Unit 3 : Data Warehouse Physical Design

Lecturer : Bijay Mishra

Physical Design

Physical design is the phase of a database design following the logical design that identifies the actual database tables and index structures used to implement the logical design.

In the physical design, you look at the most effective way of storing and retrieving the objects as well as handling them from a transportation and backup/recovery perspective.

Physical design decisions are mainly driven by query performance and database maintenance aspects.

During the logical design phase, you defined a model for your data warehouse consisting of entities, attributes, and relationships. The entities are linked together using relationships. Attributes are used to describe the entities. The unique identifier (UID) distinguishes between one instance of an entity and another.

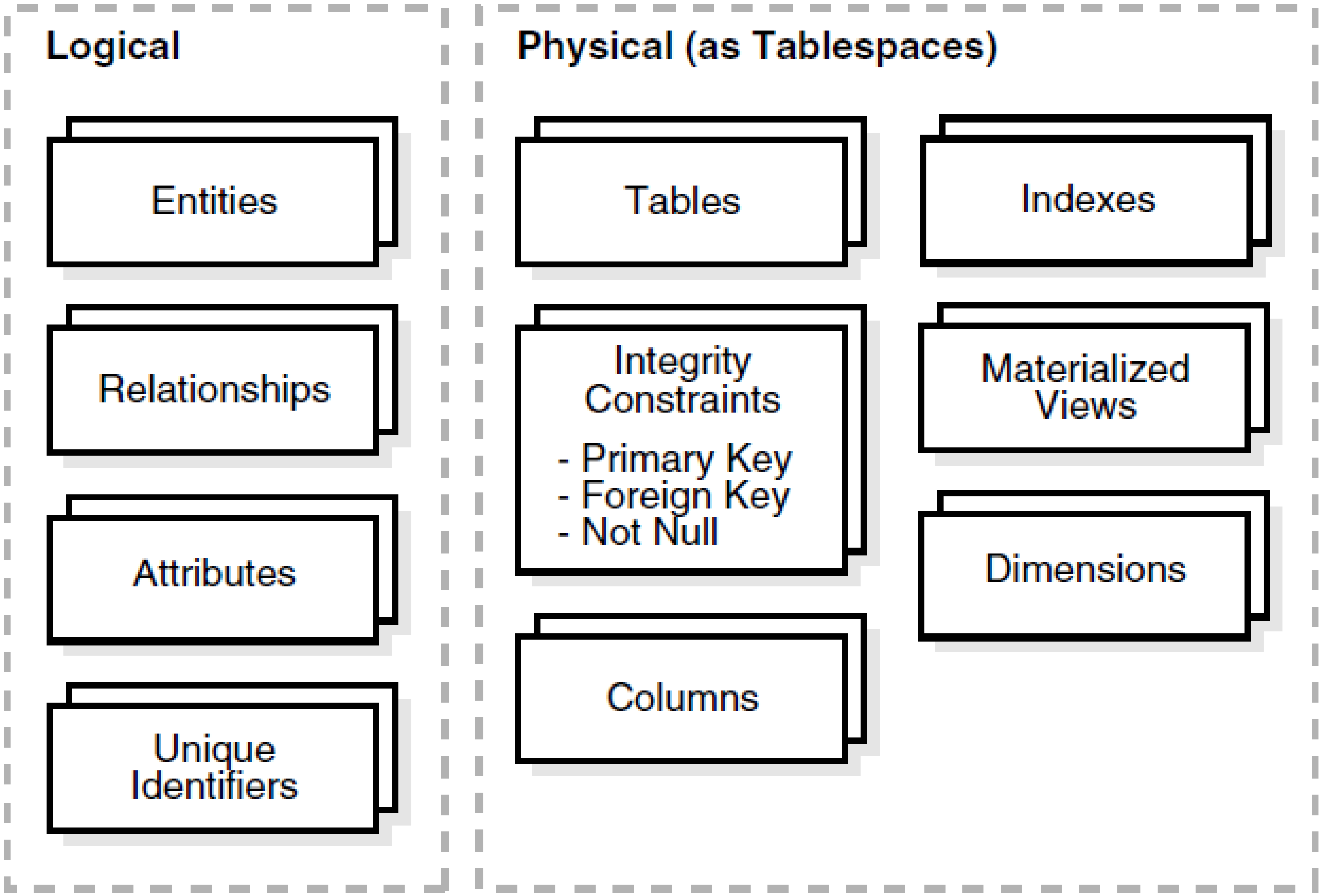


Figure: Logical Design Compared with Physical Design

During the physical design process, you translate the expected schemas into actual database structures.

At this time, you have to map:

■ Entities to tables

■ Relationships to foreign key constraints

■ Attributes to columns

■ Primary unique identifiers to primary key constraints

■ Unique identifiers to unique key constraints

Physical Data Model

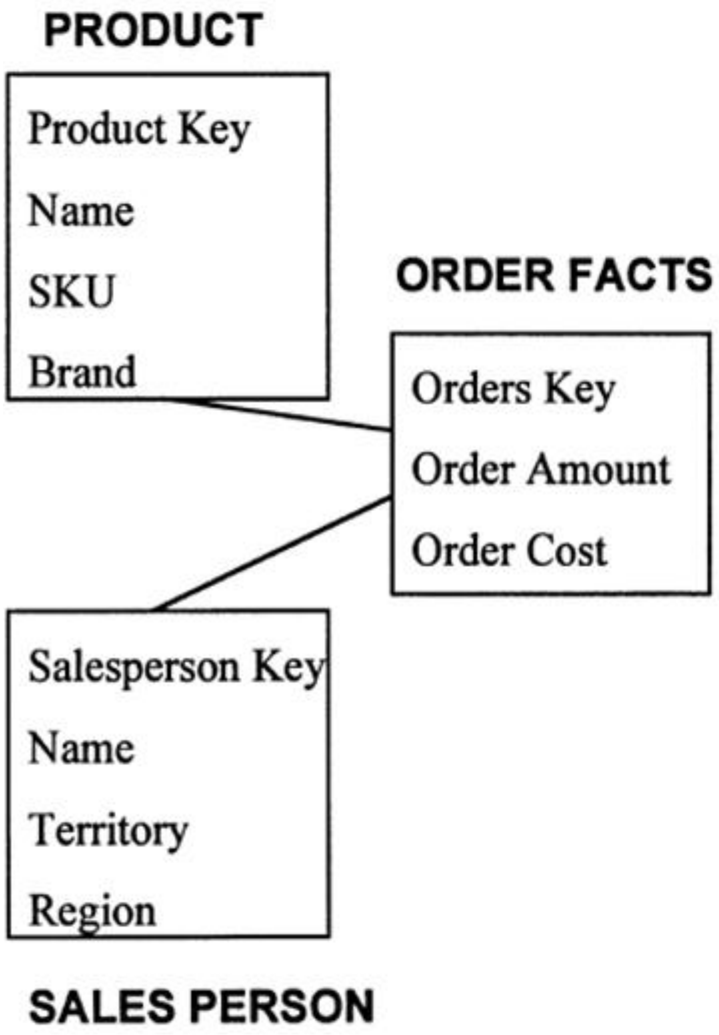
Features of physical data model include:

- ❖Specification all tables and columns.
- ❖Specification of Foreign keys.
- ❖De-normalization may be performed if necessary.
- ❖At this level, specification of logical data model is realized in the database.

The steps for physical data model design involves:

- ❖Conversion of entities into tables,
- ❖Conversion of relationships into foreign keys, Conversion of attributes into columns, and
- ❖Changes to the physical data model based on the physical constraints.

LOGICAL MODEL



PHYSICAL MODEL

Name	Type	Comments
PRODUCT		Product dimension table including all of
		company's products
product_key	integer	Primary key
product_name	char(25)	Name of product as used by marketing
product_sku	char(20)	Stock Keeping Unit in source systems
product_brand	char(25)	Name of brand as used by marketing
SALESPERSON		Sales Person dimension table includes all
		sales persons from all regions
salpers_key	integer	Primary key
salpers_name	char(30)	Name of sales person as used officially
territory	char(20)	Territory covered by the sales person
region	char(20)	Region containing the territory
ORDER FACT		Fact table containing metrics about all
		the orders received by the company
product_ref	integer	Partial primary key, also foreign key
		referencing product dimension table
salpers_ref	integer	Partial primary key, also foreign key
		referencing salesperson dimension table
order_amount	num(8,2)	Sales amount of order in dollars
order_cost	num(8,2)	Cost amount for the order in dollars

Figure: Logical model and physical model

Physical Design Objectives

❖ Involves tradeoffs among
  – Performance
  – Flexibility
  – Scalability
  – Ease of Administration
  – Data Integrity
  – Data Consistency
  – Data Availability
  – User Satisfaction

Performance

- Response time in DW typically > OLTP
  - Important to manage user expectations
- Poor performance may result from
  - Inadequate hardware
  - Inflexible data architecture
  - Poor physical design
  - Unrealistic user expectations
- Build performance bottom-up
  - Database Design and Optimization
  - Application design
  - Query efficiency
- Tune performance from top-down

Flexibility

- May include giving users flexibility to handle analysis, query, reporting needs
- Must accommodate change in today’s business environment

Scalability

- Old mainframes known for poor scalability
- Many adopt multi-server environment

Physical Design Structures

Once you have converted your logical design to a physical one, you will need to create some or all of the following structures:

■ Tablespaces

■ Tables and Partitioned Tables

■ Views

■ Integrity Constraints

■ Dimensions

Some of these structures require disk space. Others exist only in the data dictionary. Additionally, the following structures may be created for performance improvement:

■ Indexes and Partitioned Indexes

■ Materialized Views

Tablespaces

- A tablespace consists of one or more datafiles, which are physical structures within the operating system you are using.
- A datafile is associated with only one tablespace.
- From a design perspective, tablespaces are containers for physical design structures.

Tables and Partitioned Tables

- Tables are the basic unit of data storage. They are the container for the expected amount of raw data in your data warehouse.
- Using partitioned tables instead of non-partitioned ones addresses the key problem of supporting very large data volumes by allowing you to divide them into smaller and more manageable pieces.
- Partitioning large tables improves performance because each partitioned piece is more manageable.

Views

- A view is a tailored presentation of the data contained in one or more tables or other views.
- A view takes the output of a query and treats it as a table.
- Views do not require any space in the database.

Integrity Constraints

- Integrity constraints are used to enforce business rules associated with your database and to prevent having invalid information in the tables.
- In data warehousing environments, constraints are only used for query rewrite.
- NOT NULL constraints are particularly common in data warehouses.

Indexes and Partitioned Indexes

- Indexes are optional structures associated with tables.
- Indexes are just like tables in that you can partition them (but the partitioning strategy is not dependent upon the table structure)
- Partitioning indexes makes it easier to manage the data warehouse during refresh and improves query performance.

Materialized Views

- Materialized views are query results that have been stored in advance so long-running calculations are not necessary when you actually execute your SQL statements.
- From a physical design point of view, materialized views resemble tables or partitioned tables and behave like indexes in that they are used transparently and improve performance.

Hardware and I/O Consideration

- I/O performance should always be a key consideration for data warehouse designers and administrators.
- The typical workload in a data warehouse is especially I/O intensive, with operations such as large data loads and index builds, creation of materialized views, and queries over large volumes of data.
- The underlying I/O system for a data warehouse should be designed to meet these heavy requirements.

