Data Warehousing and OLAP: MOLAP and ROLAP

dr. Toon Calders
t.calders@tue.nl

Previous Lectures

- Online analytical processing
  - Data cubes as a conceptual model
  - Query languages for data cubes
- Database explosion problem
  - Materializing the complete cube is impossible
  - Partially materializing can help

This Lecture

- How is the data stored?
  - relational database (ROLAP)
  - Specialized structures (MOLAP)

- How can we speed up computation?
  - Indexing structures
    - bitmap index
    - join index

Implementation

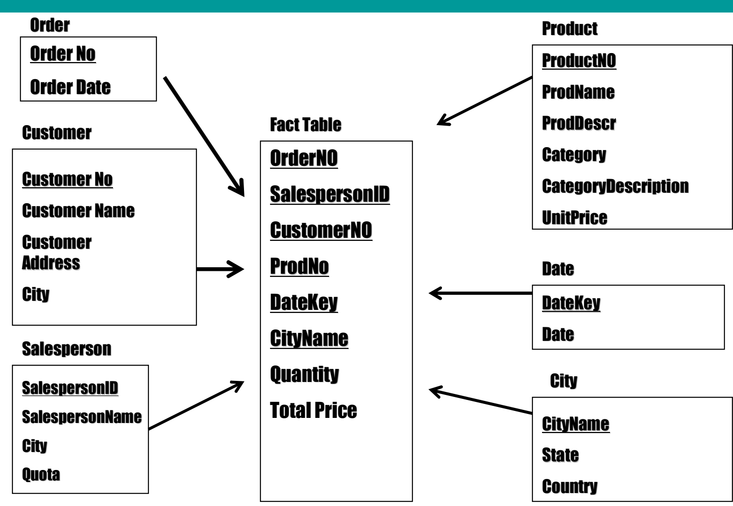
Nowadays systems can be divided in three categories:

- ROLAP (Relational OLAP)
  - OLAP supported on top of a relational database
- MOLAP (Multi-Dimensional OLAP)
  - Use of special multi-dimensional data structures
- HOLAP: (Hybrid)
  - combination of previous two

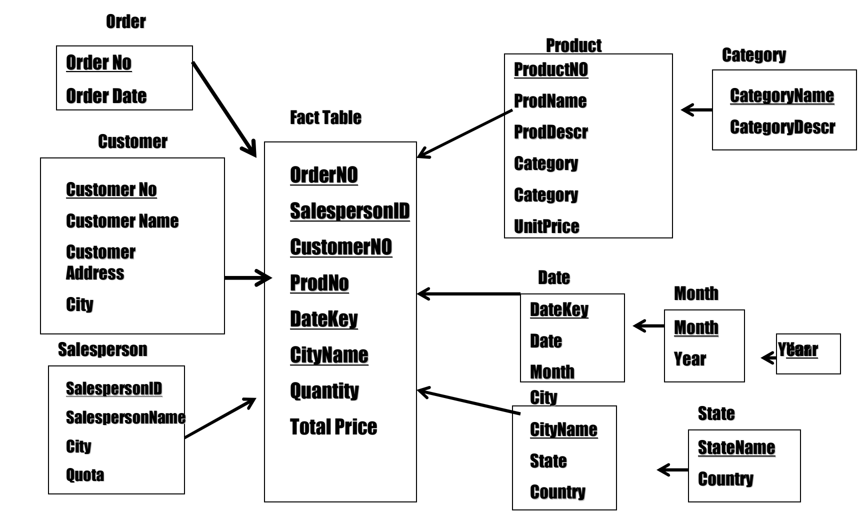
ROLAP

- Typical database scheme:
  - star schema
    - fact table is central
    - links to dimensional tables
  - Extensions:
    - snowflake schema
      - dimensions have hierarchy/extra information attached
    - Star constellation
      - multiple star schemas sharing dimensions

