Data Warehousing and OLAP

Mirek Riedewald
Computer Science Department
Cornell University

Motivation

- Large retailer
  - Several databases: inventory, personnel, sales etc.
  - High volume of updates
- Management requirements
  - Efficient support for decision making
  - Comprehensive view of all aspects of an enterprise
    - Trends, summaries, analysis of historical data
    - Information from several departments
- Why not using operational systems?

Motivation (contd.)

- Integrate data from diverse sources
  - Common schema
  - Semantic mismatches (currency, naming, normalization, databases structure)
  - Clean data (missing values, inconsistencies)
- Accumulate historical data
  - Not relevant for operational databases
- Efficient analysis
  - Complex queries versus frequent updates

Outline

- Overview of data warehousing

Terminology

- OLTP (Online Transaction Processing)
- DSS (Decision Support System)
- DW (Data Warehouse)
- OLAP (Online Analytical Processing)

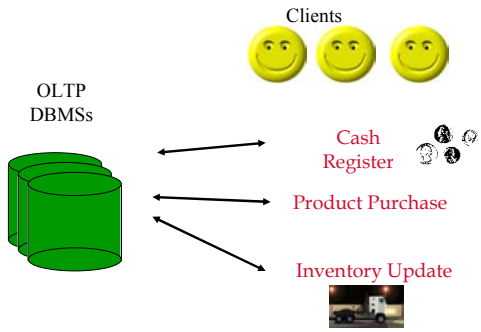
From OLTP to the Data Warehouse

- Traditionally, database systems stored data relevant to current business processes
  - Old data was archived or purged
- A database stores the current snapshot of the business:
  - Current customers with current addresses
  - Current inventory
  - Current orders
  - Current account balance

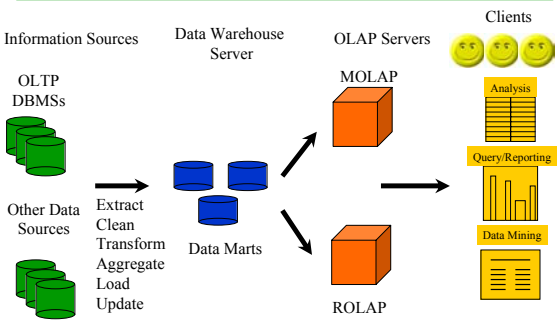
The Data Warehouse

- The data warehouse is a historical collection of all relevant data for analysis purposes
- Examples:
  - Current customers versus all customers
  - Current orders versus history of all orders
  - Current inventory versus history of all shipments
- Thus the data warehouse stores information that might be useless for the operational part of a business

OLTP Architecture



DW Architecture



Building a Data Warehouse

- Data warehouse is a collection of data marts
- Data marts contain one dimensional star schema that captures one business aspect
- Notes:
  - It is crucial to centralize the logical definition and format of dimensions and facts (political challenge; assign a dimension authority to each dimension). Everything else is a distributed effort throughout the company (technical challenge)
  - Each data mart will have its own fact table, but dimension tables are duplicated over several data marts

OLTP Versus Data Warehousing

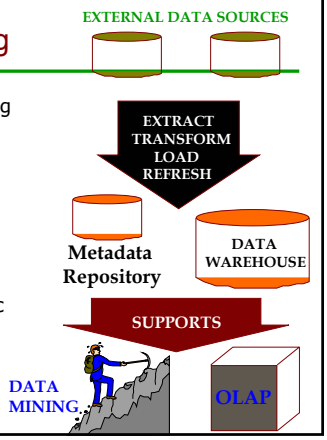
	OLTP	Data Warehouse
Typical user	Clerical	Management
System usage	Regular business	Analysis
Workload	Read/Write	Read only
Types of queries	Predefined	Ad-hoc
Unit of interaction	Transaction	Query
Level of isolation required	High	Low
No of records accessed	<100	>1,000,000
No of concurrent users	Thousands	Hundreds
Focus	Data in and out	Information out

Three Complementary Trends

- Data Warehousing: Consolidate data from many sources in one large repository
  - Loading, periodic synchronization of replicas
  - Semantic integration
- OLAP:
  - Complex SQL queries and views
  - Queries based on spreadsheet-style operations and "multidimensional" view of data
  - Interactive and "online" queries
- Data Mining: Exploratory search for interesting trends and anomalies (Another lecture!)

