Physical Design

Elena Baralis
Politecnico di Torino

Physical Design

- Workload Characterization:
  - Queries with aggregates that require access to a large portion of each table
  - Read-only access
  - Periodic update of data with eventual rebuilding of access physical structures (indexes, views, etc.)
- Physical Structures
  - Non-traditional types of indexes
    - Bitmap indexes, Join indexes, Bitmapped join indexes
    - B^+-tree index is not well-suited for:
      - Attributes with low cardinality domain
      - Queries with low selectivity
  - Materialized views:
    - Require the presence of an optimizer able to exploit them

Elena Baralis
Politecnico di Torino

Physical Design

- Optimizer characteristics
  - Must consider statistics while defining data access strategy (cost based)
  - Capability of aggregate navigation
- Physical design procedure
  - Selection of suited data structures to support the most frequent queries (or the most relevant)
  - Choice of structures that contribute to improve more queries at a time
  - Constraints:
    - Disk space
    - Available time for updating data

Elena Baralis
Politecnico di Torino

Physical Design

- Tuning:
  - *A posteriori* variation of support physical structures
  - Requires tools for workload monitoring
  - Often required for OLAP applications
- Parallelism
  - Data fragmentation
  - Queries parallelization
    - inter-query
    - intra-query
  - Join and Group By operations suitable to parallel execution

Elena Baralis
Politecnico di Torino

Physical Access Structures

- Physical access structures describe how data is stored on disk to provide efficient query execution
  - SQL select, update, …
- In relational systems
  - Physical data storage
    - Sequential structures (heap file, ordered sequential structure)
    - Hash structures
  - Indexing to increase access efficiency
    - Tree structures (B-Tree, B^+-Tree)
    - Unclustered hash index
    - Bitmap index

Elena Baralis
Politecnico di Torino

Heap file

- Tuples are sequenced in *insertion order*
  - insert is typically an *append* at the end of the file
- *All* the space in a block is completely exploited before starting a new block
- Delete or update may cause wasted space
  - Tuple deletion may leave unused space
  - Updated tuple may not fit if new values have larger size
- Sequential reading/writing is very efficient
- Frequently used in relational DBMS
  - jointly with unclustered (secondary) indices to support search and sort operations

Elena Baralis
Politecnico di Torino

Ordered sequential structures

- The order in which tuples are written depends on the value of a given key, called *sort key*
  - A sort key may contain one or more attributes
    - the sort key may be the primary key
- Appropriate for
  - Sort and group by operations on the sort key
  - Search operations on the sort key
  - Join operations on the sort key
    - when sorting is used for join

Elena Baralis
Politecnico di Torino

Ordered sequential structures

- Problem
  - preserving the sort order when inserting new tuples
    - it may also hold for update
- Solution
  - Leaving a percentage of free space in each block during table creation
    - On insertion, dynamic (re)sorting in main memory of tuples into a block
- Alternative solution
  - Overflow file containing tuples which do not fit into the correct block

Elena Baralis
Politecnico di Torino

Ordered sequential structures

- Typically used with B+-Tree clustered (primary) indices
  - the index key is the sort key
- Used by the DBMS to store intermediate operation results

