# Chapter 8 View

related to text book chapter9 (version 7)

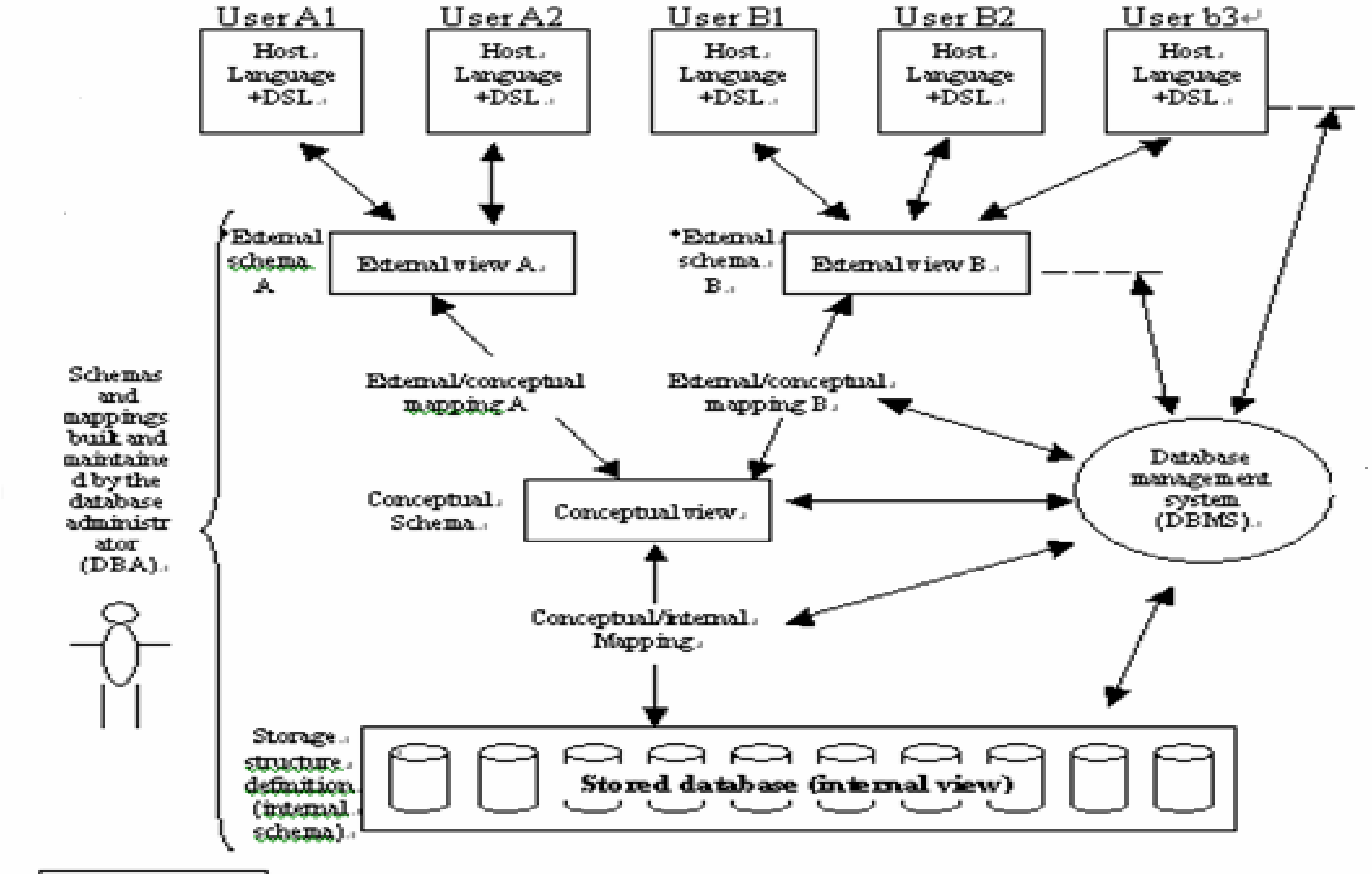
related to text book chapter 10 (version 8)

# CONTENTS

- Introduction
- What are views for
- View retrievals
- View updates
- Snapshots
- SQL facilities

# Introduction

- View is External model


User A1
Host Language +DSL
User A2
Host Language +DSL
User B1
Host Language +DSL
User B2
Host Language +DSL
User b3
Host Language +DSL
External schema A
External view A
External schema B
External view B
External/conceptual mapping A
External/conceptual mapping B
Conceptual Schema
Conceptual view
Database management system (DBMS)
Conceptual/internal Mapping
Schemas and mappings built and maintained by the database administrator (DBA)
Storage structure definition (internal schema)
Stored database (internal view)

# Introduction-cont.

- View is a named expression of the relational algebra

- View defining expression
- Derived (virtual) relvar

# Introduction-cont.

- Definition

```
VAR <relvar name>
    VIEW <relational expression>
          [<candidate key definition list>];
```

