Introduction to Databases

Maurizio Lenzerini

Sapienza
Università di Roma

Acknowledgments

This material is based on a set of slides prepared by Prof. Phokion Kolaitis (University of California, Santa Cruz, USA)

I thank Prof. Phokion Kolaitis for letting me use his material

Outline

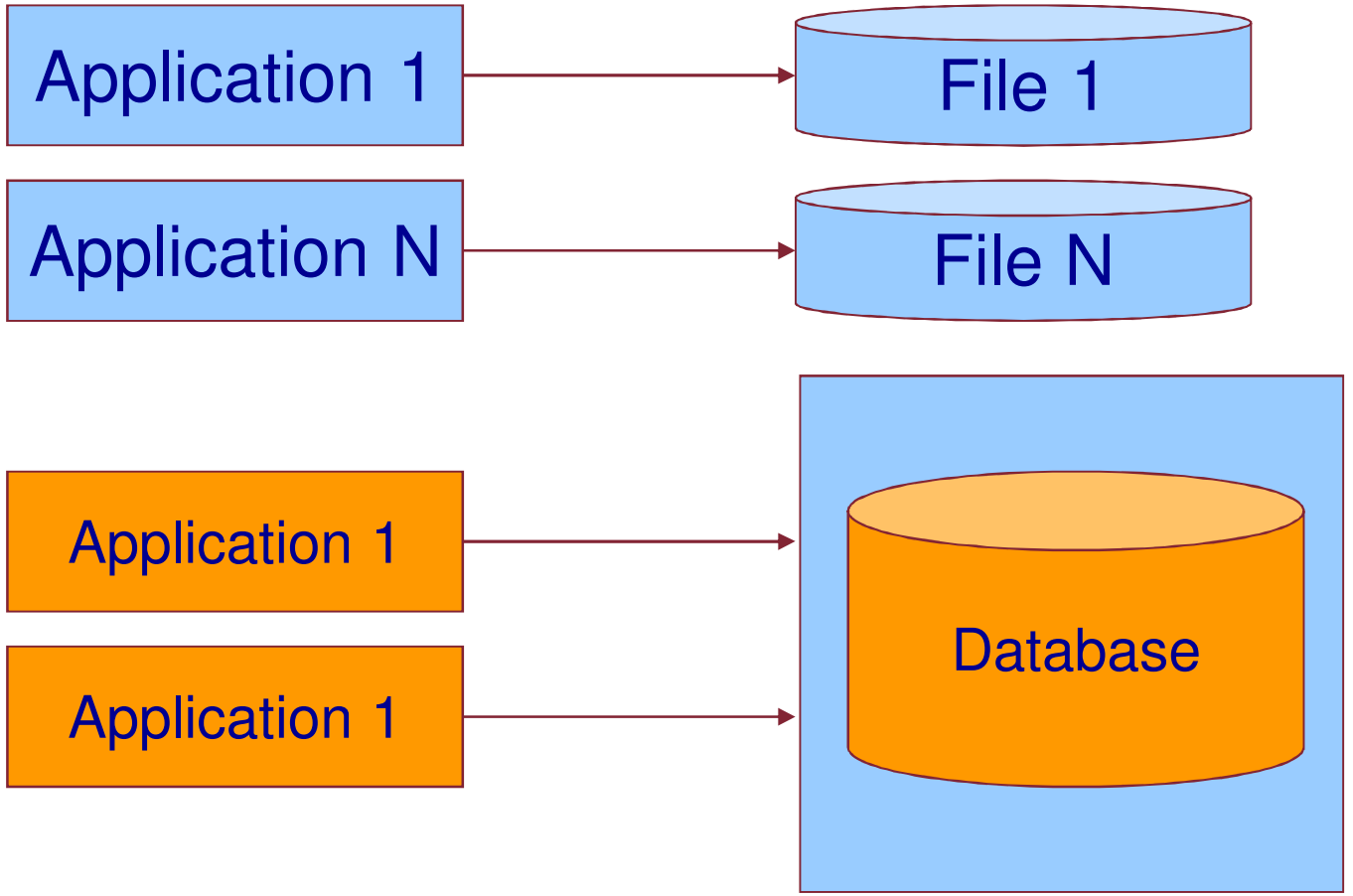
1. The notion of database
2. The relational model of data
3. The relational algebra
4. SQL

Outline

1. The notion of database
2. The relational model of data
3. The relational algebra
4. SQL

The Notion of Database

The term “database” may refer to any collection of data stored in a computing system. Here, we use it with a specific meaning: integrated repository of the set of all relevant data of an organization.