Elena Baralis
Politecnico di Torino

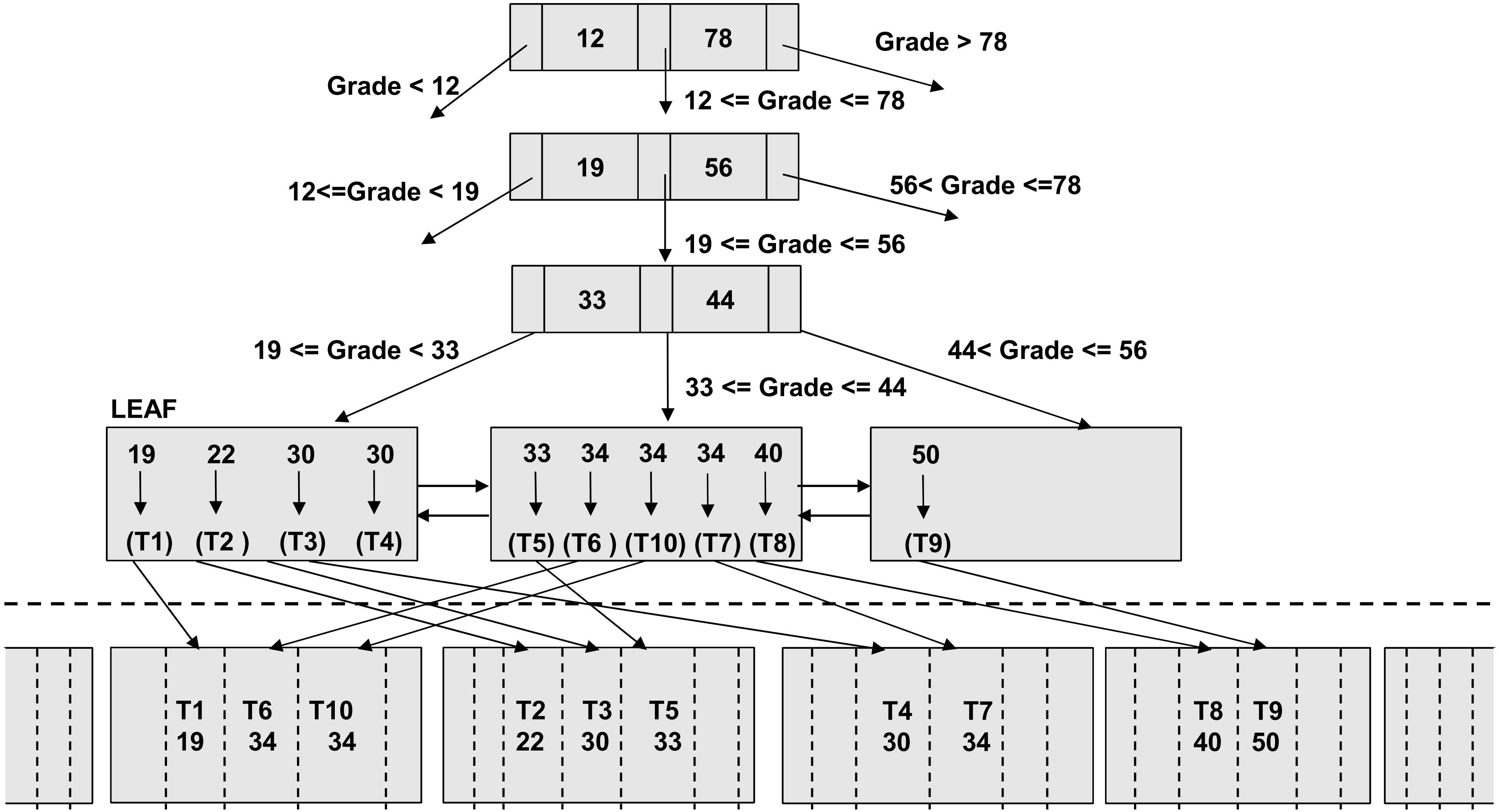
B+Tree

- Provide “direct” access to data based on the value of a key field
  - The key includes one or more attributes
- B stands for *balanced*
  - Leaves are all at the same distance from the root
  - Access time is constant, regardless of the searched value
- Unclustered
  - The leaf contains physical pointers to actual data
    - The position of tuples in a file is totally unconstrained
- Clustered
  - The tuple is contained into the leaf node
    - Constrains the physical position of tuples in a given leaf node
    - Typically used for primary key indexing

Elena Baralis
Politecnico di Torino

Example: Unclustered B^+-Tree index

STUDENT (<u>StudentId</u>, Name, *Grade*)

