Introduction to Databases

Maurizio Lenzerini

Sapienza
Università di Roma

PROMISE Winter School 2013
Bridging between Information Retrieval and Databases

Bressanone, Italy, 4–8 February 2013

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

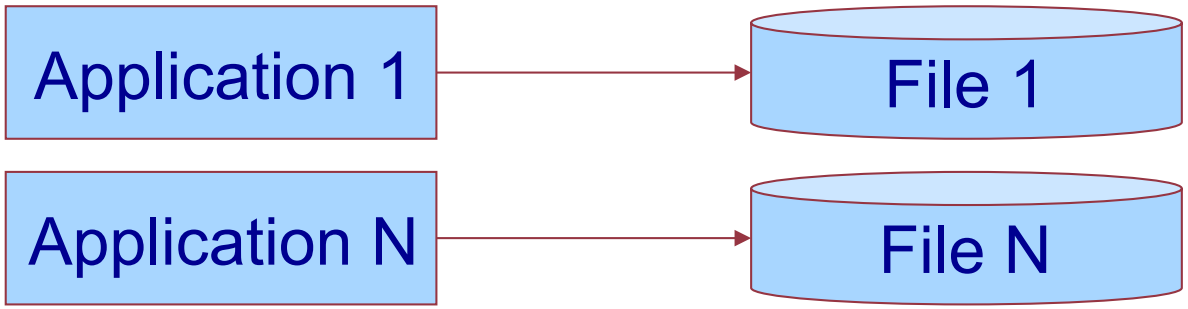
1. The notion of database
2. The relational model of data
3. The relational algebra
4. SQL
5. The relational calculus
6. Datalog
7. Conclusions

Outline

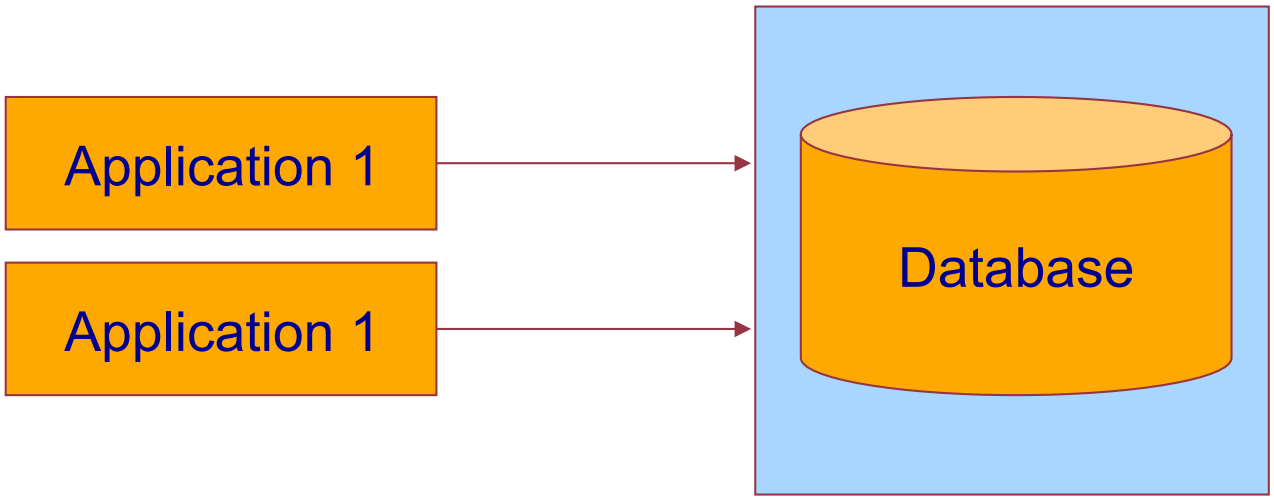
1. The notion of database
2. The relational model of data
3. The relational algebra
4. SQL
5. The relational calculus
6. Datalog
7. Conclusions

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.



