Index management

Index management

- Introduction
- Physical access structures
- Index definitions in SQL
- Physical design

Index management

Introduction

Physical data organization

- In a relational DBMS the data are represented as collections of records memorized in one or more files
  - the physical organization of the data in a file influences the time required to access the information
  - each physical data organization makes some operations efficient and others cumbersome
- There is no physical data organization that is efficient for any type of data reading and writing

Example database

Employee (ECode, Name, Surname, BirthDate, Residence, MonthlySalary)

EMPLOYEE

ECode	Name	Surname	BirthDate	Residence	MonthlySalary
D1	Elena	Rossi	02/01/1967	Torino	2.200,00
D2	Andrea	Verdi	04/05/1973	Como	1.100,00
D3	Giulia	Neri	14/04/1975	Roma	2.200,00
D4	Paolo	Bianchi	12/08/1970	Milano	3.000,00
D5	Daniele	Bruno	13/02/1968	Como	1.900,00
D6	Antonio	Bianco	25/11/1964	Venezia	1.700,00
D7	Lucia	Carta	09/04/1971	Alessandria	2.500,00
D8	Luca	Draghi	03/08/1973	Roma	2.400,00
D9	Tania	Bravo	11/06/1976	Asti	1.800,00
D10	Irene	Massa	28/04/1979	Torino	2.600,00
D11	Lia	Massa	15/05/1965	Milano	3.500,00
D12	Alessio	Morra	19/06/1969	Como	1.200,00

Example

- Table EMPLOYEE
  - is memorized by the relational DBMS in a file
- Query
  - view the information on employees resident in Como

```
SELECT *
FROM EMPLOYEE
WHERE City='Como';
```

Example: Table Employee

EMPLOYEE

ECode	Name	Surname	BirthDate	Residence	MontlySalary
D1	Elena	Rossi	02/01/1967	Torino	2.200,00
D2	Andrea	Verdi	04/05/1973	Como	1.100,00
D3	Giulia	Neri	14/04/1975	Roma	2.200,00
D4	Paolo	Bianchi	12/08/1970	Milano	3.000,00
D5	Daniele	Bruno	13/02/1968	Como	1.900,00
D6	Antonio	Bianco	25/11/1964	Venezia	1.700,00
D7	Lucia	Carta	09/04/1971	Alessandria	2.500,00
D8	Luca	Draghi	03/08/1973	Roma	2.400,00
D9	Tania	Bravo	11/06/1976	Asti	1.800,00
D10	Irene	Massa	28/04/1979	Torino	2.600,00
D11	Lia	Massa	15/05/1965	Milano	3.500,00
D12	Alessio	Morra	19/06/1969	Como	1.200,00

Example: result of the query

EMPLOYEE

ECode	Name	Surname	BirthDate	Residence	MontlySalary
D1	Elena	Rossi	02/01/1967	Torino	2.200,00
D2	Andrea	Verdi	04/05/1973	Como	1.100,00
D3	Giulia	Neri	14/04/1975	Roma	2.200,00
D4	Paolo	Bianchi	12/08/1970	Milano	3.000,00
D5	Daniele	Bruno	13/02/1968	Como	1.900,00
D6	Antonio	Bianco	25/11/1964	Venezia	1.700,00
D7	Lucia	Carta	09/04/1971	Alessandria	2.500,00
D8	Luca	Draghi	03/08/1973	Roma	2.400,00
D9	Tania	Bravo	11/06/1976	Asti	1.800,00
D10	Irene	Massa	28/04/1979	Torino	2.600,00
D11	Lia	Massa	15/05/1965	Milano	3.500,00
D12	Alessio	Morra	19/06/1969	Como	1.200,00

Example: execution of the query

- Operations carried out by the DBMS to execute the SQL query
  - sequential reading of the entire file
  - during the reading, selection of records of employees resident in Como
  - viewing of records
- Are there any physical data organizations on files that can bypass the complete scanning of the file?

Example: physical structure 1

- The records of the table EMPLOYEE are memorized in alphabetical order of Residence

Example: physical structure 1

EMPLOYEE

ECode	Name	Surname	BirthDate	Residence	MonthlySalary
D7	Lucia	Carta	09/04/1971	Alessandria	2.500,00
D9	Tania	Bravo	11/06/1976	Asti	1.800,00
D2	Andrea	Verdi	04/05/1973	Como	1.100,00
D5	Daniele	Bruno	13/02/1968	Como	1.900,00
D12	Alessio	Morra	19/06/1969	Como	1.200,00
D4	Paolo	Bianchi	12/08/1970	Milano	3.000,00
D11	Lia	Massa	15/05/1965	Milano	3.500,00
D3	Giulia	Neri	14/04/1975	Roma	2.200,00
D8	Luca	Draghi	03/08/1973	Roma	2.400,00
D1	Elena	Rossi	02/01/1967	Torino	2.200,00
D10	Irene	Massa	28/04/1979	Torino	2.600,00
D6	Antonio	Bianco	25/11/1964	Venezia	1.700,00