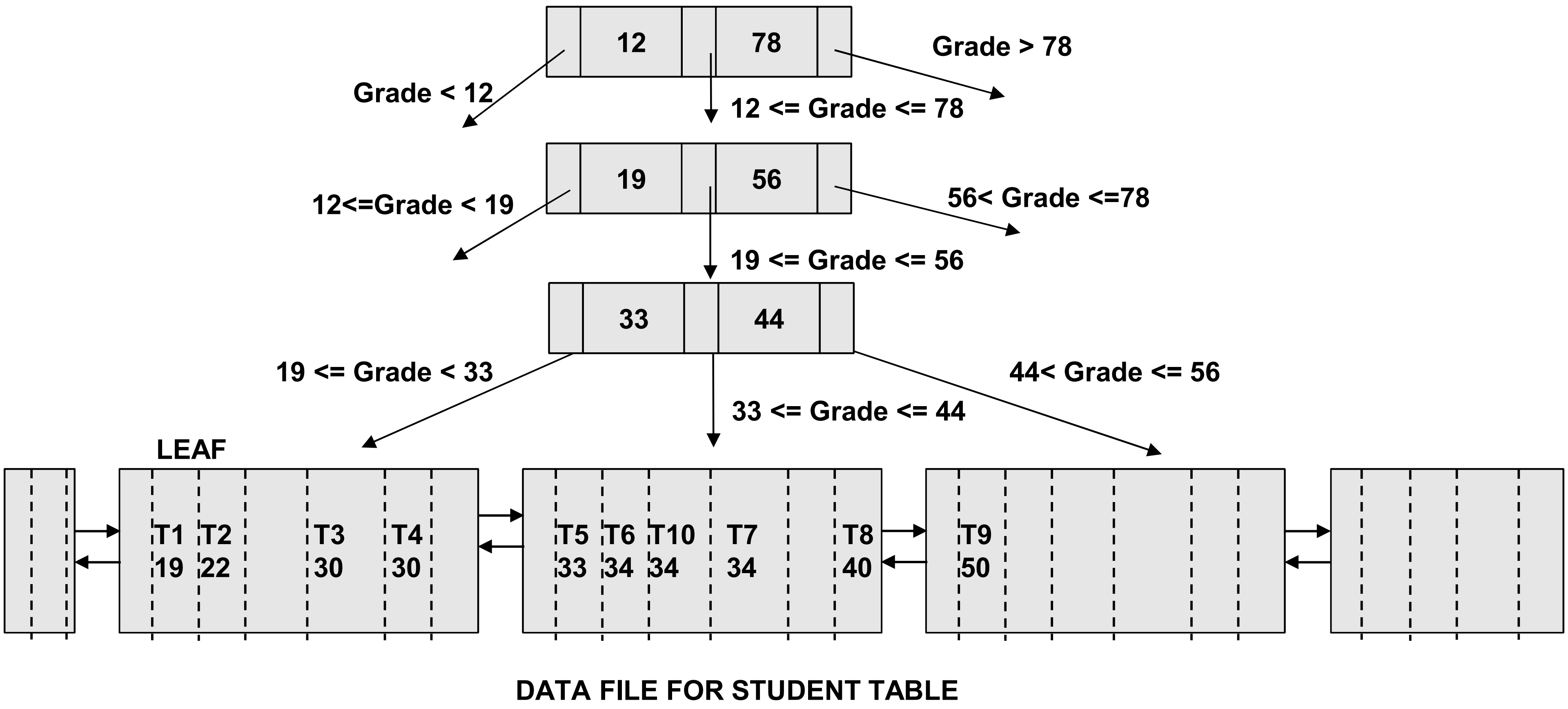


DATA FILE FOR STUDENT TABLE

Elena Baralis
Politecnico di Torino

Example: Clustered B⁺-Tree index

STUDENT (<u>StudentId</u>, Name, *Grade*)



Elena Baralis
Politecnico di Torino

Advantages and disadvantages

- Advantages
  - Very efficient for range queries
  - Appropriate for sequential scan in the order of the key field
    - Always for clustered, not guaranteed otherwise
- Disadvantages
  - Insertions may require a split of a leaf
    - possibly, also of intermediate nodes
    - computationally intensive
  - Deletions may require merging uncrowded nodes and re-balancing

Elena Baralis
Politecnico di Torino

Bitmap Index

- Composed of a bit matrix
  - A column for each different value of the indexed attribute domain
  - A row for each tuple (RID in the table)
  - The position (*i,j*) is 1 if the tuple *i* has value *j*, 0 otherwise

Example: Index on the field *Position* in the *Employee* table

Engineer – Consultant – Manager – Programmer - Assistant – Accountant

RID	Eng.	Cons.	Man.	Prog.	Assis.	Acc.
1	0	0	1	0	0	0
2	0	0	0	1	0	0
3	0	0	0	0	1	0
4	0	0	0	1	0	0
5	0	0	0	0	0	1

Taken from Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Elena Baralis
Politecnico di Torino

Bitmap index

- It guarantees direct and efficient access to data based on the value of a *key field*
  - It is based on a *bit matrix*
- The bit matrix references data rows by means of RIDs (Row IDentifiers)
  - Actual data is stored in a separate structure
  - Position of tuples is not constrained

Elena Baralis
Politecnico di Torino

Bitmap index

- The bit matrix has
  - One column for each different value of the indexed attribute
  - One row for each tuple
- Position (i, j) of the matrix is
  - 1 if tuple i takes value j
  - 0 otherwise