# GOOD_SUPPLIER

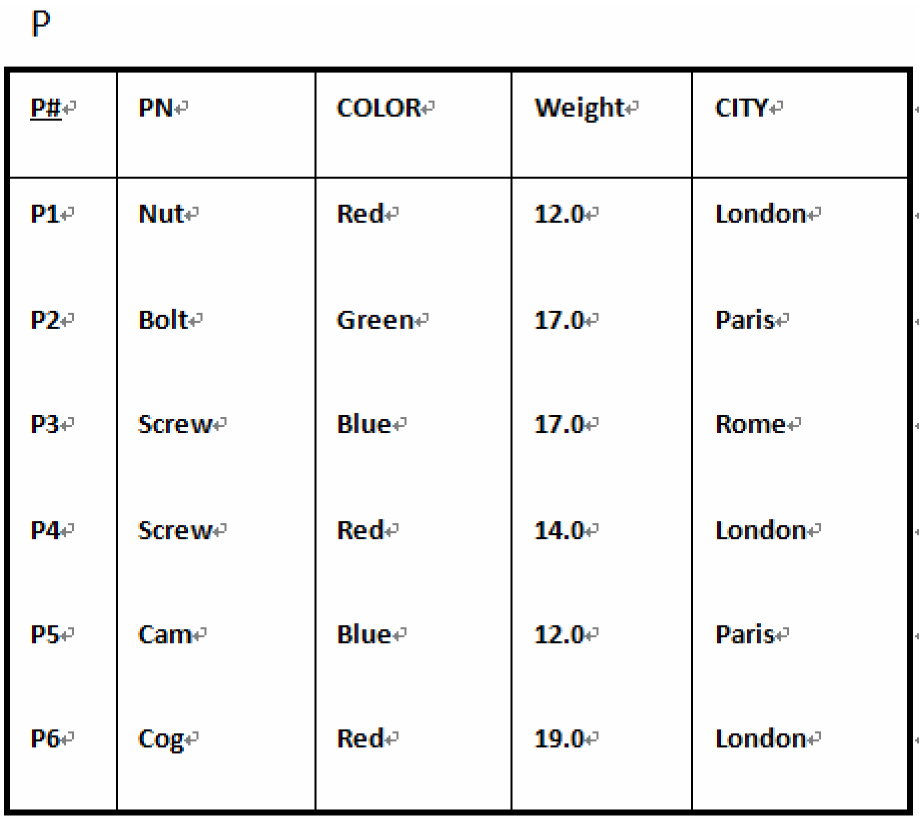
| S# | SNAME | STATUS | CITY |
|---|---|---|---|
| S1 | Smith | 20 | London |
| S2 | **Jones** | 10 | Paris |
| S3 | Blake | 30 | Paris |
| S4 | Clark | 20 | London |
| S5 | Adams | 30 | Athens |

GOOD_SUPPLIER as a view of base relvar S
(except red parts)

```
VAR good_supplier VIEW
        (S WHERE status > 15){s#, status, city}
```

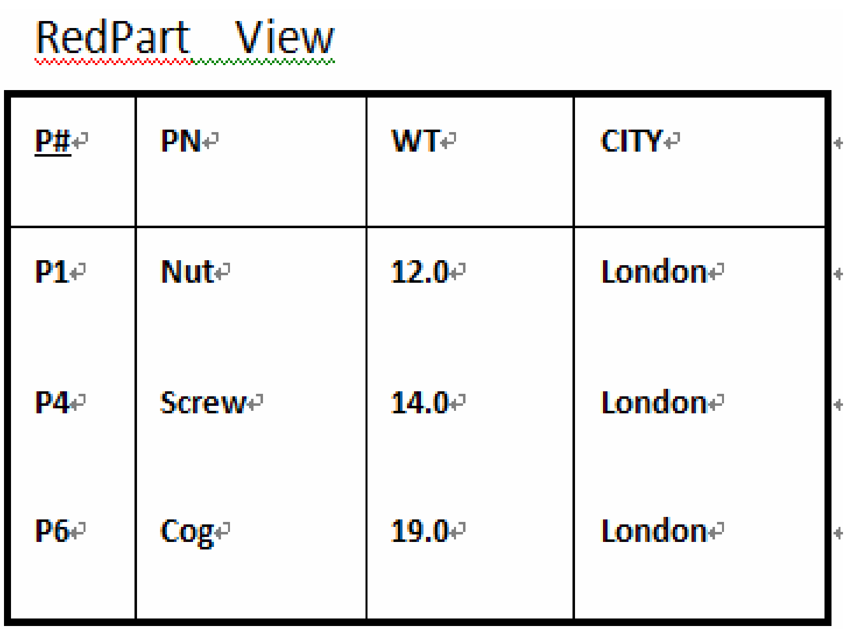
# Examples

- VAR redpart VIEW
  (( P WHERE color = color(‘red’))
  {ALL BUT color} RENAME weight AS wt;



P

| P# | PN | COLOR | Weight | CITY |
|---|---|---|---|---|
| P1 | Nut | Red | 12.0 | London |
| P2 | Bolt | Green | 17.0 | Paris |
| P3 | Screw | Blue | 17.0 | Rome |
| P4 | Screw | Red | 14.0 | London |
| P5 | Cam | Blue | 12.0 | Paris |
| P6 | Cog | Red | 19.0 | London |



RedPart View

| P# | PN | WT | CITY |
|---|---|---|---|
| P1 | Nut | 12.0 | London |
| P4 | Screw | 14.0 | London |
| P6 | Cog | 19.0 | London |

# What Are Views For

- Provide automatic security for hidden data
- Provide a shorthand or “macro” capability

```
   (city_pair WHERE scity = ‘London’ ){pcity}
```

without view:

```
  ((( S RENAME city AS scity )
      JOIN SP
      JOIN  ( P RENAME city AS pcity ))
           WHERE scity = ‘London’ ) {pcity}
```

# What Are Views For – cont.

- Allow the same data to be seen by different users in different ways at the same time
- Provide logical data independence

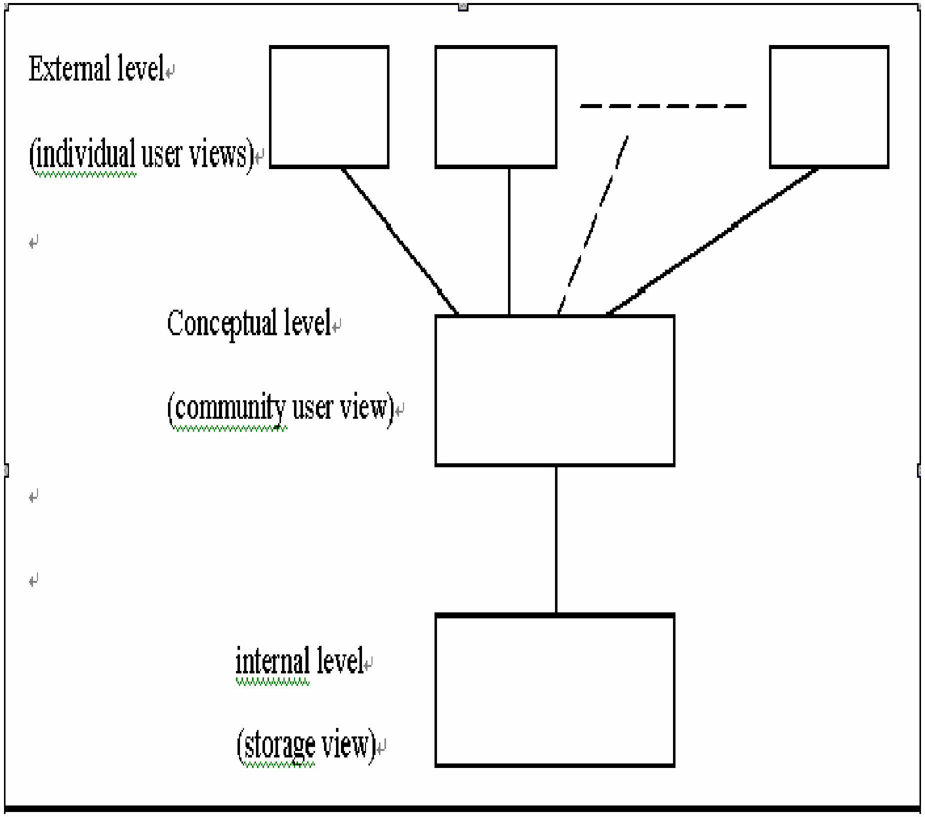
# What Are Views For – cont.

