Data Warehousing and Decision Support (mostly using Relational Databases)

CS634
Class 20

Slides based on “Database Management Systems” 3rd ed, Ramakrishnan and Gehrke, Chapter 25
Introduction

- Increasingly, organizations are analyzing current and historical data to identify useful patterns and support business strategies.

- Emphasis is on complex, interactive, exploratory analysis of very large datasets created by integrating data from across all parts of an enterprise.
  - Contrast such Data Warehousing and On-Line Analytic Processing (OLAP) with traditional On-line Transaction Processing (OLTP): mostly long queries, instead of the short update Xacts of OLTP.
  - In past, both were using “structured data” that can be fairly easily loaded into a database.
  - Today, businesses also monitor social media, web clicks, etc., which are not properly structured, hard to put in RDB.
Structured vs. Unstructured Data

• So far, we have been working with structured data

• Structured data:
  • Entities with attributes, each fitting a SQL data type
  • Relationships
  • Each row of data is precious
  • Loads into relational tables, long-term storage
  • Can be huge

• Unstructured data, realm of “big data”
  • Often doesn’t fit into E/R model, too sloppy
  • Each piece of data is not precious—it’s statistical
  • Sometimes just processed and thrown away
  • No permanent specialized repository, maybe saved in files
  • Can be really huge
Data Warehouses using RDB vs. Data Lakes using Hadoop

• Both are ways to hold huge amounts of data
• Data lakes hold “big data”, use big data techniques to query and analyze data. Hadoop provides a high-availability scalable distributed system.
• Big data can be original, uncleaned data, vs. cleaned data for RDB systems.
• A data lake can hold both original and cleaned data. Term “data lake” was invented in 2011, i.e., around same time as release of Hadoop.
• RDB Data warehouse technology ends up with data in a form easily understood by business people
• Big data is not there yet: usually need “data scientists” to interpret the data, write the queries, or at least new queries.
• Of course this is changing...
• Many big businesses have both a traditional data warehouse and a data lake, load some of same data in both Datamation article
• We may reserve “data warehouse” without adjective to encompass both RDB data warehouses and big-data warehouses that provide user-friendly access methods
Bigness of Data

Huge Data warehouses, all on Teradata systems (hard to find current sizes)

See article

- Biggest DW: Walmart, passed 1TB in 1992, 2.8 PB (petabytes) = 2800 TB in 2008, 30 PB in 2014, 40+ PB in 2017, processing 2.5PB/hour, growing...
- eBay: 9 PB DW in 2013, also has 40 PB of big data, uses Hadoop, etc.
- Apple: multiple-PB DW

- Big data:
  - Usually over 50TB, can’t fit on one machine
  - Is judged by “velocity” as well as size
  - Google: processed 24 PB of data per day in 2009, invented Map-Reduce, published 2004
Teradata

- Teradata provides a relational database with ANSI compliant SQL, targeted to data warehouses
- Proprietary, expensive ($millions)
- Uses a shared-nothing architecture on many independent nodes
- Partitioning by rows or (more recently) columns
- Scales up well: add node, add network bandwidth for it
- Now supports Hadoop as well as RDBMS: Teradata Appliance for Hadoop
Three Complementary Trends

**Data Warehousing:** Consolidate data from many sources in one large repository (relational database or data lake).
- Loading, periodic synchronization of data.
- Semantic integration, Data cleaning of data on way in (RDB only so far)
- Both simple and complex queries and views. (SQL or programmed)
- Note: SQL is available on top of big data in most systems

**OLAP/Big Data analytics**
- Queries based on spreadsheet-style operations and “multidimensional” view of data.
- Interactive queries. Look at data from different directions, granularity, etc.
- Big Data Example: Apache Kylin, originally from eBay, available 2017

**Data Mining:** Exploratory search for interesting trends and anomalies.
Data Warehousing

- Integrated data spanning long time periods, often augmented with summary information.
- Several gigabytes to terabytes common, now petabytes too.
- Interactive response times expected for complex queries; ad-hoc updates uncommon.
- Read-mostly data
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 (lineage) loading time, and other information for all data in the warehouse.
OLAP: Multidimensional data model

- A way to make complex data understandable by business user, etc.
- Example: sales data
- **Dimensions**: Product, Location, Time
- A **measure** is a numeric value like sales we want to understand in terms of the dimensions
- Example measure: dollar sales value “sales”
- **Example data point** (one row of fact/cube table):
  - Sales = 25 for pid=1, timeid=1, locid=1 is the sum of sales for that day, in that location, for that product
  - Pid=1: details in Product table
  - Locid = 1: details in Location table
- **Note aggregation here**: sum of sales is most detailed data
Multidimensional Data Model