- In fact, one of the leading causes of performance issues in a data warehouse is poor I/O configuration.
- Database administrators who have previously managed other systems will likely need to pay more careful attention to the I/O configuration for a data warehouse than they may have previously done for other environments.
- The I/O configuration used by a data warehouse will depend on the characteristics of the specific storage and server capabilities

There are following five high-level guidelines for data-warehouse I/O configurations:

- ■ Configure I/O for Bandwidth not Capacity
- ■ Stripe Far and Wide
- ■ Use Redundancy
- ■ Test the I/O System Before Building the Database
- ■ Plan for Growth

Configure I/O for Bandwidth not Capacity

- Storage configurations for a data warehouse should be chosen based on the I/O bandwidth that they can provide, and not necessarily on their overall storage capacity.
- Buying storage based solely on capacity has the potential for making a mistake, especially for systems less than 500GB is total size.

- The capacity of individual disk drives is growing faster than the I/O throughput rates provided by those disks, leading to a situation in which a small number of disks can store a large volume of data, but cannot provide the same I/O throughput as a larger number of small disks.
- While it may not be practical to estimate the I/O bandwidth that will be required by a data warehouse before a system is built, it is generally practical with the guidance of the hardware manufacturer to estimate how much I/O bandwidth a given server can potentially utilize, and ensure that the selected I/O configuration will be able to successfully feed the server.

Conclusion:

- There are many variables in sizing the I/O systems, but one basic rule of thumb is that your data warehouse system should have multiple disks for each CPU (at least two disks for each CPU at a bare minimum) in order to achieve optimal performance.

Stripe Far and Wide

- The guiding principle in configuring an I/O system for a data warehouse is to maximize I/O bandwidth by having multiple disks and channels access each database object.
- A striped file is a file distributed across multiple disks. This striping can be managed by software (such as a logical volume manager), or within the storage hardware.
- The goal is to ensure that each tablespace is striped across a large number of disks so that any database object can be accessed with the highest possible I/O bandwidth.

Use Redundancy

- Because data warehouses are often the largest database systems in a company, they have the most disks and thus are also the most susceptible to the failure of a single disk.
- Therefore, disk redundancy is a requirement for data warehouses to protect against a hardware failure.
- Like disk-striping, redundancy can be achieved in many ways using software or hardware.

- A key consideration is that occasionally a balance must be made between redundancy and performance.
- For example, a storage system like RAID configuration and its variants may be used.
- Redundancy is necessary for any data warehouse, but the approach to redundancy may vary depending upon the performance and cost constraints of each data warehouse.

Test the I/O System Before Building the Database

- The most important time to examine and tune the I/O system is before the database is even created.
- Once the database files are created, it is more difficult to reconfigure the files.
- When creating a data warehouse on a new system, the I/O bandwidth should be tested before creating all of the database datafiles to validate that the expected I/O levels are being achieved.

Plan for Growth

- A data warehouse designer should plan for future growth of a data warehouse.
- There are many approaches to handling the growth in a system, and the key consideration is to be able to grow the I/O system without compromising on the I/O bandwidth.

Parallelism

- Parallelism is the idea of breaking down a task so that, instead of one process doing all of the work in a query, many processes do part of the work at the same time.
- Parallel execution is sometimes called parallelism.
- Parallel execution dramatically reduces response time for data-intensive operations on large databases typically associated with decision support systems (DSS) and data warehouses.
- An example of this is when four processes handle four different quarters in a year instead of one process handling all four quarters by itself.

Parallelism improves processing for:

■ Queries requiring large table scans, joins, or partitioned index scans

■ Creation of large indexes

■ Creation of large tables (including materialized views)

■ Bulk inserts, updates, merges, and deletes

Parallelism benefits systems with all of the following characteristics:

■ Symmetric multiprocessors (SMPs), clusters, or massively parallel systems

■ Sufficient I/O bandwidth

■ Underutilized or intermittently used CPUs (for example, systems where CPU usage is typically less than 30%)

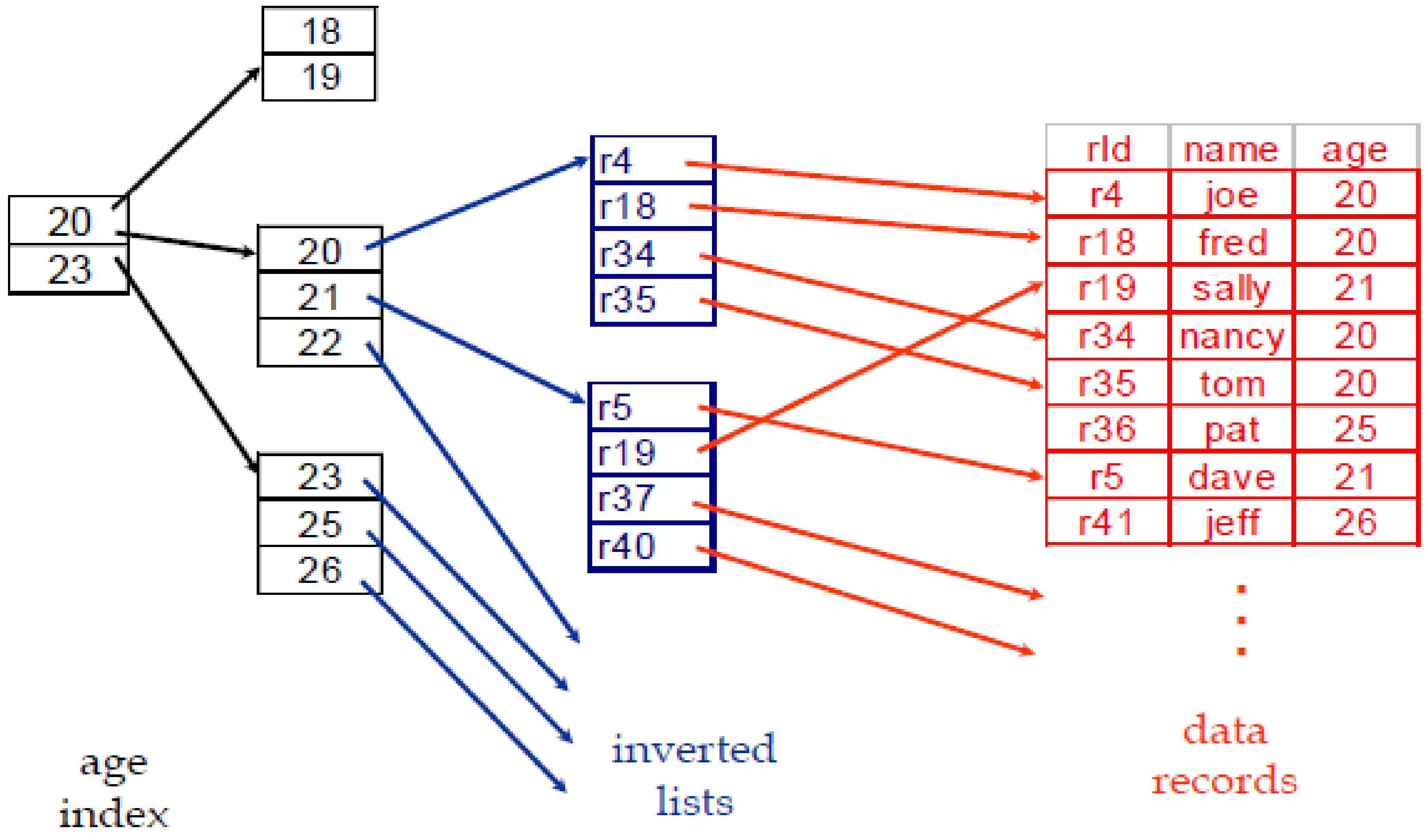
■ Sufficient memory to support additional memory-intensive processes, such as sorts, hashing, and I/O buffers