- View definition combines the external schema function and the external/ conceptual mapping function

# The three levels of the architecture

## Logical independence

When conceptual schema has changed the external schema needn’t to change, and so the user program needn’t changed too.



# Logical Independency

- Growth
  - expansion of an existing base relvar to include a new attribute (DISCOUNT attribute for supplier)
  - inclusion of a new base relvar (Project to SPJ DB)

# Logical Independency-cont.

- Restructuring

If Replace relvar S with the two relvar as follows:

VAR snc BASE RELATION {s# s#, sname

name, city char} PRIMARY KEY {s#}

VAR st BASE RELATION

{s# s#, status integer}

PRIMARY KEY {s#}

the S is: VAR S VIEW snc JOIN st;

# Logical Independency-cont.

- Two principles
  - Interchangeability

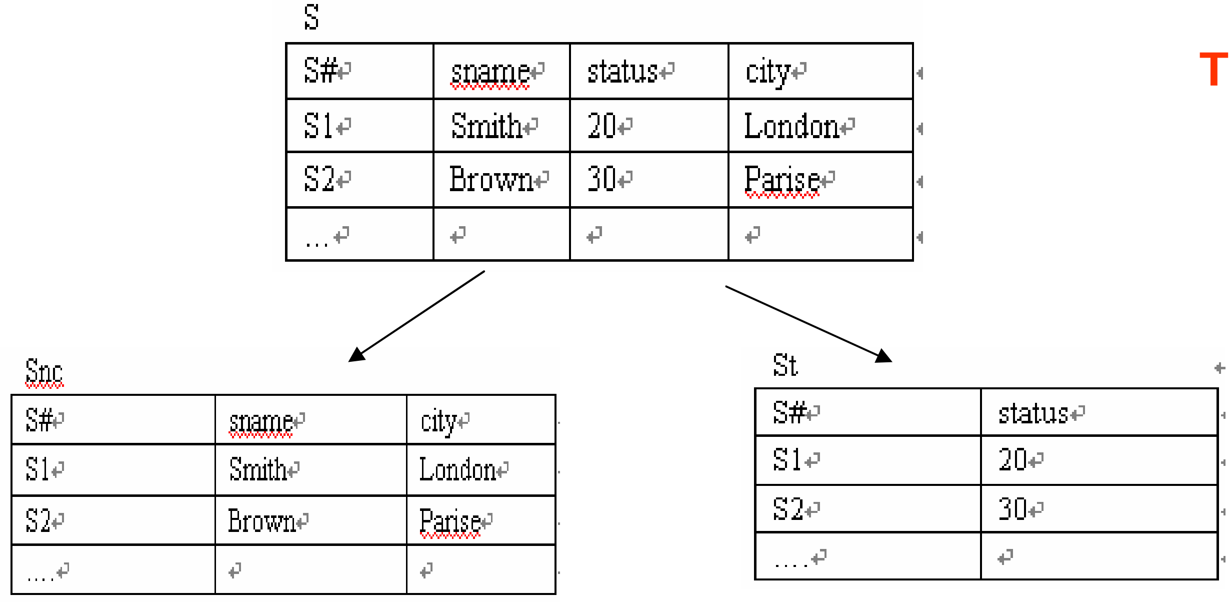
    there must be no arbitrary and unnecessary distinctions between base and derived relvar.

    S vs snc and st
  - Database relativity

    ‘real’ database

    ‘expressible’ database

Table



S

| S# | sname | status | city |
|---|---|---|---|
| S1 | Smith | 20 | London |
| S2 | Brown | 30 | Parise |
| … | | | |

Snc

| S# | sname | city |
|---|---|---|
| S1 | Smith | London |
| S2 | Brown | Parise |
| …. | | |