Example: physical structure 1

EMPLOYEE

ECode	Name	Surname	BirthDate	Residence	MonthlySalary
D7	Lucia	Carta	09/04/1971	Alessandria	2.500,00
D9	Tania	Bravo	11/06/1976	Asti	1.800,00
D2	Andrea	Verdi	04/05/1973	Como	1.100,00
D5	Daniele	Bruno	13/02/1968	Como	1.900,00
D12	Alessio	Morra	19/06/1969	Como	1.200,00
D4	Paolo	Bianchi	12/08/1970	Milano	3.000,00
D11	Lia	Massa	15/05/1965	Milano	3.500,00
D3	Giulia	Neri	14/04/1975	Roma	2.200,00
D8	Luca	Draghi	03/08/1973	Roma	2.400,00
D1	Elena	Rossi	02/01/1967	Torino	2.200,00
D10	Irene	Massa	28/04/1979	Torino	2.600,00
D6	Antonio	Bianco	25/11/1964	Venezia	1.700,00

Example: physical structure 1

- Operations to execute the query (simple version)
  - sequential reading of the file to the first record with Residence equal to Como
  - Sequential reading of all the records with Residence equal to Como, down to the first record with Residence different from Como
  - viewing of records of employees with Residence in Como
- There are more efficient methods for consulting the same data structure

Example: physical structure 1

- Physical organization designed specifically for the query proposed
  - memorization of the records in the file in alphabetical order of Residence
- File with an *ordered sequential structure*

Example: physical structure 1

- Advantages
  - sequential reading of the entire file is avoided
- Disadvantages
  - maintaining the order is computationally cumbersome
    - files need to be reorganized when records are inserted, updated or cancelled
  - this physical organization is not efficient when it is necessary to execute other query typologies
    - example: select the employees (dipendenti) who receive a monthly-salary over 2000 euro

Example: physical structure 2

- It is possible to define accessory physical structures that facilitate access to data

Example: physical structure 2

Accessory physical structure

Residence	Physical Address
Alessandria	
Asti	
Como	
Milano	
…	…
Venezia	

EMPLOYEE

ECode	…	Residence	…
D1	…	Torino	…
D2	…	Como	…
D3	…	Roma	…
D4	…	Milano	…
D5	…	Como	…
D6	...	Venezia	...
D7	...	Alessandria	...
D8	…	Roma	...
D9	…	Asti	...
D10	…	Torino	...
D11	…	Milano	...
D12	..	Como	…

Example: physical structure 2

Accessory physical structure

Residence	Physical Address
Alessandria	
Asti	
Como	
Milano	
…	…
Venezia	

EMPLOYEE

ECode	…	Residence	…
D1	…	Torino	…
D2	…	Como	…
D3	…	Roma	…
D4	…	Milano	…
D5	…	Como	…
D6	...	Venezia	...
D7	...	Alessandria	...
D8	…	Roma	...
D9	…	Asti	...
D10	…	Torino	...
D11	…	Milano	...
D12	..	Como	…

Example: physical structure 2

- Accessory physical structure with associative access to data
  - realized on the attribute Residence
- The attribute Residence is the key field of the structure
  - for each value assumed by the attribute Residence are memorized
    - all the physical locations of the records corresponding to the value of the key field
  - the physical location
    - indicates the position of a record inside a file
    - enables direct access to the record of interest (to the physical page that contains it)

Example: physical structure 2

- Operations carried out to execute the query
  - reading of the accessory physical structure to recover the physical locations of the records corresponding to Residence=Como
  - direct access only to the records of the file associated with Residence Como
  - viewing of the records of interest
- There are different methods for swiftly finding information in the accessory physical structures

Example: physical structure 2

- Advantages
  - the complete and sequential reading of the file is avoided
    - as in the case of physical structure 1
  - direct access *only* to the records of interest
  - The maintainance costs of the accessory structure are lower than the costs of maintaining the file with an ordered structure

Example: physical structure 2

- Disadvantages
  - it occupies more space
    - supplementary space is necessary to memorize the accessory physical structure
  - the accessory structure can be utilized only when the attribute Residence appears in the query
    - accessory structures may be necessary for multiple attributes or combinations of attributes

