Lecture 12

Query Optimization in Relational Databases. Evaluating Relational Algebra Operators.

SQL Statement Execution

- Client application – the SQL statement execution request - should be performed for any query and provide a minimum response time
- Stages for the statement execution:
  - client: generate SQL statement (non-procedural language) and send it to server
  - server:
    - analyze SQL statement (in syntactically manner)
    - translate statement into an internal form (by using relational algebra expression)
    - transform internal form into an optimal form (similar to relational algebra)
    - generate a procedural execution plan
    - evaluate procedural plan, send result to client

SQL Statement Execution

- The operators that are used in the query process are:

- **Selection:** $\sigma_C(R)$
- **Projection:** $\pi_\alpha(R)$
- **Cross-product:** $R_1 \times R_2$
- **Union:** $R_1 \cup R_2$
- **Intersection:** $R_1 \cap R_2$
- **Set-difference:** $R_1 - R_2$
- **Theta join:** $R_1 \otimes_\theta R_2$
- **Natural join:** $R_1 * R_2$
- **Left outer join:** $R_1 \ltimes_C R_2$
- **Right outer join:** $R_1 \rtimes_C R_2$
- **Full outer join:** $R_1 \bowtie_C R_2$
- **Left semi join:** $R_1 \rhd R_2$
- **Right semi join:** $R_1 \lhd R_2$
- **Division:** $R_1 \div R_2$
- **Duplicate elimination:** $\delta(R)$
- **Sorting:** $S_{\{list\_attributes\}}\ (R)$
- **Grouping:**$\gamma_{\{list1\_attributes\}\ GROUP\ BY\ \{list2\_attributes\}}\ (R)$

SQL Statement Execution

- An SQL query may be written in multiple ways

Example: Consider the following relational database

Specialization[<u>SpId</u>, SpecName, SpecDescription, NoOfStudents]

Group[<u>Gid</u>, NoOfStudents, StudyYear, *SpId*]

Student[<u>Sid</u>, FirstName, LastName, Age, Email, *Gid*]

SQL query: display the students (FirstName, LastName, Age, StudyYear, SpecName) from a given specialization (e.g. SpId=3 can be a parameter) with the Age>=20 (can be a parameter)

* The provided solutions are equivalent (or, provide the same answer)

Solution 1:

SELECT FirstName, LastName, Age, StudyYear, SpecName
FROM Student S, Group G, Specialization Sp
WHERE S.Gid=G.Gid AND G.SpId=Sp.SpId AND Sp.SpID=3 AND Age>=20

$$\pi_{\text{FirstName, LastName, Age, StudyYear, SpecName}}\left(\sigma_{\text{Student.Gid=Group.Gid AND Group.SpId=Specialization.SpId AND Sp.SpID=3 AND Age>=20}}\, Student \times Group \times Specialization\right)$$

SQL Statement Execution

Specialization[<u>SpId</u>, SpecName, SpecDescription, NoOfStudents]
Group[<u>Gid</u>, NoOfStudents, StudyYear, *SpId*]
Student[<u>Sid</u>, FirstName, LastName, Age, Email, *Gid*]

SQL query: display the students (FirstName, LastName, Age, StudyYear, SpecName) from a given specialization (e.g. SpId=3 can be a parameter) with the Age>=20 (can be a parameter)

Solution 2:

```
SELECT FirstName, LastName, Age, StudyYear, SpecName
FROM (Student S INNER JOIN Group G ON S.Gid=G.Gid)
INNER JOIN Specialization Sp ON G.SpId=Sp.SpId
WHERE  Sp.SpID=3 AND Age>=20
```

$$\pi_{\text{FirstName, LastName, Age, StudyYear, SpecName}}(\sigma_{Specialization.SpId=3\ AND\ Age\geq 20}(Student \otimes_{Student.Gid=Group.Gid} Group) \otimes_{Specialization.SpId=Group.SpId} Specialization))$$

SQL Statement Execution

Specialization[<u>SpId</u>, SpecName, SpecDescription, NoOfStudents]
Group[<u>Gid</u>, NoOfStudents, StudyYear, *SpId*]
Student[<u>Sid</u>, FirstName, LastName, Age, Email, *Gid*]

SQL query: display the students (FirstName, LastName, Age, StudyYear, SpecName) from a given specialization (e.g. SpId=3 can be a parameter) with the Age>=20 (can be a parameter) * *that are assign in the given group (e.g. Gid=3 can be a parameter)*