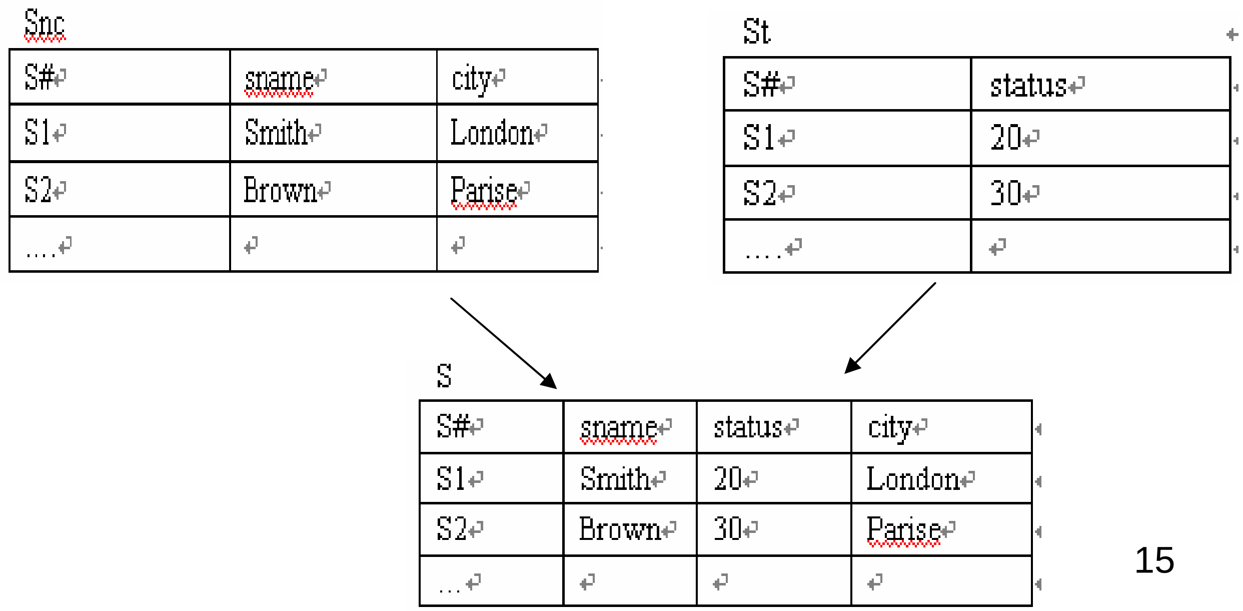
St

| S# | status |
|---|---|
| S1 | 20 |
| S2 | 30 |
| …. | |

View

---

Table



Snc

| S# | sname | city |
|---|---|---|
| S1 | Smith | London |
| S2 | Brown | Parise |
| …. | | |

St

| S# | status |
|---|---|
| S1 | 20 |
| S2 | 30 |
| …. | |

View

S

| S# | sname | status | city |
|---|---|---|---|
| S1 | Smith | 20 | London |
| S2 | Brown | 30 | Parise |
| … | | | |

# View Retrieval

- Defining expression X of view D is some function on database D

  $V = X(D)$

- Retrieval on V is

  $R(V) = R(X(D)) = X'(D)$

  so there no much difference for retrieval between base relvar and view.

# View  Retrieval-cont.

- Materializing

  X (D) has a copy of the relation

- Substitution

  X ( D ) hasn’t materialized copy, retrieval V will changed to retrieval D directly

# View Updates

- Let U be an update operation on V, then

$$U ( V ) = U ( X ( D ) )$$

translate to

$$U ( X (D) ) = X ( U' (D) )$$

So, View Update is more complicated will explained next

# View Updates-cont.

- Principles satisfied by any view updating problem
  - View updatability is a semantic issue, not a syntactic one

```
VAR V VIEW S WHERE status > 25
                   OR city = ‘Paris’;
VAR V VIEW ( S WHERE status > 25 )
        UNION ( S WHERE city = ‘Paris’ );
```

the view V’s updatability must be equal don’t care their syntax

# View Updates – cont.

- Must work correctly in the special case when the “view” is in fact a base relvar

  updating on B and

  updating on V = B UNION B has same result

- updating rule must preserve symmetry

  DELETE V = A INTERSECT B must delete tuple from both A and B.

- etc.

# View Updates – cont.

Is A’ predicate

- UNION V = A UNION B
  - INSERT New tuple must satisfy PA or PB or both

```
VAR uv VIEW
    (S WHERE status > 25)
        UNION (S WHERE city = ‘Paris’)
```

Tuple ( s6, Smith, 50, Rome)

Tuple ( s3, Blake, 30, Paris ) has side effect

UV

| S# | SNAME | STATUS | CITY |
|---|---|---|---|
| S2 | Jone | 10 | Paris |
| S3 | Blake | 30 | Paris |
| S5 | Adams | 30 | Athens |

# View Updates – cont.

SA

| S# | SAME | STATUS | CITY |
|---|---|---|---|
| S3 | Blake | 30 | Paris |
| S5 | Adams | 30 | Athens |

SB

| S# | SAME | STATUS | CITY |
|---|---|---|---|
| S2 | Jones | 10 | Paris |
| S3 | Blake | 30 | Paris |

# View Updates – cont.

- DELETE tuple deleted appears in A, It is deleted from A, If it appears in B, it is deleted from B.
- UPDATE tuple to be updated must be such that the updated version satisfies PA or PB or both.

  Tuple (s5, Adams, 30, Athens) change to

  Tuple (s5, Adams, 15, Paris )

# View Updates – cont.

- INTERSECT

  V = A INTERSECT B

  – INSERT

    new tuple must satisfy both PA and PB.

  – DELETE

    tuple to be deleted is deleted from A . If it still appear in B, it is deleted from B.

  – UPDATE

    tuple to be updated must be such that the updated version satisfies both PA and PB.

# View Updates – cont.

- Difference

  the relvar predicate is (PA) AND NOT ( PB )

- Restrict

  the predicate for V is ( PA ) AND ( P ), the predicate for A is PA.

  VAR LS View (S WHERE city = ‘London’ )

| S# | SAME | STATUS | CITY |
|---|---|---|---|
| S1 | Smith | 20 | London |
| S4 | Clark | 20 | London |

# View Updates – cont.

| | |
|---|---|
| INSERT (s6, Green, 20, London) | success |
| INSERT (**s1**, Green, 20, Paris ) | fail |
| INSERT (s6, Green, 20, Athens) | fail |
| DELETE( s1, smith, 20, London) | success |

# View Updates – cont.

- Project
- Extend
- Join

# SNAPSHOTS

- Derived relvar

  common with view

- Real database
- periodically refresh
- use for some application there need some freeze data

```
VAR p2sc SNAPSHOT
     ( (S JOIN SP) WHERE p# = p#('p2')){s#,city}
      REFRESH EVERY DAY ;
```

# SNAPSHOTS – cont.

- Definition

VAR <relvar name> SNAPSHOT <rel exp>

<candidate key definition list>

REFRESH EVERY <now and then>

<now and then> may be:

MONTH, WEEK, DAY, HOUR, n MINUTES, MONDAY, WEEKDAY, ……

# SQL Facility

- To create a view :

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

  where:

  – <query expression> is any legal expression

  – The view name is represented by *v*

# Example

```
create view good_supplier as
 (select s.s#, s.status,s.city
  from  S
  where s.status>15;

 Create view redpart
     AS select p#, pname, weight AS wt, city
          from P
          where color = ‘red’;
```