Example of a Star Schema

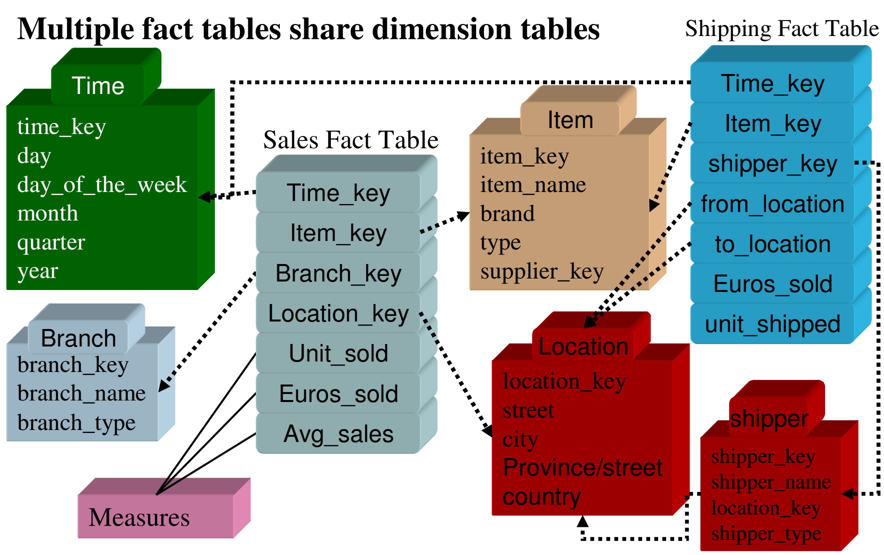


Example of a Snowflake Schema



Example of Fact Constellation

Multiple fact tables share dimension tables



This Lecture

- How is the data stored?
  - Relational database (ROLAP)
  - **Specialized structures (MOLAP)**
- How can we speed up computation?
  - Indexing structures
    - bitmap index
    - join index

MOLAP

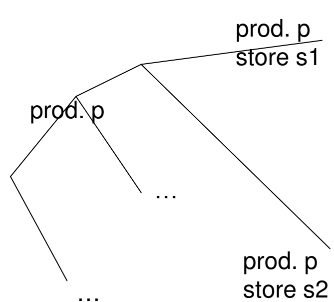
- Not on top of relational database
  - most popular design
  - specialized data structures
    - Multicubes vs Hypercubes
  - Not all subcubes are materialized

Storing the cube

- User identifies set of *sparse* attributes S, and a set of *dense* attributes D.
- Index tree is constructed on sparse dimensions.
- Each leaf points to a multidimensional array indexed by D.