RID	Val_1	Val_2	…	Val_n
1	0	0	…	1
2	0	0	…	0
3	0	0	…	1
4	1	0	…	0
5	0	1	…	0

Elena Baralis
Politecnico di Torino

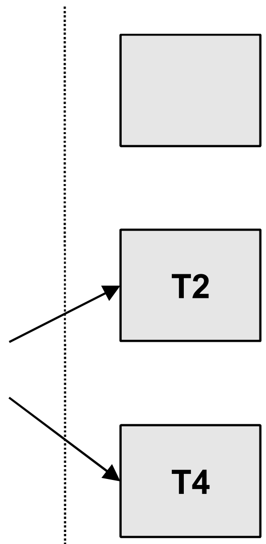
Example: Bitmap index

EMPLOYEE (<u>EmployeeId</u>, Name, *Job*)

Domain of Job attribute = {Engineer, Consultant, Manager, Programmer, Secretary, Accountant}

RID	Eng.	Cons.	Man.	Prog.	Secr.	Acc.
1	0	0	1	0	0	0
2	0	0	0	1	0	0
3	0	0	0	0	1	0
4	0	0	0	1	0	0
5	1	0	0	0	0	0

Prog.
0
1
0
1
0



DATA FILE FOR EMPLOYEE TABLE

Example: Index on the field *Position* in the *Employee* table
Engineer – Consultant – Manager – Programmer - Assistant – Accountant

Elena Baralis
Politecnico di Torino

Bitmap index

- Advantages
  - Very efficient for boolean expressions of predicates
    - Reduced to bit operations on bitmaps
  - Appropriate for attributes with limited domain cardinality
- Disadvantages
  - Not used for continuous attributes
  - Required space grows significantly with domain cardinality

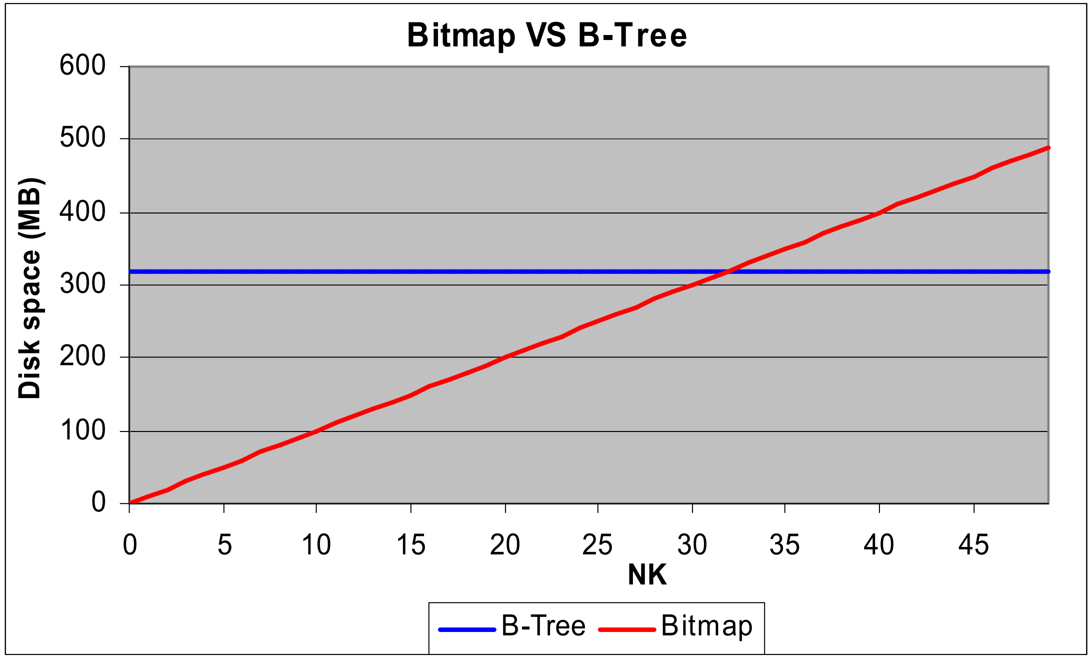
Elena Baralis
Politecnico di Torino

Bitmap Index

- Well-suited for dimensional attributes with low-cardinality domain
  - Storage requires limited space
  - If domain cardinality (NK) grows, the required space grows as well