Data Warehousing

- Integrated data spanning long time periods, often augmented with summary information
- Several gigabytes to terabytes common
- Interactive response times expected for complex queries; ad-hoc updates uncommon



Warehousing Issues

- Semantic Integration: When getting data from multiple sources, must eliminate mismatches, e.g., different currencies, schemas
- Heterogeneous Sources: Must access data from a variety of source formats and repositories
  - Replication capabilities can be exploited here
- Load, Refresh, Purge: Must load data, periodically refresh it, and purge too old data
- Metadata Management: Must keep track of source, loading time, and other information for all data in the warehouse

Outline

- Overview of data warehousing
- Dimensional Modeling

Dimensional Data Modeling

- Recall: The relational model

The dimensional data model:

- Relational model with two different types of attributes and tables
- Attribute level: Facts (numerical, additive, dependent) versus dimensions (descriptive, independent)
- Table level: Fact tables (large tables with facts and foreign keys to dimensions) versus dimension tables (small tables with dimensions)

Dimensional Modeling (contd.)

- Fact (attribute): Measures performance of a business
- Example facts:
  - Sales, budget, profit, inventory
- Example fact table:
  - Transactions (timekey, storekey, pkey, promkey, ckey, units, price)
- Dimension (attribute): Specifies a fact
- Example dimensions:
  - Product, customer data, sales person, store
- Example dimension table:
  - Customer (ckey, firstname, lastname, address, dateOfBirth, occupation, …)

OLTP versus Data Warehouse

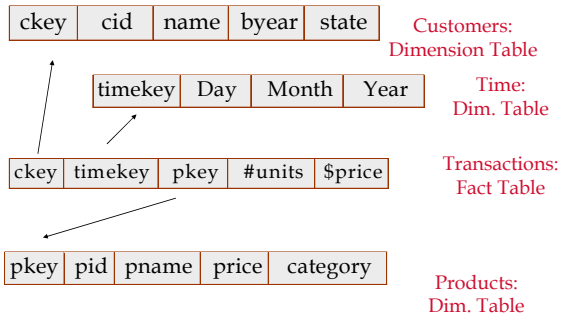
OLTP

- Regular relational schema
  - Normalized
- Updates overwrite previous values: One instance of a customer with a unique customerID
- Queries return information about the current state of affairs

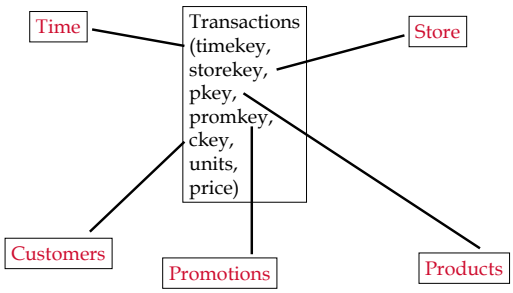
Data warehouse

- Dimensional model
  - Fact table in BCNF
  - Dimension tables not normalized: few updates, mostly queries
- Updates add new version: Several instances of the same customer (with different data, e.g., address)
- Queries return aggregate information about historical facts

Example: Dimensional Data Modeling



Another View: Star Schema



Fact versus Dimension Tables

- Fact tables are usually very large; they can grow to several hundred GB and TB
- Dimension tables are usually smaller (although can grow large, e.g., Customers table), but they have many fields
- Queries over fact tables usually involve many records

Grain

- The grain defines the level of resolution of a single record in the fact table.
- Example fact tables:
  - Transactions (timekey, storekey, pkey, promkey, ckey, units, price); grain is individual item
  - Transactions (timekey, storekey, ckey, units, price); grain is one market basket

Typical Queries

- SQL:

```
SELECT      D1.d1, …, Dk.dk, agg1(F.f1,)
FROM        Dimension D1, …,
            Dimension Dk, Fact F
WHERE       D1.key = F.key1 AND … AND
            Dk.keyk = F.keyk AND
            otherPredicates
GROUP BY    D1.d1, …, Dk.dk
HAVING      groupPredicates
```

- This query is called a "Star Join".