Solution 3:

```
SELECT FirstName, LastName, Age, StudyYear, SpecName
FROM ( (SELECT FirstName, LastName, Age, Gid
        FROM Student
        WHERE Age>=20) S
        INNER JOIN (SELECT * FROM Group WHERE SpId=3) G ON S.Gid=G.Gid
        ) INNER JOIN
        (SELECT SpId, SpecName FROM Specialization WHERE SpId=3) Sp ON G.SpId=Sp.SpId
```

$$\pi_{FirstName,LastName,Age,\, StudyYear,\, SpecName}((\pi_{FistName,\, LastName,\, Age,\, Gid}\left(\left(\sigma_{Age>=20}(Student)\right)\right) \otimes_{Gid\_condition} \left(\sigma_{SpId=3}(Group)\right))$$

$$\otimes_{SpId\_condition}\ (\pi_{SpId,SpecName}\left(\left(\sigma_{SpId=3}(Specialization)\right)\right))$$

- An evaluation tree can be constructed for a relational algebra expression
- Problems:
  - Which version is better?
  - When generating the execution plan:
    - Which parameters are optimized?
    - What information is required?
  - What can the Optimizer (a component of the DBMS) do?

Relational Algebra Operators - Evaluation

- Operands for relational operators:
  - database tables (can have attached indexes)
  - temporary tables (obtained by evaluating some relational operators)
- several evaluation algorithms could be used for a relational algebra operator
- when generating the execution plan:
  - choose the algorithm with the lowest complexity (for the current database context); take into account data from the system catalog, statistical information

Relational Algebra Operators - Evaluation

Algorithms

1. Table Scan

- A lot of operators require a full scan of the entire table
- Let b_R be the number of blocks storing a table's records
  - sequential search algorithm - approximately $b_R/2$ blocks are necessary (on average) when performing a sequential search on a key value
  - all blocks must be brought into main memory when performing a sequential search on a non-key field
- High transfer time for large tables

2. Index Seek

- Searching for a key value K_0 by having a condition of the form: $K = K_0$
- Search can be performed
  - explicit (searching a table, evaluating a join)
  - implicit (checking a key constraint)
- Examine an index (stored as a B-tree, B+ tree) created:
  - via a key constraint
  - with the CREATE INDEX statement

where K can be a simple or composite key

Relational Algebra Operators - Evaluation

Algorithms

3. Index Scan

- Evaluating $\sigma_C(R)$, where condition *C* has the form:
  - A < v, A <= v, A > v, A >= v, A IS [NOT] NULL – index built for a key A
  - A = v, A < v, A <= v, A > v, A >= v, A IS [NOT] NULL – index built for a non-key field A
- From the partial / total index scan are obtained desired records' addresses
- Get records from the relation; some blocks can be read multiple times

- A ***join*** can be defined as a cross-product followed by a selection
- Joins arise more often in practice than cross-products
- In general, the result of a cross-product is much larger than the result of a join
- It is important to implement the join without materializing the underlying cross-product, by applying selections and projections as soon as possible, and materializing only the subset of the cross-product that will appear in the result of the join

Relational Algebra Operators - Evaluation

Cross Join

- The algorithm is used to evaluate a cross-product:
  - R CROSS JOIN S
  - R INNER JOIN S ON C (C evaluates to TRUE)
  - SELECT ... FROM R, S ..., no join condition between R and S
- b_R, b_S – the number of blocks storing R and S
- m, n – the number of blocks from R and S that can simultaneously appear in the main memory (there are m+n buffers for the 2 tables)
- Algorithm used to generate the cross-product {(r, s) | r ∈ R, s ∈ S}:
  - for every group of max. m blocks in R:
    - read the group of blocks from R into main memory;
    - let M_1 be the set of records in these blocks
  - for every group of max. n blocks in S:
    - read the group of blocks from S into main memory;
    - let M_2 be the set of records in these blocks
    - for every r ∈ M_1:
      - for every s ∈ M_2:
        - add (r, s) to the result

Relational Algebra Operators - Evaluation

Cross Join

- Algorithm complexity: total number of read blocks (from the 2 tables):

$$b_R + [b_R/m] * b_S$$

(number of blocks in R; for every group of max. m blocks in R, read S)

