On-Line Analytical Processing (OLAP)

WANG, Yixuan, SIN, Wing Lam
On-Line Transactional Processing (OLTP)

Transaction-based queries
- INSERT
- UPDATE
- DELETE
- retrieving a small amount of records

Client-side System

Operational DB
We need to get the summary for the amount of products sold by each category in each region and in each month...

Problems:
- Complex queries require long computation time
- May block the service of the operational DB
We need to get the summary for the amount of products sold by **each category** in **each region** and in **each month**…
<table>
<thead>
<tr>
<th>Source</th>
<th>Operational Data</th>
<th>Consolidated Data (from OLTP DBs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Purpose</td>
<td>Support basic business operations</td>
<td>Help with planning, decision making</td>
</tr>
<tr>
<td>Data Content</td>
<td>Reveals a snapshot of ongoing business processes</td>
<td>Multidimensional views of business activities</td>
</tr>
<tr>
<td>Updates</td>
<td>Fast/Immediate</td>
<td>Periodic batch jobs</td>
</tr>
<tr>
<td>Queries</td>
<td>Simple, related to a few records</td>
<td>Complex, often involve aggregations</td>
</tr>
<tr>
<td>Speed</td>
<td>Fast</td>
<td>Update/Refresh takes long time; Query is faster (depending on the amount of records/dimension involved)</td>
</tr>
<tr>
<td>Space</td>
<td>Small (if historical data is archived)</td>
<td>Large (due to aggregation structures and historical data)</td>
</tr>
</tbody>
</table>

Source: http://datawarehouse4u.info/OLTP-vs-OLAP.html
Basic Architecture
Basic Flow: 1. Data Integration

1. Data Cleaning
To ensure data from multiple sources are consistent

a. **Data Migration**: use transformation rule
   - transform the string “men” to “male”

b. **Data Scrubbing**: use domain-specific knowledge to correct/remove the data
   - HKID: A123456(7) can be valid input
   - It is not a valid HKID number

c. **Data Auditing**: discover rules and relationship to examine variants

Basic Flow: 1. Data Integration

2. Loading
To load the data into data warehouse with additional process (e.g. checking IC, sorting)

- **Full Load** (building a new DB):
  - + current DB can still support queries
  - - take long time

- **Incremental load** (only update updated tuples):
  - + reduce data that need to update
  - - conflicts with ongoing queries

---

Basic Flow: 1. Data Integration

3. Refreshing
To propagate updates on source data to the base data and derived data in the data warehouse

- Refresh policy (when and how to refresh) is customized according to user needs and traffic

OLAP Data Structure

- **MOLAP** - Multi-dimensional OLAP
  - Data Cube

- **ROLAP** - Relational OLAP
  - Star Schema
  - Snowflakes Schema

- **HOLAP** - Hybrid OLAP
  - storing part of the data in a multidimensional database
  - another part of the data in a relational database
OLAP Data Structure

- **MOLAP**
  - Multidimensional OLAP
  - Data Cube

- **ROLAP** - Relational OLAP

- **HOLAP** - Hybrid OLAP

Figure from [CHASS Data Centre](https://www.chassdatacentre.ca) Apr. 10 2018
MOLAP- Multidimensional OLAP

- Data stored in Data Cubes
- Pre-computed and aggregated data
- Rapid query response
- May introduce data redundancy

Chapter I Conceptual Modeling Solutions for the Data Warehouse - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/The-cube-metaphor-for-multidimensional-modeling_fig1_237622501 [accessed 12 Apr, 2018]
OLAP Data Structure

- **MOLAP** - Multidimensional OLAP
- **ROLAP** - Relational OLAP
  - Fact + Dimensions
  - Star Schema
  - Snowflakes Schema
- **HOLAP** - Hybrid OLAP

OLAP Data Structure

- **MOLAP**
  - Multidimensional OLAP

- **ROLAP** - Relational OLAP
  - Fact + Dimensions
  - Star Schema
  - Snowflakes Schema

- **HOLAP** - Hybrid OLAP

OLAP Data Structure

- **MOLAP** - Multidimensional OLAP
- **ROLAP** - Relational OLAP
- **HOLAP** - Hybrid OLAP
  - storing part of the data in a multidimensional database
  - another part of the data in a relational database

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Schema</th>
<th>Dimension</th>
<th>DB size</th>
<th>Access</th>
<th>Flexibility</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROLAP</td>
<td>Star Schema</td>
<td>Can add dynamically</td>
<td>Medium-Large</td>
<td>Ad-hoc requests</td>
<td>High</td>
<td>Good with small data sets</td>
</tr>
<tr>
<td>MOLAP</td>
<td>Data Cube</td>
<td>Recreate data cube</td>
<td>Small-Medium</td>
<td>Pre-defined dimensions</td>
<td>Low</td>
<td>Faster for small/medium</td>
</tr>
</tbody>
</table>

