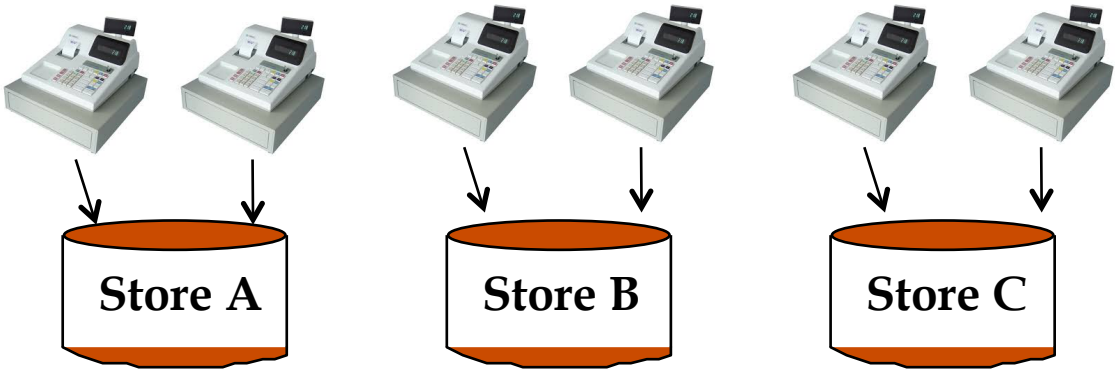
Data Warehouses

Yanlei Diao and
Michaël Thomazo

Slides Courtesy of R. Ramakrishnan and J. Gehrke

Introduction

- In the late 80s and early 90s, companies began to use their DBMSs for *complex, interactive, exploratory analysis* of *historical data*.



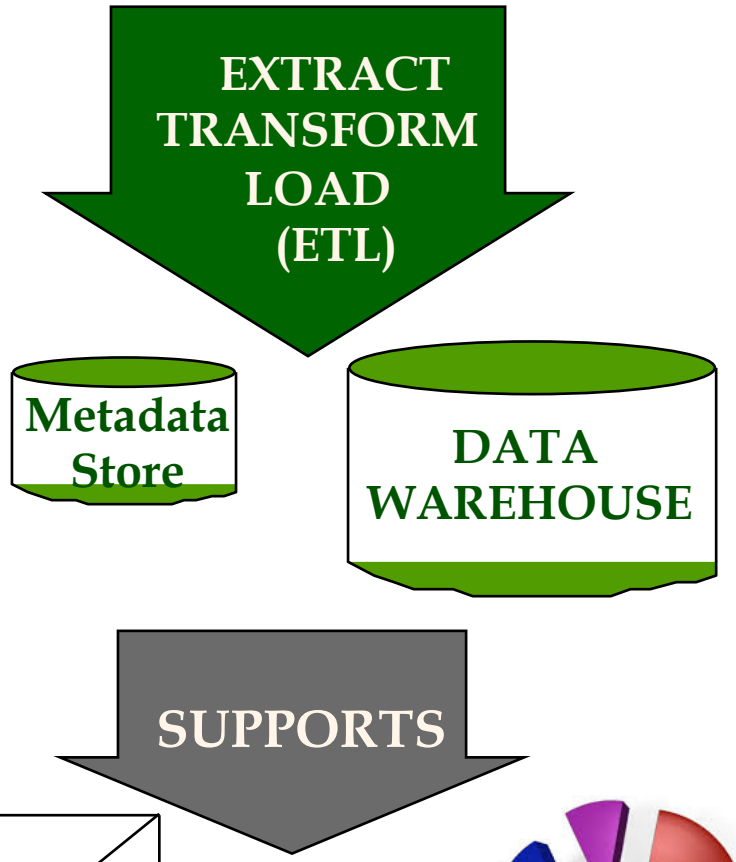
Operational Data
(purchase transactions: store, customer, products, sales, etc.)

Decision Making:

- how much of which products to order for which stores?
- when to deliver the products?
- benefits of promotional offers?

Data Warehousing

- Integrated data spanning long time periods, often augmented with summary information.
- Large volumes: several terabytes to petabytes common.
- Interactive response times expected for complex queries; ad-hoc updates uncommon.