SalesCube(pid, timeid, locid, sales)

- Collection of numeric measures, which depend on a set of dimensions.
  - E.g., measure sales, dimensions Product (key: pid), Location (locid), and Time (timeid).
  - Full table, pg. 851

Slice locid=1 is shown:

<table>
<thead>
<tr>
<th>pid</th>
<th>timeid</th>
<th>locid</th>
<th>sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
<td>1</td>
<td>25</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>1</td>
<td>30</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>1</td>
<td>50</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>13</td>
<td>3</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>2</td>
<td>35</td>
</tr>
</tbody>
</table>

- - -
Granularity of Data

- Example of last slide uses time at granularity of days
- Individual transactions (sales at cashier) have been added together to make one row in this table
- Note: “measures” can always be aggregated
- Current hardware can handle more data
- Typical data warehouses hold the original transaction data
- So such a fact table has more columns, for example
  - dateid, timeofday, prodid, storeid, txnid, clerkid, sales, ...
Data Warehouse vs. Data for OLAP

• Current DW fact table is huge, with individual transactions, large number of dimensions
• Can only use a subset of this for OLAP, because of explosion of cells
• Take DW fact table, roll up to days (say), drop less important columns, get much smaller data for OLAP
• Load data into OLAP, another tool.
• Table on pg. 851 is a cube table, not a DW fact table
• Can think of OLAP as a cache of most important aggregates of DW tables
For each dimension, the set of values can be organized in a hierarchy:

- **PRODUCT**
  - category
  - pname

- **TIME**
  - year
  - quarter
  - month
  - week
  - date

- **LOCATION**
  - country
  - state
  - city
Fact/cube table in BCNF; dimension tables not normalized.

- Dimension tables are small; updates/inserts/deletes are rare. So, anomalies less important than good query performance.

- This kind of schema is very common in DW and OLAP, and is called a star schema; computing the join of all these relations is called a star join.

- Note: in OLAP, this is not what the user sees, it’s hidden underneath

- In DW, this is the basic setup, but usually with more dimensions

- Here only one measure, sales, but can have several
OLAP (and DW) 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.
OLAP Queries: MDX (Multidimensional Expressions)

• Originally a Microsoft SQL Server project, but now supported widely in the OLAP industry: Oracle, SAS, SAP, Teradata on server side, as well as Microsoft. Allows client programs to specify OLAP datasets.

• Example from Wikipedia

```
SELECT
    { [Measures].[Store Sales] } ON COLUMNS,
    { [Date].[2002], [Date].[2003] } ON ROWS
FROM Sales
WHERE ( [Store].[USA].[CA] )
```

• The SELECT clause sets the query axes as the Store Sales member of the Measures dimension, and the 2002 and 2003 members of the Date dimension.
• The FROM clause indicates that the data source is the Sales cube.
• The WHERE clause defines the "slicer axis" as the California member of the Store dimension.
OLAP Queries

- **Drill-down**: The inverse of roll-up: go from sum to details that were added up before
  - E.g., Given total sales by state, can drill-down to get total sales by county.
  - Drill down again, see total sales by city.
  - E.g., Can also drill-down on different dimension to get total sales by product for each state.
OLAP Queries: cross-tabs

With relational DBs, we are used to tables with column names across the top, rows of data.

With OLAP, a spreadsheet-like representation is common, called a cross-tabulation:

- One dimension horizontally
- Another vertically

<table>
<thead>
<tr>
<th></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>
OLAP Queries: Pivoting

- **Example cross-tabulation:**

<table>
<thead>
<tr>
<th></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>

- **Pivoting:** switching dimensions on axes, or choosing what dimensions to show on axes
- **Switching dimensions** means pivoting around a point in the upper-left-hand corner
  - End up with “1995 1996 1997 Total” across top,
  - “WI CA Total” down the side
Oracle 11+ supports cross-tabs display

```
select * from (  
    select times_purchased, state_code
    from customers t
) pivot (  
    count(state_code)
    for state_code in ('NY','CT','NJ','FL','MO')
) order by times_purchased

Here is the output:
```

<table>
<thead>
<tr>
<th>TIMES_PURCHASED</th>
<th>NY</th>
<th>CT</th>
<th>NJ</th>
<th>FL</th>
<th>MO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16601</td>
<td>90</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>33048</td>
<td>165</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>33151</td>
<td>179</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>32978</td>
<td>173</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>33109</td>
<td>173</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

