Armazéns de Dados

Departamento de Engenharia Informática (DEI/ISEP)
Paulo Oliveira
pjo@isep.ipp.pt

Adaptado do Original de:
Fátima Rodrigues (DEI/ISEP)

1

Data Warehouse Optimization

2

Bibliography

- The Data Warehouse Lifecycle Toolkit : Expert Methods for Designing, Developing, and Deploying Data Warehouses
  Ralph Kimball, Laura Reeves, Margy Ross, Warren Thornthwaite
  Wiley, 1998

  Chapters 14, 15

- Modern Database Management
  J.Hoffer, M.Prescott, H. Topi
  Prentice Hall, 2008

  Chapter 6

3

3

Indexing

4

Indexes

- In almost all DWs, the **size of dimension tables is insignificant compared with the size of fact tables**
- **Second biggest thing in a DW** is the **size of indexes of the fact tables**
- Indexing mechanisms are used to **speed up access** to desired data
- Index file consists of records (called index entries) of the form:

search-key	pointer

- **Search key** is **an attribute** or **set of attributes** used to **look up records** in a file and **pointer points to the record** in the data table
- One of the most powerful capabilities of indexed file organization is the ability to create **multiple indexes**

Index Types

- **B-Tree index**
  - Adequate for **high cardinality attributes**
  - May be built **on multiple columns**
  - Is the **default index type** for most databases, created on the primary key of a table
- **Bitmap index**
  - Adequate for **not high cardinality attributes** (*e.g.*: marital status)
  - Are a **major advance in indexing** that benefit DW applications
  - Are used both with **dimension tables and fact tables**, where the constraint on the table results in a **not high cardinality**
- **Others**
  - **Hash indexes** - array of *n* buckets or slots, each one containing a pointer to a row.
  - Some DBMSs use **additional index structures or optimizations** strategies **adequate to the n-way join** problem, inherent to a star query (*e.g.*: Red Brick: star indexes).

Bitmap Index

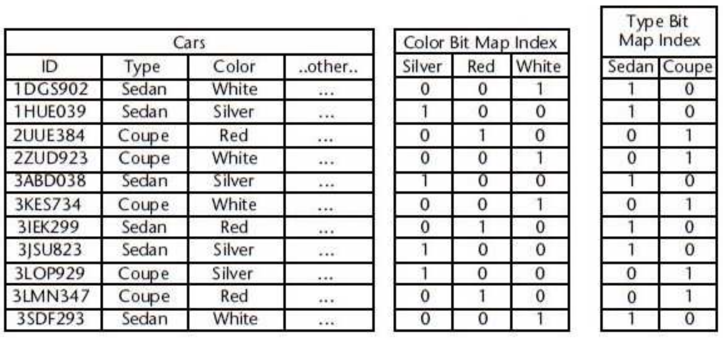
- Bitmap is simply an **array of bits**
- Bitmap index on an attribute has a **bit for each value of the attribute**
  - Bitmap has as many bits as distinct values
  - In a Bitmap for value v, the bit for a record is 1 if the record has the value v for the attribute, and is 0 otherwise

Id_Client	Gender	City	Income Level
145023	M	Brooklin	L1
145025	F	Jonestown	L2
154265	F	Perryridge	L4
265453	M	Brooklin	L1
645654	F	Perryridge	L3

B-Tree index   Bitmap index

Bitmap Index Structure

- Bitmap index for *Color* and *Type*



Cars				Color Bit Map Index			Type Bit Map Index	
ID	Type	Color	..other..	Silver	Red	White	Sedan	Coupe
1DGS902	Sedan	White	...	0	0	1	1	0
1HUE039	Sedan	Silver	...	1	0	0	1	0
2UUE384	Coupe	Red	...	0	1	0	0	1
2ZUD923	Coupe	White	...	0	0	1	0	1
3ABD038	Sedan	Silver	...	1	0	0	1	0
3KES734	Coupe	White	...	0	0	1	0	1
3IEK299	Sedan	Red	...	0	1	0	1	0
3JSU823	Sedan	Silver	...	1	0	0	1	0
3LOP929	Coupe	Silver	...	1	0	0	0	1
3LMN347	Coupe	Red	...	0	1	0	0	1
3SDF293	Sedan	White	...	0	0	1	1	0