Example Query

- "Break down sales by year and category for the last two years; show only categories with more than $1M in sales."
- SQL:

```
SELECT T.year, P.category, SUM(X.units * X.price)
FROM Time T, Products P, Transactions X
WHERE T.year = 1999 OR T.year = 2000
GROUP BY T.year, P.category
HAVING SUM(X.units * X.price) > 1000000
```

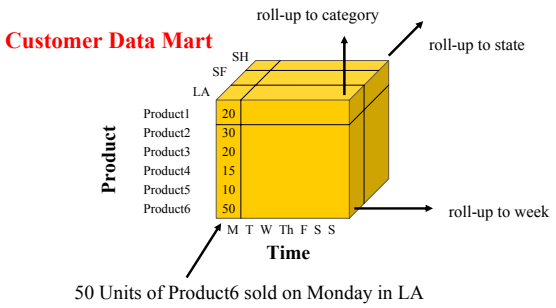
Outline

- Overview of data warehousing
- Dimensional Modeling
- Online Analytical Processing

Online Analytical Processing (OLAP)

- Ad hoc complex queries
- Simple, but intuitive and powerful query interface
  - Spreadsheet influenced analysis process
- Specialized query operators for multidimensional analysis
  - Roll-up and drill-down
  - Slice and dice
  - Pivoting

Visual Intuition: Cube

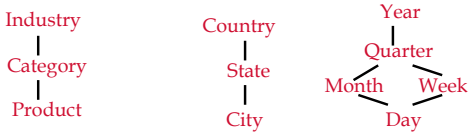


Multidimensional Data Analysis

Data warehouse:

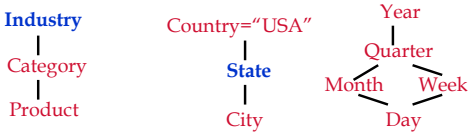
- Transactions(ckey, timekey, pkey, units, price)
- Customers(ckey, cid, name, byear, city, state, country)
- Time(tkey, day, month, quarter, year)
- Products(pkey, pname, price, pid, category, industry)

Hierarchies on dimensions:



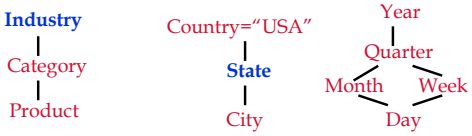
Multidimensional Data Analysis

	NY	CA	WI
Industry1	$1000	$2000	$1000
Industry2	$500	$1000	$500
Industry3	$3000	$3000	$3000



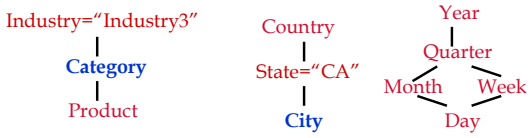
Corresponding Query in SQL

- SELECT SUM(units)
  FROM Transactions T, Products P, Customers C
  WHERE T.pkey = P.pkey AND T.ckey = C.ckey
  AND C.country = "USA"
  GROUP BY P.industry, C.state
- We think that Industry3 in CA is interesting.



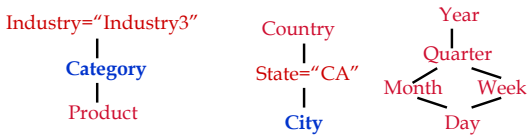
Slice and Drill-Down

	San Francisco	San Jose	Los Angeles
Category1	$300	$300	$400
Category2	$300	$300	$400
Category3	$100	$800	$100



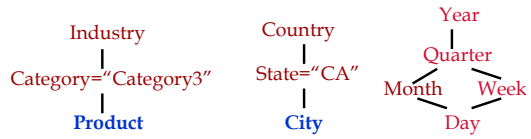
Corresponding Query in SQL

- SELECT SUM(units)
  FROM Transactions T, Products P, Customers C
  WHERE T.pkey = P.pkey AND T.ckey = C.ckey
  AND P.industry = "Industry3" AND C.state = "CA"
  GROUP BY P.category, C.city
- We think that Category3 is interesting.



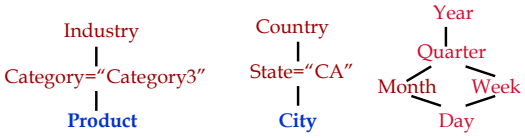
Slice and Drill-Down

	San Francisco	San Jose	Los Angeles
Product1	$20	$160	$20
Product2	$20	$160	$20
Product3	$60	$480	$60



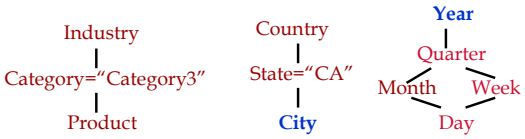
Corresponding Query in SQL

- SELECT SUM(units)
  FROM Transactions T, Products P, Customers C
  WHERE T.pkey = P.pkey AND T.ckey = C.ckey
  AND C.state = "CA" AND P.category = "Category3"
  GROUP BY P.product, C.city
- Nothing new in this view of the data.