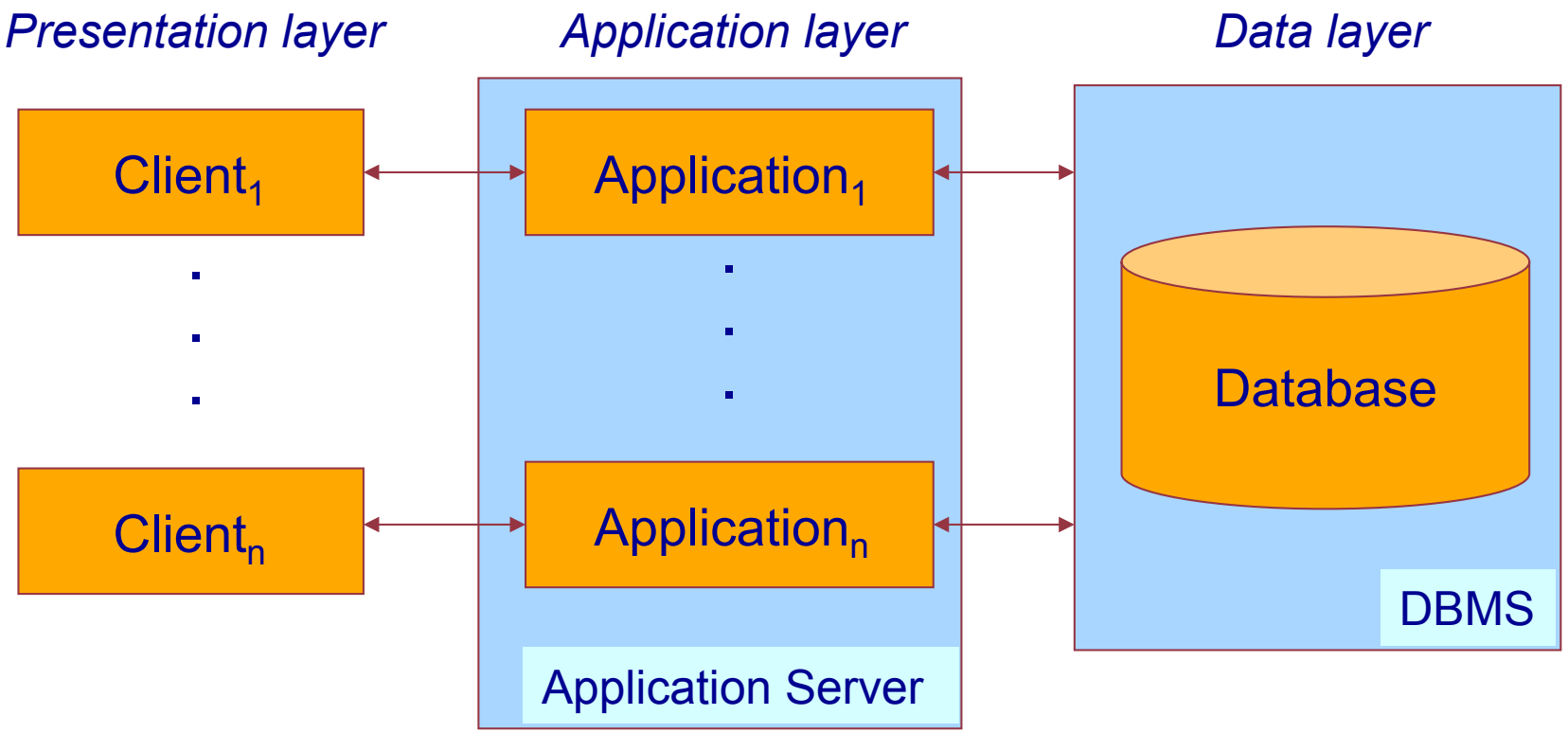
pre-database situation (60’s-70’s)



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

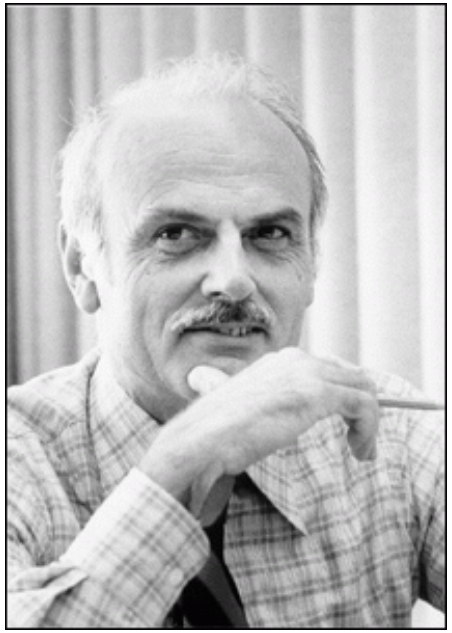
Relational Databases: A Very Brief History

- 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
5. The relational calculus
6. Datalog
7. Conclusions

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
5. The relational calculus
6. Datalog
7. Conclusions

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
    $R \cup S = \{(a_1,\ldots,a_k)$: $(a_1,\ldots,a_k)$ 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 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:
  - $R - R = \varnothing$
  - 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.

