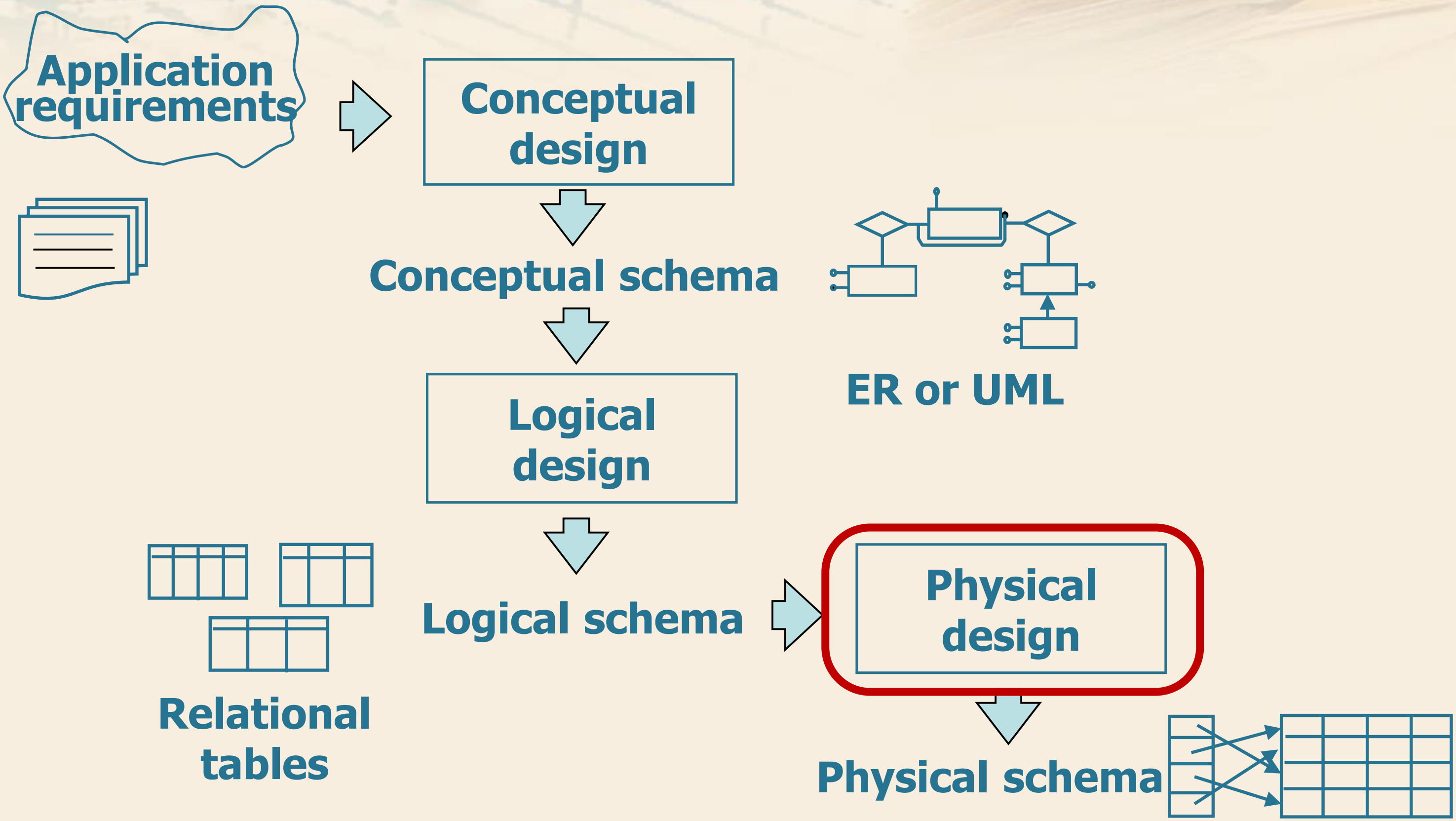
Database Management Systems

Physical Design

Phases of database design



Physical design

- Goal
  - Providing good performance for database applications
- Application software is not affected by physical design choices
  - Data independence
- It requires the selection of the DBMS product
  - Different DBMS products provide different
    - storage structures
    - access techniques

Physical design: Inputs

- Logical schema of the database
- Features of the selected DBMS product
  - e.g., index types, page clustering
- Workload
  - Important queries with their estimated frequency
  - Update operations with their estimated frequency
  - Required performance for relevant queries and updates