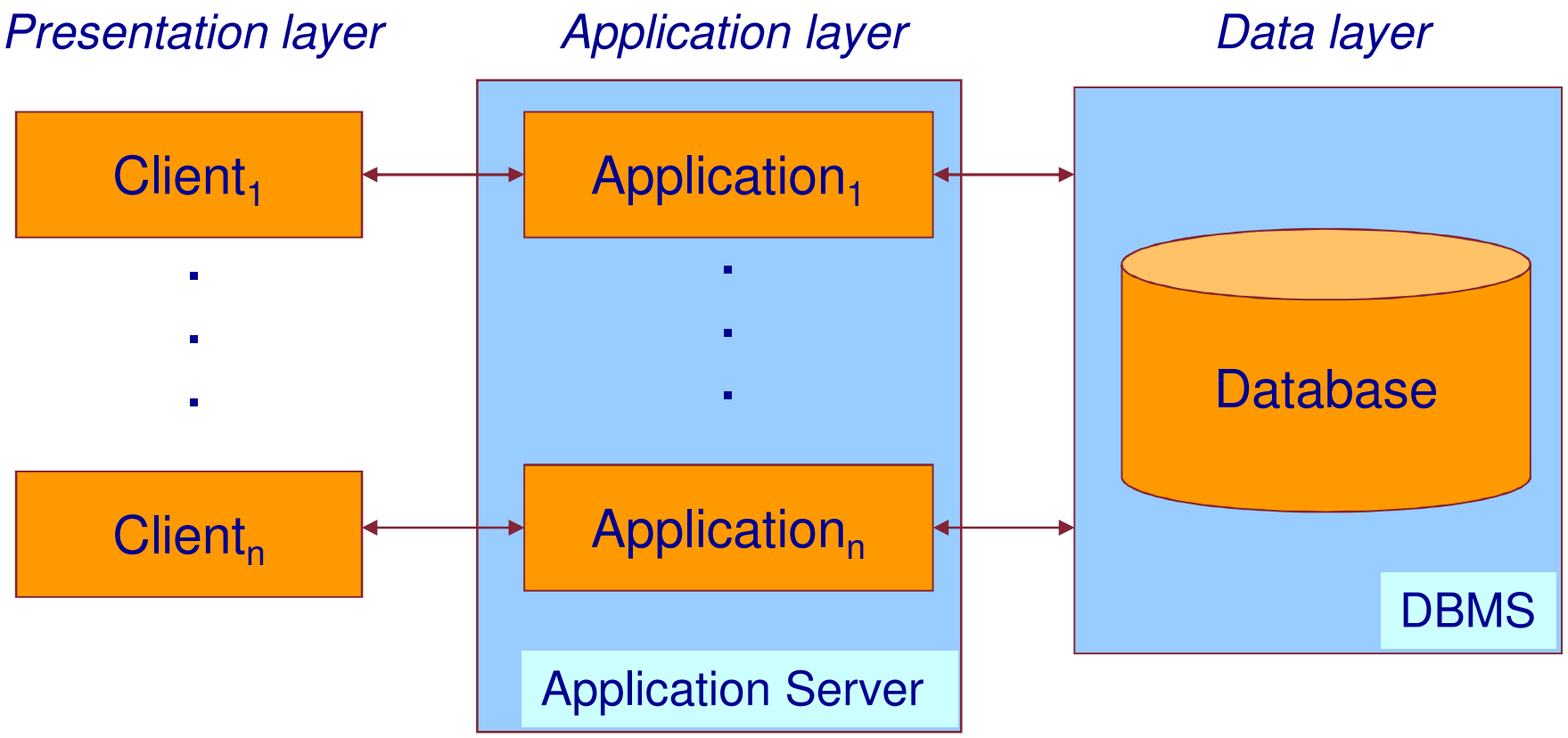


pre-database situation (60’s-70’s)

post-database situation

Three-layer Software Architecture

The Database Management System (DBMS) is the software system responsible of managing the database. Data in the database are accessible only through such system.



