Data Management

Maurizio Lenzerini

Dipartimento di Informatica e Sistemistica "Antonio Ruberti"
Sapienza Università di Roma

Academic year 2019/2020

Part 1
Introduction to the course

http://www.dis.uniroma1.it/~lenzerini/home/?q=node/53

This course is for …

- Students of the Master of Science in Engineering in Computer Science and Master in Ingegneria Gestionale
  - 6 credits
- Prerequisites
  - A good knowledge of the fundamentals of Programming Structures and Programming Languages
  - A good knowledge of the fundamentals of Databases, in particular SQL, relational data model, Entity-Relationship data model, conceptual and logical database design

Objectives

- *Knowledge on the structure and the functionalities of Data Management systems from the point of view of data administrators*
- *Knowledge on the structure and the functionalities of Data Management systems from the point of view of Data Management tool designers*
- *Other advanced topics in data management*

Organization of the course

Teacher: Maurizio Lenzerini

Home page of Prof. Maurizio Lenzerini

http://www.dis.uniroma1.it/~lenzerini

Home page of the course:

http://www.dis.uniroma1.it/~lenzerin/home/?q=node/53

Office hours:

- Tuesday, 5:00 pm
- Dipartimento di Ingegneria Informatica Automatica e Gestionale Antonio Ruberti,

  Via Ariosto 25, 2nd floor, room B203 (if available), or room B217 (otherwise)

Organization of the course

- Lectures (via Eudossiana 18):
  - Monday, 10:00 am – 13:00 am (Classroom 41)
  - Wednesday, 08:00 pm – 10:00 pm (Classroom 41)

- Exercises during lectures

- Exam
  - written exam
  - oral exam (if needed)

Organization of the course

Material

- M. Lenzerini, Lecture notes, Available for download from the Moodle system
- R. Ramakrishnan, J. Gehrke. Database Management Systems. McGraw-Hill
- Papers on specific topics

- More material available in the course web site
  - exercises
  - problems proposed in past exams

Course topics

- The structure of a Data Base Management System (DBMS)
  - Relational data and queries
  - Buffer manager
- Transaction management
  - The concept of transaction
  - Concurrency management
- Crash management
  - Classification of failures
  - Recovery
- Physical structures for data bases
  - File organizations for data base management
  - Principles of physical database design
- Query processing
  - Evaluation of relational algebra operators
  - Fundamentals of query optimization
- Advanced topics in data management
  - Datawarehousing
  - OLTP vs OLAP
  - NoSQL systems

The Relational Data Model (E.F. Codd – 1970)

CHECKING-ACCOUNT Table

branch-name	account-no	customer-name	balance
Orsay	10991-06284	Abiteboul	$3,567.53
Hawthorne	10992-35671	Hull	$11,245.75
…	…	…	…

- The Relational Data Model uses the mathematical concept of a relation as the formalism for describing and representing data.
- Question: What is a relation?
- Answer:
  – Mathematically speaking, a relation is a subset of a cartesian product of sets.
  – A relation can be considered as a "table" with rows and columns.

Query Languages for the Relational Data Model

Codd introduced two different query languages for the relational data model:

- Relational Algebra, which is a procedural language.
  - It is an algebraic formalism in which queries are expressed by applying a sequence of operations to relations.
- Relational Calculus, which is a declarative language.
  - It is a logical formalism in which queries are expressed as formulas of first-order logic.

Codd's Theorem: Relational Algebra and Relational Calculus are *essentially equivalent in terms of expressive power*.

DBMSs are based on yet another language, namely SQL, a hybrid of a procedural and a declarative language that combines features from both relational algebra and relational calculus.

The Five Basic Operations of Relational Algebra

Operators of Relational Algebra:

- Group I: Three standard set-theoretic binary operations:
  - Union
  - Difference
  - Cartesian Product
- Group II. Two special unary operations on relations:
  - Projection
  - Selection

Note: Renaming can be expressed by Projection

- Relational Algebra consists of all expressions obtained by combining these five basic operations in syntactically correct ways.