B-tree	NR×*Len*(Pointer)
Bitmap	NR × NK × 1 bit

Len(*Pointer*) = 4×8 bit



Taken from Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

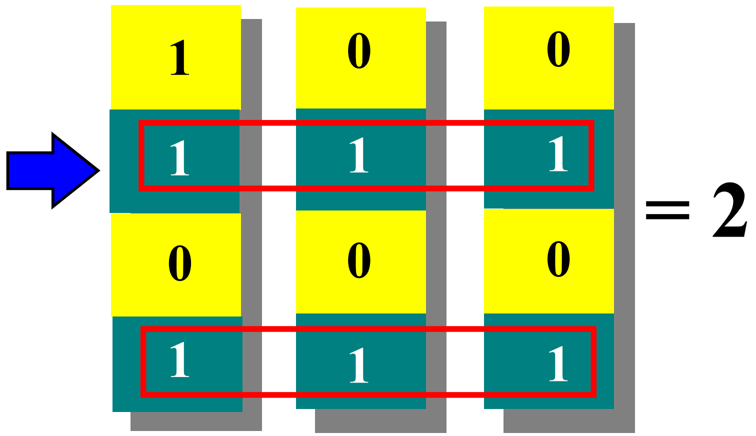
Elena Baralis
Politecnico di Torino

Bitmap Index

- Efficient for verifying Boolean expressions of predicates
  - Bit-wise and/or on bitmaps

Example: "*How many males in Romagna are insured?*"

RID	Gender	Ins.	Region
1	M	No	LO
2	M	Yes	E/R
3	F	No	LA
4	M	Yes	E/R



Example taken from Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Elena Baralis
Politecnico di Torino

Join index

- Precomputes the join between two tables
  – Stores the RID couples of tuples that satisfy the join predicate

Sale

IDStore	IDWeek	IDProd	Quantity	Income
123	13	41	100	10.00
123	24	17	150	15.00
367	36	94	350	35.00
367	27	11	120	12.00

Store

IDStore	Store	City	Country	SaleManager
123	N1	RM	I	R1
367	N3	MI	I	R2

RID Sale	RID Store
1	1
2	1
3	2
4	2

RID couples that satisfy Join condition:
Sells. IDStore = Store.IDStore

Example taken from Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Star index

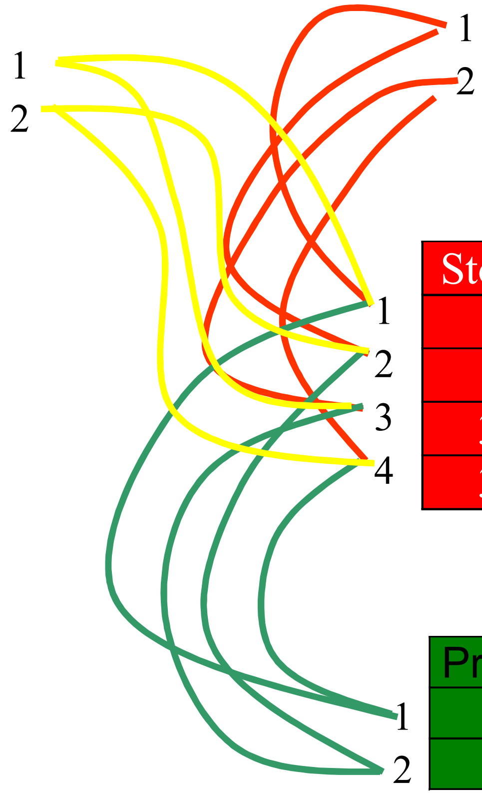
- Precomputes the join between two or more tables
  - Stores the RID n-uples of the tuples that satisfy the join predicate

Week

WeekID	Week	Month
13	Jan1	Jan.
24	Jan2	Jan.

Shop

StoreID	Store	City	Country
123	N1	RM	I
367	N3	MI	I



Sale

StoreID	WeekID	ProdID	Quantity	Income
123	13	41	100	10.00
123	24	17	150	15.00
367	13	17	350	35.00
367	24	41	120	12.00

SaleRID	SRID	WID	PID
1	1	1	1
2	1	2	2
3	2	1	2
4	2	2	1

Product

ProdID	Product	Type	Category	Supplier
41	P1	A	X	F1
17	P2	A	X	F1

Example taken from Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Star index

- Advantages
  - Efficient computation of Joins involving initial index columns (or all columns)
- Disadvantages
  - Useful only for specific Join combinations
    - It is necessary to store a high number of indexes in order to achieve generalization
  - The storage space may become big
    - Joins always include the fact table