Data Warehouses (Chaudhuri & Dayal 97)

- A **data warehouse** is an organization-wide data repository, used for decision making
  - An integrated enterprise warehouse collects info about all subjects, e.g. customers, products, sales, assets, personnel.
  - The data is used to assist in decision making
    - e.g., how much of which products to order for which stores, when to deliver the products, the benefits of various promotional offers, etc.
  - Also called **On-Line Analytic Processing** (**OLAP**).
  - OLAP tasks slowed down the normal operation of the company, called **On-Line Transaction Processing** (**OLTP**), hence leading to the separation of Data Warehouses from from operational Databases.

OLTP vs OLAP

OLTP / Operational / Production	**OLAP/ Data Warehouse / DSS**
Operate the business / Clerks	Diagnose the business / Managers
Short queries, small amts of data	Large queries, large amts of data
Current data	Current and historical data
Queries change data	Queries are mostly read-only
Examples: customer inquiry, order entry	OLAP, statistics, visualization, data mining, etc.
Legacy applications, heterogeneous databases	Opposite
Often distributed	Often integrated and centralized (Warehouse)

OLTP vs OLAP (cont'd)

Operational	Warehouse
General E-R Diagrams	Multidimensional data model
Locks necessary	No Locks necessary
Crash recovery required	Crash recovery optional
Smaller volume of data	Huge volume of data
Indexes designed to access small amounts of data	Indexes designed to access large volumes of data

Overview of Topics

1. Introduction
   - Operational vs. Warehouse

2. Multidimensional Data
   - Multidimensional Design
   - Star/Snowflake Schemas
   - OLAP Queries
   - CUBE Operator
   - MOLAP vs ROLAP

- Implementation Issues
  - Bitmap Index
- Constructing a Data Warehouse (ETL)
- Views
  - Materialized View Example
  - Materialized View is an Index
  - Issues in Materialized Views
  - Maintaining Materialized Views

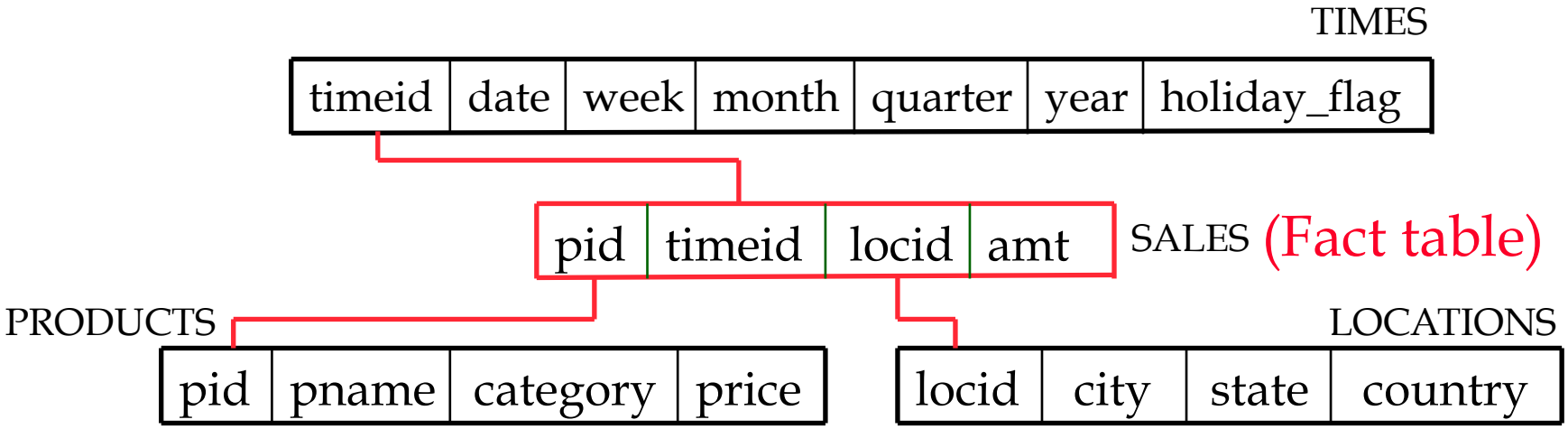
Multidimensional Data (Ch 25.2)

- To support OLAP, warehouse data is often structured multidimensionally, as measures and dimensions.
  - Measure: numeric attribute, e.g. sales amount
  - Dimension: attribute categorizing the measure, e.g. product, store, date of sale.
- Star schema:
  - The central fact table contains a foreign key for each dimension, plus an attribute for each measure.
  - There will also be a dimension table for each dimension.