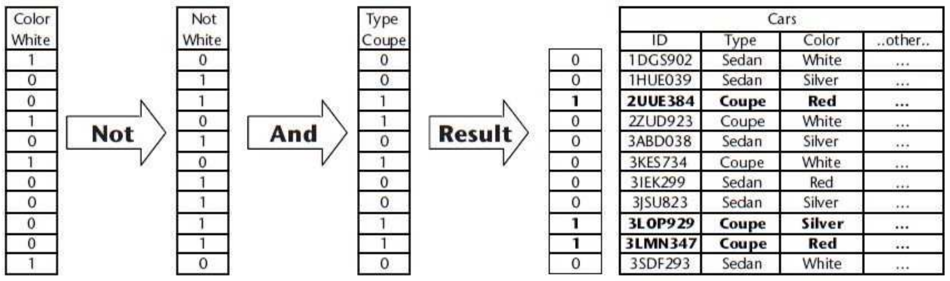
- Bitmap index **often requires less storage space** than a conventional B-tree index
- For an attribute with **many distinct values**, can **exceed the storage space** of a conventional B-tree index

Using Bitmap Indexes

- Bitmap indexes **are useful for queries on multiple attributes** but not particularly useful for single attribute queries
- Queries are answered using **bitmap operations**
  - Intersection (AND)
  - Union (OR)
  - Negation (NOT)
- Each operation takes a bitmap vector and **applies the operation** to get the result bitmap vector
- Bit manipulation and searching is so fast, that the speed of **query processing with a bitmap index can be 10 times faster** than with a conventional B-tree index
- Databases support: DB2; Informix; Ingres; Oracle; PostgreSQL

Queries on Bitmap Indexes

- Example of a query: find all cars that are not white and which are coupes



Color White	Not White	Type Coupe	Result	ID	Type	Color	..other..
1	0	0	0	1DGS902	Sedan	White	...
0	1	0	0	1HUE039	Sedan	Silver	...
0	1	1	**1**	**2UUE384**	**Coupe**	**Red**	**...**
1	0	1	0	2ZUD923	Coupe	White	...
0	1	0	0	3ABD038	Sedan	Silver	...
1	0	1	0	3KES734	Coupe	White	...
0	1	0	0	3IEK299	Sedan	Red	...
0	1	0	0	3JSU823	Sedan	Silver	...
0	1	1	**1**	**3LOP929**	**Coupe**	**Silver**	**...**
0	1	1	**1**	**3LMN347**	**Coupe**	**Red**	**...**
1	0	0	0	3SDF293	Sedan	White	...

- Database is able to resolve the query using the **bitmap vectors** and **Boolean operations**
- It does not need to **touch the data** until it has **isolated the rows that answer the query**

Indexing Dimension Tables

- Dimension tables have a **single column primary key** – must have **one unique index on that key**
- Small dimension tables seldom benefit from additional indexing
- Large dimension tables (*e.g.*: customer, product)
  - **Single-column bitmap or B-tree indexes** on **dimension attributes** that are most commonly used for:
    - **applying filters** (only makes sense in ROLAP)
    - **grouping** (only makes sense in ROLAP)
    - **used in a join condition**

Indexing Fact Tables

- Fact table index must be a **B-Tree index** on the **primary key**
- Primary key, and the primary key index, must have **date-key in the first position** in the primary key index
  - Incremental loads are keyed by date
  - Most DW queries are constrained by date
- Create a **single-column index on each fact table key** and let the optimizer combine those indexes as appropriate to answer the queries
  - Only makes sense in ROLAP
- If many queries constrain fact column values (amount, quantity) they must also be included in indexes – **non-key fact table indexes** are **single-column indexes**
  - Only makes sense in ROLAP