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

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

Algebraic Laws for the Selection Operation

- $\sigma_{\Theta_1}(\sigma_{\Theta_2}(R)) = \sigma_{\Theta_2}(\sigma_{\Theta_1}(R))$
- $\sigma_{\Theta_1}(\sigma_{\Theta_2}(R)) = \sigma_{\Theta_1 \wedge \Theta_2}(R)$
- $\sigma_{\Theta}(R \wedge S) = \sigma_{\Theta}(R) \wedge S$
  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) =

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

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×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 × 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 $\bowtie$.

Derived Operation: Quotient (Division)

- Motivating Example:

  Given ENROLLS(stud-name,course) and TEACHES(fac-name,course), find the names of students who take every course taught by V. Vianu.

- Other Motivating Examples:
  - Find the names of customers who have an account in every branch of Wachovia in San Jose.
  - Find the names of Netflix customers who have rented every film in which Paul Newman starred.

- These and other similar queries can be answered using the Quotient (Division) operation.

Quotient (Division)

- Definition: Let R be a relation of arity r and let S be a relation of arity s, where r > s.

  The quotient (or division) R ÷ S is the relation of arity r – s consisting of all tuples $(a_1,\ldots,a_{r-s})$ such that for every tuple $(b_1, \ldots,b_s)$ in S, we have that $(a_1,\ldots,a_{r-s}, b_1,\ldots,b_s)$ is in R.

- Example: Given

  ENROLLS(studname,course) and TEACHES(facname,course), find the names of students who take every course taught by V. Vianu.

  - Find the courses taught by V. Vianu

    $$\pi_{course}\ (\sigma_{\ facname\ =\ \text{“V. Vianu”}}\ (TEACHES))$$

  - The desired answer is given by the expression:

    $$ENROLLS\ \div\ \pi_{course}\ (\sigma_{\ facname\ =\ \text{“V. Vianu”}}\ (TEACHES))$$

Quotient (Division)

Fact: The quotient operation is expressible in relational algebra.

Proof: For concreteness, assume that R has arity 5 and S has arity 2.

Key Idea: to compute R÷S, use the difference operation

- $R \div S = \pi_{1,2,3}(R)$ – "tuples in $\pi_{1,2,3}(R)$ that do not make it to R÷S"

- What is the set of all tuples that fail the test for membership in R÷S? It is the projection on 1,2,3 on the set denoted by the relational algebra expression $(\pi_{1,2,3}(R) \times S) - R$.

  Hence,

$$R \div S = \pi_{1,2,3}(R) - \pi_{1,2,3}(\pi_{1,2,3}(R) \times S) - R).$$

Relational Completeness

- Definition (Codd – 1972): A database query language L is relationally complete if it is at least as expressive as relational algebra, i.e., every relational algebra expression E has an equivalent expression F in L.

- Relational completeness provides a benchmark for the expressive power of a database query language.

- Every commercial database query language should be at least as expressive as relational algebra.

Independence of the Basic Relational Algebra Operations

- Question: Are all five basic relational algebra operations really needed? Can one of them be expressed in terms of the other four?

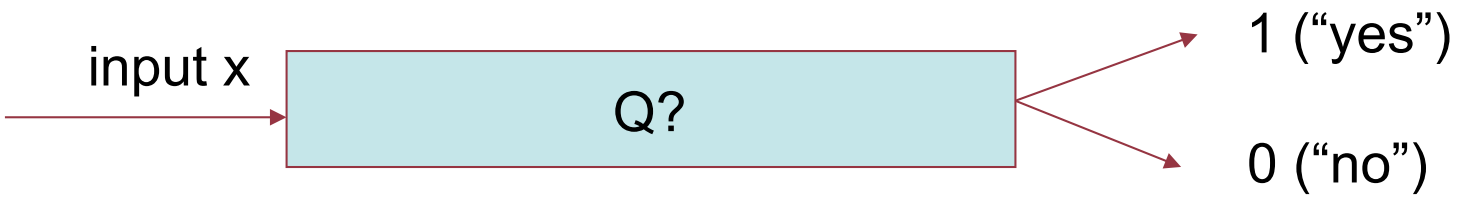
- Theorem: Each of the five basic relational algebra operations is independent of the other four, that is, it cannot be expressed by a relational algebra expression that involves only the other four.