Elena Baralis
Politecnico di Torino

Bitmapped join index

- Bit matrix that precomputes the join between a dimension and the fact table
  - A column for each dimension RID
  - A row for each fact table RID
  - The position (*i,j*) is 1 if the tuple *i* of the dimension is joined with the tuple *j* of the fact table, 0 otherwise
- Can be used together with traditional bitmap indexes to compute complex queries with conditions on dimensions and multiple joins

RID of the STORE table

RID of SELLS table

RID	1	2	3	…
1	1	0	0	…
2	0	1	0	…
3	0	0	1	…
4	0	1	0	…
5	1	0	0	…
…	…	…	…	…

The row 4 of table SELLS is joined with the row 2 of the table STORE

Taken from Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

Elena Baralis
Politecnico di Torino

Bitmapped join index

Executing a bit-wise OR, the system obtains RID_i that satisfy all conditions for a dimensional table

Bitmap Index on attribute $DT_i.b_i$

RID	Val_1	Val_2	…	Val_i	…	Val_h
1	1	0	…	0	…	0
2	0	0	…	0	…	1
3	0	1	…	0	…	0
4	0	0	…	1	…	0
5	0	0	…	1	…	0
…	…	…	…	…	…	…

1

Bitmapped join index $FT.a_i = DT_i.a_i$

RID	1	2	3	4	5	…
1	0	0	0	1	0	…
2	0	0	0	1	0	…
3	0	1	1	0	0	…
4	1	0	0	0	0	…
5	0	0	0	0	1	…
6	0	1	0	0	0	…
…	…	…	…	…	…	…

2

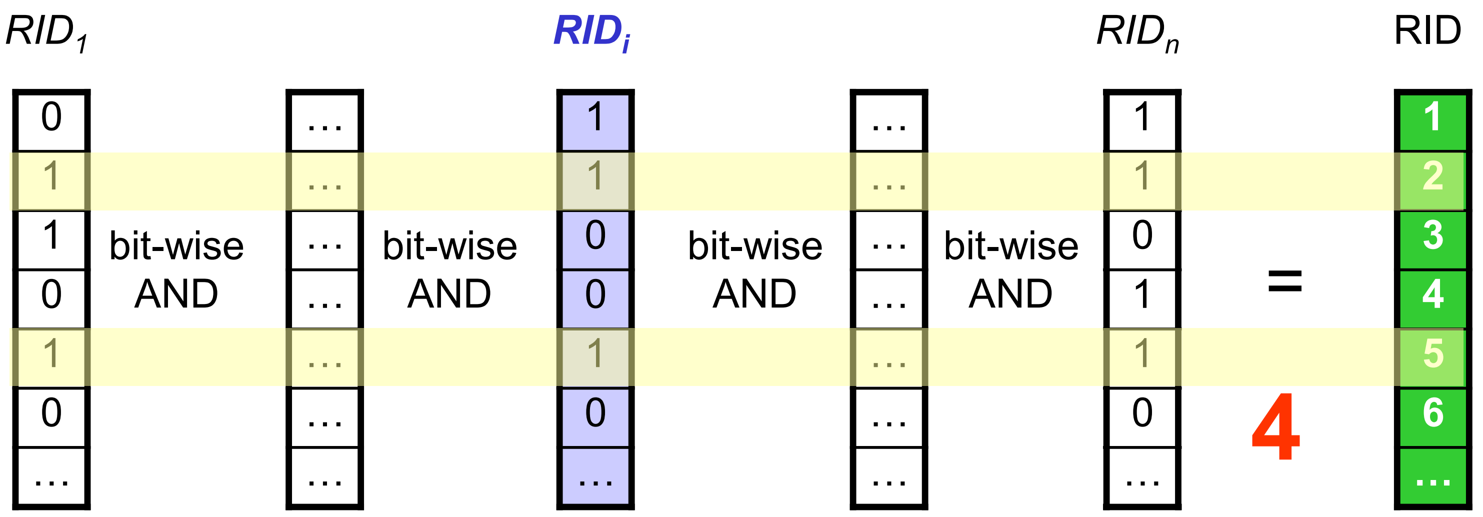
RID 4	RID 5	RID_i
1	0	1
1	0	1
0	0	0
0	0	0
0	1	1
0	0	0
…	…	…

3 Bit-wise OR: RID 4 OR RID 5 = RID_i

Taken from Golfarelli, Rizzi,”Data warehouse, teoria e pratica della progettazione”, McGraw Hill 2006

Elena Baralis
Politecnico di Torino

Bitmapped join index



The fact table tuples that satisfy the query are computed with a bit-wise AND between the *n* vector previously created