# Example – cont.

```
Create view PQ
 AS select p#, SUM(qty) AS totqty
        from SP
        group by p#;

 create view city_pair
   AS select distinct s.city scity, p.city pcity
        from S, SP, P
        where s.s#=sp.s# and sp.p# = p.p#
```

# Example – cont.

```
Create view
     dept_summary ( name, minsal, maxsal,
                                avgsal)
   AS    select dname,  min(sal),  max(sal),
                     ave(sal)
         from   EMP, DEPT
         where  dept.d# = emp.d#
         group by  dname;
```

# View Retrieval

- Find all weight great than 20 pounds part from redpart view.

```
select *
from  redpart
where wt > 20;
```

# View Retrieval – cont.

- it will changed to operate on base relation:

```
select p#, pname, weight, city
 from P
 where color = ‘red’ and weight > 20;
```

# View Retrieval – cont.

- Find all department that its employee has average salary 2000.

```
select name from dept_summary
 where avesal = 2000;
```

# View Retrieval – cont.

- It change to operate on base relation:

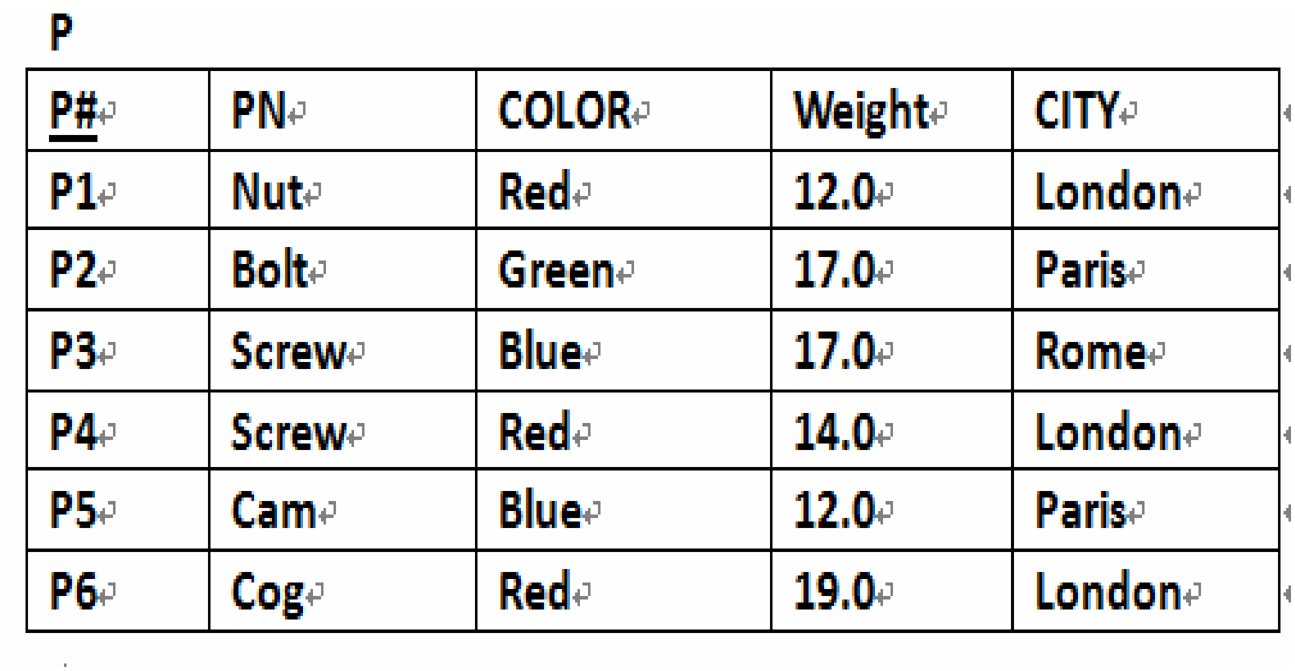
```
select  dname
 from emp , dept
 where emp.d# = dept.d#
 group by dname
 having  avg(sal) = 2000;
```

# Update of a View

- simple View  vs complex view
  - Simple view must satisfy
    - do not contain key word  JOIN, UNION, INTERSECT, EXCEPT
    - do not contain key word DISTINCT
    - SELECT clause only contain single column name

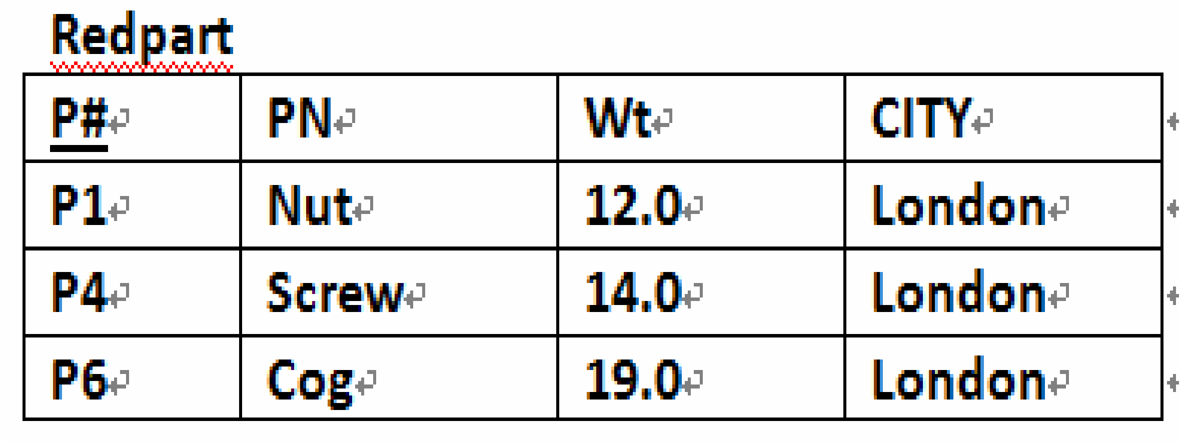
# Update of a View-cont.