  Proof Idea: For each relational algebra operation, we need to discover a property that is possessed by that operation, but is not possessed by any relational algebra expression that involves only the other four operations.

Computational Complexity of the Relational Algebra

- What is the computational complexity of the relational algebra?

- Definition (informal): A decision problem Q consists of a set of inputs and a question with a "yes" or "no" answer for each input.



- Definition:
  - Σ^* is the set of all strings over a finite alphabet Σ.
  - A language over Σ is a set $L \subseteq \Sigma^*$
  - Every language L gives rise to the following decision problem:
    - Given $x \in \Sigma^*$, is $x \in L$?
  - Conversely, every decision problem can be thought of as arising from a language, namely, the language consisting of all inputs with a "yes" answer.

The Five Basic Computational Complexity Classes

- LOGSPACE (or, L): All decision problems solvable by a TM using extra memory bounded by a logarithmic amount in the input size.
- NLOGSPACE (or, NL): All decision problems solvable by a NTM using extra memory bounded by a logarithmic amount in the input size.
- P (or, PTIME): All decision problems solvable by a TM in time bounded by some polynomial in the input size.
- NP: All decision problems solvable by a NTM in time bounded by some polynomial in the input size (deterministic time will be exponential).
- PSPACE: All decision problems solvable by a TM using memory bounded by a polynomial in the input size (time will be exponential)

The Five Basic Computational Complexity Classes

Theorem:

- The following inclusions hold:

  $$\text{LOGSPACE} \subseteq \text{NLOGSPACE} \subseteq \text{P} \subseteq \text{NP} \subseteq \text{PSPACE}.$$

- Moreover, it is known that $\text{LOGSPACE} \subset \text{PSPACE}$.

- **No** other proper inclusion between these classes is known at present. In particular, it is **not** known whether P = NP.

Note:

- The question: “is P = NP?” is the central open problem in computational complexity.
- It is one of the Millennium Prize Problems – see http://www.claymath.org/millennium/

Complexity of the Query Evaluation Problem

- The Query Evaluation Problem: Given a query q and a database I, find the answers q(I) to q wrt I. (As a decision problem: Given a query q, a database I, and a tuple **t**, decide whether **t** is in the answers q(I)).

- "Obvious" algorithm for evaluating a relational algebra expression E over a database I:
  - For each relational algebra operator, write a procedure for computing the result of an expression with only one application of such operator. Note that the time complexity of each procedure is at most quadratic wrt the size |I| of the database I
  - Evaluate the expression by calling the appropriate procedure. The time complexity is $O(|I|^{|E|})$, where |I| is the size of I and |E| is the size of the expression E.

- Theorem: The Query Evaluation Problem for Relational Algebra is PSPACE-complete.

The Query Evaluation Problem for Relational Algebra

- **Paradox:**
  - The Query Evaluation Problem for Relational Algebra has very high combined complexity (PSPACE-complete, so "harder" than NP-complete).
  - Yet, database systems evaluate SQL queries "efficiently".

Vardi's Taxonomy of the Query Evaluation Problem

Let L be a database query language

- The most general complexity measure consider the input to the problem to be both the query Q and the database I – combined complexity
- When we want to measure the complexity of L wrt to the size of the database I only, we consider the input of the problem to be only I, with the query Q fixed, and we measure the complexity of evaluating Q over I – data complexity
- When we want to measure the complexity of L wrt to the size of the query Q only, we consider the input of the problem to be only Q, with the database I fixed, and we measure the complexity of evaluating Q over I – query complexity

The Query Evaluation Problem for Relational Algebra

- **Paradox:**
  - The Query Evaluation Problem for Relational Algebra has very high combined complexity (PSPACE-complete, so "harder" than NP-complete).
  - Yet, database systems evaluate SQL queries "efficiently".

- **Resolution of the Paradox:**
  - As we said, the combined complexity of the query evaluation problem for relational algebra is $O(|I|^{|E|})$, and this tells that the source of exponentiality is the size of the query.
  - In practice, we deal with the data complexity of the query evaluation problem, because we typically have a small fixed collection of queries to answer (while of course the database instances vary and is much larger than the queries).
  - The data complexity of the query evaluation problem for Relational Algebra is in PTIME (actually, in LOGSPACE); so, in principle, it is a tractable problem.