RIDs that satisfy all conditions

Taken from Golfarelli, Rizzi,"Data warehouse, teoria e pratica della progettazione", McGraw Hill 2006

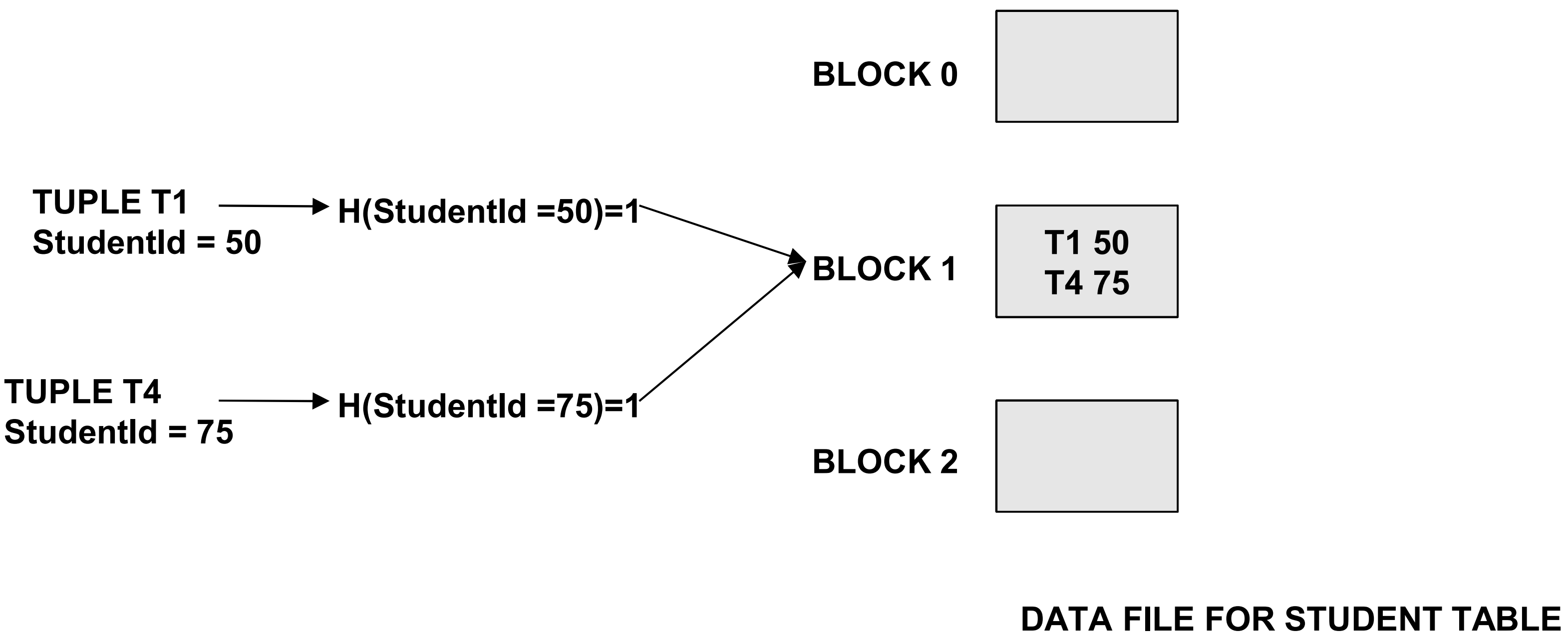
Hash structure

- It guarantees direct and efficient access to data based on the value of a *key field*
  - The hash key may include one or more attributes
- Suppose the hash structure has B blocks
  - The hash function is applied to the key field value of a record
    - It returns a value between 0 and B-1 which defines the position of the record
  - Blocks should never be completely filled
    - To allow new data insertion

Elena Baralis
Politecnico di Torino

Example: hash index

STUDENT (*StudentId*, Name, Grade)



Elena Baralis
Politecnico di Torino

Hash index

- Advantages
  - Very efficient for queries with equality predicate on the key
  - No sorting of disk blocks is required
- Disadvantages
  - Inefficient for range queries
  - Collisions may occur