Physical design: Outputs

- Physical schema of the database
  - table organization, indices
- Set up parameters for database storage and DBMS configuration
  - e.g., initial file size, extensions, initial free space, buffer size, page size
  - Default values are provided

Physical design: Selection dimensions

- Physical file organization
  - unordered (heap)
  - ordered (clustered)
  - hashing on a hash-key
  - clustering of several relations
    - Tuples belonging to different tables may be interleaved

DBMG

Physical design: Selection dimensions

- Indices
  - different structures
    - e.g., B^+-Tree, hash
  - clustered (or primary)
    - *Only one* index of this type can be defined
  - unclustered (or secondary)
    - *Many* different indices can be defined

Characterization of the workload

- For each query
  - accessed tables
  - visualized attributes
  - attributes involved in selections and joins
  - selectivity of selections
- For each update
  - attributes and tables involved in selections
  - selectivity of selections
  - update type (Insert / Delete / Update) and updated attributes (if any)

Selection of data structures

- Selection of
  - physical storage of tables
  - indices
- For each table
  - file structure
    - heap or clustered
  - attributes to be indexed
    - hash or B^+-Tree
    - clustered or unclustered

Selection of data structures

- Changes in the logical schema
  - alternatives which preserve BCNF (Boyce Codd Normal Form)
  - alternatives *not* preserving BCNF
    - e.g., in data warehouses
  - partitioning on different disks

Physical design strategies

- No general design methodology is available
  - trial and error design process
  - general criteria
  - “common sense” heuristics
- Physical design may be improved after deployment
  - database tuning

General criteria

- The primary key is usually exploited for selections and joins
  - index on the primary key
    - clustered or unclustered, depending on other design constraints

DBMG

General criteria

- Add more indices for the most common query predicates
  - Select a frequent query
  - Consider its current evaluation plan
  - Define a new index and consider the new evaluation plan
    - if the cost improves, add the index
  - Verify the effect of the new index on
    - modification workload
    - available disk space

Heuristics

- Never index *small* tables
  - loading the entire table requires few disk reads
- Never index attributes with *low cardinality domains*
  - low selectivity
    - e.g., gender attribute
  - not true in data warehouses
    - different workloads and exploitation of bitmap indices

Heuristics

- For attributes involved in *simple predicates* of a where clause
  - Equality predicate
    - Hash is preferred
    - B^+-Tree
  - Range predicate
    - B^+-Tree
- Evaluate if using a clustered index improves in case of slow queries

Heuristics

- For where clauses involving *many* simple predicates
  - Multi attribute (composite) index
  - Select the appropriate key order
    - the order of attributes affects the usability of the index
- Evaluate maintenance cost

Heuristics

- To improve *joins*
  - Nested loop
    - Index on the *inner* table join attribute
  - Merge scan
    - B^+-Tree on the join attribute (if possible, clustered)

Heuristics

- For *group by*
  - Index on the grouping attributes
    - Hash index or B^+-tree
- Consider *group by push down*
  - anticipation of group by with respect to joins
  - not available in all systems
    - observe the execution plan

Example: Group by push down

- Tables

  PRODUCT (Prod#, PName, PType, PCategory)

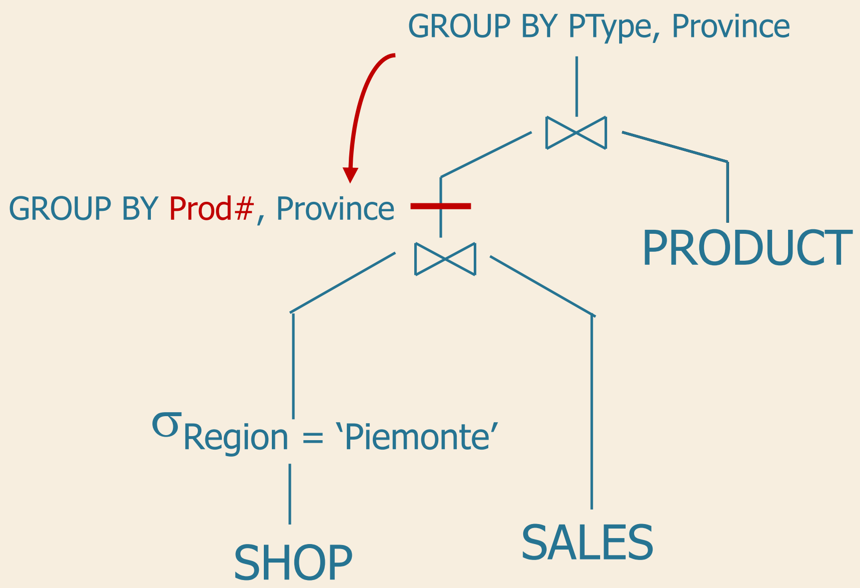
  SHOP (Shop#, City, Province, Region, State)

