Motivation

• Data analytics or Business Intelligence
  – Identify useful patterns and trends
  – Support business strategies and decision-making
  – Use both current and historical data

• Emphasis is on
  – Complex, interactive, exploratory analysis
  – Very large datasets that are fairly static
  – Integrating data from all parts of an enterprise
  – On-Line Analytic Processing (OLAP) vs traditional On-line Transaction Processing (OLTP)
  – Long queries vs short update transactions.
What is a Data Warehouse?

• Defined in many different ways, but not rigorously.
  – A decision support database that is maintained separately from the organization’s operational database
  – Support information processing by providing a solid platform of consolidated, historical data for analysis.

• “A data warehouse is a subject-oriented, integrated, time-variant, and nonvolatile collection of data in support of management’s decision-making process.”—W. H. Inmon

• Data warehousing:
  – The process of constructing and using data warehouses
Subject-Oriented

• Organized around major subjects, such as
  – customer,
  – product,
  – sales.

• Focused on the modeling and analysis of data for decision makers
  – *not on* daily operations or transaction processing.

• Provide a simple and concise view around particular subject issues by excluding data that are not useful in the decision support process.
Integrated

• Constructed by integrating multiple, heterogeneous data sources
  – relational databases, flat files, on-line transaction records
• Data cleaning and data integration techniques are applied.
  – Ensure consistency in naming conventions, encoding structures, attribute measures, etc. among different data sources
    • E.g., Hotel price: currency, tax, breakfast covered, etc.
  – When data is moved to the warehouse, it is converted.
Time Variant

• The time horizon for the data warehouse is significantly longer than that of operational systems.
  – Operational database: current value data.
  – Data warehouse data: provide information from a historical perspective (e.g., past 5-10 years)

• Every key structure in the data warehouse
  – Contains an element of time, explicitly or implicitly
  – But the key of operational data may or may not contain “time element”.
Non-Volatile

- A physically separate store of data transformed from the operational environment.
- Operational update of data does not occur in the data warehouse environment.
  - Does not require transaction processing, recovery, and concurrency control mechanisms
  - Requires only two operations in data accessing:
    - initial loading of data and access of data.
Data Warehouse vs. Operational DBMS

• **OLTP (on-line transaction processing)**
  – Major task of traditional relational DBMS
  – Day-to-day operations: purchasing, inventory, banking, manufacturing, payroll, registration, accounting, etc.

• **OLAP (on-line analytical processing)**
  – Major task of data warehouse system
  – Data analysis and decision making

• Distinct features (OLTP vs. OLAP):
  – User and system orientation: customer vs. market
  – Data contents: current, detailed vs. historical, consolidated
  – Database design: ER + application vs. star + subject
  – View: current, local vs. evolutionary, integrated
  – Access patterns: update vs. read-only but complex queries
# OLTP vs OLAP

<table>
<thead>
<tr>
<th></th>
<th>OLTP</th>
<th>OLAP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>users</strong></td>
<td>clerk, IT professional</td>
<td>knowledge worker</td>
</tr>
<tr>
<td><strong>function</strong></td>
<td>day to day operations</td>
<td>decision support</td>
</tr>
<tr>
<td><strong>DB design</strong></td>
<td>application-oriented</td>
<td>subject-oriented</td>
</tr>
<tr>
<td><strong>data</strong></td>
<td>current, up-to-date</td>
<td>historical, summarized, multidimensional</td>
</tr>
<tr>
<td></td>
<td>detailed, flat relational</td>
<td>integrated, consolidated</td>
</tr>
<tr>
<td></td>
<td>isolated</td>
<td></td>
</tr>
<tr>
<td><strong>usage</strong></td>
<td>repetitive</td>
<td>ad-hoc</td>
</tr>
<tr>
<td><strong>access</strong></td>
<td>read/write</td>
<td>lots of scans</td>
</tr>
<tr>
<td></td>
<td>index/hash on prim. key</td>
<td></td>
</tr>
<tr>
<td><strong>unit of work</strong></td>
<td>short, simple transaction</td>
<td>complex query</td>
</tr>
<tr>
<td><strong># records accessed</strong></td>
<td>tens</td>
<td>millions</td>
</tr>
<tr>
<td><strong>#users</strong></td>
<td>thousands</td>
<td>hundreds</td>
</tr>
<tr>
<td><strong>DB size</strong></td>
<td>100MB-GB</td>
<td>100GB-TB</td>
</tr>
<tr>
<td><strong>metric</strong></td>
<td>transaction throughput</td>
<td>query throughput, response</td>
</tr>
</tbody>
</table>
Why Separate Data Warehouse?

• **High performance for both systems**
  – DBMS—tuned for OLTP: access methods, indexing, concurrency control, recovery
  – Warehouse—tuned for OLAP: complex OLAP queries, multidimensional view, consolidation.