Aggregates

13

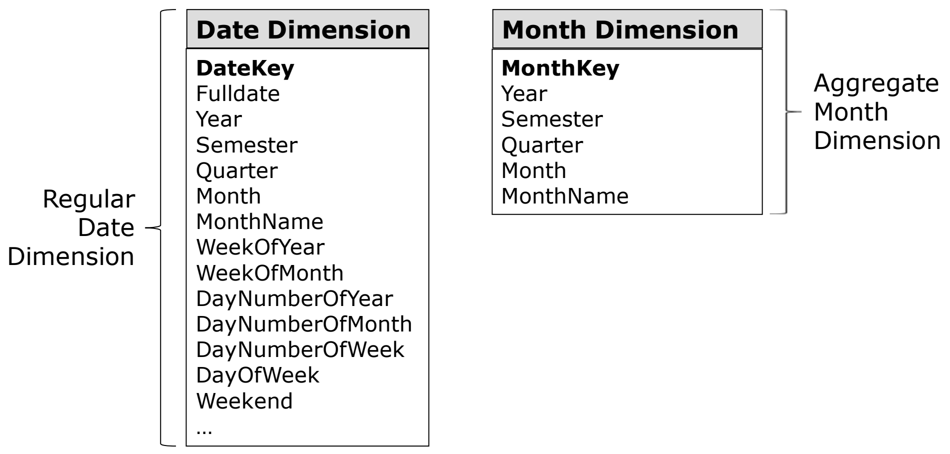
Aggregates

- **Summarization of a set of measures**, stored into fact tables with the purpose of **accelerating queries**
- Always associated with **one or more dimensions** that are **aggregated or not**
  - Example: **Sales** by ***product category***, by ***store*** , by ***date***

    Fact aggregate dimension dimensions
- Can have a very significant **effect on performance**
  - Speeding queries by a factor that goes from 100 to 1000
  - No other means exist to provide such performance gain

14

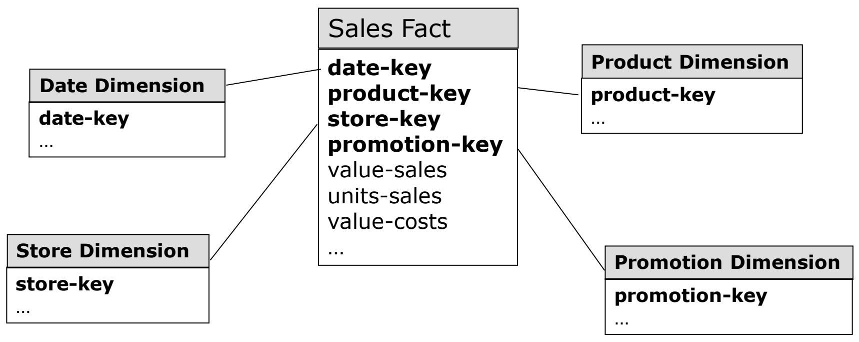
Regular vs Aggregate Dimension



Aggregates Advantages/Disadvantages

- **Advantages**
  - Improve performance of the DW
  - Transparent to end-users and applications
  - Shared by many users
- **Disadvantages**
  - Only speed up the answers for the questions previously known, i.e., previously calculated and stored
  - Need constant attention by the DW administrator to:
    - Build new aggregates adequate to the more frequent questions made by the users
    - Eliminate aggregates that are not useful
  - Spend disk space

Aggregates: Grocery Store



- Aggregates:
  - Totals by category (Product dimension)
  - Totals by region (Store dimension)
  - Totals by month (Date dimension)
- How many **aggregate fact tables** are necessary?