- To minimize this value, m should be maximized (the other operands are constants); one buffer can be used for S (so n = 1), while the remaining space can be used for R (m max.)
- Switch the 2 relations (in the algorithm and when computing the complexity) => complexity:

$$b_S + [b_S/n] * b_R$$

- Choose better version
- If b_R<=m or b_S<=n => complexity $b_R + b_S$

Nested Loops Join

- The Cross Join algorithm can be used to evaluate a join between 2 tables
- For every element (r, s) in the cross-product, evaluate the condition in the join operator
- The elements (r, s) that don't meet the join condition are eliminated

Relational Algebra Operators - Evaluation

Indexed Nested Loops Join

- The algorithm is used to evaluate $R \otimes_C S$, where $C \equiv (R.A=S.B)$, and there is an index on A (in R) or on B (in S)
- For the algorithm, it is assumed that there is an index on column B in table S
    - for every block in R:
        - read the block into main memory
        - let M be the set of records in the block
        - for every $r \in M$:
            - determine $v = \pi_A (r)$
            - use the index on B in S to determine records $s \in S$ with value v for B
            - for every such record s, the pair (r,s) is added to the result

Depending on the type of index, there can be at most 1 / multiple matching records in S

Relational Algebra Operators - Evaluation

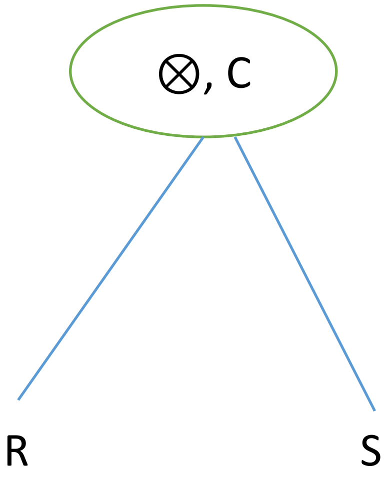
Merge Join

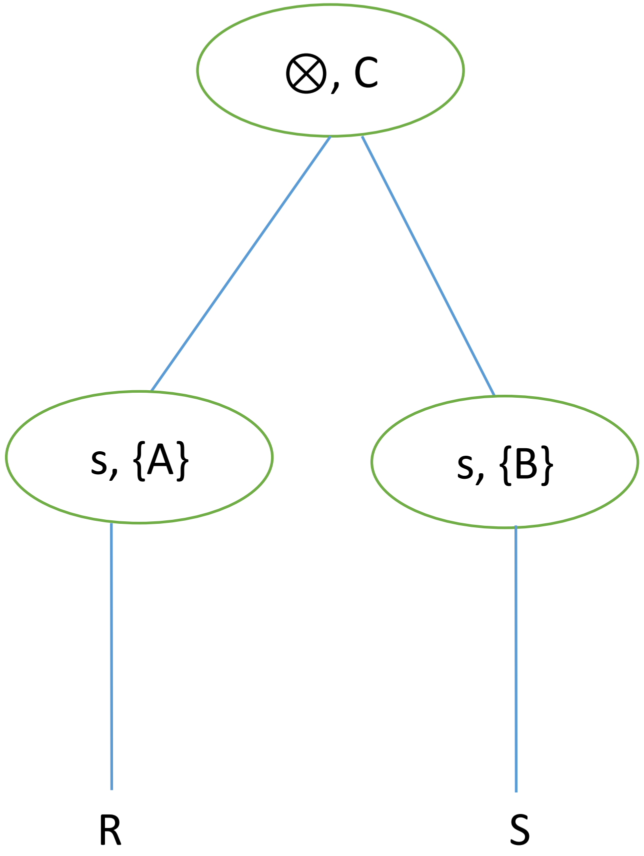
- The algorithm is used to evaluate R $\otimes_C$ S, where C ≡ (R.A=S.B), and there are no indexes on A (in R) and B (in S)
  - sort R and S on the columns used in the join: R on A, S on B
  - scan obtained tables
  - let r in R and s in S be 2 current records
    - if r.A = s.B
      - add (r', s') to the result; r' is in the set of all consecutive records in R with A = r.A
      - add (r', s') to the result; s' is in the set of all consecutive records in S with B = s.B
      - next(r)
      - next(s) (get a record with the next value for A and B)
    - if r.A < s.B
      - next(r) (determine record in sorted R with the next value for A)
    - if r.A > s.B
      - next(s) (determine record in sorted S with the next value for B)