Indexes

- Indexes are optional structures associated with tables and clusters.
- Indexes are structures actually stored in the database, which users create, alter, and drop using SQL statements.
- You can create indexes on one or more columns of a table to speed SQL statement execution on that table.
- In a query-centric system like the data warehouse environment, the need to process queries faster dominates.
- Among the various methods to improve performance, indexing ranks very high.
- Indexes are typically used to speed up the retrieval of records in response to search conditions.

- **Indexes** can be unique or non-unique.
- Unique indexes guarantee that no two rows of a table have duplicate values in the key column (or columns).
- Non-unique indexes do not impose this restriction on the column values.
- Index structures applied in warehouses are:
  - Inverted lists
  - Bitmap indexes
  - Join indexes
  - Text indexes
  - B-Tree Index

Inverted Lists



rId	name	age
r4	joe	20
r18	fred	20
r19	sally	21
r34	nancy	20
r35	tom	20
r36	pat	25
r5	dave	21
r41	jeff	26

Query: Get people with age = 20 and name = “fred”

List for age = 20: r4, r18, r34, r35
List for name = “fred”: r18, r52
Answer is intersection: r18

Bitmap Indexes

- The concept of bitmap index was first introduced by Professor Israel Spiegler and Rafi Maayan in their research "Storage and Retrieval Considerations of Binary Data Bases", published in 1985.
- A bitmap index is a special kind of database index that uses bitmaps and are used widely in multi-dimensional database implementation.
- Bitmap indexes are primarily intended for data warehousing applications where users query the data rather than update it.
- They are not suitable for OLTP applications with large numbers of concurrent transactions modifying the data.

- ❖ Bitmap indexes use bit arrays (commonly called bitmaps) and answer queries by performing bitwise logical operations on these bitmaps.
- ❖ In a bitmap index, a bitmap for each key value replaces a list of row ids.
- ❖ Each bit in the bitmap corresponds to a possible rowid, and if the bit is set, it means that the row with the corresponding rowid contains the key value.

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

Table

Index on City:

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

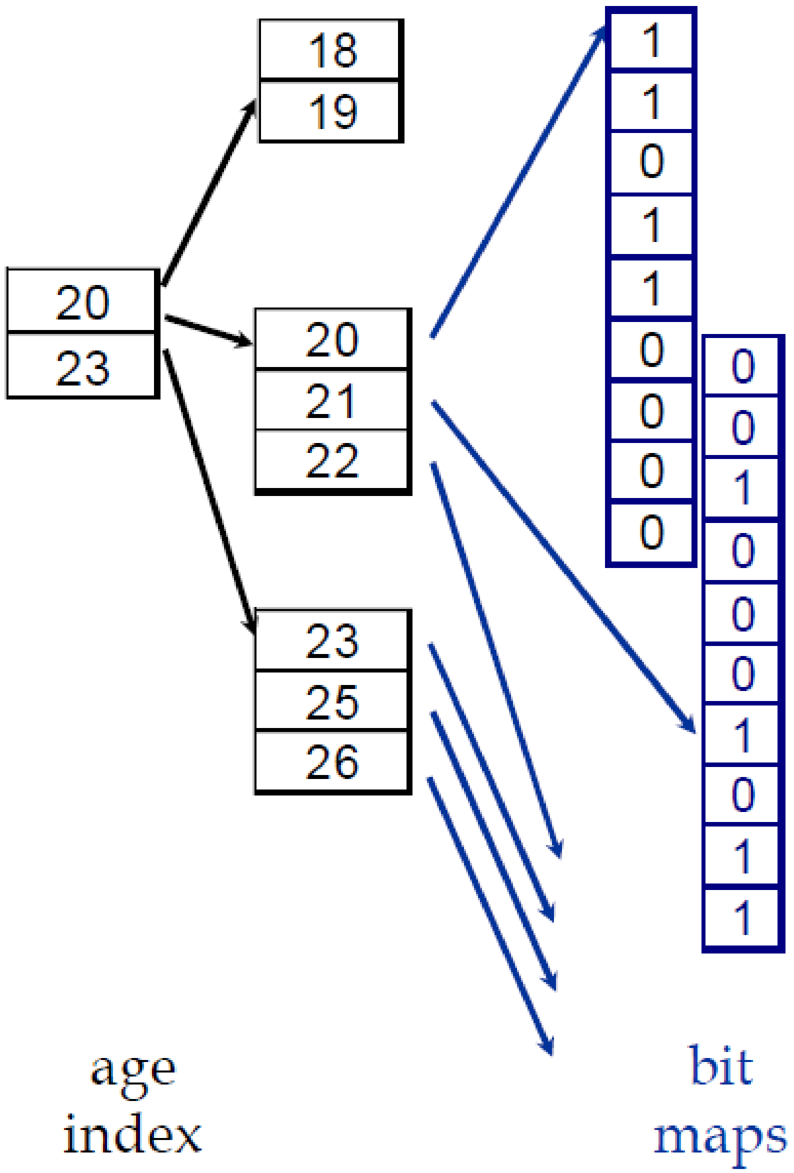
bitmaps

Index on Car:

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

- ❖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.
- ❖With efficient hardware support for bitmap operations (AND, OR, XOR, NOT), bitmap index offers better access methods for certain queries.



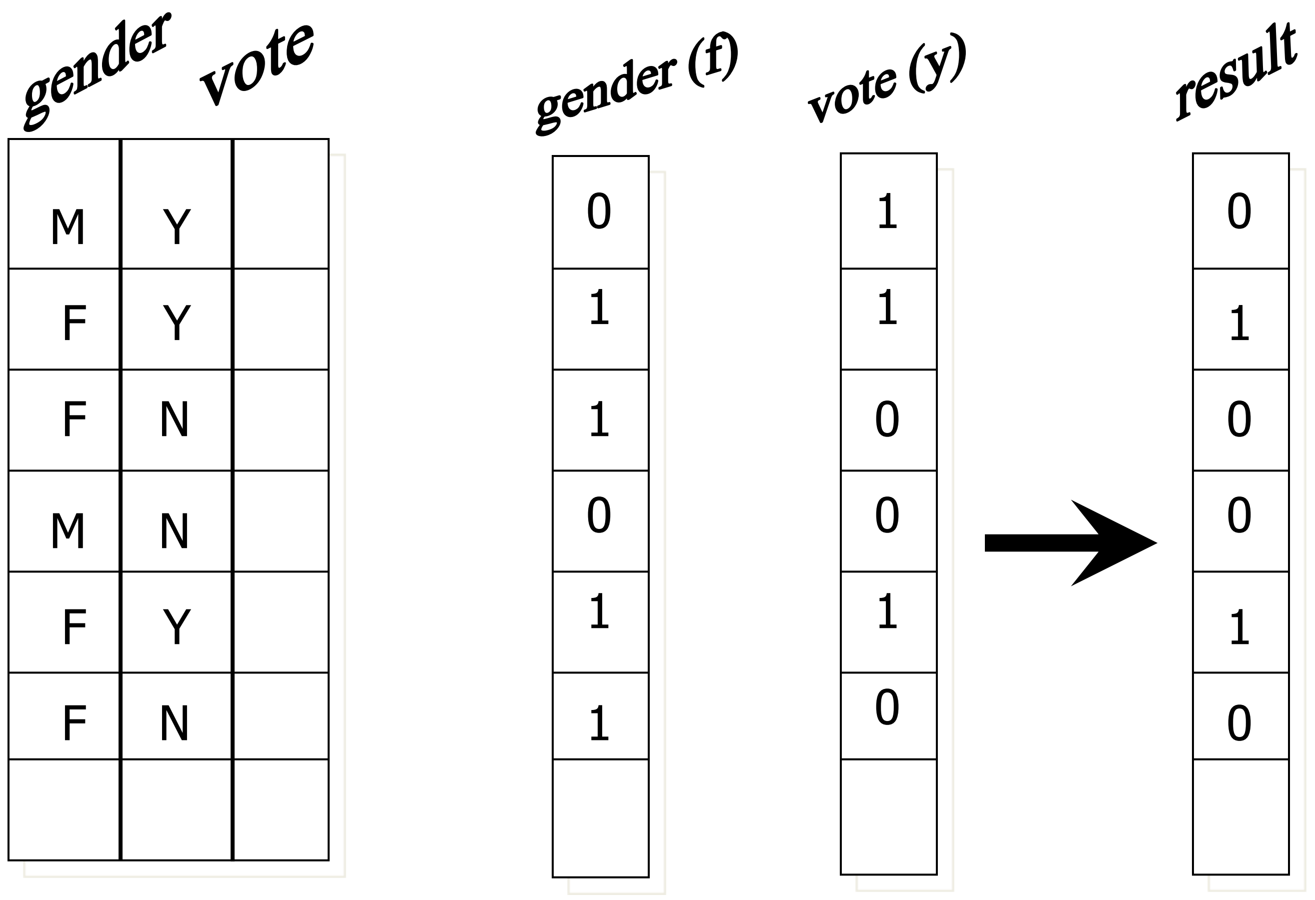
id	name	age
1	joe	20
2	fred	20
3	sally	21
4	nancy	20
5	tom	20
6	pat	25
7	dave	21
8	jeff	26

⋮

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

Example: the attribute sex has values M and F. A table of 100 million people needs 2 lists of 100 million bits



gender	vote	
M	Y	
F	Y	
F	N	
M	N	
F	Y	
F	N	

Customer

gender (f)	vote (y)	result
0	1	0
1	1	1
1	0	0
0	0	0
1	1	1
1	0	0

Query : select * from customer where gender = 'F' and vote = 'Y'

Base Table

Cust	Region	Rating
C1	N	H
C2	S	M
C3	W	L
C4	W	H
C5	S	L
C6	W	L
C7	N	H

Region Index

Row ID	N	S	E	W
1	1	0	0	0
2	0	1	0	0
3	0	0	0	1
4	0	0	0	1
5	0	1	0	0
6	0	0	0	1
7	1	0	0	0

Rating Index

Row ID	H	M	L
1	1	0	0
2	0	1	0
3	0	0	0
4	0	0	0
5	0	1	0
6	0	0	0
7	1	0	0

Customers where **Region = W** **And** **Rating = M**

Conclusion:

- ❖ Bitmap indexes are useful in data warehousing applications for joining a large fact table to smaller dimension tables such as those arranged in a star schema.