- If you want to try using Relational Algebra, go to https://users.cs.duke.edu/~junyang/radb/

Relational Algebra: Standard Set-Theoretic Operations

- Union
  - Input: Two k-ary relations R and S, for some k.
  - Output: The k-ary relation $R \cup S$, where
    $R \cup S = \{(a_1,\ldots,a_k): (a_1,\ldots,a_k)$ is in R or $(a_1,\ldots,a_k)$ is in S$\}$
- Difference:
  - Input: Two k-ary relations R and S, for some k.
  - Output: The k-ary relation R - S, where
    $R - S = \{(a_1,\ldots,a_k): (a_1,\ldots,a_k)$ is in R and $(a_1,\ldots,a_k)$ is not in S$\}$

- Note:
  - In relational algebra, both arguments of the union and the difference must be relations of the same arity.
  - In SQL, there is the additional requirement that the corresponding attributes must have the same data type.
  - However, the corresponding attributes need not have the same names; the corresponding attribute in the result can be renamed arbitrarily.

Union

Employee

Code	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Director

Code	Name	Age
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Employee ∪ Director

Code	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45
9297	Neri	33

Difference

Employee

Code	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Director

Code	Name	Age
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Employee – Director

Code	Name	Age
7274	Rossi	42

Relational Algebra: Cartesian Product

- Cartesian Product
  - Input: An m-ary relation R and an n-ary relation S
  - Output: The (m+n)-ary relation $R \times S$, where

    $R \times S = \{(a_1,\ldots,a_m,b_1,\ldots,b_n): (a_1,\ldots a_m)$ is in R and $(b_1,\ldots,b_n)$ is in S$\}$

- Note:

$$|R \times S| = |R| \times |S|$$

Relational Algebra: Cartesian Product

Employee

Emp	Dept
Rossi	A
Neri	B
Bianchi	B

Dept

Code	Chair
A	Mori
B	Bruni

Employee × Dept

Emp	Dept	Code	Chair
Rossi	A	A	Mori
Rossi	A	B	Bruni
Neri	B	A	Mori
Neri	B	B	Bruni
Bianchi	B	A	Mori
Bianchi	B	B	Bruni

The Projection Operation

- Motivation: It is often the case that, given a table R, one wants to rearrange the order of the columns and/or suppress/rename some columns

- Projection is a family of unary operations of the form

$$\pi_{\text{<attribute list>}} (\text{<relation name>})$$

or

$$\text{PROJ}_{\text{<attribute list>}} (\text{<relation name>})$$

- The intuitive description of the projection operation is as follows:
  - When the projection is applied to a relation R, it removes all columns whose attributes do not appear in the <attribute list>
  - The remaining columns may be re-arranged (and also renamed) according to the order in the <attribute list>
  - Any duplicate rows are eliminated

The Projection Operation

– Show name and Site of employees

Employee

Name	Site
Neri	Napoli
Neri	Milano
Rossi	Roma

$$\text{PROJ}_{\text{Name, Site}}(\text{Employee})$$

– To rename: $\text{PROJ}_{N \leftarrow \text{Name},\ A \leftarrow \text{Age}}(\text{Employee})$

More on the Syntax of the Projection Operation

- In relational algebra, attributes can be referenced by position number
- Projection Operation:
  - Syntax: $\pi_{i_1,\ldots,i_m}(R)$, where R is of arity k, and $i_1,\ldots,i_m$ are distinct integers from 1 up to k.
  - Semantics:

$$\pi_{i_1,\ldots,i_m}(R) = \{ (a_1,\ldots,a_m) : \text{there is a tuple } (b_1,\ldots,b_k) \text{ in } R \text{ such that } a_1=b_{i_1}, \ldots, a_m=b_{i_m} \}$$

- Example: If R is R(A,B,C,D), then $\pi_{C,A}(R) = \pi_{3,1}(R)$

$\pi_{3,1}(R) = \{(a_1,a_2)$: there is (a,b,c,d) in R such that $a_1=c$ and $a_2=a\}$