Databases and Database Management Systems

- A **database** is a collection of inter-related data organized in particular ways, and managed by a DBMS.
- A **database management system** (**DBMS**) is a set of programs that allows one to carry out at least the following tasks:
  - Create a (persistent) database.
  - Insert, delete, modify (update) data in a database.
  - Query a database "efficiently (ask questions and extract information from the database).
  - Ensuring "correctness" and "availability" in data management

- DBMS's are different from File Systems
  - Example: "Find all customers whose address has 95060 as zip code" is an easy task for a DBMS, but may require a new program to be written in a file system.

Key Characteristics of DBMS's

Every DBMS must provide support for:

- A **Data Model**: A mathematical abstraction for representing/organizing data.

- At least one high-level **Data Language:** Language for defining, updating, manipulating, and retrieving data.

- **Mechanisms for specifying and checking Integrity Constraints**:
  Rules ad restrictions that the data at hand must obey – e.g., *different people must have different SSNs.*

- **Transaction management**, **concurrency control** & **recovery mechanisms:**
  Must not confuse simultaneous actions – e.g.,
  two deposits to the same account must each credit the account.

- **Access control:**
  Limit access of certain data to certain users.

Applications of Database Management Systems

- Traditional applications:
  - Institutional records
    - Government, Corporate, Academic, …
    - Payroll, Personnel Records, …
  - Airline Reservation Systems
  - Banking Systems

- Numerous new applications:
  - Scientific Databases
  - Electronic Health Records
  - Information Integration from Heterogeneous Sources
  - Databases are behind most of the things one does on the web:
    - Google searches, Amazon purchases, eBay auctions, …

Data Languages

A Data Language has two parts:

- A Data Definition Language (DDL) has a syntax for describing "database templates" in terms of the underlying data model.

- A Data Manipulation Language (DML) supports the following operations on data:
  - Insertion
  - Deletion
  - Update
  - Retrieval and extraction of data (query the data).

  The first three operations are fairly standard. However, there is much variety on data retrieval and extraction (Query Languages).

A Brief History of Data Models

- Earlier Data Models (before 1970)
  - Hierarchical Data Model
    - Based on the mathematical notion of a tree.
  - Network Data Model
    - Based on the mathematical notion of a graph.
- Relational Data Model – 1970
  - Based on the mathematical notion of a relation.
- Entity-Relationship Model – 1976
  - Conceptual model; used mainly as a design tool.
- Semi-structured Data Model and XML – late 1990s
  - Based on SGML and the mathematical notion of a tree (the Hierarchical Model strikes back!).
- Data Model of Graph-databases – 2000s

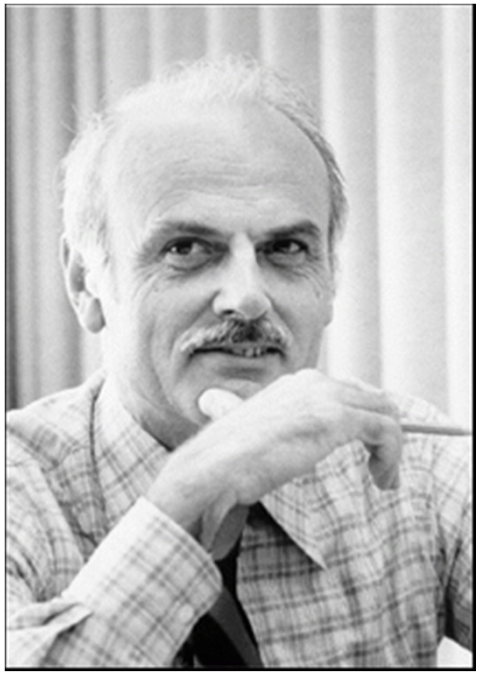
Relational Databases: A Very Brief History

- The history of relational databases is the history of a scientific and technological revolution.

- The scientific revolution started in 1970 by Edgar (Ted) F. Codd at the IBM San Jose Research Laboratory (now the IBM Almaden Research Center)

- Codd introduced the relational data model and two database query languages: relational algebra and relational calculus.
  - "A relational model for data for large shared data banks", CACM, 1970.
  - "Relational completeness of data base sublanguages", in: Database Systems, ed. by R. Rustin, 1972.