Relational Algebra Operators - Evaluation

Merge Join

- The algorithm replaces an evaluation tree with another evaluation tree





Relational Algebra Operators - Evaluation

Hash Join

- The algorithm is used to evaluate $R \otimes_C S$, where $C \equiv (R.A = S.B)$

a. partitioning phase – hash R and S on the join column, use the same hash function *h* => partitions

b. probing phase – tuples in partition R_x are compared only with tuples in partition S_x (tuples in partition R_1 cannot join with tuples in partition S_2, for instance, as they have a different hash value)

Outer Joins

- The condition join algorithms has to be adapted

Operations on Sets of Records: $R \cup S$, $R - S$, $R \cap S$

- The previous algorithms should be adapted
- For intersection:
  - sort R using all columns
  - sort S using all columns
  - scan sorted R and S, write in the result only the tuples in R that also appear in S

Relational Algebra Equivalences

- SQL statement - transformed into a relational algebra expression (based on a set of transformation rules for the clauses that appear in the statement)
- transform relational expression (such that the evaluation algorithm has a lower complexity)
- certain transformation rules are used (mathematical properties of the relational operators)

- $\sigma_C(\pi_\alpha(R)) = \pi_\alpha(\sigma_C(R))$
  - selection reduces the number of records for projection
  - in the second expression, the projection operator analyzes fewer records
  - optimization - algorithm that evaluates both operators in a single pass of R

- perform one pass instead of 2:

$$\sigma_{C1}(\sigma_{C2}(R)) = \sigma_{C1\ AND\ C2}(R)$$

- replace cross-product and selection by condition join (a number of condition join algorithms don't evaluate the cross-product):

$$\sigma_C(R \times S) = R \otimes_C S$$

where C - join condition between R and S

Relational Algebra Equivalences

- Let R and S be compatible schemas

  $\sigma_C(R \cup S) = \sigma_C(R) \cup \sigma_C(S)$

  $\sigma_C(R \cap S) = \sigma_C(R) \cap \sigma_C(S)$

  $\sigma_C(R - S) = \sigma_C(R) - \sigma_C(S)$

- $\sigma_C (R \times S)$ with the particular cases:
  - C contains only attributes from R:

    $$\sigma_C (R \times S) = \sigma_C (R) \times S$$

  - C = C1 AND C2, C1 contains only attributes from R, C2 - only attributes from S:

    $$\sigma_{C1 \text{ AND } C2} (R \times S) = \sigma_{C1} (R) \times \sigma_{C2} (S)$$

  - C = C1 AND C2, C2 - join condition between R and S:

    $$\sigma_{C1 \text{ AND } C2} (R \times S) = \sigma_{C1}(R \otimes_{C2} S)$$

- $\pi_\alpha (R \cup S) = \pi_\alpha(R) \cup \pi_\alpha (S)$
- $\pi_\alpha (R \otimes_C S) = \pi_\alpha (\pi_{\alpha 1}(R) \otimes_C \pi_{\alpha 2}(S))$

where, $\alpha 1$ are attributes in R that appear in α or C and $\alpha 2$ are attributes in S that appear in α or C

Relational Algebra Equivalences

- associativity and commutativity for some relational operators
  - associativity and commutativity for ∪ and ∩
  - associativity for the cross-product and the natural join
  - "equivalent" results (same records, but different column order) when commuting operands in × and certain join operators
    - R × S = S × R – when using the Cross Join algorithm, the order of the data sources is important

- transitivity of some relational operators for the join operators - additional filters could be applied before the join:
  - (A>B AND B>5) ≡ (A>B AND B>5 AND A>5)

Example: A is in R, B is in S: $R \otimes_{A>B \text{ AND } B>5} S = (\sigma_{A>5}(R)) \otimes_{A>B} (\sigma_{B>5} (S))$

  - (A=B AND B=5) ≡ (A=B AND B=5 AND A=5)

Example: A is in R, B is in S: $R \otimes_{A=B \text{ AND } B=5} S = (\sigma_{A=5}(R)) \otimes_{A=B} (\sigma_{B=5} (S))$

- evaluating $\sigma_C (R)$, where C ≡ (R. A ∈ $\delta(\pi_{\{B\}}(S))$); avoid evaluating C for every record of R; the initial evaluation is equivalent to: $R \otimes_{R.A=S.B} (\delta(\pi_{\{B\}}(S)))$

