What Is a Data Warehouse?

• “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
OLAP Operations
Data Warehouse Schema Design

• Query answering efficiency
  – Subject orientation
  – Integration

• Tradeoff between time and space
  – Universal table versus fully normalized schema
Star Schema

### Time Dimension
- time_key
- day
- day_of_the_week
- month
- quarter
- year

### Location Dimension
- location_key
- street
- city
- state_or_province
- country

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

### Item Dimension
- item_key
- item_name
- brand
- type
- supplier_type

### Branch Dimension
- branch_key
- branch_name
- branch_type

### Measures
- item_key
- item_name
- brand
- type
- supplier_type
Snowflake Schema

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

- **branch**
  - branch_key
  - branch_name
  - branch_type

- **Measure**
  - units_sold
  - dollars_sold
  - avg_sales

- **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_key

- **supplier**
  - supplier_key
  - supplier_type

- **location**
  - location_key
  - street
  - city_key

- **city**
  - city_key
  - city
  - state_or_province
  - country
(Good) Aggregate Functions

- **Distributive:** there is a function $G()$ such that
  $F(\{X_{i,j}\}) = G(\{F(\{X_{i,j} | i=1,\ldots,l_j\}) | j=1,\ldots,n\})$
  - Examples: COUNT(), MIN(), MAX(), SUM()
  - $G=$SUM() for COUNT()

- **Algebraic:** there is an $M$-tuple valued function $G()$ and a function $H()$ such that
  $F(\{X_{i,j}\}) = H(\{G(\{X_{i,j} | i=1,\ldots,l\}) | j=1,\ldots,n\})$
  - Examples: AVG(), standard deviation, MaxN(), MinN()
  - For AVG(), $G()$ records sum and count, $H()$ adds these two components and divides to produce the global average
Holistic Aggregate Functions

- There is no constant bound on the size of the storage needed to describe a sub-aggregate.
  - There is no constant $M$, such that an $M$-tuple characterizes the computation $F(\{X_{i,j} \mid i=1,\ldots,l\})$.
- Examples: Median(), MostFrequent() (also called the Mode()), and Rank()
Index Requirements in OLAP

• Data is read only
  – (Almost) no insertion or deletion
• Query types
  – Point query: looking up one specific tuple (rare)
  – Range query: returning the aggregate of a (large) set of tuples, with group by
  – Complex queries: need specific algorithms and index structures, will be discussed later
OLAP Query Example

- In table (cust, gender, …), find the total number of male customers
- Method 1: scan the table once
- Method 2: build a B+ tree index on attribute gender, still need to access all tuples of male customers
- Can we get the count without scanning many tuples, even not all tuples of male customers?
Bitmap Index

- For n tuples, a bitmap index has n bits and can be packed into \([n \div 8]\) bytes and \([n \div 32]\) words.
- From a bit to the row-id: the j-th bit of the p-th byte \(\rightarrow\) row-id = \(p \times 8 + j\)

<table>
<thead>
<tr>
<th>cust</th>
<th>gender</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jack</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>Cathy</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Nancy</td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

1 0 ... 0
Using Bitmap to Count

- Shcount[] contains the number of bits in the entry subscript
  - Example: shcount[01100101]=4

```c
count = 0;
for (i = 0; i < SHNUM; i++)
    count += shcount[B[i]];
```
Advantages of Bitmap Index

- Efficient in space
- Ready for logic composition
  - $C = C_1 \text{ AND } C_2$
  - Bitmap operations can be used
- Bitmap index only works for categorical data with low cardinality
  - Naively, we need 50 bits per entry to represent the state of a customer in US
  - How to represent a sale in dollars?
Bit-Sliced Index

- A sale amount can be written as an integer number of pennies, and then be represented as a binary number of N bits
  - 24 bits is good for up to $167,772.15, appropriate for many stores
- A bit-sliced index is N bitmaps
  - Tuple j sets in bitmap k if the k-th bit in its binary representation is on
  - The space costs of bit-sliced index is the same as storing the data directly
Using Indexes

SELECT SUM(sales) FROM Sales WHERE C;
– Tuples satisfying C is identified by a bitmap B

• Direct access to rows to calculate SUM: scan the whole table once
• B+ tree: find the tuples from the tree
• Projection index: only scan attribute sales
• Bit-sliced index: get the sum from $\sum (B \text{ AND } B_k) \times 2^k$
Cost Comparison

- Traditional value-list index (B+ tree) is costly in both I/O and CPU time
  - Not good for OLAP
- Bit-sliced index is efficient in I/O
- Other case studies in [O’Neil and Quass, SIGMOD’97]

<table>
<thead>
<tr>
<th>Method</th>
<th>I/O</th>
<th>CPU contributions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add from Rows</td>
<td>1,341K</td>
<td>I/O + 2M*(25 ins)</td>
</tr>
<tr>
<td>Projection index</td>
<td>100K</td>
<td>I/O + 2M*(10 ins)</td>
</tr>
<tr>
<td>Value-List index</td>
<td>103K</td>
<td>I/O + 100M*(10 ins)</td>
</tr>
<tr>
<td>Bit-Sliced index</td>
<td>69K</td>
<td>I/O + 197M*(1 ins)</td>
</tr>
</tbody>
</table>
Horizontal or Vertical Storage

- A fact table for data warehousing is often fat
  - Tens of even hundreds of dimensions/attributes
- A query is often about only a few attributes
- Horizontal storage: tuples are stored one by one
- Vertical storage: tuples are stored by attributes

<table>
<thead>
<tr>
<th>A₁</th>
<th>A₂</th>
<th>…</th>
<th>A₁₀₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁</td>
<td>x₂</td>
<td>…</td>
<td>x₁₀₀</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>z₁</td>
<td>z₂</td>
<td>…</td>
<td>z₁₀₀</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A₁</th>
<th>A₂</th>
<th>…</th>
<th>A₁₀₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>x₁</td>
<td>x₂</td>
<td>…</td>
<td>x₁₀₀</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>z₁</td>
<td>z₂</td>
<td>…</td>
<td>z₁₀₀</td>
</tr>
</tbody>
</table>
Horizontal Versus Vertical

• Find the information of tuple $t$
  – Typical in OLTP
  – Horizontal storage: get the whole tuple in one search
  – Vertical storage: search 100 lists

• Find $\text{SUM}(a_{100}) \ \text{GROUP BY} \ \{a_{22}, a_{83}\}$
  – Typical in OLAP
  – Horizontal storage (no index): search all tuples $O(100n)$, where $n$ is the number of tuples
  – Vertical storage: search 3 lists $O(3n)$, 3% of the horizontal storage method

• Projection index: vertical storage