Examples of Multi-Dimensional Data

- Purchase(ProductID, StoreID, DateID, Amt)
  - Product(ID, SKU, size, brand)
  - Store(ID, Address, Sales District, Region, Manager)
  - Date (ID, Week, Month, Holiday, Promotion)
- Claims(ProvID, MembID, ProcedureID, DateID, Cost)
  - Providers(ID, Practice, Address, ZIP, City, State)
  - Members(ID, Contract, Name, Address)
  - Procedure (ID, Name, Type)
- Telecomm (CustID, SalesRepID, ServiceID, DateID)
  - SalesRep(ID, Address, Sales District, Region, Manager)
  - Service(ID, Name, Category)

Example Multidimensional Design



- This kind of schema is very common in OLAP applications
- It is called a star schema
- Is this a good design of the schema?

Star/Snowflake Schemas

- Why normalize?
  - Space
  - Redundancy, anomalies
- Why unnormalize?
  - Performance
- Which is more important in Data Warehouses?
- If normalized, it is a snowflake schema

Star/Snowflake Schemas

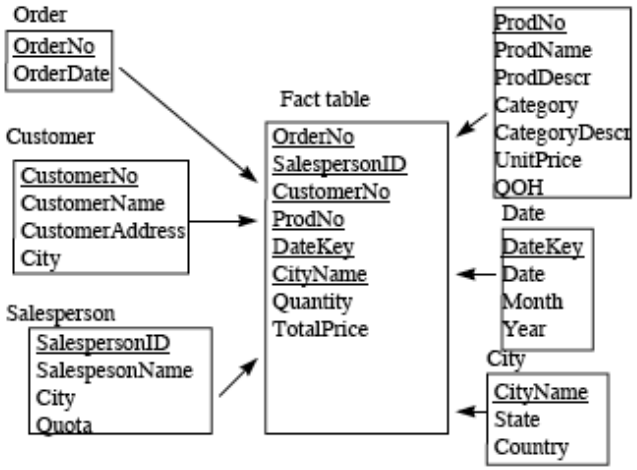


Figure 3. A Star Schema.

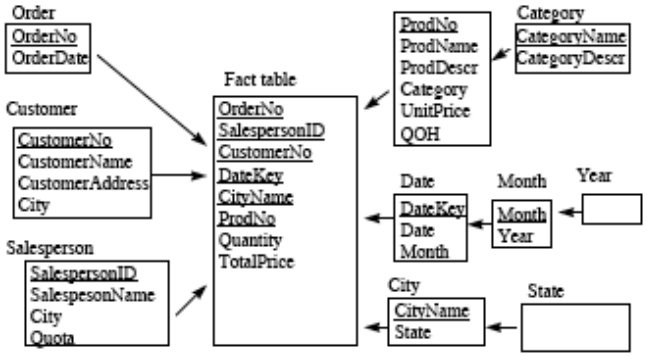


Figure 4. A Snowflake Schema.