Example

- product, store are sparse dimensions
- date and customer-type are dense

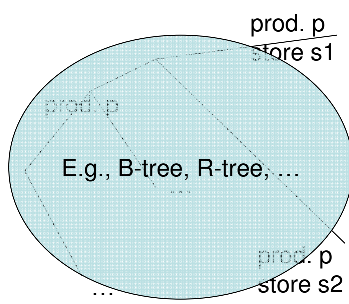


	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

Example

- product, store are sparse dimensions
- date and customer-type are dense

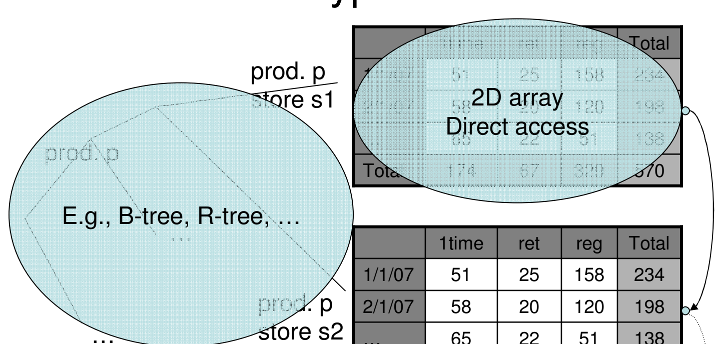


	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

Example

- product, store are sparse dimensions
- date and customer-type are dense

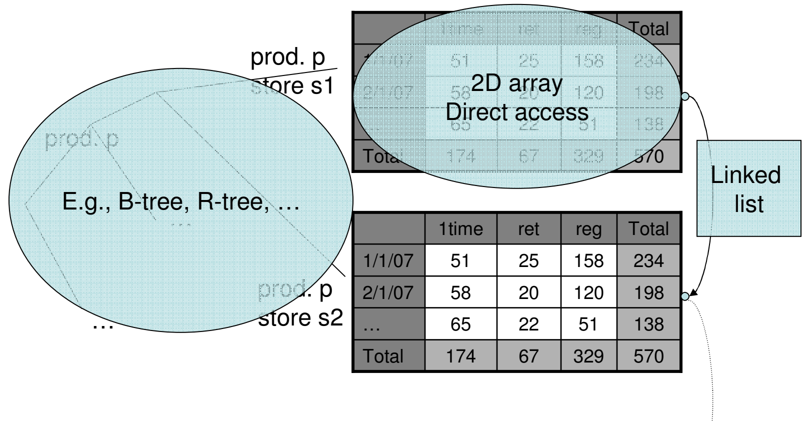


	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

Example

- product, store are sparse dimensions
- date and customer-type are dense



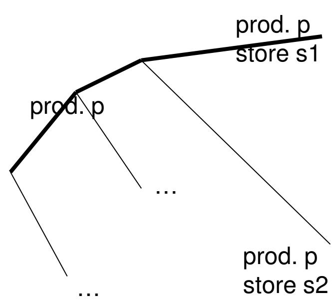
	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

Queries

- Efficiency depends on:
  - does index on sparse dimensions fit into memory?
  - Type of queries:
    - Restrictions on all dimensions
    - Restrictions only on dense
    - Restrictions only on some sparse and dense

Queries

- Selection on all attributes: (p,s1,ret,all)

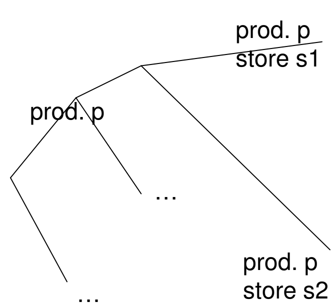


	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

Queries

- Only on dense attributes: (-,-,ret,"2/1/07")

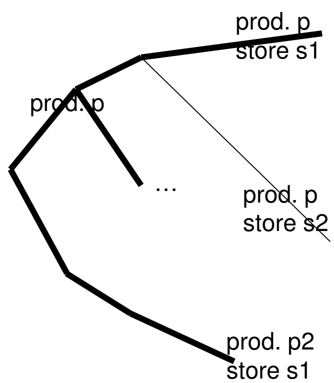


	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

Queries

- Only some sparse and dense attributes: (-,s1,ret,"2/1/07")

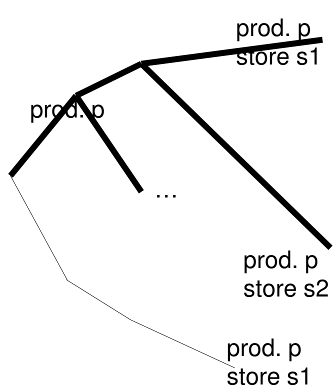


	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

Queries

- Only some sparse and dense attributes: (p,-,-,"2/1/07")



	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

	1time	ret	reg	Total
1/1/07	51	25	158	234
2/1/07	58	20	120	198
…	65	22	51	138
Total	174	67	329	570

Storing the Cube

- Dense combinations of dimensions can be stored in multi-dimensional arrays
- For every combination of sparse dimensions
  - one sub-cube
- Sub-cubes indexed by sparse dimensions
  - E.g., B-tree
  - Order of the dimensions plays a role

This Lecture

- How is the data stored?
  - relational database (ROLAP)
  - Multi-dimensional structure (MOLAP)
- How can we speed up computation?
  - Indexing structures
    - bitmap index
    - join index

Specialized Indexing Structures

- B-trees, (covered in other courses)
- Bitmapped indices,
- Join indices,
- Spatial data structures (covered later)

Index Structures

- Indexing principle:
  - mapping key values to records for associative direct access
- Most popular indexing techniques in relational database: B+-trees
- For multi-dimensional data, a large number of indexing techniques have been developed: R-trees

Bitmap Indexes

- Bitmap index: indexing technique that has attracted attention in multi-dimensional DB implementation

table

Customer	City	Car
c1	Detroit	Ford
c2	Chicago	Honda
c3	Detroit	Honda
c4	Poznan	Ford
c5	Paris	BMW
c6	Paris	Nissan