Join Indexes

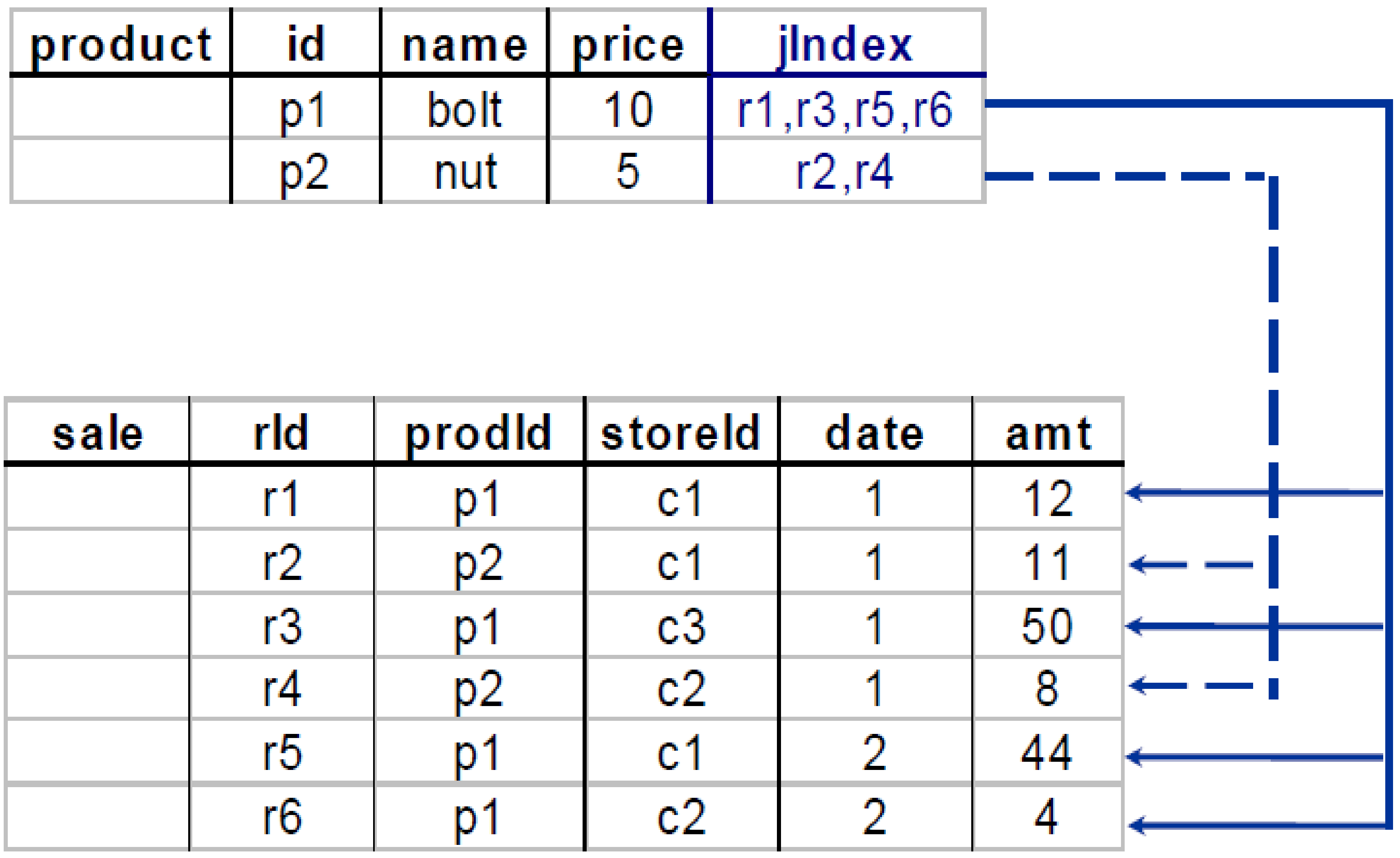
- Join indexes map the tuples in the join result of two relations to the source tables.
- In data warehouse cases, join indexes relate the values of the dimensions of a star schema to rows in the fact table.
  - For a warehouse with a Sales fact table and dimension city, a join index on city maintains for each distinct city a list of RIDs of the tuples recording the sales in the city
- Join indexes can span multiple dimensions