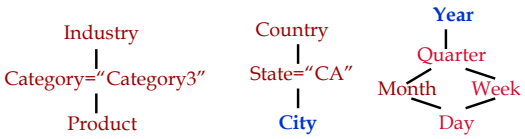
Pivot To (City, Year)

	San Francisco	San Jose	Los Angeles
1997	$20	$100	$20
1998	$20	$600	$20
1999	$60	$100	$60



Corresponding Query in SQL

- SELECT SUM(units)
  FROM Transactions T, Products P, Customers C
  WHERE T.pkey = P.pkey AND T.ckey = C.ckey
  AND C.state = "CA" AND P.category = "Category3"
  GROUP BY **C.city, T.year**



Multidimensional Data Analysis

Set of data manipulation operators

- Roll-up: Go up one step in a dimension hierarchy (e.g., month -> quarter)
- Drill-down: Go down one step in a dimension hierarchy (e.g., quarter -> month)
- Slice: Select a value of a dimension (e.g., all categories -> only Category3)
- Dice: Select range of values of a dimension (e.g., Year > 1999)
- Pivot: Select new dimensions to visualize the data (e.g., pivot to Time(quarter) and Customer(state))

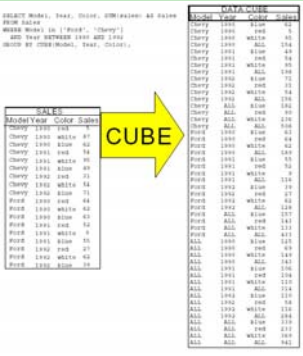
The CUBE Operator

- Generalizing GROUP BY and aggregation
  - 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:

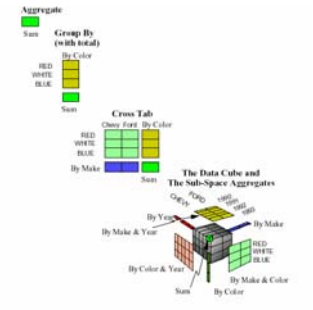
Lots of recent work on optimizing the CUBE operator!

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

Example



Multidimensional View Of CUBE



OLAP Server Architectures

- Relational OLAP (ROLAP)
  - Relational DBMS stores data mart (star schema)
  - OLAP middleware:
    - Aggregation and navigation logic
    - Optimized for DBMS in the background, but slow and complex
- Multidimensional OLAP (MOLAP)
  - Specialized array-based storage structure
- Desktop OLAP (DOLAP)
  - Performs OLAP directly at your PC
- Hybrids and Application OLAP

Summary: Multidimensional Analysis

- Spreadsheet style data analysis
- Roll-up, drill-down, slice, dice, and pivot your way to interesting cells in the CUBE
- Mainstream technology
- Established enterprises already have OLAP installations

Outline

- Overview of data warehousing
- Dimensional Modeling
- Online Analytical Processing
- Implementation Issues

Query Challenges

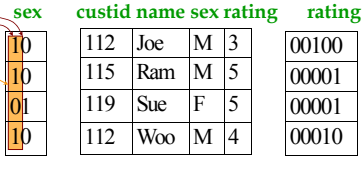
- Ad hoc queries
  - Simple, but powerful interface
- Complex queries
  - Group by, aggregation, joins
- Interactive response time

How can this be done for data warehouses with terabytes of data?

New DW And OLAP Techniques

- Bitmap indexes, join indexes, array representations, compression, precomputation of aggregations, etc.
- E.g., Bitmap index:

Bit-vector:
1 bit for each possible value.
Many queries can be answered using bit-vector ops!