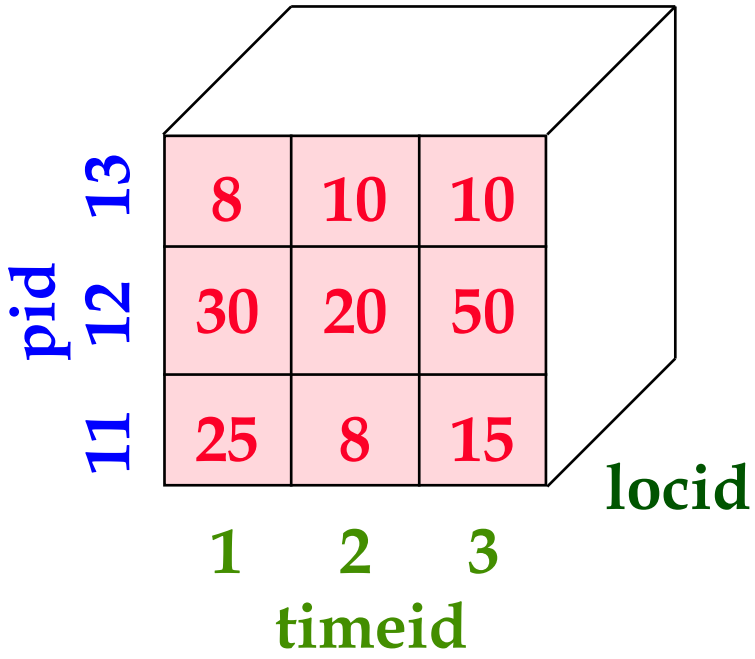
MOLAP vs ROLAP

- Multidimensional data can be stored physically in a (disk-resident, persistent) array; called MOLAP systems. Alternatively, can store as a relation; called ROLAP systems.
- The main relation, which relates dimensions to a measure, is called the fact table. Each dimension can have additional attributes and an associated dimension table.
  - E.g., **Products(pid, locid, timeid, amt)**
  - Fact tables are *much* larger than dimensional tables.

Multidimensional Data Model

- Collection of numeric measures, which depend on a set of dimensions.
  - E.g., measure **Amt**, dimensions **Product** (key: pid), **Location** (locid), and **Time** (timeid).

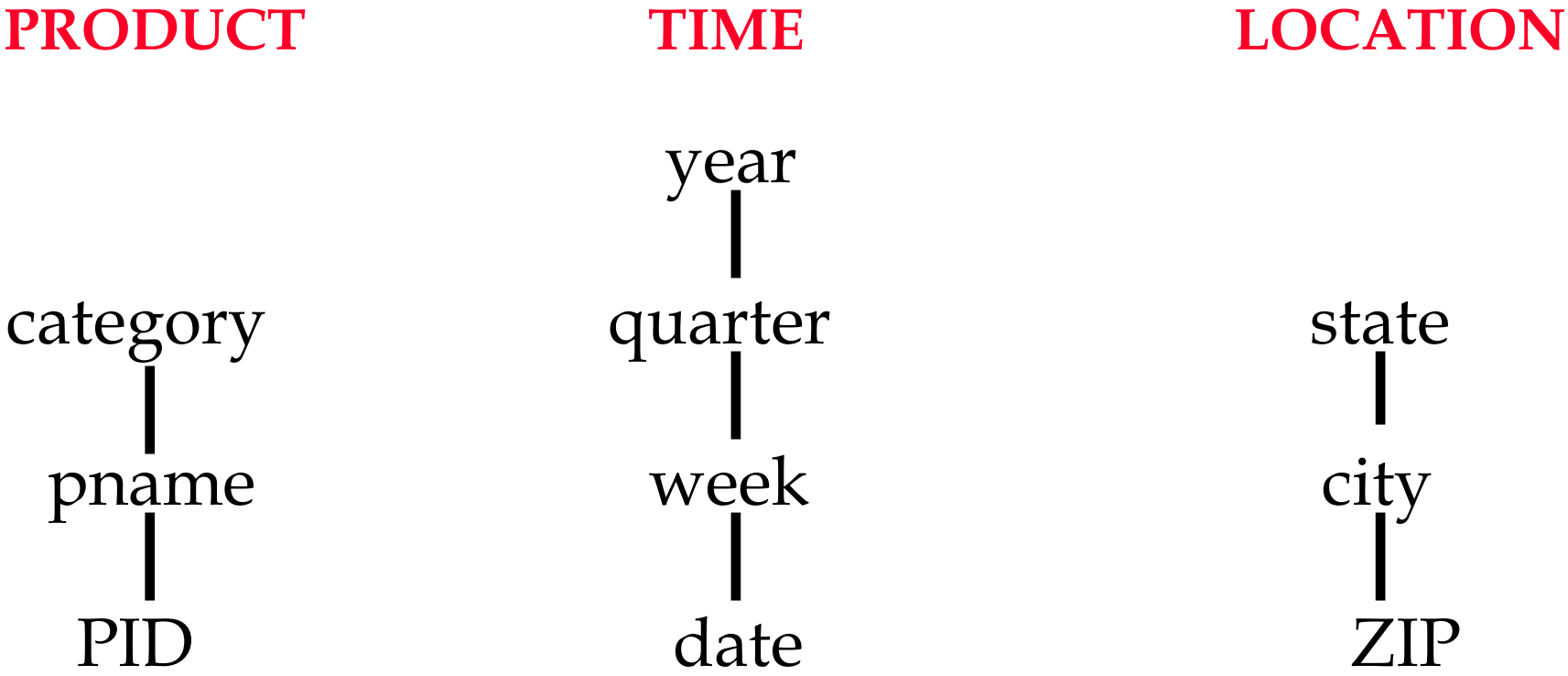
Slice locid=1 is shown:



pid	timeid	locid	amt
11	1	1	25
11	2	1	8
11	3	1	15
12	1	1	30
12	2	1	20
12	3	1	50
13	1	1	8
13	2	1	10
13	3	1	10
11	1	2	35

Dimension Hierarchies

- For each dimension, some of the attributes may be organized in a hierarchy:



OLAP Queries (Ch 25.3)

- Influenced by SQL and by spreadsheets.
- A common operation is to aggregate a measure over one or more dimensions.
  - Find total sales.
  - Find total sales for each city, or for each state.
  - Find top five products ranked by total sales.
- Roll-up: Aggregating at different levels of a dimension hierarchy.
  - E.g., Given total sales by city, we can roll-up to get sales by state.

OLAP Queries

- Drill-down: The inverse of roll-up.
  - E.g., Given total sales by state, can drill-down to get total sales by city.
  - E.g., Can also drill-down on different dimension to get total sales by product for each state.
- Pivoting: Aggregation on selected dimensions.
  - E.g., Pivoting on State and Year **cross-tabulation**:

yields this

	OR	CA	Total
2007	63	81	144
2008	38	107	145
2009	75	35	110
Total	176	223	339

- Slicing and Dicing: Equality and range selections on one or more dimensions.

Comparison with SQL Queries

- The cross-tabulation obtained by pivoting can also be computed using a collection of SQLqueries:

```
SELECT T.year, L.state, SUM(S.amt)
FROM   Sales S, Times T, Locations L
WHERE  S.timeid=T.timeid AND S.locid=L.locid
GROUP BY T.year, L.state
```

```
SELECT T.year, SUM(S.amt)
FROM   Sales S, Times T
WHERE  S.timeid=T.timeid
GROUP BY T.year
```

```
SELECT L.state,SUM(S.amt)
FROM   Sales S, Location L
WHERE  S.locid=L.locid
GROUP BY L.state
```

The CUBE Operator

- Generalizing the previous example, if there are k dimensions, we have 2^k possible SQL GROUP BY queries that can be generated through pivoting on a subset of dimensions.
- CUBE pid, locid, timeid BY SUM Sales
  - Equivalent to rolling up Sales on all eight subsets of the set {pid, locid, timeid}; each roll-up corresponds to an SQL query of the form:

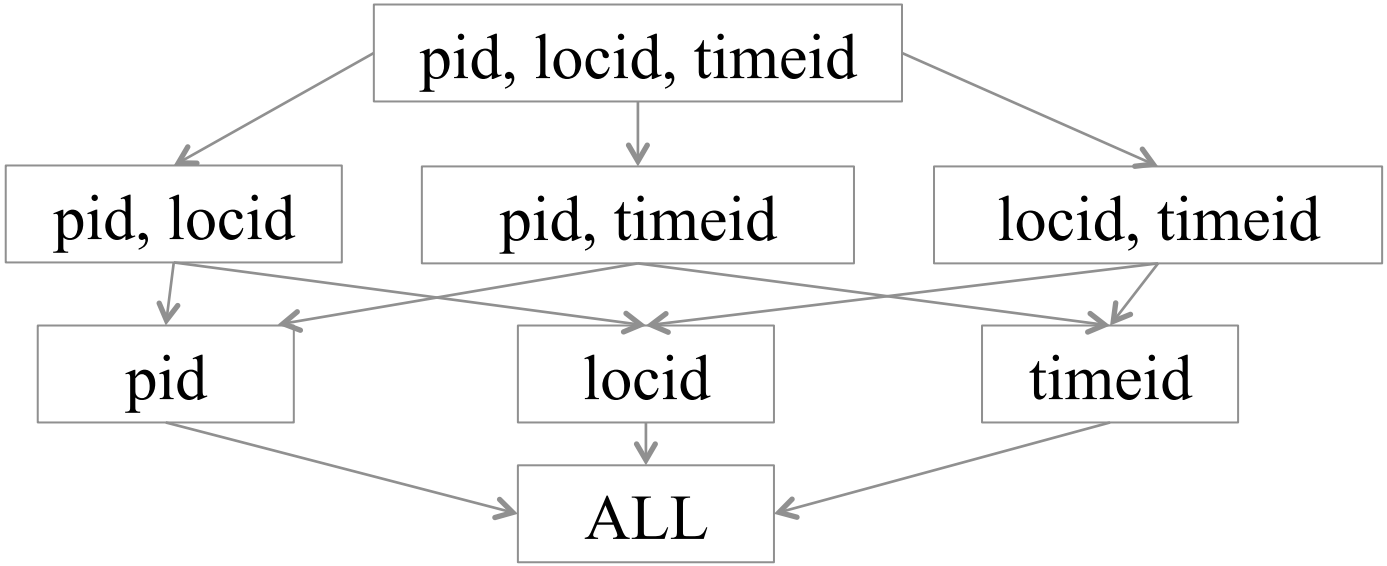
```
SELECT SUM(S.amt)
FROM   Sales S
GROUP BY grouping-list
```

Implementing the Cube Operator

- CUBE pid, locid, timeid BY SUM Sales
  - Equivalent to rolling up Sales on all eight subsets of the set {pid, locid, timeid}; each roll-up corresponds to an SQL query of the form:

```
SELECT SUM(S.amt)
FROM   Sales S
GROUP BY grouping-list
```

- A lot of work on optimizing the CUBE operator



Online Aggregation (Ch 25.5.2)

- Consider an aggregate query, e.g., finding the average sales by state. Can we provide the user with some information before the exact average is computed for all states?
  - Can show the current "running average" for each state as the computation proceeds.
  - Even better, if we use statistical techniques and sample tuples to aggregate instead of simply scanning the aggregated table, we can provide bounds such as "the average for Oregon is 2000±102 with 95% probability".
    - Should also use nonblocking algorithms!
    - Control sampling (reading of the data) to enable statistical bounds.

Constructing A Data Warehouse (25.7)

- Extract
  - Is the data in native format?
- Clean
  - How many ways can you spell Mr.?
  - Errors, missing information
- Transform
  - Fix semantic mismatches.
    - E.g. Last+first vs. Name
    - E.g. Convert all sales to Dollar
- Load
  - Do it in parallel or else….
- Refresh
  - Both data and indexes

Views and Decision Support (25.8,9)

- In large databases, precomputation is necessary for fast response times
  - Examples: brain, google
- Example: Precompute daily sums for the cube.
  - What can be derived from those precomputations?
- These precomputed queries are called **Materialized Views**
  - Another name: Indexed views in SQL Server.

Materialized View Example

Mat. View

```
CREATE VIEW DailySum(date, sumamt) AS
  SELECT date, SUM(amt)
  FROM Times Join Sales USING(timeid)
  GROUP BY date
```

Query

```
SELECT week, SUM(amt)
FROM Times Join Sales USING(timeid)
GROUP BY week
```

Modified Query

```
SELECT week, SUM(sumamt)
FROM Times Join DailySum USING (week)
GROUP BY week
```

Views

- A view is a named query

- A materialized view is the stored result of a query

Why are views interesting?

Views

- A view is a named query

- A materialized view is the stored result of a query

Why are views interesting?

- Query optimization

- Independence of the physical and the logical views

- Access rights management

Query containment and equivalence

- A query Q is said to be contained in Q' if for all database instance D, the set of tuples computed for Q is a subset of those computed for Q'
- Q and Q' are equivalent if Q is contained in Q' and conversely
- Food for thought: how hard is it to decide if two queries are equivalent?

Query Rewriting using Views

- Let V={V_1,...,V_n} be a set of views, Q be a query. Is it always possible to express Q using only the views in V?

