Views

Content from Database Systems-The Complete Book and Database System Concepts. Credit to Dr. Burns and Dr. Widom

Scenario: restricted view

instructor

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000

Problem: some administrators may need CS instructor info, but aren't allowed to see salaries.

```
sqlite> select ID, name, dept_name from instructor;
ID          name        dept_name
10101       Srinivasan  Comp. Sci.
12121       Wu          Finance
15151       Mozart      Music
22222       Einstein    Physics
32343       El Said     History
33456       Gold        Physics
45565       Katz        Comp. Sci.
…
```

Scenario: define Spring 2010 catalog

course

course_id	title	dept_name	credits
BIO-101	Intro. to Biology	Biology	4
BIO-301	Genetics	Biology	4

section

course_id	sec_id	semester	year	building	room_number	time_slot_id
BIO-101	1	Summer	2009	Painter	514	B
BIO-301	1	Summer	2010	Painter	514	A

time_slot

time_slot_id	day	start_hr	start_min	end_hr	end_min
A	M	8	0	8	50
A	W	8	0	8	50

Problem: existing tables aren't convenient for showing the Spring 2010 classes

Exercise: write query to create a new one.

```
sqlite> select course_id, title, credits, day, start_hr, building
   ...>           from course natural join section natural join time_slot
   ...>          where semester="Spring" and year="2010";
course_id   title                       credits     day         start_hr    building
CS-101      Intro. to Computer Science  4           R           14          Packard
CS-101      Intro. to Computer Science  4           T           14          Packard
CS-315      Robotics                    3           F           13          Watson
```

Scenario: summary tables

census

usid	age	workclass	education	occupation
1	39	State_gov	Bachelors	Adm_clerical
2	50	Self_emp_not_inc	Bachelors	Exec_managerial
3	38	Private	HS_grad	Handlers_cleaners
4	53	Private	11th	Handlers_cleaners
5	28	Private	Bachelors	Prof_specialty
6	37	Private	Masters	Exec_managerial
7	49	Private	9th	Other_service

