Views and Indexes

Slides adapted from http://infolab.stanford.edu/~ullman/fcdb.html

1

Views

- A *view* is a relation defined in terms of stored tables (called *base tables*) and other views.
- Two kinds:
  1. *Virtual* = not stored in the database; just a query for constructing the relation.
  2. *Materialized* = actually constructed and stored.

2

Declaring Views

- Declare by:
  CREATE [MATERIALIZED] VIEW <name> AS <query>;
- Default is virtual.

3

Example: View Definition

- CanDrink(drinker, beer) is a view "containing" the drinker-beer pairs such that the drinker frequents at least one bar that serves the beer:

```
CREATE VIEW CanDrink AS
   SELECT drinker, beer
   FROM Frequents, Sells
   WHERE Frequents.bar = Sells.bar;
```

4

Example: Accessing a View

- Query a view as if it were a base table.
  - limited ability to modify views if it makes sense as a modification of one underlying base table.
- Example query:

```
SELECT beer FROM CanDrink
WHERE drinker = 'Sally';
```

5

Insertion on Views

- Generally, it is impossible to modify a virtual view, because it doesn't exist. Insertions are only allowed if:
  - The view is constructed from a single base table while the attributes not in the view can be set to null
  - The view definition does not contain aggregates, expressions and DISTINCT
  - The user has the privilege to modify the base table

6

Materialized Views

- Problem: each time a base table changes, the materialized view may change.
  - Cannot afford to recompute the view with each change.
- Solution: Periodic reconstruction of the materialized view, which is otherwise "out of date."

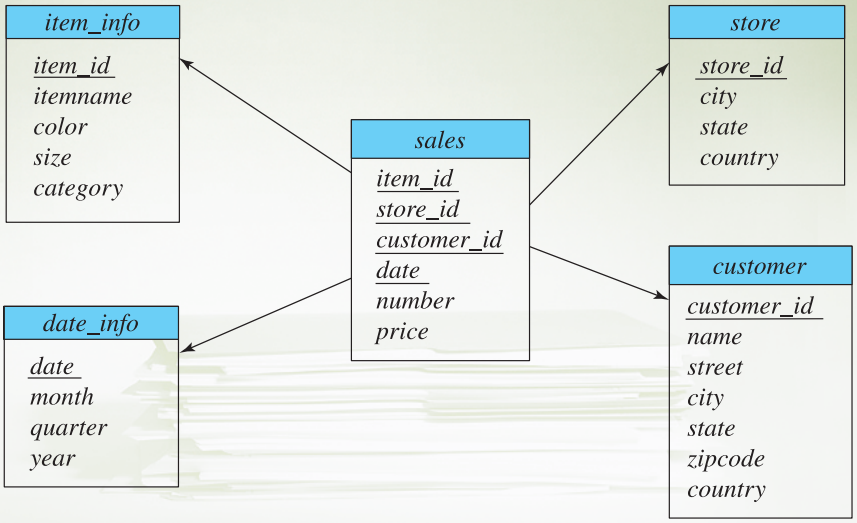
7

Example: A Data Warehouse

- Wal-Mart stores every sale at every store in a database.
- Overnight, the sales for the day are used to update a *data warehouse* = materialized views of the sales.
- The warehouse is used by analysts to predict trends and move goods to where they are selling best.

8

Data Warehouse Schema



sample sales relation

item_name	color	clothes_size	quantity
skirt	dark	small	2
skirt	dark	medium	5
skirt	dark	large	1
skirt	pastel	small	11
skirt	pastel	medium	9
skirt	pastel	large	15
skirt	white	small	2
skirt	white	medium	5
skirt	white	large	3
dress	dark	small	2
dress	dark	medium	6
dress	dark	large	12
dress	pastel	small	4
dress	pastel	medium	3
dress	pastel	large	3
dress	white	small	2
dress	white	medium	3
dress	white	large	0
shirt	dark	small	2
shirt	dark	medium	6
shirt	dark	large	6
shirt	pastel	small	4
shirt	pastel	medium	1
shirt	pastel	large	2
shirt	white	small	17
shirt	white	medium	1
shirt	white	large	10
pant	dark	small	14
pant	dark	medium	6
pant	dark	large	0
pant	pastel	small	1
pant	pastel	medium	0
pant	pastel	large	1
pant	white	small	3
pant	white	medium	0
pant	white	large	2