Average for medium/large
OLAP Operations

- Aggregation: Roll up
- Navigation to detail: Drill down
- Define a sub-cube: Slice & Dice
- Visualization operation: Pivot
OLAP Operation: Roll up & Drill Down

OLAP Operation: Slice & Dice

Slice

Dice

Example: OLAP Operations with SQL

Suppose the sales amount of markets among regions are stored into

- Sales(Market_Id, Product_Id, Time_Id, Sales_Amt)
- Market(Market_Id, City, State, Region)
- Time(Time_Id, Week, Month, Quarter)
- Product(Product_Id, Name, Category, Price)

```
SELECT S.Product_Id, M.Region, SUM (S.Sales_Amt)
FROM Sales S, Market M
WHERE M.Market_Id = S.Market_Id
GROUP BY S.Product_Id, M.Region
```

```
SELECT S.Product_Id, M.State, SUM (S.Sales_Amt)
FROM Sales S, Market M
WHERE M.Market_Id = S.Market_Id
GROUP BY S.Product_Id, M.State
```
Example: OLAP Operations with SQL

Suppose the sales amount of markets among regions are stored into:
- Sales(Market_Id, Product_Id, Time_Id, Sales_Amt)
- Market(Market_Id, City, State, Region)
- Time(Time_Id, Week, Month, Quarter)
- Product(Product_Id, Name, Category, Price)

```
SELECT S.Product_Id, SUM(Sales_Amt)
FROM Sales S, Time T
WHERE T.Time_Id = S.Time_Id AND T.Week = "Week12"
GROUP BY S.Product_Id
```

**Slicing**

**Pivot**: View the data as a multi-dimensional cube and group on a subset of the axes.
Brief Intro to an OLAP example: Pinot

Distributed Multi-dimensional OLAP
Columnar + indexes
No joins
Latency: low ms to sub-second

At LinkedIn, it powers more than 50+ applications such as:

Who Viewed My Profile,
The Jobs you may be interested in,
Ad campaign creation and tracking

Pinot Query Execution:

1. **Query**
2. **Fetch routing table from Helix**
3. **Scatter Request**
4. **Process Request**
5. **Gather Response**
6. **Return Response**

Who Viewed My Profile: ProfileViews Metric

Count up the records of the tracking data is being emitted by the profile viewed.

```
ProfileViewEvent
Record 1:
{   ...
    "pageKey": "profile_page",
    ...
}

Metric
ProfileViews = sum(ProfileViewEvent
    where pagekey=profile_page
    or new profile_page
)
```

Reference: Strata 2017: Building a healthy data ecosystem around Kafka and Hadoop: Lessons learned at LinkedIn
Improving OLAP Performance: Materializing Views

- To store query results (usually involve aggregation) as materialized views in Data Warehouse for faster query in the future

- Problem:
  - A huge amount of possible views
  - Impossible to store all the views in the data warehouse

Steps for View Materialization

1. **View Size Estimation**
   - Examines the distribution and estimate the expected size of the view

2. **View Selection**
   - Select views by balancing the capacity and benefits (methods will be discussed in the next slides)

3. **View Maintenance**
   - Update the view when there are updates

4. **Query Optimization**
   - Consider which of the materialized views can be used to answer the queries
View Selection Algorithms:

1. Greedy Algorithm on Hypercube Lattice Structure (HRU)

Select \( \text{SUM(amount)} \) FROM Sale_Record GROUP BY Customer, Part, Supplier

Edge: \{p, s\} can be derived from \{c, p, s\}
View Selection Algorithms: HRU

Benefit calculation:
- If we materialize \{p,s\}, the result \{s\}, \{p\} and {}, can be derived from \{p,s\}
- \(= (6M(\text{Original Cost } \{c, p, s\}) - 0.8M(\text{Cost of } \{p,s\})) \times 4\)
View Selection Algorithms: HRU

Benefit calculation:
- If we materialize \{p,s\}, the result \{s\}, \{p\} and \{}, can be derived from \{p,s\}
- \(= (6M(\text{Original Cost \{c, p, s\}}) - 0.8(\text{Cost of \{p,s\}})) \times 4\)

### Iteration 1 Benefit

<table>
<thead>
<tr>
<th>Set</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>{c, p, s}</td>
<td>5.2M x 4 = 20.8M</td>
</tr>
<tr>
<td>{c, s}</td>
<td>0 x 4 = 0</td>
</tr>
<tr>
<td>{c, p}</td>
<td>0 x 4 = 0</td>
</tr>
<tr>
<td>{s}</td>
<td>5.99M x 2 = 11.98M</td>
</tr>
<tr>
<td>{p}</td>
<td>5.8M x 2 = 11.6M</td>
</tr>
<tr>
<td>{c}</td>
<td>5.9M x 2 = 11.8M</td>
</tr>
<tr>
<td>{}</td>
<td>6M - 1</td>
</tr>
</tbody>
</table>

