  from loan

  Add a new tuple to *branch_loan*

  insert into branch_loan
  values (‘L-307’, ‘La Jolla’,)

  This insertion leads to the insertion of the tuple

  (‘L-307’, ‘La Jolla’, null)

  into the *loan* relation

View Updates

- Update on views without aggregates, group-by, or tuple aliases, defined on a single base table, maps naturally to an update of the underlying base table
- For other views, mapping updates to base tables is not always possible
- Most SQL implementations allow updates only on simple views (without aggregates, group-by or tuple aliases) defined on a single base table

View Update Example

```
create view Berto-titles as
select title from movie where director = 'Bertolucci'
```

Delete a title T in view

→ delete all tuples with title T from movie

Insert a title T in view

→ insert <T, 'Bertolucci', NULL> in movie

Update "Sky" to "Sheltering Sky" in view

→ update movie

set title = 'Sheltering Sky'

where director = 'Bertolucci' and title = 'Sky'

View Update Example

```
create view Same as
select t.theater, s.theater
from schedule t, schedule s
where t.title = s.title
```

Same contains pairs of theaters showing the same title

- Suppose I insert <Ken, Hillcrest> in Same
  Problem: Cannot be mapped to an update of movie because the common title is unknown
- Similar problem for deletes and updates
- Such view updates are prohibited

Assertions

- An assertion defines a constraint the database must satisfy
- Syntax

  An assertion in SQL takes the form
  create assertion <assertion-name> **check** <predicate>
- When an assertion is made, the system tests it for validity, and tests it again on every update that may violate the assertion

  Testing may introduce a significant amount of overhead; hence assertions should be used with great care.
- Asserting

  for all X, P(X)

  is achieved in a round-about fashion using

  not exists X such that not P(X)

Using General Assertions

- Specify a query that violates the condition
  include inside a NOT EXISTS clause
- Query result must be empty
  if the query result is not empty, the assertion has been violated

Assertion Example

- Example

  Every loan has at least one borrower who maintains an account with a minimum balance or $1000.00

```
create assertion balance_constraint check (not exists
(select * from loan
  where not exists
        (select *
              from borrower, depositor, account
              where loan.loan_number = borrower.loan_number
                  and borrower.customer_id = depositor.customer_id
                  and depositor.account_number = account.account_number
                  and account.balance >= 1000.00)))
```

Assertion Example

- Example

  The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.

```
create assertion sum_constraint check
    (not exists (select *
                 from branch
                 where (select sum(amount )
                        from loan
                        where loan.branch_name =
                              branch.branch_name )
                    >= (select sum (amount )
                        from account
                        where account.branch_name =
                              branch.branch_name )))
```

Assertion Example

- Example

  The salary of an employee must not be greater than the salary of the manager of the department that the employee works for

```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS
   (SELECT *
    FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
    WHERE     E.SALARY > M.SALARY
          AND  E.DNO=D.NUMBER
          AND  D.MGRSSN=M.SSN))
```

SQL Triggers

- Objective

  Monitor a database and take action when a condition occurs

- Syntax

  Triggers are expressed in a syntax similar to assertions and include the following:

  - event (e.g., an update operation)
  - condition
  - action (to be taken when the condition is satisfied)

SQL Triggers: Example

- Example

  A trigger to compare an employee's salary to his/her supervisor during insert or update operations:

```
CREATE TRIGGER INFORM_SUPERVISOR
BEFORE INSERT OR UPDATE OF
        SALARY, SUPERVISOR_SSN ON EMPLOYEE
FOR EACH ROW
   WHEN (NEW.SALARY >
            (SELECT SALARY FROM EMPLOYEE
             WHERE SSN=NEW.SUPERVISOR_SSN))
   INSERT INTO INFORM_SUPERVISOR VALUES
      (NEW.SUPERVISOR_SSN, SSN);
```

SQL Triggers

- Many variations in syntax, functionality
- Many triggering semantics possible:
  before/after event, immediate/deferred execution, etc.
- Behavior can be hard to anticipate
  sometimes results in non-terminating computations!
- Sub-area of databases: “Active databases”

A safe form of trigger: Cascade

- Enforces referential integrity
- Example

```
create table account
(account_number char(10),
branch_name      char(15),
balance          integer,
primary key      (account_number),
foreign key      (branch_name) references branch
                  on delete cascade,
                  on update cascade)
```

Semantics of "on delete cascade": if a tuple deletion in branch causes a violation of referential integrity for some tuple t in account, the tuple t is also deleted

A safe form of trigger: Cascade

- Enforces referential integrity
- Example

```
create table account
(account_number char(10),
branch_name     char(15),
balance         integer,
primary key     (account_number),
foreign key     (branch_name) references branch
                 on delete cascade,
                 on update cascade)
```

Semantics of "on update cascade": if an update of the primary key in branch causes a violation of referential integrity for some tuple t in account, the tuple t.branch_name is also updated to the new value