Query Rewriting using Views

- Let $V=\{V_1,...,V_n\}$ be a set of views, Q be a query. Is it always possible to express Q using only the views in V?
- Maximally contained rewritings: let Q be a query, $V=\{V_1,...,V_n\}$ be a set of views, L be a query language. Q' is a maximally contained rewriting of Q using V wrt L if:
  - Q' is in L and refers only to views in V
  - Q' is contained in Q
  - There is no query Q'' such that Q' is contained in Q'' and Q'' uses only views in V and Q' is in L

Certain Answers

- Let V = {V_1,...,V_n} be a set of views, Q be a query, v_i the extension of V_i.
  - A tuple a is a certain answer to the query Q under the closed-world assumption given v_1,...,v_n if a is in Q(D) for all database instance D such that $V_i(D) = v_i$, for all i
  - A tuple a is a certain answer to the query Q under the open-world assumption given v_1,...,v_n if a is in Q(D) for all database instance D such that

    $v_i \subseteq V_i(D)$, for all i

Using a View for a Query

- Query:

select Advises.prof, Advises.student, Registered.quarter

from Registered, Teaches, Advises

where Registered.c-number = Teaches.c-number **and**

Registered.quarter = Teaches.quarter **and**

Advises.prof = Teaches.prof **and** Advises.student = Registered.student **and** Registered.quarter >= “winter98”.

- V1:

select Registered.student, Teaches.prof, Registered.quarter

from Registered, Teaches

where Registered.c-number = Teaches.c-number **and** Registered.quarter = Teaches.quarter **and** Registered.quarter > “winter97”.

Using a View for a Query

- Query:

select Advises.prof, Advises.student, Registered.quarter

from Registered, Teaches, Advises

where Registered.c-number = Teaches.c-number **and**

Registered.quarter = Teaches.quarter **and**

Advises.prof = Teaches.prof **and** Advises.student = Registered.student **and** Registered.quarter >= “winter98”.

- V2:

select Registered.student, Registered.quarter

from Registered, Teaches

where Registered.c-number = Teaches.c-number **and** Registered.quarter = Teaches.quarter **and** Registered.quarter > “winter97”.

Using a View for a Query

- Query:

select Advises.prof, Advises.student, Registered.quarter

from Registered, Teaches, Advises

where Registered.c-number = Teaches.c-number **and**

Registered.quarter = Teaches.quarter **and**

Advises.prof = Teaches.prof **and** Advises.student = Registered.student **and** Registered.quarter >= “winter98”.

- V3:

select Registered.student, Teaches.prof, Registered.quarter

from Registered, Teaches

where Registered.c-number = Teaches.c-number **and** Registered.quarter >= “winter98”.

Using a View for a Query

- Query:

select Advises.prof, Advises.student, Registered.quarter

from Registered, Teaches, Advises

where Registered.c-number = Teaches.c-number **and**

Registered.quarter = Teaches.quarter **and**

Advises.prof = Teaches.prof **and** Advises.student = Registered.student **and** Registered.quarter >= “winter98”.

- V4:

select Registered.student, Teaches.prof, Registered.quarter

from Registered, Teaches

where Registered.c-number = Teaches.c-number **and** Registered.quarter = Teaches.quarter **and** Registered.quarter > “winter99”.

Refreshing Materialized Views

- How often should we refresh the materialized view?
- Many enterprises refresh warehouse data only weekly/nightly, so can afford to completely rebuild their materialized views.
- Others want their warehouses to be current, so materialized views must be updated incrementally if possible.
- Let's look at some simple examples.

Maintaining Materialized Views (25.10)

- Incremental view maintenance
  - make changes in view that correspond to changes in the base tables

- Example: View V = **`SELECT a FROM R`**
  - How is V modified if r is inserted to R?

  - How is V modified if r is deleted from R?

Maintaining Materialized Views

- Consider V = R ⋈ S
  - How is V modified if r is inserted to R?
  - How is V modified if r is deleted from R?

- Consider V = SELECT g,COUNT(*)
  FROM R GROUP BY g
  - How is V modified if r is inserted to R?
  - How is V modified if r is deleted from R
  - What if the aggregate is AVG, MIN, MAX

- For more general cases, see [348]

Questions