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 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 the 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.

### 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

### 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, txid, clerkid, sales, ...

### 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:**
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

Dimension Hierarchies: OLAP, DW

- For each dimension, the set of values can be organized in a hierarchy:

```
<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>TIME</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>category</td>
<td>week</td>
<td>month</td>
</tr>
<tr>
<td>pname</td>
<td>date</td>
<td>timeid</td>
</tr>
<tr>
<td>pid</td>
<td>year</td>
<td>quarter</td>
</tr>
<tr>
<td>locid</td>
<td>price</td>
<td>locale</td>
</tr>
</tbody>
</table>
```

Schema underlying OLAP, used in RDB DW

```
timeid | date | week | month | quarter | year | holiday_flag
pid | timeid | local | sales

PRODUCTS
pid | pname | category | price | locid | city | state | country

SALES (Fact table)
```

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](https://en.wikipedia.org/wiki/Multidimensional_expression)
```
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 WHERE 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

```sql
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:

```
TIMES_PURCHASED 'NY'  'CT'  'NJ'   'FL'    'MO'
---------------
----------
----------
----------
----------
- 0
1 16601
90 165
0 0 0
1 33048
165 0 0 0
2 33151
179 0 0 0
3 33198
173 0 1 0
0
---
```

SQL Queries for cross-tab entries

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

```sql
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
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.timeid = L.timeid 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 Query, pg. 857

```sql
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:

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

Oracle 10+ supports CUBE queries

```sql
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):
```

```
YEAR STORE_STATE SUM(DOLLAR_SALES)
--------- -------------- ---------------
1994     AZ                 30854.0
1994     CA                 77020.82
1994     CO                 38313.26 [some rows deleted]
1994     TX                 45866.14
1994     NM                 28481.18
1994     AZ 1995.04
1994     CA 1996.04
1994     CO 1997.33
1994     DC 2000-01-18 ... from db2 output
```
**DW data → 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](https://www.youtube.com/watch?v=eGhjklYys6Y) of UCI (Cal state U/Channel Islands)
- 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 store_key = store.store_key AND
    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)
  - 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>sex</th>
<th>custid</th>
<th>name</th>
<th>sex</th>
<th>rating</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>112</td>
<td>Joe</td>
<td>M</td>
<td>3</td>
<td>00100</td>
</tr>
<tr>
<td>M</td>
<td>115</td>
<td>Ram</td>
<td>M</td>
<td>5</td>
<td>00001</td>
</tr>
<tr>
<td>F</td>
<td>119</td>
<td>Sue</td>
<td>F</td>
<td>5</td>
<td>00001</td>
</tr>
<tr>
<td>M</td>
<td>112</td>
<td>Woo</td>
<td>M</td>
<td>4</td>
<td>00010</td>
</tr>
</tbody>
</table>

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

- 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

- 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’ed 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