- *sub query clause cann’t reference the same table with out query*
- *there are no GROUP BY clause*
- *etc.*



P

| P# | PN | COLOR | Weight | CITY |
|---|---|---|---|---|
| P1 | Nut | Red | 12.0 | London |
| P2 | Bolt | Green | 17.0 | Paris |
| P3 | Screw | Blue | 17.0 | Rome |
| P4 | Screw | Red | 14.0 | London |
| P5 | Cam | Blue | 12.0 | Paris |
| P6 | Cog | Red | 19.0 | London |



Redpart

| P# | PN | Wt | CITY |
|---|---|---|---|
| P1 | Nut | 12.0 | London |
| P4 | Screw | 14.0 | London |
| P6 | Cog | 19.0 | London |

# Update of a View – cont.

- Add a new tuple to *redpart*

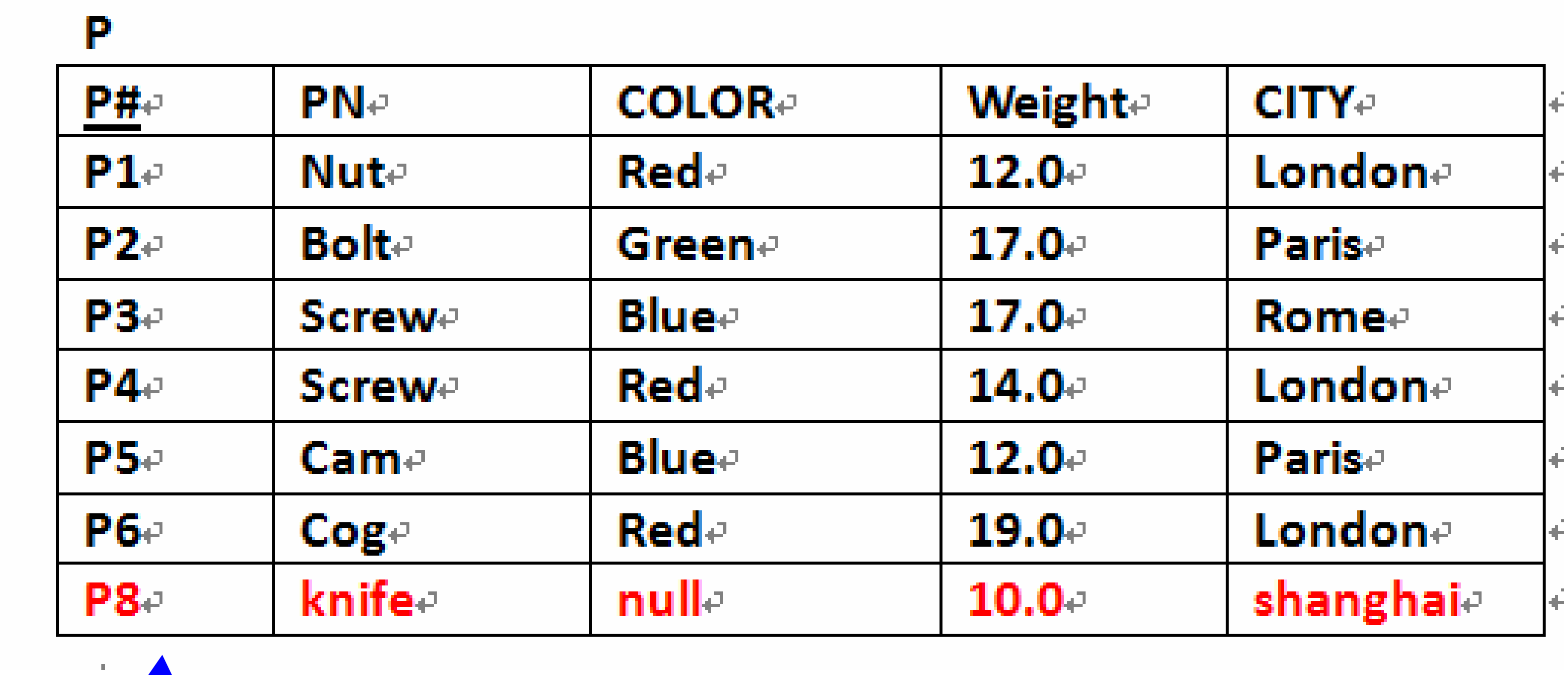
  **insert into** *redpart*

  **values** (p8, knife, 10, ‘shanghai’)

  This insertion must be represented by the insertion of the tuple

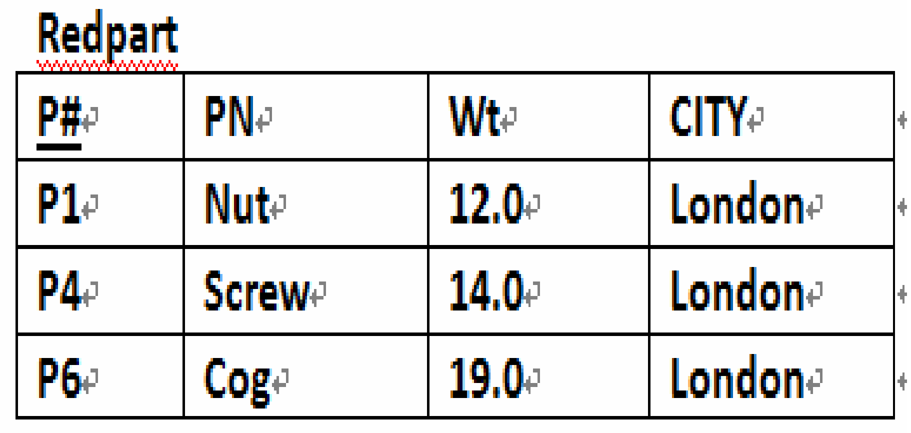
  (p8, knife, null, 10, ‘shanghai’)

  into the *P* relation



P

| P# | PN | COLOR | Weight | CITY |
|---|---|---|---|---|
| P1 | Nut | Red | 12.0 | London |
| P2 | Bolt | Green | 17.0 | Paris |
| P3 | Screw | Blue | 17.0 | Rome |
| P4 | Screw | Red | 14.0 | London |
| P5 | Cam | Blue | 12.0 | Paris |
| P6 | Cog | Red | 19.0 | London |
| P8 | knife | null | 10.0 | shanghai |