  SALES (Prod#, Shop#, Date, Qty)

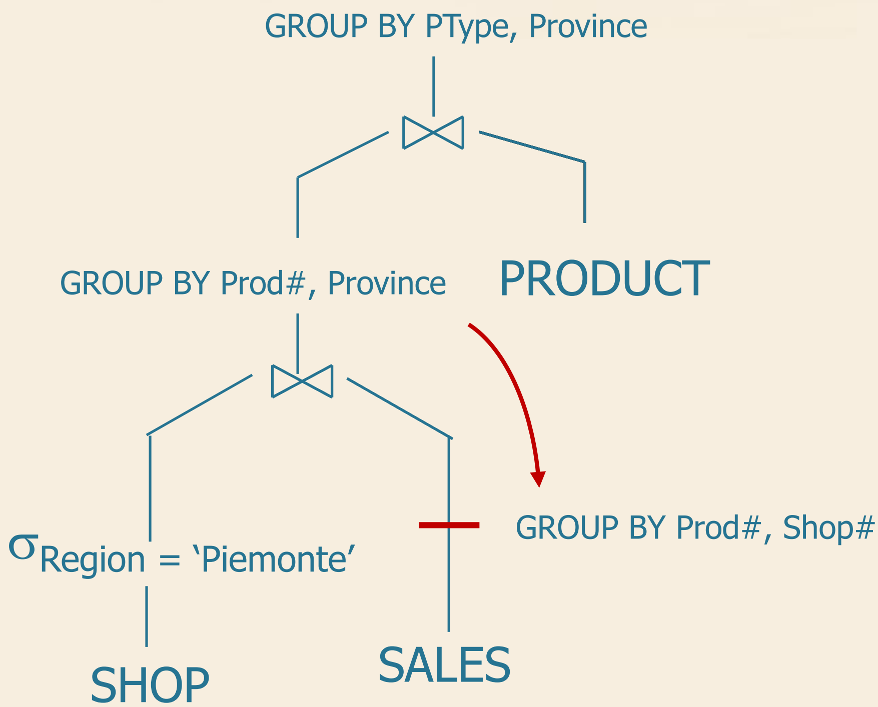
- SQL query

```
SELECT PType, Province, SUM (Qty)
FROM Sales S, Shop SH, Product P
WHERE S.Shop# = SH.Shop#
AND S.Prod# = P.Prod#
AND Region ='Piemonte'
GROUP BY PType, Province;
```

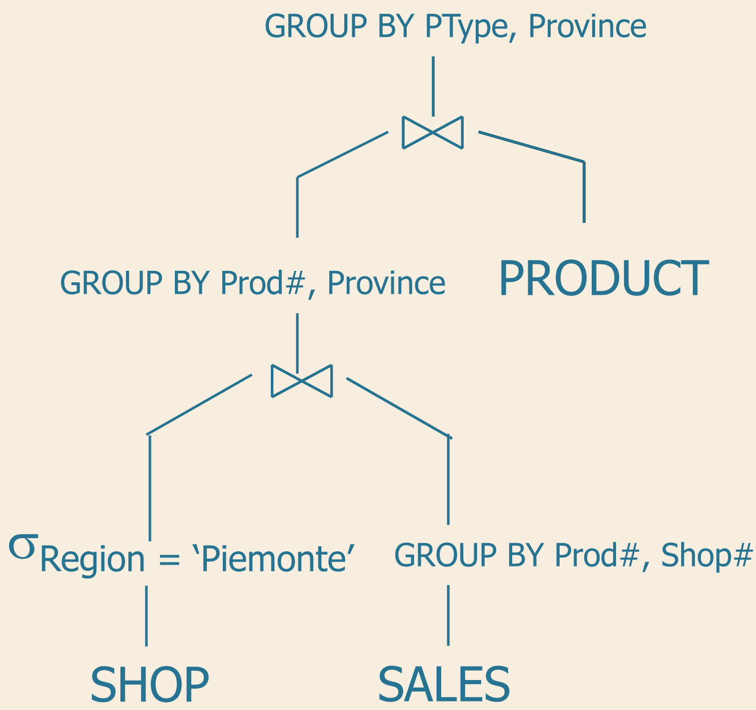
Example: Initial query tree



Example: Rewritten query tree (1)



Example: Rewritten query tree (2)



If it does not work?