Problem: it would be handy to have a table showing # people in each occupation

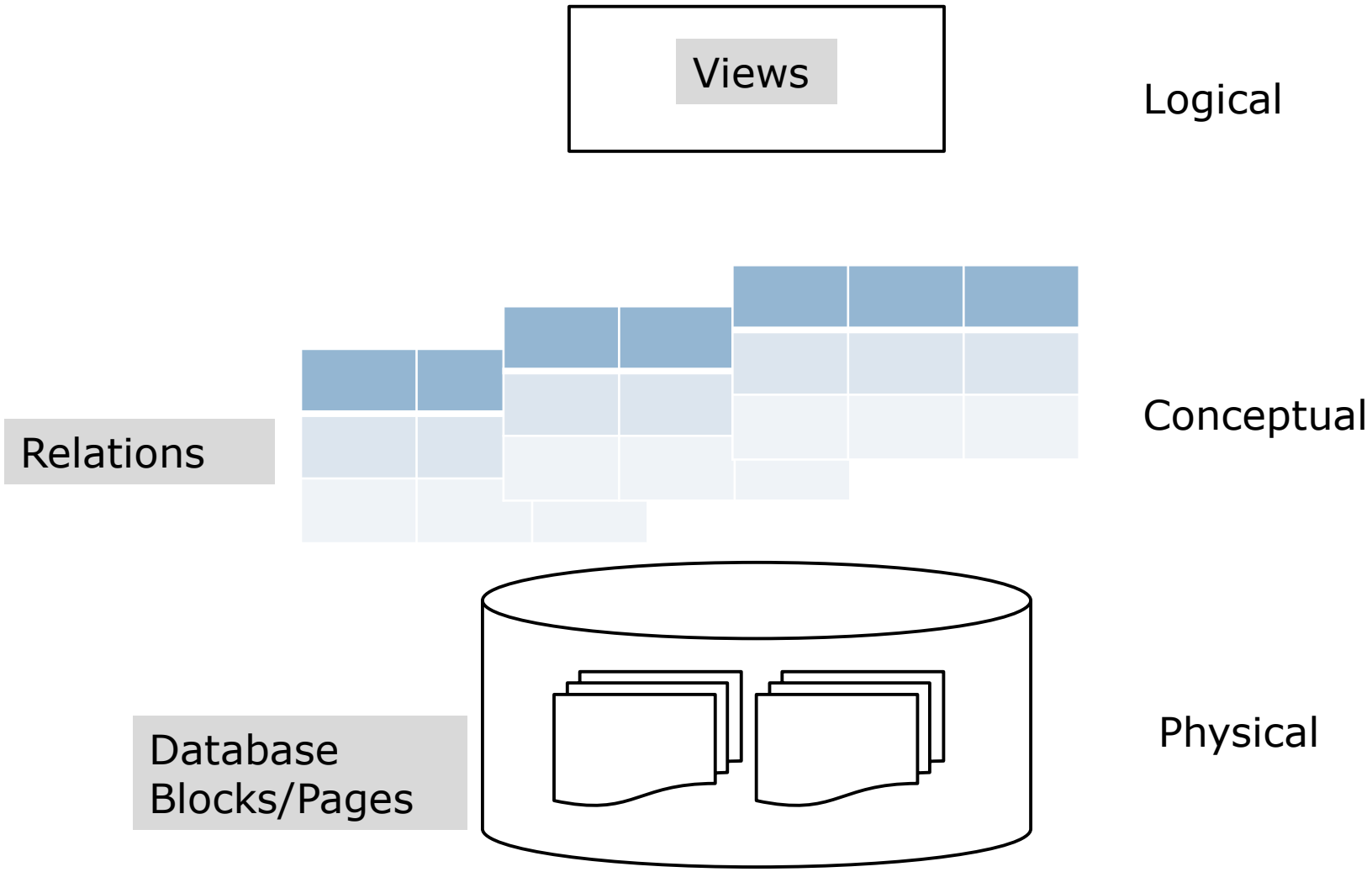
Exercise: write the query to answer this

```
sqlite> select occupation, count(*) as count from census group by occupation;
occupation    count
Adm_clerical  3770
Armed_Forces  9
Craft_repair  4099
Exec_manager  4066
Farming_fish  994
Handlers_cle  1370
Machine_op_i  2002
NA            1843
…
```

Why do we need views?

- Data security - Hide data from other users
- Make some queries easier
- Intermediate tables
- Modularity to database access

Abstraction levels



Database View

- View is just a named query
- Virtual (consists of definition only)
- View V = Query(R1,R2,…,Rn)
- Schema of V is schema of query result

CREATE VIEW viewname AS

<QUERY>

```
CREATE VIEW csfaculty AS
    SELECT id, name, dept_name
      FROM instructor
      WHERE dept_name = ‘Comp. Sci.’;
```

Database Views…

- View can be created on another view
  - View V = Query(R1,R2)
  - View V2 = Query(R1,V)
  - View V3 = Query(V,V2,R2)

```
CREATE VIEW csfaculty AS
    SELECT id, name, dept_name
      FROM instructor
      WHERE dept_name = ‘Comp. Sci.’;
```

```
SELECT course_id, name
  FROM csfaculty, teaches
  WHERE teaches.id = csfaculty.id
  AND year = 2020;
```

View evaluation by DBMS - conceptually

USE CASE 1: Temporary table for Views

```
SELECT course_id, name
  FROM csfaculty, teaches
  WHERE teaches.id = csfaculty.id
  AND year = 2020;
```

View

```
CREATE temporary TABLE T as
    SELECT id, name, dept_name
      FROM instructor
      WHERE dept_name = 'Comp. Sci.';

SELECT course_id, name
  FROM T, teaches
  WHERE teaches.id = T.id
  AND year = 2020;

DROP TABLE T;
```

View evaluation by DBMS…

USE CASE 2: Query re-written to use base tables

```
SELECT course_id, name
  FROM csfaculty, teaches
  WHERE teaches.id = csfaculty.id
  AND year = 2020;
```

```
SELECT course_id, name
  FROM (SELECT id, name, dept_name
        FROM instructor
        WHERE dept_name = 'Comp. Sci.') csfaculty,
       teaches
  WHERE teaches.id = csfaculty.id
  AND year = 2020;
```

View evaluation by DBMS…

USE CASE 3: DMBS re-writes into simpler query

```
SELECT course_id, name
  FROM csfaculty, teaches
  WHERE teaches.id = csfaculty.id
  AND year = 2020;
```

```
SELECT course_id, name
FROM instructor, teaches
WHERE dept_name = ‘Comp. Sci.’
AND teaches.id = instructor.id
AND year = 2020;
```

Updating a view

Can we allow insert/delete/update on the csfaculty view?