Edgar F. Codd, 1923-2003



Relational Databases: A Very Brief History

- Researchers at the IBM San Jose Laboratory embark on the System R project, the first implementation of a relational database management system (RDBMS) – see the paper by Astrahan et al.
  - In 1974-1975, they develop SEQUEL, a query language that eventually became the industry standard SQL.
  - System R evolved to DB2 – released first in 1983.
- M. Stonebraker and E. Wong embark on the development of the Ingres RDBMS at UC Berkeley in 1973.
  - Ingres is commercialized in 1983; later, it became PostgreSQL, a free software OODBMS (object-oriented DBMS).
- L. Ellison founds a company in 1979 that eventually becomes Oracle Corporation; Oracle V2 is released in 1979 and Oracle V3 in 1983.
- Ted Codd receives the ACM Turing Award in 1981.
- Database research is still very active today

Outline

1. The notion of database
2. The relational model of data
3. The relational algebra
4. SQL

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

- 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:
  - Formally, a relation is a subset of a cartesian product of sets.
  - Informally, a relation is a "table" with rows and columns.

CHECKING-ACCOUNT Table

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

Basic Notions from Discrete Mathematics

- A k-tuple is an ordered sequence of k objects (need not be distinct)
  - (2,0,1) is a 3-tuple; (a,b,a,a,c) is a 5-tuple, and so on.

- If $D_1, D_2, \ldots, D_k$ are k sets, then the cartesian product $D_1 \times D_2 \ldots \times D_k$ of these sets is the set of all k-tuples $(d_1,d_2, \ldots,d_k)$ such that $d_i \subseteq D_i$, for $1 \leq i \leq k$.

- Fact: Let $|D|$ denote the cardinality (# of elements) of a set D. Then $|D_1 \times D_2 \times \ldots \times D_k| = |D_1| \times |D_2| \times \ldots \times |D_k|$.

- Example: If $D_1 = \{0,1\}$ and $D_2 = \{a,b,c,d\}$, then $|D_1 \times D_2| = 8$.

- Warning: In general, computing a cartesian product is an expensive operation!

Basic Notions from Discrete Mathematics

- A k-ary relation R is a subset of a cartesian product of k sets, i.e., $R \subseteq D_1 \times D_2 \times \ldots \times D_k$.

- Examples:
  - Unary $R = \{0,2,4,\ldots,100\}$ $(R \subseteq N)$
  - Binary $L = \{(m,n): m < n\}$ $(L \subseteq N \times N)$
  - Binary $T = \{(a,b)$: a and b have the same birthday$\}$
  - Ternary $S = \{(m,n,s): s = m+n\}$
  - …

Relations and Attributes

$R \subseteq D_1 \times D_2 \times \ldots \times D_k$ can be viewed as a table with k columns

Definition: An attribute is the name of a position (column) of a relation (table).

In the **CHECKING-ACCOUNT** Table below, the attributes are **branch-name**, **account-no**, **customer-name**, and **balance**.

CHECKING-ACCOUNT Table

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

Relation Schemas and Relations

Definition: A k-ary relation schema **R**(A_1,A_2,…,A_K) is a named ordered sequence (A_1,A_2,…,A_k) of k attributes (where each attribute may have a data type declared).

Examples:

- **COURSE**(course-no, course-name, term, instructor, room, time)
- **CITY-INFO**(name, state, population)
- Option: course-no:integer, course-name:string

Thus, a k-ary relation schema is a "blueprint", a "template" or a "structure specification" for some k-ary relation.

Definition: An instance of a relation schema is a relation conforming to the schema:

- The arities must match;
- If declared, the data types must match.

Relational Database Schemas and Relational Databases

Definition: A relational database schema is a set of relation schemas $\mathbf{R_i}(A_1,A_2,\ldots,A_{k_i})$, for $1 \leq i \leq m$.

Example: BANKING relational database schema with relation schemas

– CHECKING-ACCOUNT(branch, acc-no, cust-id, balance)

– SAVINGS-ACCOUNT(branch, acc-no, cust-id, balance)

– CUSTOMER(cust-id, name, address, phone, email)

– ….

Definition: A relational database instance or, simply, a relational database of a relational schema is a set of relations R_i each of which is an instance of the corresponding relation schema $\mathbf{R_i}$, for each $1 \leq i \leq m$.

Relational Database Schemas - Examples

Examples:

- BANKING relational database schema with relation schemas
  - CHECKING-ACCOUNT(branch, acc-no, cust-id, balance)
  - SAVINGS-ACCOUNT(branch, acc-no, cust-id, balance)
  - CUSTOMER(cust-id, name, address, phone, email)
  - ....
- UNIVERSITY relational database schema with relation schemas
  - STUDENT(student-id, student-name, major, status)
  - FACULTY(faculty-id, faculty-name, dpt, title, salary)
  - COURSE(course-no, course-name, term, instructor)
  - ENROLLS(student-id, course-no, term)
  - ...