Redpart

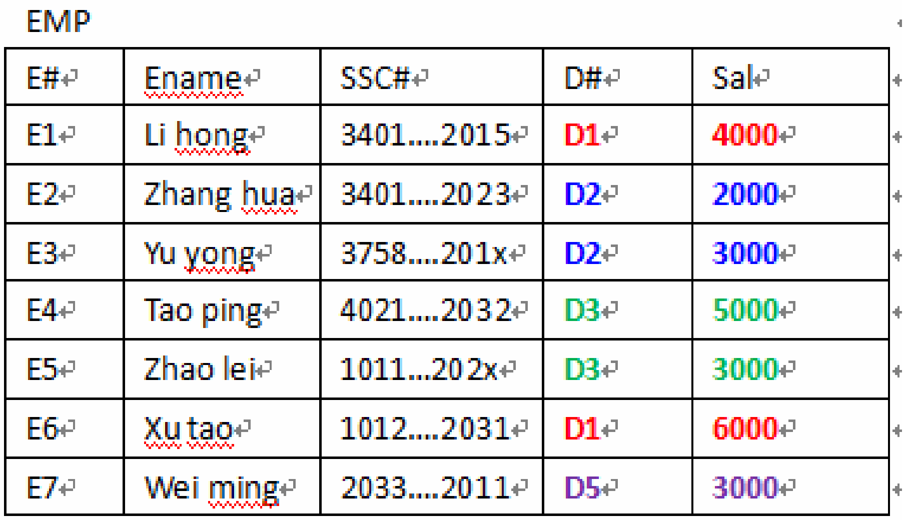
| P# | PN | Wt | CITY |
|---|---|---|---|
| P1 | Nut | 12.0 | London |
| P4 | Screw | 14.0 | London |
| P6 | Cog | 19.0 | London |

Insert Into…..

# Update of a View – cont.

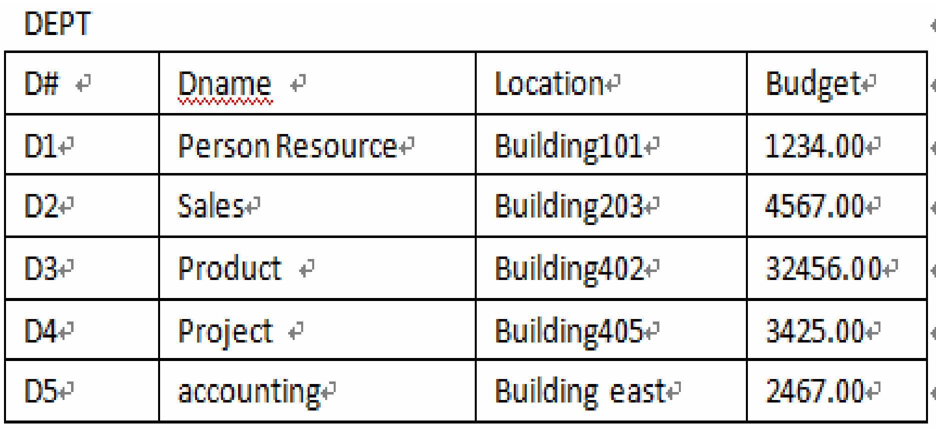
- Updates on more complex views are difficult or impossible to translate, and hence are disallowed.

```
Update dept_summary
  set  avgsal = avgsal*1.1
  where avgsal < 4000;
```



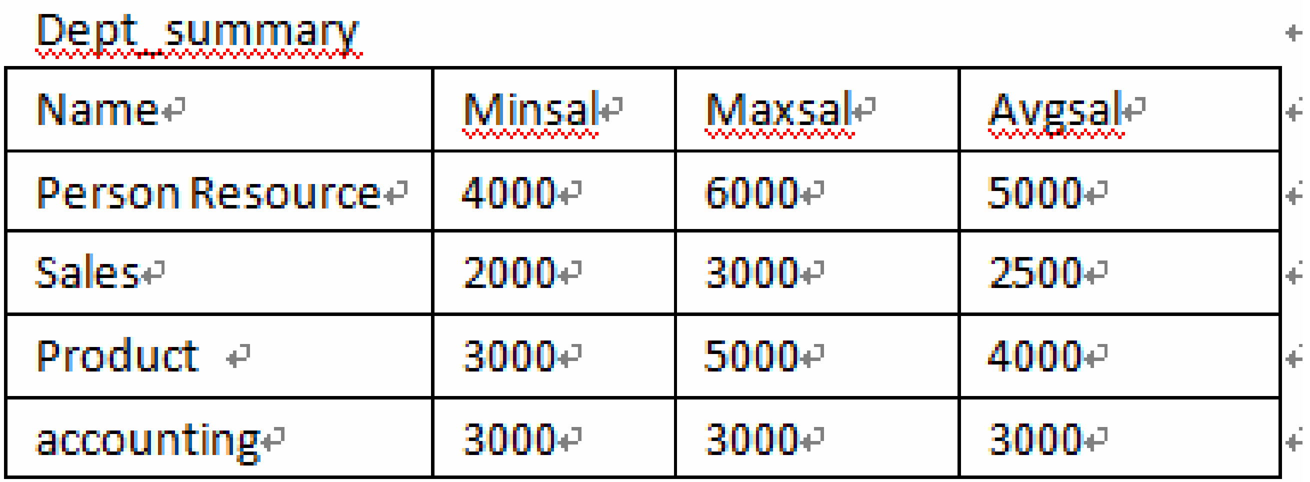
EMP

| E# | Ename | SSC# | D# | Sal |
|---|---|---|---|---|
| E1 | Li hong | 3401....2015 | D1 | 4000 |
| E2 | Zhang hua | 3401....2023 | D2 | 2000 |
| E3 | Yu yong | 3758....201x | D2 | 3000 |
| E4 | Tao ping | 4021....2032 | D3 | 5000 |
| E5 | Zhao lei | 1011...202x | D3 | 3000 |
| E6 | Xu tao | 1012....2031 | D1 | 6000 |
| E7 | Wei ming | 2033....2011 | D5 | 3000 |