- Query execution is not as fast as you expect
  - or you are not satisfied yet
- Remember to update database statistics!
- Database tuning
  - Add and remove indices
    - May be performed also after deployment
- Techniques to affect the optimizer decision
  - Should almost *never* be used
    - called hints in oracle
  - Data independence is *lost*

Database Management Systems

Physical Design Examples

DBMG

Example tables

- Tables
  - EMP (Emp#, EName, Dept#, Salary, Age, Hobby)
  - DEPT (Dept#, DName, Mgr)
    - In EMP
      Dept# FOREIGN KEY REFERENCES DEPT.Dept#
    - In DEPT
      Mgr FOREIGN KEY REFERENCES EMP.Emp#

Example 1

- SQL query

```
SELECT *
FROM   EMP
WHERE Salary/12 = 1500;
```

- Index on the salary attribute (B^+-Tree)
  - The index may be disregarded because of the arithmetic expression

Example 2

- SQL query

```
SELECT *
FROM   EMP
WHERE Salary = 18000;
```

- The index is used but it does not provide any benefit
  - Consider Salary data distribution
    - The value is very frequent and index access is not appropriate

Example 3

- Suppose that table EMP has block factor (number of tuples per block) equal to 30
    a) Card(DEPT)= 50
    b) Card(DEPT) = 2000

  For accessing Dept# in the EMP table, would you define a secondary index on Emp.Dept#?

Example 3

- Case A: Card (DEPT) = 50
  - Indexing is *not* appropriate
  - Each page on average contains almost all departments
    - sequential scan is better
- Case B: Card(DEPT) = 2000
  - Indexing is appropriate
  - Each page contains tuples belonging to few departments

Example 4

- SQL query

```
SELECT EName, Mgr
FROM   EMP E, DEPT D
WHERE E.Dept# = D.Dept#
AND DName = 'Toys';
```

- Index definition
  - Hash Index on DName for the selection condition
  - Hash Index on Emp. Dept# for a nested loop with Emp as *inner* table

DBMG

Example 5

- SQL query

```
SELECT EName, Mgr
FROM   EMP E, DEPT D
WHERE E.Dept# = D.Dept#
AND DName = 'Toys'
AND Age=25;
```

- Index definition
  - An index on Age may be considered
    - it depends on the selectivity of the condition

Example 6

- SQL query

```
SELECT EName, Mgr
FROM   EMP E, DEPT D
WHERE E.Dept# = D.Dept#
AND Salary BETWEEN 10000 AND 12000
AND Hobby='Tennis';
```

Example 6: selection

- Alternatives for the selection on EMP
  - hash index on Hobby
  - B^+-Tree on Salary
- Usually equality predicates are more selective
- One index is always considered by the optimizer
- Two indices may be exploited by smart optimizers
  - compute the intersection of RIDs before reading tuples

Example 6: join

- Alternatives for join
  - Hash join
  - Nested loop
    - EMP outer
      - because of selection predicates
    - DEPT inner
      - plus index on DEPT.Dept#

        not appropriate if DEPT table is very small

Example 7

- SQL query

```
SELECT Dept#, Count(*)
FROM   EMP
WHERE Age>20
GROUP BY Dept#
```

Example 7

- If the selection condition on Age is not very selective
  - *no* B^+-Tree on Age
- For group by
  - Clustered index on Dept#
    - Avoids sorting and reads blocks ready for group by
      - Good!
  - Secondary index on Dept#
    - May cause too many reads
      - Consider if appropriate

Example 8

- SQL query

```
SELECT Dept#, COUNT(*)
FROM   EMP
GROUP BY Dept#
```

DBMG

Example 8

- Unclustered (secondary) index on Dept#
  - It avoids reading table EMP
- It is a *covering index*
  - it answers the query without requiring access to table data

Example 9

- SQL query

```
SELECT Mgr
FROM   DEPT, EMP
WHERE DEPT.Dept#=EMP.Dept#
```

- Unclustered index on EMP.Dept#
  - It avoids reading table EMP

DBMG

Example 10

- SQL query

```
SELECT AVG(Salary)
FROM   EMP
WHERE Age = 25
AND Salary BETWEEN 3000 AND 5000
```

Example 10

- Composite index on <Age,Salary>
  - Fastest solution
  - This order is the best if the condition on Age is more selective

- Issues in composite indices
  - Order of the fields in a composite index is important
  - Update overhead grows