Relational Algebra Equivalences

Example: Consider the (same) relational database

Specialization[SpId, SpecName, SpecDescription, NoOfStudents]

Group[Gid, NoOfStudents, StudyYear, *SpId*]

Student[Sid, FirstName, LastName, Age, Email, *Gid*]

SQL query: display the students (FirstName, LastName, Age, StudyYear, SpecName) from a given specialization (e.g. SpId=3 can be a parameter) with the Age>=20 (can be a parameter)

```
SELECT FirstName, LastName, Age, StudyYear, SpecName
FROM Student S, Group G, Specialization Sp
WHERE S.Gid=G.Gid AND G.SpId=Sp.SpId AND Sp.SpID=3 AND Age>=20
```

- Denote by C ≡ S.Gid=G.Gid AND G.SpId=Sp.SpId AND Sp.SpID=3 AND Age>=20
- Denote by β = {FirstName, LastName, Age, StudyYear, SpecName} – the attributes from the SELECT clause
- So, the relational expression is

$$\pi_{\beta}\left(\sigma_{\mathrm{C}}\left(Student \times Group \times Specialization\right)\right)$$

Relational Algebra Equivalences

Pay attention to the following transformations, by using the previously discussed rules:

- Associativity for ×
  - $Student \times Group \times Specialization$ = $(Student \times Group) \times Specialization$
  - $Student \times Group \times Specialization$ = $Student \times (Group \times Specialization)$
- Commute σ with × (use a particular case); the transitivity of the equality operator will be
  - (G.SpId=Sp.SpId AND Sp.SpID=3) ≡ (G.SpId=Sp.SpId AND Sp.SpID=3 AND *G.SpId=3*)
- For

S.Gid=G.Gid AND	G.SpId=Sp.SpId AND	Sp.SpID=3 AND	Age>=20 AND	*G.SpId=3*
C1	C2	C3	C4	C5

$\sigma_C\,(Student \times Group \times Specialization) = \sigma_{C1\ AND\ C2}\left(\left(\sigma_{C4}(Student) \times \sigma_{C5}(Group)\right) \times \sigma_{C3}(Specialization)\right)$

or

$\sigma_C\,(Student \times Group \times Specialization) = \sigma_{C1\ AND\ C2}(\sigma_{C4}(Student)) \times (\sigma_{C5}(Group) \times \sigma_{C3}(Specialization))$

- Replace selection and cross-product with condition join

$\sigma_C\,(Student \times Group \times Specialization) = ((\sigma_{C4}(Student)) \otimes_{C1} (\sigma_{C5}(Group))) \otimes_{C2} (\sigma_{C3}(Specialization))$

or

$\sigma_C\,(Student \times Group \times Specialization) = ((\sigma_{C4}(Student))) \otimes_{C1} ((\sigma_{C5}(Group)) \otimes_{C2} (\sigma_{C3}(Specialization)))$

Relational Algebra Equivalences

- Choose a version based on statistical information from the database; next it is considered the first version:

$$e = \pi_\beta(((\sigma_{C4}(Student)) \otimes_{C1} (\sigma_{C5}(Group)))\otimes_{C2}(\sigma_{C3}(Specialization)))$$

- commute π with join
  - $\beta 1 = \{FirstName, LastName, Age, Gid\}$ - useful for β and join
  - $\beta 2 = \{Gid, StudyYear, SpId\}$ - useful for β and join
  - $\beta 3 = \{SpId, SpecName\}$ - useful for β and join

$$e = \pi_\beta(((\pi_{\beta 1}(\sigma_{C4}(Student)) \otimes_{C1}(\pi_{\beta 2}(\sigma_{C5}(Group))))\otimes_{C2}(\pi_{\beta 3}(\sigma_{C3}(Specialization))))$$