Elena Baralis
Politecnico di Torino

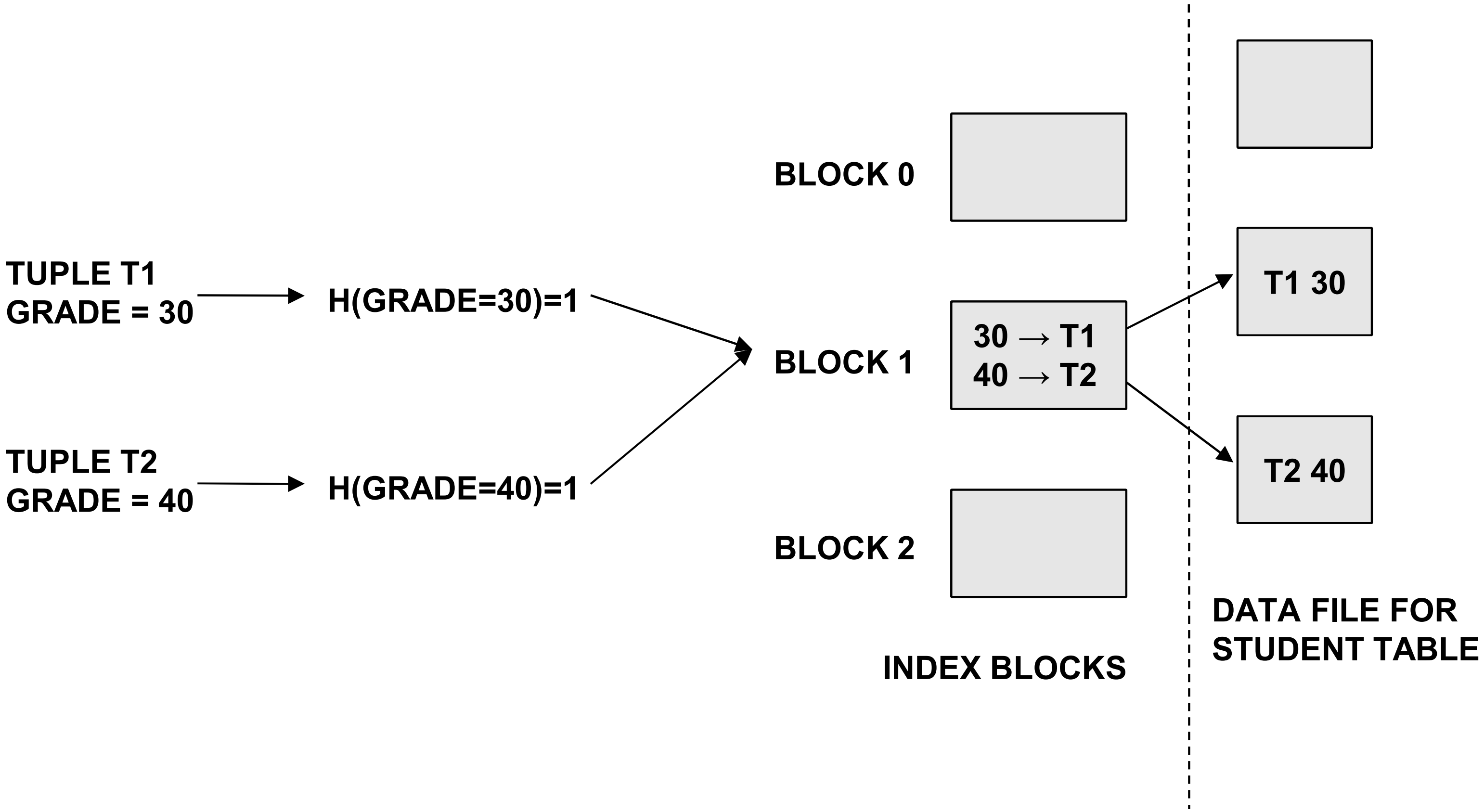
Unclustered hash index

- It guarantees direct and efficient access to data based on the value of a *key field*
  - Similar to hash index
- Blocks contain pointers to data
  - Actual data is stored in a separate structure
  - Position of tuples is not constrained to a block
    - Different from hash index

Elena Baralis
Politecnico di Torino

Example: Unclustered hash index

STUDENT (<u>StudentId</u>, Name, *Grade*)



Elena Baralis
Politecnico di Torino

Index choice

- Indexing of dimensions
  - Attribute frequently involved in selection predicates
  - If the domain has high cardinality, B-tree index
  - If the domain has low cardinality, bitmap index
- Indexes for Join
  - It is seldom necessary to index only external keys of the fact table
  - Be careful when using Star Join Indexes (problems related to column ordering)
  - Bitmapped join index are recommended
- Indexes for Group By
  - Use Materialized Views

Elena Baralis
Politecnico di Torino