• **Different functions and different data:**
  – **missing data**: Decision support requires historical data which operational DBs do not typically maintain
  – **data consolidation**: DS requires consolidation (aggregation, summarization) of data from heterogeneous sources
  – **data quality**: different sources typically use inconsistent data representations, codes and formats which have to be reconciled
Data Warehousing

- Integrated data spanning long time periods, often augmented with summary information.
- Several 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.
Multidimensional Data Model

• Collection of numeric measures, which depend on a set of dimensions.
  – E.g., measure Sales, dimensions Product (key: pid), Location (locid), and Time (timeid).
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, pname, category, price)
  – Fact tables are much larger than dimensional tables.
Dimension Hierarchies

• For each dimension, the set of values can be organized in a hierarchy:
Conceptual Design of Data Warehouses

- Fact table in BCNF; dimension tables un-normalized.
  - Dimension tables are small; updates/inserts/deletes are rare. So, anomalies less important than query performance.
- This kind of schema is very common in OLAP applications, and is called a **star schema**; computing the join of all these relations is called a **star join**.
Example: Star Schema

A fact table in the middle connected to a set of dimension tables

- **Time**
  - time_key
  - day
  - day_of_the_week
  - month
  - quarter
  - year

- **Item**
  - item_key
  - item_name
  - brand
  - type
  - supplier_type

- **Branch**
  - branch_key
  - branch_name
  - branch_type

- **Location**
  - location_key
  - street
  - city
  - province_or_street
  - country

**Measures**

- units_sold
- dollars_sold
- avg_sales

Sales Fact Table

A fact table in the middle connected to a set of dimension tables.
some dimensional hierarchy is normalized into a set of smaller dimension tables, forming a shape similar to snowflake
Example: Constellation

galaxy schema or fact constellation: Multiple fact tables share dimension tables

Sales Fact Table
- time_key
- item_key
- branch_key
- location_key
- units_sold
- dollars_sold
- avg_sales

item
- item_key
- item_name
- brand
- type
- supplier_type

Shipping Fact Table
- time_key
- item_key
- shipper_key
- from_location
- to_location
- dollars_cost
- units_shipped

shipper
- shipper_key
- shipper_name
- location_key
- shipper_type

branch
- branch_key
- branch_name
- branch_type

Measures
- time
  - day
  - day_of_the_week
  - month
  - quarter
  - year

location
- location_key
- street
- city
- province_or_street
- country

Lipyeow Lim -- University of Hawaii at Manoa
OLAP Queries

• 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.
More 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 Location and Time yields this **cross-tabulation**:

• **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 SQL queries:

<table>
<thead>
<tr>
<th>Year</th>
<th>WI</th>
<th>CA</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>63</td>
<td>81</td>
<td>144</td>
</tr>
<tr>
<td>1996</td>
<td>38</td>
<td>107</td>
<td>145</td>
</tr>
<tr>
<td>1997</td>
<td>75</td>
<td>35</td>
<td>110</td>
</tr>
<tr>
<td>Total</td>
<td>176</td>
<td>223</td>
<td>339</td>
</tr>
</tbody>
</table>

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

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

SELECT SUM(S.sales)
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** $	ext{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:

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

Lots of work on optimizing the CUBE operator
Querying Sequences in SQL:1999

• Trend analysis is difficult to do in SQL-92:
  – Find the % change in monthly sales
  – Find the top 5 product by total sales
  – Find the trailing $n$-day moving average of sales
  – The first two queries can be expressed with difficulty, but the third cannot even be expressed in SQL-92 if $n$ is a parameter of the query.

• The \texttt{WINDOW} clause in SQL:1999 allows us to write such queries over a table viewed as a sequence (implicitly, based on user-specified sort keys)
The WINDOW Clause

SELECT L.state, T.month, AVG(S.sales) OVER W AS movavg
FROM Sales S, Times T, Locations L
WHERE S.timeid=T.timeid AND S.locid=L.locid
WINDOW W AS (PARTITION BY L.state
ORDER BY T.month
RANGE BETWEEN INTERVAL `1' MONTH PRECEDING
AND INTERVAL `1' MONTH FOLLOWING)

• Let the result of the FROM and WHERE clauses be “Temp”.
• (Conceptually) Temp is partitioned according to the PARTITION BY clause.
  – Similar to GROUP BY, but the answer has one row for each row in a partition, not one row per partition!
• Each partition is sorted according to the ORDER BY clause.
• For each row in a partition, the WINDOW clause creates a “window” of nearby (preceding or succeeding) tuples.
  – Can be value-based, as in example, using RANGE
  – Can be based on number of rows to include in the window, using ROWS clause
• The aggregate function is evaluated for each row in the partition using the corresponding window.
  – New aggregate functions that are useful with windowing include RANK (position of a row within its partition) and its variants DENSE_RANK, PERCENT_RANK, CUME_DIST.
Top K 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 at 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 $\text{cost}<c$ 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.
Example: Top K 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  
FETCH FIRST 10 ROWS ONLY

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

- FETCH FIRST 10 ROWS ONLY is not in SQL99
- Cut-off value c is chosen by optimizer
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!
Data Warehousing Architecture

Data Sources

- Operational DBs
- other sources

Data Storage

- Data Warehouse
- Metadata
- OLAP Server
- Serve
- Analysis
- Query
- Reports
- Data mining

OLAP Engine

- OLAP Engine
- Analysis
- Query
- Reports
- Data mining

Front-End Tools

- Front-End Tools
- Monitor & Integrator
- Serve
- OLAP Server
- Extract
- Transform
- Load
- Refresh

Data Marts

- Data Marts
- Data Warehouse
- Monitor & Integrator
- OLAP Engine
- Front-End Tools

Operational DBs

- Operational DBs
- Other sources