The Selection Operation

- Motivation: Given SAVINGS(branch-name, acc-no, cust-name, balance) we may want to extract the following information from it:
  - Find all records in the Aptos branch
  - Find all records with balance at least $50,000
  - Find all records in the Aptos branch with balance less than $1,000
- Selection is a family of unary operations of the form

$$\sigma_{\Theta}(R) \qquad \text{OR} \qquad SEL_{\Theta}(R)$$

  where R is a relation and Θ is a condition that can be applied as a test to each row of R.
- When a selection operation is applied to R, it returns the subset of R consisting of all rows that satisfy the condition Θ
- Question: What is the precise definition of a "condition"?

The Selection Operation

- Definition: A condition in the selection operation is an expression built up from:
  - Comparison operators =, <, >, ≠, ≤, ≥ applied to operands that are constants or attribute names or component numbers.
    - These are the basic (atomic) clauses of the conditions.
  - The Boolean logic operators ∧, ∨, : applied to basic clauses.
- Examples:
  - balance > 10,000
  - branch-name = “Aptos”
  - (branch-name = “Aptos”) ∧ (balance < 1,000)

The Selection Operator

- Note:
  - The use of the comparison operators <, >, ≤, ≥ assumes that the underlying domain of values is totally ordered.
  - If the domain is not totally ordered, then only = and ≠ are allowed.
  - If we do not have attribute names (hence, we can only reference columns via their component number), then we need to have a special symbol, say $, in front of a component number. Thus,
    - $4 > 100 is a meaningful basic clause
    - $1 = "Aptos" is a meaningful basic clause, and so on.

The Selection Operator

– Show the employees whose salary is greater than 50

Employee

Code	Name	Site	Salary
7309	Rossi	Roma	55
5998	Neri	Milano	64
5698	Neri	Napoli	64

$$\sigma_{\text{Salary} > 50}\ (\text{Employee})$$

Relational Algebra Expression

- Definition: A relational algebra expression is an expression obtained from relation schemas using union, difference, cartesian product, projection, and selection.
- Context-free grammar for relational algebra expressions:

  $E := R, S, \ldots \mid (E_1 \cup E_2) \mid (E_1 - E_2) \mid (E_1 \times E_2) \mid \pi_X(E) \mid \sigma_\Theta(E)$,

  where
  - R, S, … are relation schemas
  - X is a list of attributes
  - Θ is a condition.

Derived Operation: Intersection

- Intersection
  - Input: Two k-ary relations R and S, for some k.
  - Output: The k-ary relation $R \cap S$, where

  $R \cap S = \{(a_1,\ldots,a_k)$: $(a_1,\ldots,a_k)$ is in R and $(a_1,\ldots,a_k)$ is in S$\}$

- Fact: $R \cap S = R - (R - S) = S - (S - R)$

Thus, intersection is a derived relational algebra operation.

Intersection: example

Employee

Code	Name	Age
7274	Rossi	42
7432	Neri	54
9824	Verdi	45

Director

Code	Name	Age
9297	Neri	33
7432	Neri	54
9824	Verdi	45

Employee $\cap$ Director

Code	Name	Age
7432	Neri	54
9824	Verdi	45

Derived Operation: Θ–Join and Beyond

Definition: A Θ-Join is a relational algebra expression of the form

$$\sigma_{\Theta}(R \times S) \quad \text{OR} \quad R \text{ JOIN}_{\Theta} \; S$$

Note:

- If R and S have attribute A in common, then we use the notation R.A and S.A to disambiguate.
- The Θ-Join selects those tuples from $R \times S$ that satisfy the condition Θ. In particular, if every tuple in $R \; \Theta \; S$ satisfies Θ, then

$$\sigma_{\Theta}(R \times S) = R \times S$$

Θ–Join and Beyond

- Θ-joins are often combined with projection to express interesting queries.
- Example: F(name, dpt, salary), C(dpt, name), where F stands for FACULTY and C stands for CHAIR
  - Find the salaries of department chairs

    C-SALARY(dpt,salary) =

    $\pi_{F.dpt,\ F.salary}(\sigma_{F.name\ =\ C.name}\ (F \times C))$

Note: The Θ-Join in this example is an equijoin, since Θ is a conjunction of equality basic clauses.