Views are not stored!

For example:

```
insert into csfaculty values (“30765”, “Green”, “Music”);
```

Short answer:

- Some views are updatable – it depends on the query
- Some database don’t support modification to views, in that case use Triggers

INSERT into a view

```
insert into csfaculty values (“30765”, “Green”, “Music”);
translated to:
insert into instructor values (“30765”, “Green”, “Music”, NULL);
```

- Default values are used instead of NULL when defined.

SQL Updatable Views - conditions

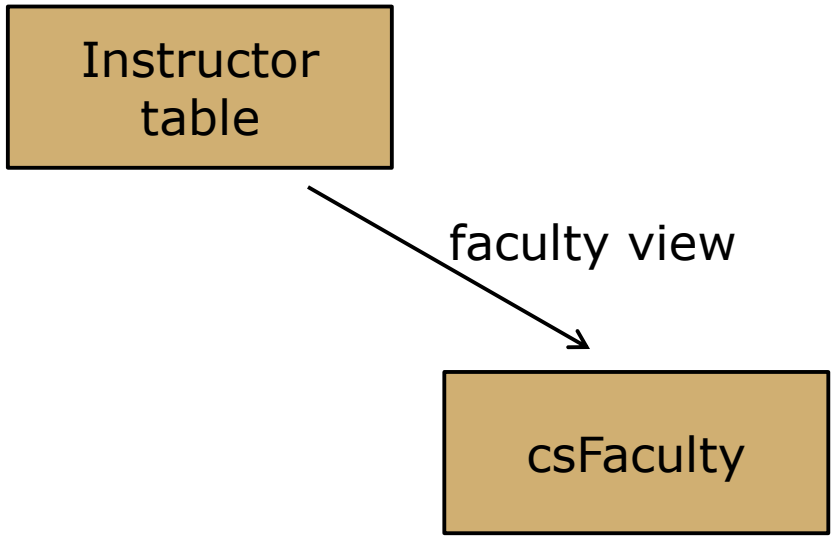
- SELECT on a Single base table
- No DISTINCT
- No aggregations (AVG)
- Attributes of base table not projected in view allowed to be NULL or DEFAULT values
- View with SubQueries must not refer to same base table

Expensive to re-compute view

Maybe we use the ‘csfaculty’ view all the time.

It is expensive to keep re-running the query.

Solution?



Materialized Views

- Materialized MV = VQuery(R1,R2,…,Rn)
- Create table MV with schema of query result
- Execute VQuery and populate table MV with the results
- Queries (SELECT * FROM MV) access table MV without rewriting
  - Performance gain
  - Helpful for workloads that consists of lots of queries but not so many updates

Use Materialized Views all the times?

- Materialized MV = VQuery(R1,R2,…,Rn)
- Materialized MV could be extremely large
  - MV relation saved in database could be large
- Modification to base data invalidate MV
- If R1,R2,...Rn are modified (tuples inserted, deleted, updated) over which MV is defined, then
  - DBMS modify MV stored table based on the changes of base tables
  - Or completely recompute the MV

Materialized View maintenance

instructor

ID	name	dept_name	salary
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000

```
CREATE MATERIALIZED VIEW csfaculty AS
    SELECT id, name, dept_name
      FROM instructor
      WHERE dept_name = 'Comp. Sci.';
```

Eager strategy:

If the instructor table changes, re-run the csfaculty view

Lazy strategy:

If the instructor table changes, don't re-run the csfaculty view until it is needed

Incremental strategy:

Incremental maintenance rather than full recomputation