sex
10
10
01
10

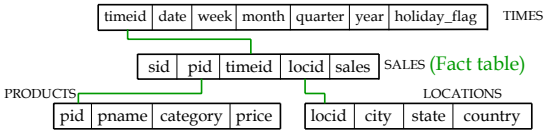
custid	name	sex	rating
112	Joe	M	3
115	Ram	M	5
119	Sue	F	5
112	Woo	M	4

rating
00100
00001
00001
00010

Join Indexes

- Consider join of Sales, Products, Times, and Locations
  - Join index can be constructed to speed up such joins
    - Index contains [s,p,t,l] if there are tuples (with sid) s in Sales, p in Products, t in Times and l in Locations that satisfy the join (and selection) conditions
- Problem: Large number of join indexes
  - A variant of the idea addresses this problem: For each column with an additional selection (e.g., country), build an index with [c,s] in it if a dimension table tuple with value c in the selection column joins with a Sales tuple with sid s; if indexes are bitmaps, called bitmapped join index

Bitmapped Join Index



- Consider a query with conditions price=10 and country="USA". Suppose tuple (with sid) s in Sales joins with a tuple p with price=10 and a tuple l with country ="USA". There are two join indexes; one containing [10,s] and the other [USA,s]
- Intersecting these indexes tells us which tuples in Sales are in the join and satisfy the given selection

Using Views (Evaluate On Demand)

View

```
CREATE VIEW RegionalSales(category,sales,state)
        AS SELECT P.category, S.sales, L.state
           FROM Products P, Sales S, Locations L
           WHERE P.pid=S.pid AND S.locid=L.locid
```

Query

```
SELECT R.category, R.state, SUM(R.sales)
FROM RegionalSales AS R
GROUP BY R.category, R.state
```

Modified Query

```
SELECT R.category, R.state, SUM(R.sales)
FROM (SELECT P.category, S.sales, L.state
        FROM Products P, Sales S, Locations L
        WHERE P.pid=S.pid AND S.locid=L.locid) AS R
GROUP BY R.category, R.state
```

View Materialization (Precomputation)

- Suppose we precompute RegionalSales and store it with a clustered B+ tree index on [category,state,sales]
  - Query can be answered by an index only scan

```
SELECT R.state, SUM(R.sales)
FROM RegionalSales R
WHERE R.category="Laptop"
GROUP BY R.state
```

Index on precomputed view is great!

```
SELECT R.state, SUM(R.sales)
FROM RegionalSales R
WHERE R. state="Wisconsin"
GROUP BY R.category
```

Index is less useful (must scan entire leaf level).

Views and Decision Support

- Precomputation is essential for interactive response times
- CUBE is collection of aggregate queries, and precomputation is especially important: lots of research on what is best to precompute
- Warehouse = collection of asynchronously replicated tables and periodically maintained views
  - Has renewed interest in view maintenance
- Tradeoffs for storing more precomputed data
  - Faster queries
  - Higher storage and maintenance cost

Issues in View Materialization

- What views should we materialize, and what indexes should we build on the precomputed results?
- Given a query and a set of materialized views, can we use the materialized views to answer the query?
- How frequently should we refresh materialized views to make them consistent with the underlying tables? (And how can we do this incrementally?)

Interactive Queries: Beyond Materialization

- Top N Queries: If you want to find the 10 (or so) cheapest cars, it would be nice if the DB could avoid computing the costs of all cars before sorting to determine the 10 cheapest.
  - Idea: Guess a cost c such that the 10 cheapest all cost less than c, and that not too many more cost less. Then add the selection cost<c and evaluate the query.
    - If the guess is right, great, we avoid computation for cars that cost more than c.
    - If the guess is wrong, need to reset the selection and recompute the original query.

Top N Queries

```
SELECT P.pid, P.pname, S.sales
FROM Sales S, Products P
WHERE S.pid=P.pid AND S.locid=1 AND S.timeid=3
ORDER BY S.sales DESC
OPTIMIZE FOR 10 ROWS
```

```
SELECT P.pid, P.pname, S.sales
FROM Sales S, Products P
WHERE S.pid=P.pid AND S.locid=1 AND S.timeid=3
      AND S.sales > c
ORDER BY S.sales DESC
```

- OPTIMIZE FOR construct is not in SQL:1999!
- Cut off value c is chosen by optimizer.

Interactive Queries: Beyond Materialization

- Online Aggregation: 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 Wisconsin is 2000±102 with 95% probability.
    - Should also use nonblocking algorithms!

Summary

- Decision support is a rapidly growing subarea of databases
- Involves the creation of large, consolidated data repositories called data warehouses
- Warehouses are exploited using sophisticated analysis techniques: complex SQL queries and OLAP "multidimensional" queries (influenced by both SQL and spreadsheets)
- New techniques for database design, indexing, view maintenance, and interactive querying need to be supported