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

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



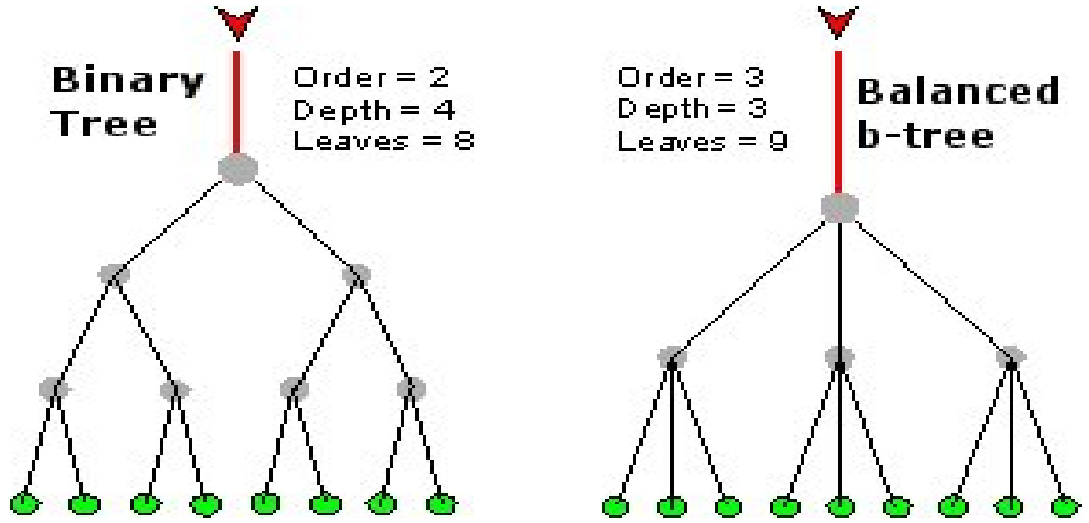
product	id	name	price	jIndex
	p1	bolt	10	r1,r3,r5,r6
	p2	nut	5	r2,r4

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

Figure: Join Index

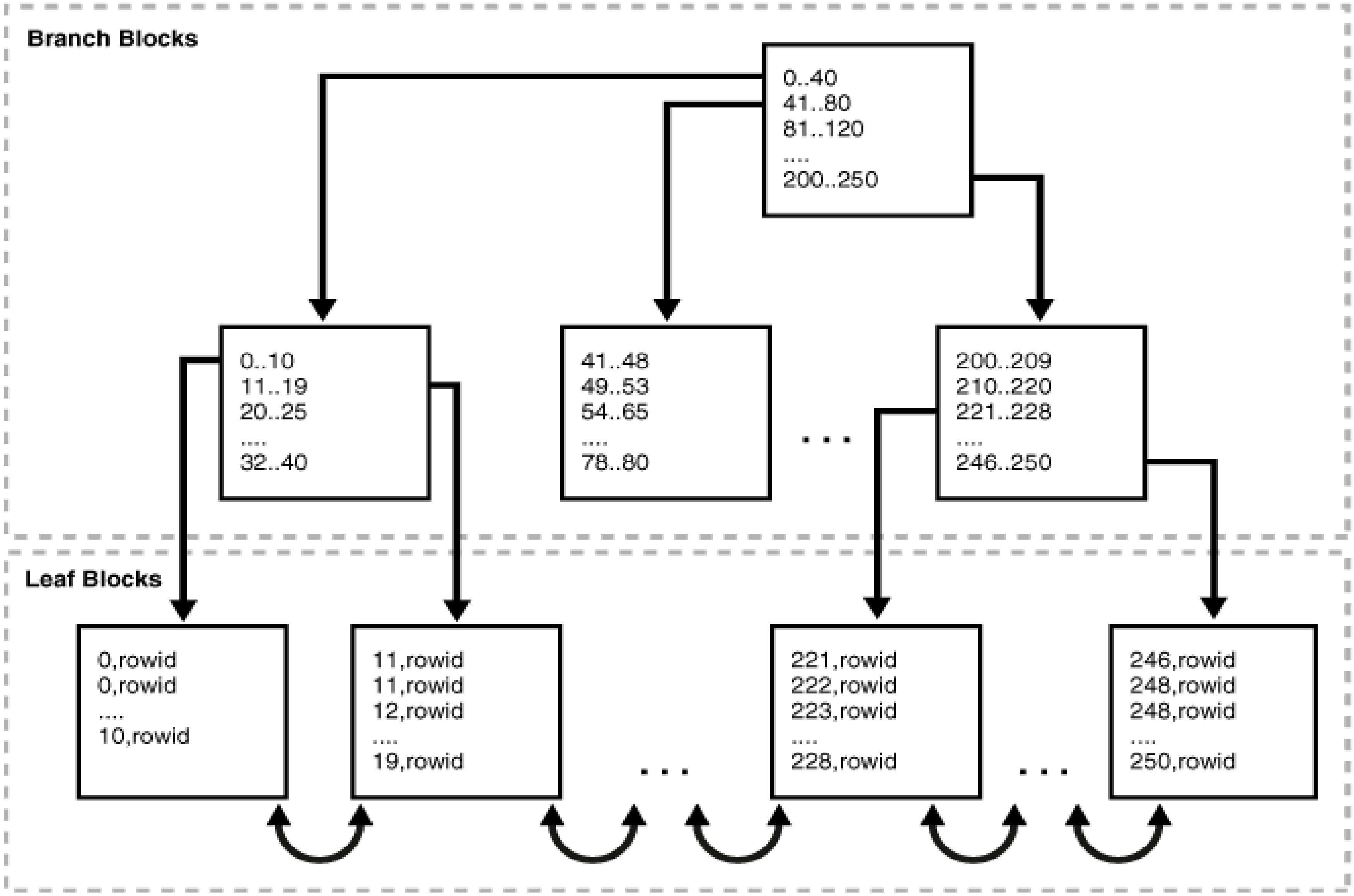
B-Tree Index

- B-trees, short for **balanced trees**, are the most common type and default database index.
- A **B-tree** is a tree data structure that keeps data sorted and allows searches, sequential access, insertions, and deletions.



- The B-tree is a generalization of a binary search tree in that a node can have more than two children.
- Figure below shows an example of a B-Tree Index.

Example: A B-Tree Index



- A B-tree index has two types of blocks: **branch blocks** for searching and **leaf blocks** that store values.
- The upper-level branch blocks of a B-tree index contain index data that points to lower-level index blocks.
- In above figure, the root branch block has an entry 0-40, which points to the leftmost block in the next branch level.
- This branch block contains entries such as 0-10 and 11-19.
- Each of these entries points to a leaf block that contains key values that fall in the range.