DEPT

| D# | Dname | Location | Budget |
|---|---|---|---|
| D1 | Person Resource | Building101 | 1234.00 |
| D2 | Sales | Building203 | 4567.00 |
| D3 | Product | Building402 | 32456.00 |
| D4 | Project | Building405 | 3425.00 |
| D5 | accounting | Building east | 2467.00 |



Dept_summary

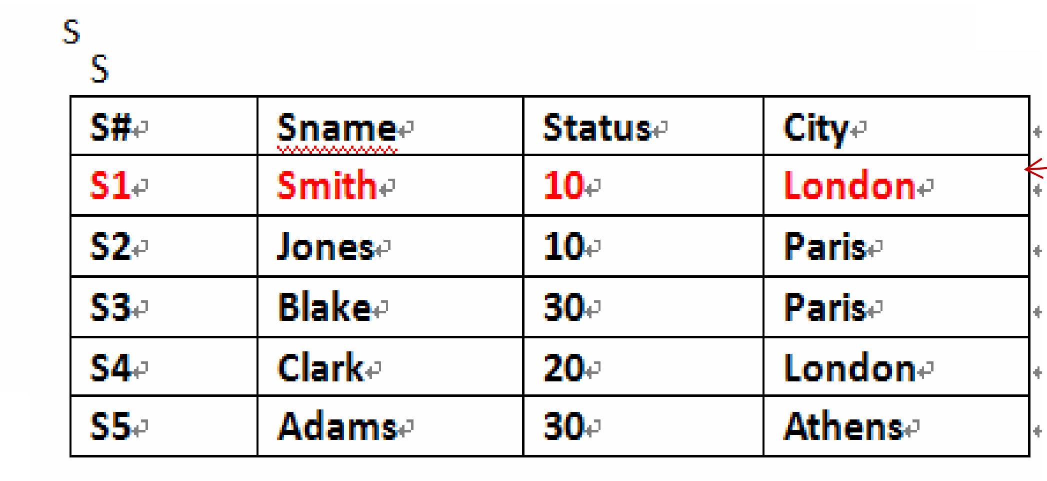
| Name | Minsal | Maxsal | Avgsal |
|---|---|---|---|
| Person Resource | 4000 | 6000 | 5000 |
| Sales | 2000 | 3000 | 2500 |
| Product | 3000 | 5000 | 4000 |
| accounting | 3000 | 3000 | 3000 |

# Update of a View – cont.

- Most SQL implementations allow updates only on simple views (without aggregates) defined on a single relation
- even though there still have problems

```
Update  good_supplier
 set   status = 10
 where s# = ‘s1’;
```

suppose we have know s1 is in good_supplier.

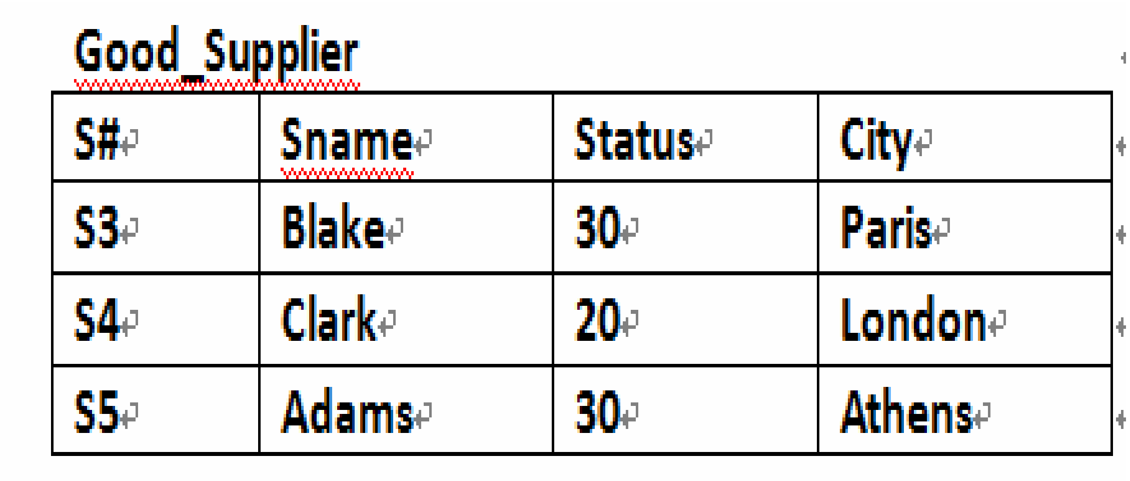


S

S

| S# | Sname | Status | City |
|---|---|---|---|
| S1 | Smith | 10 | London |
| S2 | Jones | 10 | Paris |
| S3 | Blake | 30 | Paris |
| S4 | Clark | 20 | London |
| S5 | Adams | 30 | Athens |

View Update ……..



Good_Supplier

| S# | Sname | Status | City |
|---|---|---|---|
| S3 | Blake | 30 | Paris |
| S4 | Clark | 20 | London |
| S5 | Adams | 30 | Athens |

**S1 not satisfy View definition now**

# Update of a View – cont.

- SQL use ‘with check option’ for updated view definition

```
create view good_supplier as
     select s.s#, s.status,s.city
     from  S
     where s.status>15
      with check option;
```

- Now

```
Update  good_supplier
    set   status = 10
    where s# = ‘s1’;
```

The operation is fail now

# Update of a View – cont.

```
Create view redpart
        AS select p#, pname, weight AS wt, city
              from P
              where color = ‘red’
              with check option;
```

# Update of a View – cont.

```
create view EMP-DATA
    AS  select e#, ename, job, mgr, sal, d#
          from EMP
          where sal between 1000 and 2000
           AND  mgr in
                   ( select distinct e# from EMP)
           AND  d# in ( select d# from DEPT)
           with check option;
```

# Update of a View – cont.

```
Create view EMP_DETAILS
  AS   select e#, ename, job, d#
       from EMP
       where ename = user
       AND tochar(sysdate, ‘HH’) between 9 and 17
       AND tochar(sysdate, ‘D’) between 2 and 6
       with check option;
```

- exercises
- version 7

  9.2, 9.5

  9.6 use SQL language
- Version 8

  10.2, 10.4, 10.5