### Iteration 2 Benefit

<table>
<thead>
<tr>
<th>Set</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>{p, s}</td>
<td>0 x 2 = 0</td>
</tr>
<tr>
<td>{c, s}</td>
<td>0 x 2 = 0</td>
</tr>
<tr>
<td>{c, p}</td>
<td>0.79M x 2 = 1.58M</td>
</tr>
<tr>
<td>{s}</td>
<td>0.6M x 2 = 1.2M</td>
</tr>
<tr>
<td>{p}</td>
<td>5.9M x 2 = 11.8M</td>
</tr>
<tr>
<td>{}</td>
<td>0.8M - 1</td>
</tr>
</tbody>
</table>
View Selection Algorithms:

2. Polynomial Greedy Algorithm

View Selection Algorithms: PGA

View Selection Algorithms: PGA

View Selection Algorithms: PGA

View Selection Algorithms:

2. Polynomial Greedy Algorithm

View Selection Algorithms: PGA

View Selection Algorithms: PGA

View Selection Algorithms: PGA

Materializing Data Cubes

Steps:
1. Defining a set of initial data cube
   - Each data cube is corresponding to each frequently asked query

2. Optimizing the data cube set by merging redundant data cubes
   - overlapping or query can be derived from other cubes in the set
     a. Query Level
        Removing as few queries as possible
     b. Attribute Level
        Answering all queries left with minimum maintenance-cost

Materializing Data Cubes

2. Optimization Step (Query Level):
   ● **Optimal Removing (OR)**
     - Start with all possible combinations of queries from initial data cube set Q
     - Check if a set C of data cubes that can answer query set P and lower than the maintenance-cost bound is found
       - If not, return the bound is too strict
     - Check if the cost of answering all queries in P using C is smaller than the query-cost bound
       - If yes, return C
       - If not, decrease the size of the cube set Q by 1, and repeat the steps

Materializing Data Cubes

2. Optimization Step (Query Level):
   ● **Greedy Removing (GR)**
     ○ Check if keeping all data cubes $Q$ fit in the maintenance-cost bound
       ■ If yes, return $Q$
       ■ If not, consider all possible ways of removing one data cube
         ● Remove a data cube $c$ from $C$ if the resulting set $C\{-c\}$ gives us a minimum query cost
         ● repeat until a set $C$ fit in the maintenance-cost bound and query-cost bound

Materializing Data Cubes

2. Optimization Step (Attribute Level):

● **Optimal Merging (OM)**
  ○ Start with all possible groupings of data cubes \( G \) from data cube set \( Q \)
  ○ Check if a set \( H \) from \( G \) is lower than the maintenance-cost bound and query-cost bound
    ■ If yes, assign \( H \) to \( C \)
  ○ Return set \( C \) or empty when the bounds are too strict that an empty set is resulted
Materializing Data Cubes

2. Optimization Step (Attribute Level):
   - **2-Greedy Merging (2GM)**
     - Select two data cube set D such that D obtain maximum value in the evaluation
     - Remove D from C and add A (obtained by merging two cubes in D) to C
     - Check if the maintenance-cost bound is satisfied
       - If yes, return C
       - If not, repeat until only one data cube remains, then return an empty set

Materializing Data Cubes

2. Optimization Step (Attribute Level):

- **2-Greedy Merging with Multiple paths (2GMM)**
  - Similar to 2GM
  - 2GM can be viewed as finding a path of data cube set C to satisfy maintenance-cost bound
  - Every intermediate node (data cube sets produced from merging two cubes)
  - 2GMM gives two/more branches
  - If a solution is found, record its query cost
    - Branch will stop growing if its query cost exceed the recorded minimum
  - Solution with the minimum query cost is returned

Materializing Data Cubes

Table 1. Table of execution time of 2GM, 2GMM and OM (in sec.) \((MC_1 \text{ by Formula 2.2})\) in the data set DS1.

<table>
<thead>
<tr>
<th>Program</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>2GM</td>
<td>0.007</td>
<td>0.012</td>
<td>0.013</td>
<td>0.028</td>
<td>0.089</td>
<td>0.067</td>
<td>0.143</td>
<td>0.285</td>
<td>0.200</td>
<td>0.245</td>
</tr>
<tr>
<td>2GMM</td>
<td>0.040</td>
<td>0.048</td>
<td>0.255</td>
<td>0.155</td>
<td>0.677</td>
<td>0.200</td>
<td>0.403</td>
<td>0.688</td>
<td>0.357</td>
<td>0.837</td>
</tr>
<tr>
<td>OM</td>
<td>0.01</td>
<td>0.01</td>
<td>0.15</td>
<td>0.70</td>
<td>5.10</td>
<td>27.8</td>
<td>185.7</td>
<td>1444</td>
<td>9100</td>
<td>159019</td>
</tr>
</tbody>
</table>