Indexes

- *Indexes* are the accessory physical structures provided by the relational DBMS to improve the efficiency of data access operations
  - indexes are realized using different types of physical structures
    - trees
    - hash tables
- The instructions for managing the indexes are not part of the standard SQL

Index management

Physical access structures

Physical access structures

- Physical access structures describe how the data are organized in the secondary memory to guarantee *efficient* search and data modification operations
- These structures can be classified as
  - sequential structures
  - tree structures
  - calculated access structures

Physical access structures

- Each relational DBMS has different variations of the basic physical structures
  - the description of the internal structures of data memorization is not publicly available
  - the physical structures are different for different DBMSs

Sequential structure

- Is characterized by
  - a sequential arrangement of the records in the secondary memory
  - Blocks of consecutive memory in the file
- System sequence
  - the sequence of records depends on the value of a system field, called a *system key*
    - composed of one or more attributes

Tree structure

- Efficient associative data access, based on the value of a key
  - the key can be composed of one or more attributes
- A *tree structure* enables access to the set of physical locations of the records corresponding to the selected value of the key field
  - The physical location of a record indicates the physical position of the record in the file in the secondary memory
- Examples: B-tree, B^+-tree

Calculated access structure

- Efficient associative data access, based on the value of a key field
  - the key can be composed of one or more attributes
- It requires a *calculation algorithm* to locate the physical block of the file containing the records corresponding to the value of the key field
- It does not require a specific record system in the secondary memory
- Example: hash structure

Index management

Index definition in SQL

Index definition in SQL

- SQL language provides the following instructions for defining indexes
  - to create an index
    - CREATE INDEX
  - to cancel an index
    - DROP INDEX
- The instructions for the management of indexes are not part of the standard SQL

To create an index

CREATE INDEX *IndexName*
ON T*ableName (AttributeList)*

- Create an index
  - with the name *IndexName*
  - on the table *TableName*
  - defining its attributes in *AttributeList*

To create an index

CREATE INDEX *IndexName*
ON T*ableName (AttributeList)*

- The order in which the attributes appear in *AttributeList is important*
  - the order of the index keys is
    - first on the basis of the first attribute in *AttributeList*
    - equal in value to the first attribute on the values of the second attribute
    - and so on, in order, until the last attribute

Example database

EMPLOYEE

ECode	Name	Surname	BirthDate	Residence	MonthlySalary
D1	Elena	Rossi	02/01/1967	Torino	2.200,00
D2	Andrea	Verdi	04/05/1973	Como	1.100,00
D3	Giulia	Neri	14/04/1975	Roma	2.200,00
D4	Paolo	Bianchi	12/08/1970	Milano	3.000,00
D5	Daniele	Bruno	13/02/1968	Como	1.900,00
D6	Antonio	Bianco	25/11/1964	Venezia	1.700,00
D7	Lucia	Carta	09/04/1971	Alessandria	2.500,00
D8	Luca	Draghi	03/08/1973	Roma	2.400,00
D9	Tania	Bravo	11/06/1976	Asti	1.800,00
D10	Irene	Massa	28/04/1979	Torino	2.600,00
D11	Lia	Massa	15/05/1965	Milano	3.500,00
D12	Alessio	Morra	19/06/1969	Como	1.200,00

Example n.1

- Creation of an index on the attribute Residence of the table EMPLOYEE

```
CREATE INDEX ResidenceIndex
ON EMPLOYEE (Residence)
```

Example n.2

- Creation of an index on a combination of attributes Surname and Name of the table EMPLOYEE

```
CREATE INDEX SurnameNameIndex
ON EMPLOYEE (Surname,Name)
```

- The index is jointly defined on the two attributes
- The index keys are ordered
  - first on the basis of the value of the attribute Surname
  - of equal value to the attribute Surname, on the value of the attribute Name

To cancel an index

DROP INDEX *IndexName*

- Eliminate the index with the name *IndexName*
- This command is used when
  - the index is no longer utilized
  - the improvement in performance is insufficient
    - reduced reduction in response time for the queries
    - slowing down of updates due to index maintenance