Final Scheme: Grocery Store

- **Star base**: Fact table + 4 dimensions
- **3 aggregate dimensions**:
  - Category (Product dimension)
  - Region (Store dimension)
  - Month (Date dimension)
- **Aggregate fact tables**:
  1. Totals by category, store, year
  2. Totals by category, store, month
  3. Totals by category, region, year
  4. Totals by category, region, month
  5. Totals by region, product, year
  6. Totals by region, product, month
  7. Totals by store, product, year
  8. …

These tables are not visible to end-users

Deciding What to Aggregate

- Two different areas need to be considered when selecting which aggregates to build
  - **Common business users' requests**
    - Major geographic groupings (regions, districts, …)
    - Major product groupings (category, subcategory, …)
    - Regular reporting time periods groupings (months, quarters, …)
    - …
    - Combinations of these attributes
  - **Statistical distribution of data**

19

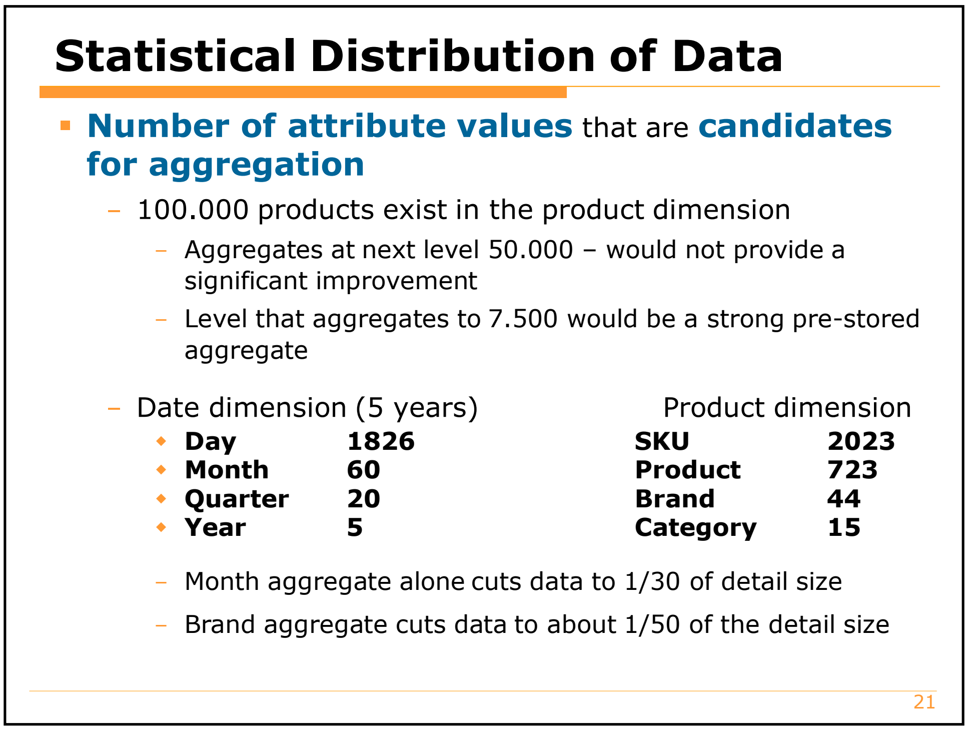
19

Common Business Users’ Requests

- Reviews should be performed to determine **which attributes are commonly used for grouping**, considering:
  - Each attribute individually
  - Combinations of attributes
    - Within a dimension
    - Among dimensions
- Not all attributes or combinations of attributes **are used together**