Note: In general, a relational schema may have infinitely many different relational database instances.

Schemas vs. Instances

Keep in mind that there is a clear distinction between

- relation schemas and instances of relation schemas

and

- relational database schemas and relational database instances.

Syntactic Notion	**Semantic Notion (discrete mathematics notion)**
Relation Schema	Instance of a relation schema (i.e., a relation)
Relational Database Schema	Relational database instance (i.e., a database)

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.

Desiderata for a Database Query Language

Desiderata:

I. The language should be sufficiently high-level to secure physical data independence, i.e., the separation between the physical level and the conceptual level of databases.

II. The language should have high enough expressive power to be able to pose useful and interesting queries against the database.

III. The language should be efficiently implementable to allow for the fast retrieval of information from the database.

Warning:

- There is a well-understood tension between desideratum II and desideratum III.
- Increase in expressive power comes at the expense of efficiency.

Outline

1. The notion of database
2. The relational model of data
3. The relational algebra
4. SQL

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

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

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

- Note:
  - In relational algebra, both arguments to 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 × 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:

  As stated earlier,

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

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

Algebraic Laws for the Basic Set-Theoretic Operation

- Union:
  - $R \cup R = R$ -- idempotence law
  - $R \cup S = S \cup R$ -- commutativity law, order is unimportant
  - $R \cup (S \cup T) = (R \cup S) \cup T$

    -- associativity law, can drop parentheses

- Difference:
  - $R - R = \emptyset$
  - In general, $R - S \neq S - R$
  - Associativity does not hold for the difference

- Cartesian Product:
  - In general, $R \times S \neq S \times R$
  - $R \times (S \times T) = (R \times S) \times T$
  - $R \times (S \cup T) = (R \times S) \cup (R \times T)$ (distributivity law)

Algebraic Laws

- Question:
  - Why are algebraic laws important?

- Answer:
  - Algebraic laws are important in query processing and optimization to transform a query to an equivalent one that may be less costly to evaluate
  - Applying correct algebraic laws ensures the correctness of the transformations.

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 some columns

- Projection is a family of unary operations of the form

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

- The intuitive description of the projection operation is as follows:
  - When 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 according to the order in the <attribute list>.
  - Any duplicate rows are also eliminated.

The Projection Operation

Show name and Site of employees

Employee

Name	Site
Neri	Napoli
Neri	Milano
Rossi	Roma

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

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:

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

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

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

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

$$\sigma_{\Theta}(R)$$

  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.

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

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

Algebraic Laws for the Selection Operation

- $\sigma_{\Theta_1}(\sigma_{\Theta_2}(R)) = \sigma_{\Theta_2}(\sigma_{\Theta_1}(R))$
- $\sigma_{\Theta_1}(\sigma_{\Theta_2}(R)) = \sigma_{\Theta_1 \wedge \Theta_2}(R)$
- $\sigma_{\Theta}(R \times S) = \sigma_{\Theta}(R) \times S$
  provided Θ mentions only attributes of R.

Note: These are very useful laws in query optimization.

Relational Algebra Expression

- Definition: A relational algebra expression is a string 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

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

- 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 ∩ 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

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

Note:

- If R and S have an attribute A in common, then we use the notation R.A and S.A to disambiguate.

- The Θ-Join selects those tuples from R × S that satisfy the condition Θ. In particular, if every tuple in R Θ 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) =

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

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 ∩ 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) =

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

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:

$$\text{ENROLLS} \bowtie \text{TEACHES}$$

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 $R \bowtie S$.

Natural Join

Some Algebraic Laws for Natural Join

- $R \bowtie S = S \bowtie R$ (up to rearranging the columns)
- $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$
- $(R \bowtie R) = R$
- If A is an attribute of R, but not of S, then
  $\sigma_{A = c} (R \bowtie S) = \sigma_{A = c} (R) \bowtie S$
- …

Fact: The most FAQs against databases involve the natural join operation $\bowtie$.

Outline

1. The notion of database
2. The relational model of data
3. The relational algebra
4. SQL

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

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

References

- Raghu Ramakrishnan, Johannes Gehrke, “Database Management Systems”, McGraw-Hill Science Engineering, 2002

  Deals with all aspects of database management (and design)

- Serge Abiteboul, Richard Hull, Victor Vianu, “Foundations of databases”, Addison-Wesley, 1995

  THE database theory book