... and so on ...
(We have Oracle 10, unfortunately)
**SQL Queries for cross-tab entries**

The cross-tabulation values can be computed using a collection of SQL queries:

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

<table>
<thead>
<tr>
<th></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>
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 Query, pg. 857

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

- 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.sales)
FROM Sales S
GROUP BY grouping-list
```
Oracle 10+ supports CUBE queries

```
select t.year, s.store_state, sum(dollar_sales)
from salesfact f, times t, store s
where f.time_key = t.time_key and s.store_key = f.store_key
group by cube(t.year, s.store_state);
```

<table>
<thead>
<tr>
<th>YEAR</th>
<th>STORE_STATE</th>
<th>SUM(DOLLAR_SALES)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>781403.59</td>
</tr>
<tr>
<td></td>
<td>AZ</td>
<td>35684</td>
</tr>
<tr>
<td></td>
<td>CA</td>
<td>77420.82</td>
</tr>
<tr>
<td></td>
<td>CO</td>
<td>38335.26</td>
</tr>
<tr>
<td></td>
<td>TX</td>
<td>40886.54</td>
</tr>
<tr>
<td></td>
<td>WA</td>
<td>39540.16</td>
</tr>
<tr>
<td>1994</td>
<td>AZ</td>
<td>17903.04</td>
</tr>
<tr>
<td>1994</td>
<td>CA</td>
<td>38966.54</td>
</tr>
<tr>
<td>1994</td>
<td>CO</td>
<td>17870.33</td>
</tr>
<tr>
<td>1994</td>
<td>DC</td>
<td>20901.18</td>
</tr>
</tbody>
</table>

... from dbs2 output
DW data $\rightarrow$ OLAP

• The CUBE query can do the roll-ups on DW data needed for OLAP
Excel is the champ at OLAP queries

• Next time will do Excel pivot table demo
• Based on video by Minder Chen of UCI (Cal state U/Channel Islands)
  • https://www.youtube.com/watch?v=eGhjklYyv6Y
• Setup:
  • His MS Access database with star schema for sales
  • Create view of fact joined with desired dimension data (a star join)
  • Point Excel at this big view, ask it to create pivot table
  • Pivot table: drill down, roll up, pivot, ...
Excel can use Oracle data too

- The database from Chen’s demo is now in dbs2’s Oracle
- We could point Excel to an Oracle view of joined tables.
- How does that work?
- Use ODBC (Open Database Connectivity), older than JDBC, but roughly same idea
  - Provides client API for accessing multiple databases
  - Each database provides a ODBC driver
  - Unfortunately, it’s not easy to set up ODBC on a Windows system even though Microsoft invented it
  - Another way: MDX driver to allow Excel to use live Oracle OLAP data
Star queries

- Oracle definition: a query that joins a large (fact) table to a number of small (dimension) tables, with provided WHERE predicates on the dimension tables to reduce the result set to a very small percentage of the fact table.
- The select list still has sum(sales), etc., as desired.

```
SELECT store.sales_district, 
    time.fiscal_period, SUM(sales.dollar_sales) 
FROM sales, store, time 
WHERE sales.store_key = store.store_key AND 
    sales.time_key = time.time_key AND 
    store.sales_district IN ('San Francisco', 
    'Los Angeles') AND time.fiscal_period IN ('3Q95', 
    '4Q95', '1Q96') 
GROUP BY 
    store.sales_district, time.fiscal_period;
```
Star queries

- **Oracle**: A better way to write the query would be: (i.e., give the QP a hint on how to do it)