Example

- Cancel the index ResidenceIndex

```
DROP INDEX ResidenceIndex
```

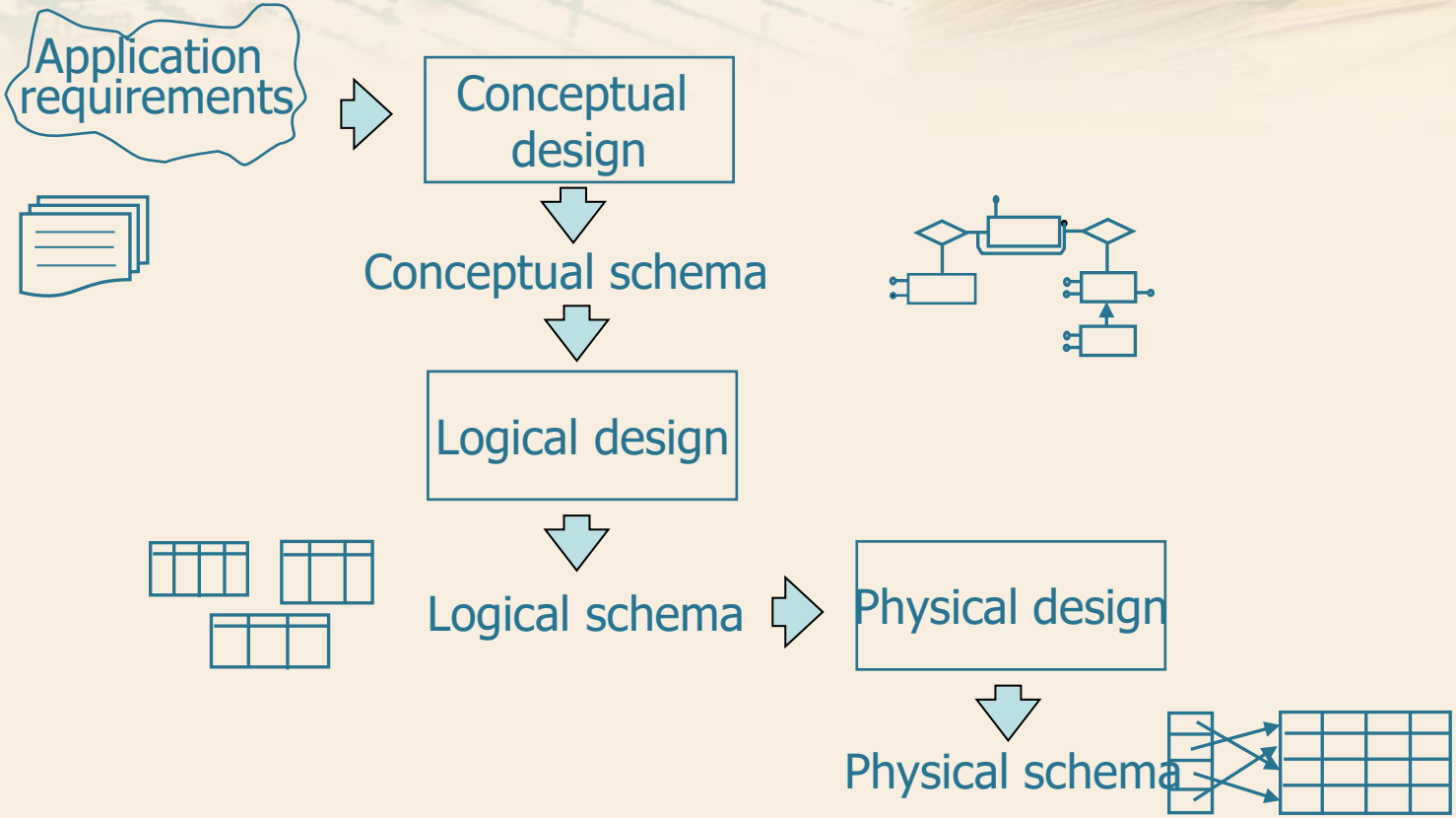
Index management

Physical design

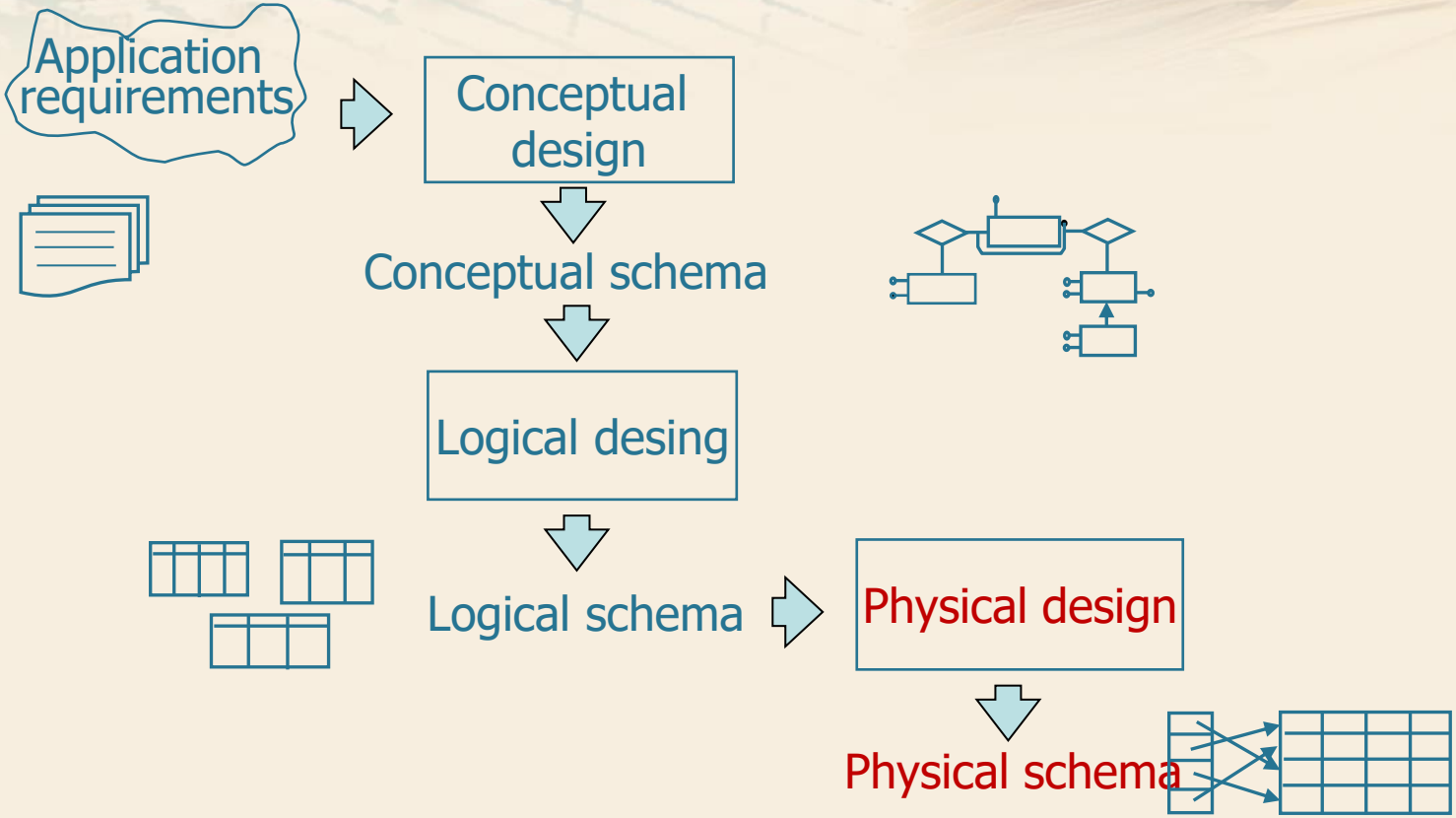
Physical design

- This is the final phase of database design
  - it requires choosing the DBMS to be utilized
  - it is linked to the characteristics of the product chosen

Database design phases



Database design phases



Physical design: entry data

- Logical scheme of the database
- Characteristics of the chosen DBMS
  - physically available options
    - physical memory structures
    - indexes
- Data volumes
  - cardinality of tables
  - cardinality and distribution of the attribute values domain

Physical design: entry data

- Estimate of applicative load
  - most important queries and their frequency
  - most important updating operations and their frequency
  - response time requirements for important queries/updates

Physical design: result

- Physical scheme of the database
  - physical organization of tables
  - indexes
- Memorization and functioning parameters
  - Initial file sizes, expansion possibilities, free space at outset, ...

Procedure

- Physical design is carried out empirically, using a trial and error approach
  - there are no reference methodologies

Procedure

- Characterization of applicative load
  - for each important query it is necessary to define
    - access relationships
    - attributes to be viewed
    - attributes involved in selections/joins
    - degree of selectivity of selection conditions
  - for each important update it is necessary to define
    - type of update
      - Insertion, cancellation, modification
    - relation to any attributes involved
    - degree of selectivity of selection conditions

Procedure

- Choices to be made
  - physical structuring of the files containing the tables
    - ordered, unordered
  - choice of attributes to index
    - piloted by estimating applicative load and data volume
  - definition of type for each index
    - e.g. hash or B-tree
  - any variations of the scheme
    - horizontal partitioning in the secondary memory
    - denormalization of tables
      - utilized in data warehouses

Tuning

- If the result is not satisfactory
  - *Tuning*, adding and removing indexes
- This is a procedure guided by tools that enable
  - verification of the execution plan adopted by the chosen DBMS
    - the execution plan defines the sequence of activities carried out by the DBMS to execute a query
      - data access methods
      - join methods
  - assess the execution cost of various alternatives