- ❖Branch blocks store the minimum key prefix needed to make a branching decision between two keys.
- ❖The branch blocks contain a pointer to the child block containing the key.
- ❖The leaf blocks contain every indexed data value and a corresponding rowid used to locate the actual row.
- ❖The leaf blocks themselves are also doubly linked.
- ❖In above figure, the leftmost leaf block (0-10) is linked to the second leaf block (11-19).

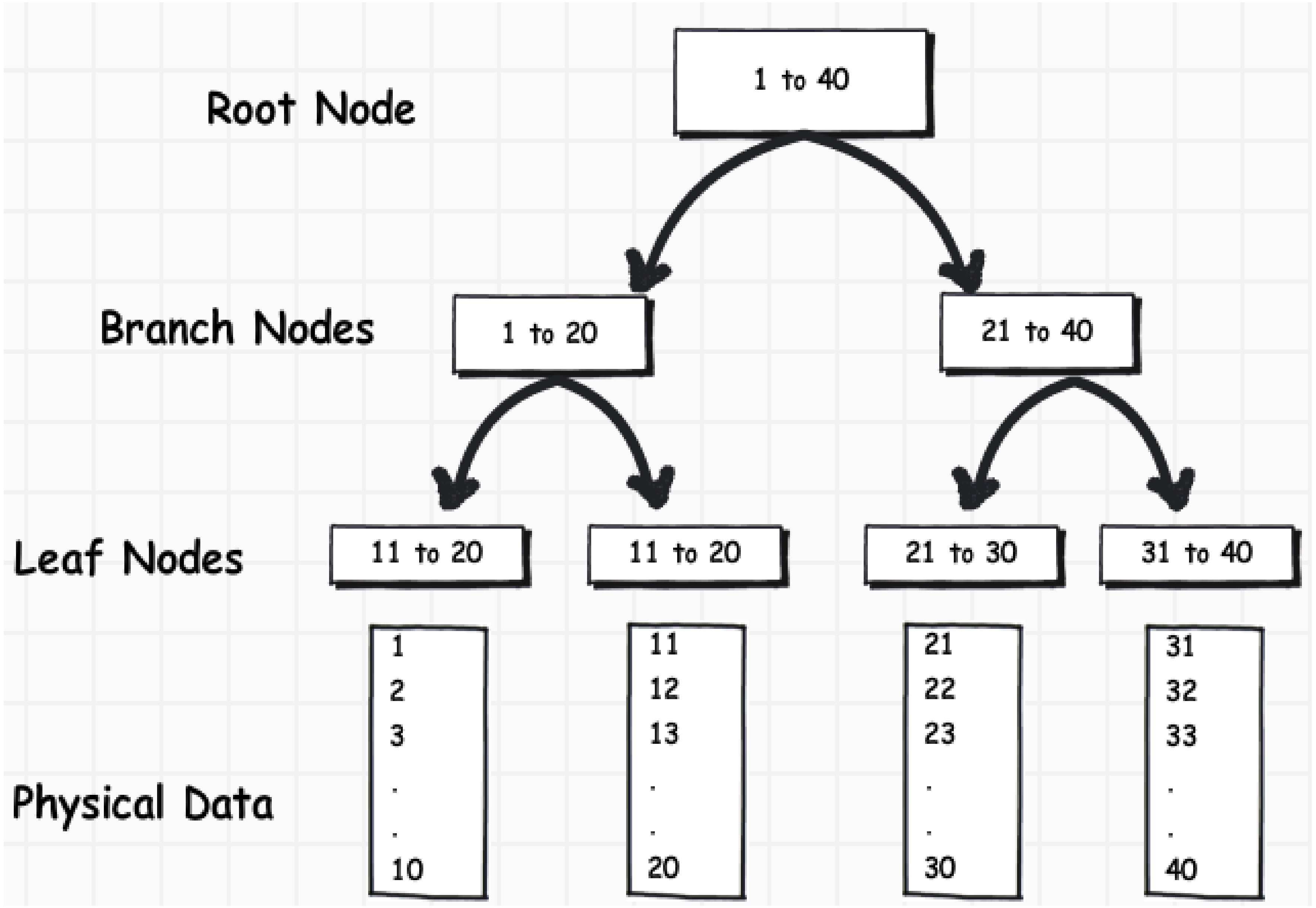


Figure: B-Tree index example

- ❖ Notice the tree structure with the root at the top.
- ❖ The index consists of a B-Tree (a balanced binary tree) structure based on the values of the indexed column.
- ❖ Suppose we have to search value 25 in an indexed column, the query engine will first look in the “Root Node” to determine which node to refer in the “Branch Nodes”.
- ❖ In the above example first “Branch Node” has Value 1 to 20 and the second “Branch Node” has Value 21 to 40, so the query engine will go to the second “Branch Node” and will skip the first “Branch Node” as we have to search Value 25.

Conclusion:

- ❖ B-tree indexes are created to decrease the amount of I/O required to find and load a set of data.

Assignment

1. In your opinion what may be the other factors for hardware and i/o consideration while making Data Warehouse.
2. Discuss about parallelism and parallel computing. Mention and explain some of the parallelism technique that could be adopted.
3. Give some of the suitable examples for the various indexing schemes.

Questions?

References

1. Sam Anahory, Dennis Murray, "Data warehousing In the Real World", Pearson Education.
2. Kimball, R. "The Data Warehouse Toolkit", Wiley, 1996.
3. Teorey, T. J., "*Database Modeling and Design: The Entity-Relationship Approach*", Morgan Kaufmann Publishers, Inc., 1990.
4. "An Overview of Data Warehousing and OLAP Technology", S. Chaudhuri, Microsoft Research
5. "Data Warehousing with Oracle", M. A. Shahzad
6. "Data Mining Concepts and Techniques", Morgan Kaufmann J. Han, M Kamber Second Edition ISBN : 978-1-55860-901-3

End of Unit 3

Thank you !!!