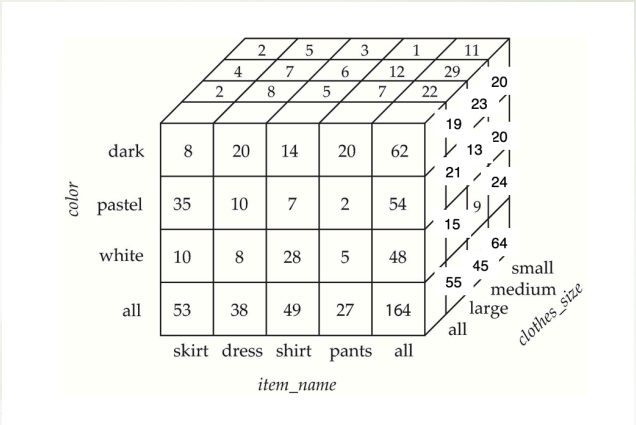
Cross Tabulation of *sales* by *item_name* and *color*

clothes_size **all**

item_name	*color*			
	dark	pastel	white	total
skirt	8	35	10	53
dress	20	10	5	35
shirt	14	7	28	49
pants	20	2	5	27
total	62	54	48	164

Data Cube of Sales

A **data cube** is a multidimensional generalization of a cross-tab

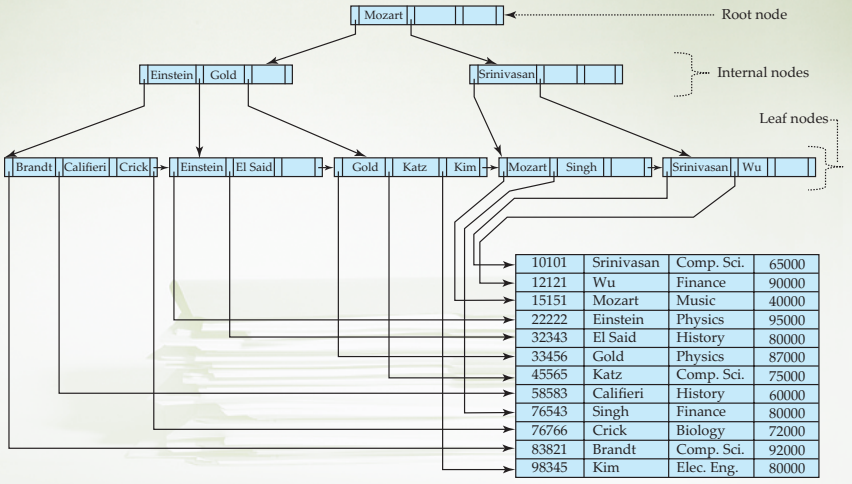


Indexes

- *Index* = data structure used to speed access to tuples of a relation, given values of one or more attributes.
- Could be a hash table, but in a DBMS it is always a balanced search tree with giant nodes (a full disk page) called a *B+-tree*.

13

Example of B+-tree



14

Declaring Indexes

- Typical syntax:

```
CREATE INDEX BeerInd ON Beers(manf);
CREATE INDEX SellInd ON Sells(bar, beer);
```

15

Using Indexes

- Given a value *v*, the index takes us to only those tuples that have *v* in the attribute(s) of the index.
- Example: use BeerInd and SellInd to find the prices of beers manufactured by Pete's and sold by Joe. (next slide)

16

Using Indexes --- (2)

```
SELECT price FROM Beers, Sells
WHERE manf = 'Pete''s' AND
   Beers.name = Sells.beer AND
   bar = 'Joe''s Bar';
```

1. Use BeerInd to get all the beers made by Pete's.
2. Then use SellInd to get prices of those beers, with bar = 'Joe''s Bar'

17

Database Tuning

- A major problem in making a database run fast is deciding which indexes to create.
- Pro: An index speeds up queries that can use it.
- Con: An index slows down all modifications on its relation because the index must be modified too.

18

Example: Tuning

- Suppose the only things we did with our beers database was:
  1. Insert new facts into a relation (10%).
  2. Find the price of a given beer at a given bar (90%).
- Then SellInd on Sells(bar, beer) would be wonderful, but BeerInd on Beers(manf) would be harmful.

19

Tuning Advisor

- Tuning advisor gets a *query load*, e.g.:
  - Choose random queries from the history of queries run on the database, or programmer provides a sample workload.
- Tuning advisor generates candidate indexes and evaluates each index on the workload.
  - Feed each sample query to the query optimizer, which assumes only this one index is available.
  - Measure the improvement/degradation in the average running time of the queries.

20