VIEWS

- A virtual relation that is defined from other pre-existing relations. Called the “defining relations” of the view.
- A view supports multiple user perspectives on the database corresponding to different information organizations, avoiding the need for data duplication or information consistency problems.
- Additional motivation: security (privacy concerns, users need only access/modify selected attributes in the data without being able to access the other attributes).
Differences between a Conceptual Database and a View

VIEW

- Employee
- Salary
- Age

CONCEPTUAL DATABASE

- Employee
- Department
- Salary
- Birth date
  ...

Constructable, but not actually present in the database
EXAMPLE: Airline Databases

External/USER
- Reservation Department
- Dispatcher
- Personnel Dept.

Conceptual
- Passengers
- Flights
- Aircraft
- Crews

Physical
- Files, Records
- Bits

High level of Abstraction

Low level of Abstraction

• Low level of Abstraction
• High level of Abstraction
DATA INDEPENDENCE

Logical Data Independence
- Many modifications of conceptual scheme can be made without affecting views
- No Changes in application programs necessary

Physical Data Independence
- Physical schema can be changed without alerting the conceptual level
- Allows for tuning
Analogy from the programming language world

<table>
<thead>
<tr>
<th>View level</th>
<th>Concept Level</th>
<th>Physical Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function f(i)</td>
<td>Integer Array</td>
<td></td>
</tr>
<tr>
<td>$f(i) = \sum_{j=1}^{m} A(i,j)$</td>
<td>$A[1..n,1..m]$</td>
<td></td>
</tr>
</tbody>
</table>

![Diagram of integer array and summation](image)
Views are defined by a query that generates the desired virtual relation from existing relations:

In Relational Algebra:

Create view DEPHEADS As

From Department

<table>
<thead>
<tr>
<th>DNAME</th>
<th>DNO</th>
<th>FNAME</th>
<th>LNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pebody</td>
<td>5</td>
<td>Robin</td>
<td>Wang</td>
</tr>
<tr>
<td>Admin</td>
<td>4</td>
<td>Jenifer</td>
<td>Veallau</td>
</tr>
<tr>
<td>CS</td>
<td>1</td>
<td>James</td>
<td>Borg</td>
</tr>
</tbody>
</table>

From Employee

Views typically include selected projections with optional selects and joins.
CREATE VIEW DEPHEADS AS

SELECT DNAME, DNUMBER, FNAME, LNAME
FROM DEPARTMENT, EMPLOYEE
WHERE DEPARTMENT.MGRSSN=EMPLOYEE.SSN

The SQL statement defining view are typically executed at query time, thus additions/changes in the defining/base relations are reflected in the virtual relation (view) transparently.

Allow Query a views:

SELECT FNAME, LNAME
FROM DEPHEADS
WHERE DNUMBER=5

The information is a virtual relation is always “up to date” (automatically reflect database update)
Complex Views

• Views may include complex calculations

```
CREATE VIEW EMP_AGE(LNAME,AGE) AS
SELECT LNAME,MONTHS-BETWEEN(SYSDATE,BDATE)/12 FROM EMPLOYEE
```

<table>
<thead>
<tr>
<th>LNAME</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SMITH</td>
<td>40.86</td>
</tr>
<tr>
<td>WANG</td>
<td>50.91</td>
</tr>
<tr>
<td>ZELAYA</td>
<td>38.12</td>
</tr>
</tbody>
</table>

• Views creation/definition may contain aggregate operation

```
CREATE VIEW DEPT-INFO(DEPT_NAME,NUM_EMPS,TOTAL_SAL) AS
SELECT DNAME,COUNT(*),SUM(SALARY) FROM DEPARTMENT,EMPLOYEE
WHERE DNUMBER=DNO
GROUP BY DNAME;
```

<table>
<thead>
<tr>
<th>DEPT_NAME</th>
<th>NUM EMPLOYER</th>
<th>TOTAL SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESEARCHER</td>
<td>4</td>
<td>135,000</td>
</tr>
<tr>
<td>ADMIN</td>
<td>3</td>
<td>93,000</td>
</tr>
<tr>
<td>HEADQUATER</td>
<td>1</td>
<td>35,000</td>
</tr>
</tbody>
</table>