Exercise: Show that the intersection $R \cap S$ can be expressed using a combination of projection and an equijoin.

Θ–Join and Beyond

Example: F(name, dpt, salary), C-SALARY(dpt, salary)

Find the names of all faculty members of the EE department who earn a bigger salary than their department chair.

HIGHLY-PAID-IN-EE(Name) =

$\pi_{F.name} (\sigma_{F.dpt = \text{“EE”} \wedge F.dpt = C.dpt \wedge F.salary > C.salary} (F \times \text{C-SALARY}))$

Note: The Θ-Join above is not an equijoin.

Derived Operation: Natural Join

The natural join between two relations is essentially the equi-join on common attributes.

Given TEACHES(facname, course, term) and
ENROLLS(studname, course, term), we compute the natural join
TAUGHT-BY(studname, course, term, facname) by:

$$\pi_{\text{E.studname, E.course, E.term. ,E.course, T.facname}} (\sigma_{\text{T.course = E.course} \wedge \text{T.term = E.term}} (\text{ENROLLS} \times \text{TEACHES}))$$

The resulting expression can be written using this notation:

ENROLLS $\bowtie$ TEACHES   OR   ENROLLS JOIN TEACHES

Natural Join

- Definition: Let A1, …, Ak be the common attributes of two relation schemas R and S. Then

$$R \bowtie S = \pi_{<list>} (\sigma_{R.A1=S.A1 \wedge \ldots \wedge R.A1=S.Ak}(R \times S)),$$

where <list> contains all attributes of $R \times S$, except for S.A1, …, S.Ak (in other words, duplicate columns are eliminated).

- Algorithm for $R \bowtie S$:

  For every tuple in R, compare it with every tuple in S as follows:
  - test if they agree on all common attributes of R and S;
  - if they do, take the tuple in $R \times S$ formed by these two tuples,
    - remove all values of attributes of S that also occur in R;
    - put the resulting tuple in $R \bowtie S$.

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the Relational Algebra queries for:

1. Find the cities with at least one school with a student who graduated with 100.
2. Find the schools where no student has graduated with 100.
3. Find the cities where all schools have a student who graduated with 100
4. For all school, find the student(s) who graduated with the minimum grade in the school.

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the Relational Algebra query for:

1. Find the cities with at least one school with a student who graduated with 100.

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the Relational Algebra query for:

1. Find the cities with at least one school with a student who graduated with 100.

$PROJ_{city}$ ($SEL_{mark=100}$ (Graduated) $JOIN_{school=scode}$ School))

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the Relational Algebra query for:

2. Find the schools where no student has graduated with 100.

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the Relational Algebra query for:

2. Find the schools where no student has graduated with 100.

$PROJ_{scode}$ (School) – $PROJ_{school}$ ($SEL_{mark=100}$ (Graduated))

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the Relational Algebra query for:

2. Find the schools where no student has graduated with 100.

$\text{PROJ}_{\text{scode}}$ (School) –

$\boxed{\text{PROJ}_{\text{scode}}\ (\text{SEL}_{\text{mark=100}}\ (\text{Graduated})\ \text{JOIN}_{\text{school=scode}}\ \text{School}))}$

The scode of the schools with at least one student graduated with 100

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the Relational Algebra query for:

3. Find the cities where all schools have a student who graduated with 100

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the Relational Algebra query for:

3. Find the cities where all schools have a student who graduated with 100

The cities with some school with no student graduated with 100

$PROJ_{city}$ (School) –

$PROJ_{city}$ (School –

$PROJ_{scode,city}$($SEL_{mark=100}$ (Graduated) $JOIN_{school=scode}$ School)

)

The schools with at least one student graduated with 100

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the Relational Algebra query for:

4. For all school, find the student(s) who graduated with the minimum grade in the school.

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the Relational Algebra query for:

4. For all school, find the student(s) who graduated with the minimum grade in the school.

$$\text{PROJ}_{\text{school,gcode}} (\text{Graduated} - \text{PROJ}_{\text{gcode,mark,school}}(\text{Graduated JOIN}_{\text{mark>m and school=s}} \text{PROJ}_{\text{m} \leftarrow \text{mark, s} \leftarrow \text{school}} (\text{Graduated})))$$

SQL: Structured Query Language

- SQL is the standard language for relational DBMSs
- We will present the syntax of the core SQL constructs and then will give rigorous semantics by interpreting SQL to Relational Algebra.
- Note: SQL typically uses multiset semantics, but we ignore this property here, and we only consider the set-based semantics (adopted by using the keyword DISTINCT in queries)

SQL: Structured Query Language

- The basic SQL construct is:

  SELECT DISTINCT <attribute list>

  FROM <relation list>

  WHERE <condition>

- More formally,

  SELECT DISTINCT R_{i1}.A1, … , R_{im}.Am

  FROM R_1, … ,R_K

  WHERE γ

Restrictions:

- R_1, … ,R_K are relation names (possibly, with aliases for renaming, where an alias S for relation name R_i is denoted by R_i AS N)
- Each R_{ij}.Aj is an attribute of R_{ij}
- γ is a condition with a precise (and rather complex) syntax.

SQL vs. Relational Algebra

SQL	**Relational Algebra**
SELECT	Projection
FROM	Cartesian Product
WHERE	Selection

Semantics of SQL via interpretation to Relational Algebra:

SELECT DISTINCT $R_{i1}.A1, \ldots, R_{im}.Am$

FROM $R_1, \ldots, R_K$

WHERE γ

corresponds to

$$\pi_{R_{i1}.A1, \ldots, R_{im}.Am} (\sigma_\gamma (R_1 \times \ldots \times R_K))$$

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the SQL queries for:

1. Find the cities with at least one school with a student who graduated with 100.
2. Find the schools where no student has graduated with 100.
3. Find the cities where all schools have a student who graduated with 100
4. For all school, find the student(s) who graduated with the minimum grade in the school.

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the SQL query for:

1. Find the cities with at least one school with a student who graduated with 100.

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the SQL query for:

1. Find the cities with at least one school with a student who graduated with 100.

```
select city
from Graduated join School on school = scode
where mark = 100
```

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the SQL query for:

2. Find the schools where no student has graduated with 100.

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the SQL query for:

2. Find the schools where no student has graduated with 100.

```
select scode
from School
where scode not in (select scode
                    from Graduate join School on school=scode
                    where mark = 100)
```

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the SQL query for:

3. Find the cities where all schools have a student who graduated with 100

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the SQL query for:

3. Find the cities where all schools have a student who graduated with 100

```
select city from School
where city not in (select city from School
                          where scode not in (select scode from Graduate
                                                     where mark = 100)
```

Exercises

Consider a database with relations

Graduated(gcode,mark,school)

School(scode,city)

and write the SQL query for:

4. For all school, find the student(s) who graduated with the minimum grade in the school.

SQL: Structured Query Language

- SQL is the standard language for relational DBMSs
- We will present the syntax of the core SQL constructs and then will give rigorous semantics by interpreting SQL to Relational Algebra.
- Note: SQL typically uses multiset semantics, but we ignore this property here, and we only consider the set-based semantics (adopted by using the keyword DISTINCT in queries)

SQL: Structured Query Language

- The basic SQL construct is:

  SELECT DISTINCT <attribute list>

  FROM <relation list>

  WHERE <condition>

- More formally,

  SELECT DISTINCT R_{i1}.A1, … , R_{im}.Am

  FROM R_1, … ,R_K

  WHERE γ

Restictions:

- R_1, … ,R_K are relation names (possibly, with aliases for renaming, where an alias S for relation name R_i is denoted by R_i AS N)
- Each R_{ij}.Aj is an attribute of R_{ij}
- γ is a condition with a precise (and rather complex) syntax.