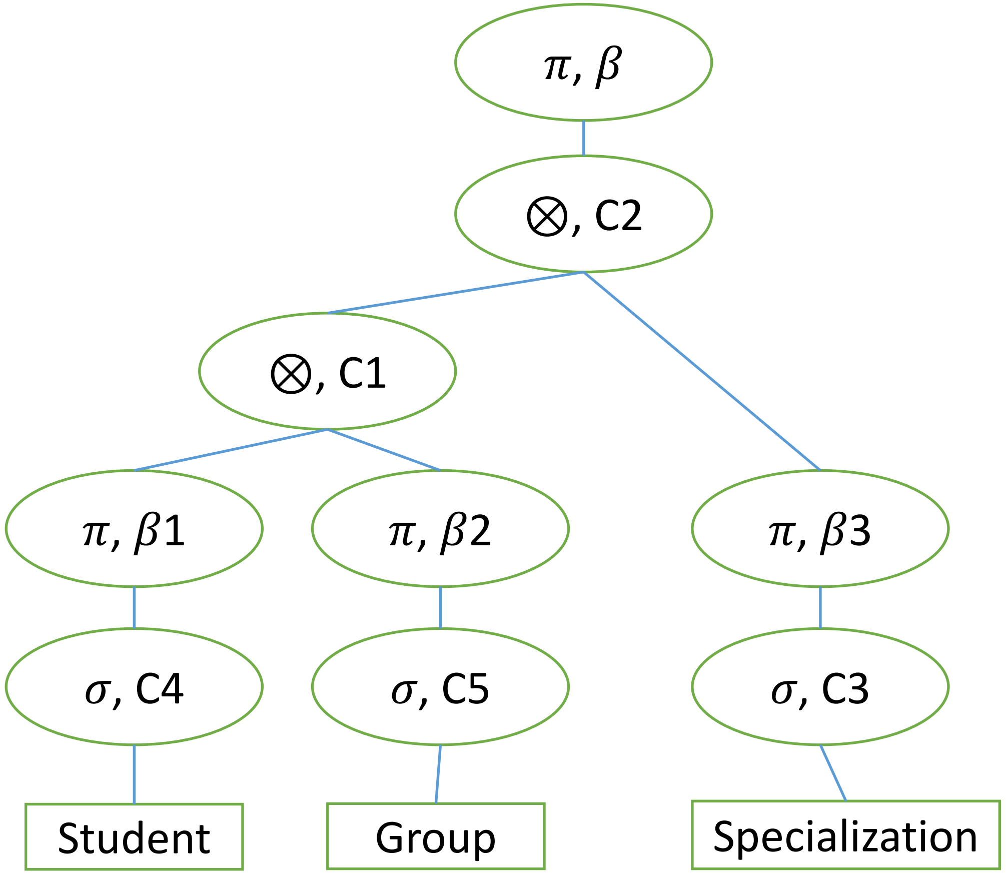
- This expression correspond to *Solution 3:*

```
SELECT FirstName, LastName, Age, StudyYear, SpecName
FROM (
        (SELECT FirstName, LastName, Age, Gid
        FROM Student
        WHERE Age>=20) S
        INNER JOIN (SELECT * FROM Group WHERE Gid=3) G ON S.Gid=G.Gid
        ) INNER JOIN
        (SELECT SpId, SpecName FROM Specialization WHERE SpId=3) Sp ON G.SpId=Sp.SpId
```

Relational Algebra Equivalences

- An evaluation tree can be constructed for the last version of the relational algebra expression
- Using information from the system catalog and possibly statistical information, an execution plan can be generated from the last version of the expression; every relational operator is replaced by an evaluation algorithm

$$e = \pi_{\beta}(((\pi_{\beta 1}(\sigma_{C4}(Student)) \otimes_{C1} (\pi_{\beta 2}(\sigma_{C5}(Group))))\otimes_{C2}(\pi_{\beta 3}(\sigma_{C3}(Specialization))))$$



References:

- C.J. Date, *An Introduction to Database Systems (8th Edition)*, Addison-Wesley, 2003.
- H. Garcia-Molina, J. Ullman, J. Widom, *Database Systems: The Complete Book, Prentice Hall Press*, 2008.
- G. Hansen, J. Hansen, *Database Management And Design (2nd Edition),* Prentice Hall, 1996.
- R. Ramakrishnan, J. Gehrke, *Database Management Systems*, McGraw- Hill, 2007. http://pages.cs.wisc.edu/~dbbook/openAccess/thirdEdition/slides/slides3ed.html
- R. Ramakrishnan, J. Gehrke, *Database Management Systems (2nd Edition),* McGraw-Hill, 2000.
- A. Silberschatz, H. Korth, S. Sudarshan, *Database System Concepts*, McGraw-Hill, 2010. http://codex.cs.yale.edu/avi/db-book/
- L. Țâmbulea, *Curs Baze de date*, Facultatea de Matematică și Informatică, UBB, 2013-2014.
- J. Ullman, J. Widom, *A First Course in Database Systems*, http://infolab.stanford.edu/~ullman/fcdb.html