Remaining attributes
Built op.
See Chapter 7
Changes to the database via views

**Syntax:**

```
UPDATE <VIEW-NAME>
SET <LIST OF CHANGES TO VIEW ATTRIBUTES>
WHERE <condition based on view attributes>
```

mapped to necessary updates in the defining relation

OK if simple name change

**Example:**

```
UPDATE DEPHEAD
SET DNAME = 'research'
WHERE LNAME = 'Wallace' OR LNAME = 'SMITH'
```

*Rename* all departments manager by ‘Wallace’ or ‘Smith’ to ‘Research’
PROBLEMS WITH VIEWS

Insert into a view based on a join

**Insert into DEPHEADS**

**Values**(‘SALES”,6,’John’,’Wilson’)

---

**DEPARTMENT**

<table>
<thead>
<tr>
<th>DNAME</th>
<th>DNO</th>
<th>MGRSSN</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESEARCH</td>
<td>5</td>
<td>33344455</td>
<td>401</td>
</tr>
<tr>
<td>ADMIN</td>
<td>4</td>
<td>98316738</td>
<td>201</td>
</tr>
<tr>
<td>SALES</td>
<td>6</td>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>

---

**EMPLOYEE**

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
<th>SSN</th>
<th>BDATE</th>
<th>ADDRES</th>
<th>SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>419324</td>
<td>11,July</td>
<td>223333</td>
<td>30000</td>
</tr>
<tr>
<td>James</td>
<td>Bary</td>
<td>123123</td>
<td>10 Nov</td>
<td>111222</td>
<td>55000</td>
</tr>
<tr>
<td>John</td>
<td>Wilson</td>
<td>......</td>
<td>......</td>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>

---

Null values in the fields projected out of the defining relations by the view.

---

**ANOMOLY**

```
SELECT LNAME
FROM DEPHEADS
WHERE DNAME=‘SALES’
```

```
SELECT EMPLOYEE.LNAME
FROM DEPARTMENT,EMPLOYEE
WHERE MGRSSN=SSN
    AND DNAME=‘SALES’
```

Fields null result. Join fail because null join attribute.
Additional Problem with views

\[
\text{UPDATE DEPT\_INFO} \\
\text{SET TOTAL\_SAL = 100000} \\
\text{WHERE DNAME = ‘RESEARCH’}
\]

Problem when view attribute is defined as an aggregate quantity

how can the constraint

\[
\text{sum(salary)=100000}
\]

be realized as an update on the individual salary attributes for dept with > 1 employee
RESTRICTIONS ON VIEWS

to avoid consistency problems

• In general, updates are only allowed when there is only one possible update in the base relation to accomplish the view update.

1). A view with a single defining table is updatable, if
   a) The view attribute contain the primary key and all other “not null” attributes.
      (still problem of nulls in the defining relations)

2). Views defines using
   --- joins
   --- grouping
   --- aggregate functions
generally not updatable

⇒ But generally no restrictions on read-only views
View Implementation Issue

Strategy #1: QUERY MAPPING
Convert query on view to query on base relation
Problem: may be inefficient if the view involves complex calculation like aggregate function.

Strategy #2: VIEW MATERIALIZATION
Create temporary table to reflect the view structure
--- efficient if many queries to few updates
Temporary table must be updated (recomputed) if updates to the defining relations
--- full recomputation costly
--- minimal update difficult to determine
--- goal of avoiding data duplication
### View Implementation Issue

#### Strategy #1: QUERY MAPPING
- Oracle approach
  - Convert query on view to query on base relation
  - Problem: may be inefficient if the view involves complex calculation like aggregate function.

#### Strategy #2: VIEW MATERIALIZATION
- Create temporary table to reflect the view structure
  - --- efficient if many queries to few updates
  - Temporary table must be updated (recomputed) if updates to the defining relations
    - --- full recomputation costly
    - --- minimal update difficult to determine
    - --- fail to avoid relational count of eliminating data duplication

---

(Runtime macro)

Cache