Bitmap Indexes

- The index consists of bitmaps:

ec1	Chicago	Detroit	Paris	Poznan
1	0	1	0	0
2	1	0	0	0
3	0	1	0	0
4	0	0	0	1
5	0	0	1	0
6	0	0	1	0

bitmaps

ec1	BMW	Ford	Honda	Nissan
1	0	1	0	0
2	1	0	1	0
3	0	0	1	0
4	0	1	0	0
5	1	0	0	0
6	0	0	0	1

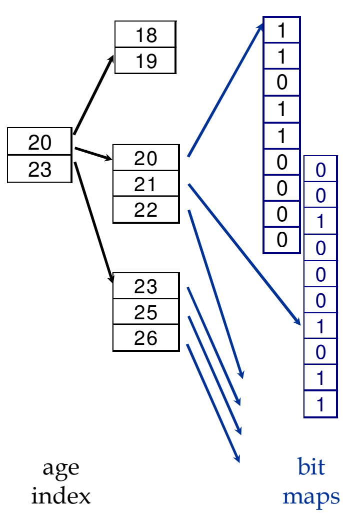
bitmaps

- Index on a particular column
- Index consists of a number of bit vectors - bitmaps
- Each value in the indexed column has a bit vector (bitmaps)
- The length of the bit vector is the number of records in the base table
- The i-th bit is set if the i-th row of the base table has the value for the indexed column

Bitmap Indexes

- Index on a particular column
- Index consists of a number of bit vectors - bitmaps
- Each value in the indexed column has a bit vector (bitmaps)
- The length of the bit vector is the number of records in the base table
- The *i*-th bit is set if the *i*-th row of the base table has the value for the indexed column

Bitmap Indexes



id	name	age
1	joe	20
2	fred	20
3	sally	21
4	nancy	20
5	tom	20
6	pat	25
7	dave	21
8	jeff	26

⋮

data records

Query:
Get people with age = 20 and name = "fred"

List for age = 20:
1101100000
List for name = "fred":
0100000001
Answer is intersection:
0100000000

Suited well ***for domains with small cardinality***

Bitmap Index

- Size of bitmaps can be further reduced
  - use run-length encoding

    1111000111100000001111000 is encoded as
    4x1;3x0;4x1;7x0;4x1;3x0

  - Can reduce the storage space significantly
  - Logical operations can work directly on the encoding

Bitmap Index – Summary

- With efficient hardware support for bitmap operations (AND, OR, XOR, NOT), bitmap index offers better access methods for certain queries
  - e.g., selection on two attributes
- Some commercial products have implemented bitmap index
- Works poorly for high cardinality domains since the number of bitmaps increases
- Difficult to maintain - need reorganization when relation sizes change (new bitmaps)

This Lecture

- How is the data stored?
  - relational database (ROLAP)
  - Specialized structures (MOLAP)
- How can we speed up computation?
  - Indexing structures
    - bitmap index
    - join index

Join Indexes

- **Traditional indexes**: value → rids.
  Join indices: tuples in the join → to rids in the source tables.
- Data warehouse:
  - values of dimensions of star schema → rows in fact table.
- Join indexes can span multiple dimensions

Join

- "Combine" SALE, PRODUCT relations
- In SQL: SELECT * FROM SALE, PRODUCT

sale	prodId	storeId	date	amt
	p1	c1	1	12
	p2	c1	1	11
	p1	c3	1	50
	p2	c2	1	8
	p1	c1	2	44
	p1	c2	2	4

product	id	name	price
	p1	bolt	10
	p2	nut	5

joinTb	prodId	name	price	storeId	date	amt
	p1	bolt	10	c1	1	12
	p2	nut	5	c1	1	11
	p1	bolt	10	c3	1	50
	p2	nut	5	c2	1	8
	p1	bolt	10	c1	2	44
	p1	bolt	10	c2	2	4

Join Indexes

join index

product	id	name	price	jIndex
	p1	bolt	10	r1,r3,r5,r6
	p2	nut	5	r2,r4

sale	rId	prodId	storeId	date	amt
	r1	p1	c1	1	12
	r2	p2	c1	1	11
	r3	p1	c3	1	50
	r4	p2	c2	1	8
	r5	p1	c1	2	44
	r6	p1	c2	2	4

OLAP - Summary

- Data warehouse is a specialized database to support analytical queries = OLAP queries
- Data cube as conceptual model
- Implementation of Data Cube
  - View selection problem
  - Explosion problem
  - ROLAP vs. MOLAP
  - Indexing structures