```sql
SELECT ... FROM sales
WHERE store_key IN
  ( SELECT store_key FROM store
      WHERE sales_district IN ('WEST', 'SOUTHWEST'))
AND time_key IN
  ( SELECT time_key FROM time
      WHERE quarter IN ('3Q96', '4Q96', '1Q97'))
AND product_key IN
  ( SELECT product_key FROM product
      WHERE department = 'GROCERY')
GROUP BY ...;
```

- Oracle will rewrite the query this way if you add the STAR_TRANSFORMATION hint to your SQL, or the DBA has set STAR_TRANSFORMATION_ENABLED
Excel can do Star queries

• Recall GROUP BY queries for individual crosstab entries
• A Star query is of this form, plus WHERE clause predicates on dimension tables such as
  • store.sales_district IN ('WEST', 'SOUTHWEST')
  • time.quarter IN ('3Q96', '4Q96', '1Q97')
• Excel allows “filters” on data that correspond to these predicates of the WHERE clause
Indexes related to data warehousing

- New indexing techniques: Bitmap indexes, Join indexes, array representations, compression, precomputation of aggregations, etc.
- E.g., Bitmap index:

<table>
<thead>
<tr>
<th>custid</th>
<th>name</th>
<th>sex</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>112</td>
<td>Joe</td>
<td>M</td>
<td>3</td>
</tr>
<tr>
<td>115</td>
<td>Ram</td>
<td>M</td>
<td>5</td>
</tr>
<tr>
<td>119</td>
<td>Sue</td>
<td>F</td>
<td>5</td>
</tr>
<tr>
<td>112</td>
<td>Woo</td>
<td>M</td>
<td>4</td>
</tr>
</tbody>
</table>

**Bit-vector:**

1 bit for each possible value.

Many queries can be answered using bit-vector ops!
Bitmap Indexes

• A bitmap index uses one bit vector (BV) for each distinct keyval
• The number of bits = #rows
• Example of last slide, 4 rows, 2 columns with bitmap indexes
  • Sex = ‘M’: BV = 1101
  • Sex = ‘F’: BV = 0010
  • Rating = 3, BV = 1000
  • Rating = 4, BV = 0001
  • Rating = 5, BV = 0110
  Bitmap index for sex column
  Bitmap index for rating column
• Underlying idea: it’s not hard to convert between a table’s row numbers and the row RIDs
• RIDs have file#, page#, row# within page, where file# is fixed for one heap table, and page# ranges from 0 up to some limit.
• For the kind of read-mostly data that bitmap indexes are used, the pages are full, so the RIDs (page#, row# in a certain file) look like (0,0), (0,1), (0,2), (1,0), (1,1), ... easily converted to row indexes 0, 1, 2, 3, 4, 5, ... and back again
Bitmap Indexes

• Implementation: B+-tree of key values, bitmap for each key
• Size = \#values*\#rows/8 if not compressed
• Bitmaps can be compressed, done by Oracle and others
• Main restriction: slow row insert/delete, so NG for OLTP
  • But great for data warehouses:
  • Data warehouses are updated only periodically, traditionally
• Low cardinality (\#values in column) a clear fit
  • Example: rating, with 10 values
• But in fact, cardinality can be fairly high with compression
• **Oracle example**: bitmap index on unique column!
Bitmap Indexes

- Oracle: create bitmap index sexx on custs(sex);
- Bitmap indexes can be used with AND and OR predicates
- Example
  
  Select name from sailors s
  where s.rating = 10 and sex = 'M' or sex = 'F'

  \[ \text{BV1} \quad \text{BV2} \quad \text{BV3} \]

  \[ \text{ResultBV} = \text{BV1} \& \text{BV2} | \text{BV3} \]

  - Each bit on in ResultBV shows a row that satisfies the predicate
  - Loop through on-bits, finding rows and output name
Oracle Bitmap index plan

- EXPLAIN PLAN FOR SELECT * FROM t WHERE c1 = 2 AND c2 <> 6 OR c3 BETWEEN 10 AND 20;

- EXPLAIN PLAN FOR
- SELECT * FROM t WHERE c1 = 2 AND c2 <> 6 OR c3 BETWEEN 10 AND 20;
- SELECT STATEMENT
- TABLE ACCESS T BY INDEX ROWID
- BITMAP CONVERSION TO ROWID -- get ROWIDs for each on-bit
- BITMAP OR --top level OR
- BITMAP MINUS --to remove null values of c2
- BITMAP MINUS -- to calc c1 = 2 AND c2 <> 6
- BITMAP INDEX C1_IND SINGLE VALUE --c1= 2 BV
- BITMAP INDEX C2_IND SINGLE VALUE --c2 = 6 BV
- BITMAP INDEX C2_IND SINGLE VALUE --c2 = null BV (no not null on col)
- BITMAP MERGE --merge BV’s over C3 range
- BITMAP INDEX C3_IND RANGE SCAN
Bitmaps for star schemas, to be continued

• The dimension tables are not large, maybe 100 rows
• Thus the FK columns in the fact table have only 100 values
• Bitmap indexes can pinpoint rows once determined.
• Bitmaps can be AND’d and OR’d
• Example: `time.fiscal_period IN ('3Q95', '4Q95')` matches say 180 days in time table, so 180 FK values in fact’s time_key column
• OR together the 180 bitmaps, get a bitmap locating all fact rows that satisfy this predicate