20

20

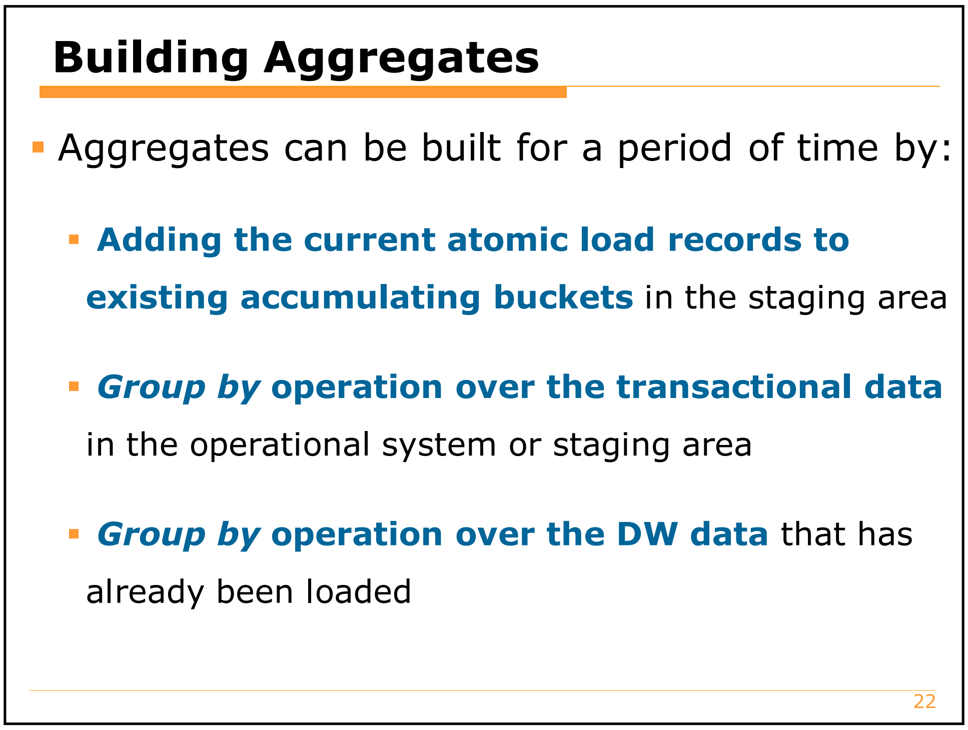


Statistical Distribution of Data

- **Number of attribute values** that are **candidates for aggregation**
  - 100.000 products exist in the product dimension
    - Aggregates at next level 50.000 – would not provide a significant improvement
    - Level that aggregates to 7.500 would be a strong pre-stored aggregate
  - Date dimension (5 years)

Date dimension (5 years)		Product dimension	
Day	**1826**	**SKU**	**2023**
Month	**60**	**Product**	**723**
Quarter	**20**	**Brand**	**44**
Year	**5**	**Category**	**15**

- Month aggregate alone cuts data to 1/30 of detail size
- Brand aggregate cuts data to about 1/50 of the detail size

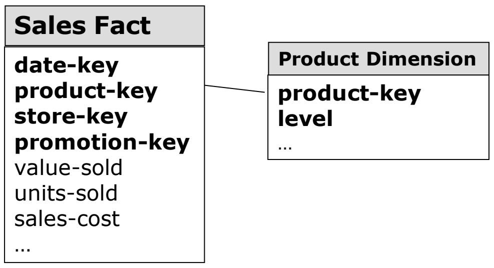


Building Aggregates

- Aggregates can be built for a period of time by:
  - **Adding the current atomic load records to existing accumulating buckets** in the staging area
  - ***Group by* operation over the transactional data** in the operational system or staging area
  - ***Group by* operation over the DW data** that has already been loaded

Techniques to Store Aggregates

1. Aggregate facts and aggregate dimensions are stored in **new tables, separated from the base atomic data**
2. Aggregate facts are stored in the atomic fact table, and **level attributes** are stored in the dimensions
   - *Level* attribute show the aggregation level of each row



- Original rows are filled with *level* = 'Base'
- Aggregates for *Category* are filled with the *level* = '*Category*'

Comparing the Two Methods

- Number of records created **is the same** in both methods
- **Separated Tables**
  - Tables that correspond to the aggregates are not visible to end-users
  - Aggregates in separated tables can be easily created, deleted, loaded and indexed
- **Level Attributes**
  - Can conduct a **double-count additive facts totals** – all the queries must restrict the level attribute – if not, all the values are included/added
  - **Wasting disk storage** – adding the aggregates in the base fact table implies to increase the attribute width for all the records
  - **Dimension tables are more complicated** – for the records corresponding to the aggregates, many attributes are filled with '*not applicable*' or with *null*