Outline

1. The notion of database
2. The relational model of data
3. The relational algebra
4. SQL
5. The relational calculus
6. Datalog
7. Conclusions

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

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

Outline

1. The notion of database
2. The relational model of data
3. The relational algebra
4. SQL
5. The relational calculus
6. Datalog
7. Conclusions

Relational Calculus

- In addition to relational algebra, Codd introduced relational calculus.

- Relational calculus is a declarative database query language based on first-order logic.

- Codd's main technical result is that relational algebra and relational calculus have essentially the same expressive power.

Logic and Databases

- There is a strong connection between logic and relational databases
- In its simpler form, the connection establishes that there is a parallel between

Logic	Databases
Interpretation over a signature	Database over a database schema
Closed formula	Boolean Query
Evaluation (in terms of truth-value) of a closed formula in an interpretation	Boolean Query evaluation over the database
Formula with free variables	Non-boolean Query
Evaluation (in terms of an n-ary relation) of a formula with n free variables in an interpretation	Evaluation (in terms of an n-ary relation) of a n-ary query over a database

Relational Calculus (First-Order Logic for Databases)

- First-order variables: x, y, z, …, x_1, …,x_k,…
  - They range over values that may occur in tables.
- Relation symbols: R, S, T, … of specified arities (names of relations)
- Atomic (Basic) Formulas:
  - $R(x_1,\ldots,x_k)$, where R is a k-ary relation symbol
    (alternatively, $(x_1,\ldots,x_k) \in R$; the variables need not be distinct)
  - (x op y), where op is one of =, ≠, <, >, ≤, ≥
  - (x op c), where c is a constant and op is one of =, ≠, <, >, ≤, ≥.
- Relational Calculus Formulas:
  - Every atomic formula is a relational calculus formula.
  - If φ and ψ are relational calculus formulas, then so are:
    - $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $\neg \psi$, $(\varphi \rightarrow \psi)$ (propositional connectives)
    - $(\exists x\, \varphi)$ (existential quantification)
    - $(\forall x\, \varphi)$ (universal quantification).

Relational Calculus Expression

Definition:

- A relational calculus expression is an expression of the form

$$\{ (x_1,\ldots,x_k):\ \varphi(x_1,\ldots x_k) \},$$

  where $\varphi(x_1,\ldots,x_k)$ is a relational calculus formula with $x_1,\ldots,x_k$ as its free variables.
- When applied to a relational database I, this relational calculus expression returns the k-ary relation that consists of all k-tuples $(a_1,\ldots,a_k)$ that make the formula "true" on I.
- Thus, every relational calculus expression as above defines a k-ary query.

Example: The relational calculus expression

$$\{ (x,y):\ \exists z\, (E(x,z) \wedge E(z,y) \}$$

returns the set P of all pairs of nodes (a,b) that are connected via a path of length 2 (ONE-STOP query, if E stands for FLIGHTS).

Algebra Operators in Relational Calculus

Let R(A,B) and S(C,D) be two relational schema

- $R \cup S$ can be expressed by $\{ (x,y): R(x,y) \vee S(x,y) \}$
- $R - S$ can be expressed by $\{ (x,y): R(x,y) \wedge \neg S(x,y) \}$
- $R \times S$ can be expressed by $\{ (x,y,w,z): R(x,y) \wedge S(w,z) \}$
- $\pi_A(R)$ can be expressed by $\{ (x): \exists y\, R(x,y) \}$
- $\sigma_\Theta(R)$ can be expressed by $\{ (x,y): R(x,y) \wedge \Theta \}$

Relational Calculus – example

Abbreviation:

- $\exists x_1,\ldots,x_k$ stands for $\exists x_1,\ldots,\exists x_k$
- $\forall x_1,\ldots,x_k$ stands for $\forall x_1,\ldots,\forall x_k$

Example: Given relation FACULTY(name, dpt, salary), find the names of the highest paid faculty in the CS department

$$\{ (x): \exists y,z\ (FACULTY(x,y,z) \wedge y = \text{“CS”} \wedge (\forall u,v,w(FACULTY(u,v,w) \wedge v = \text{“CS”} \rightarrow z \geq w))) \}$$

Exercise: Express this query in relational algebra and in SQL.

Natural Join in Relational Calculus

Example: Let R(A,B,C) and S(B,C,D) be two ternary relation schemas.

- Recall that, in relational algebra, the natural join $R \bowtie S$ is given by

$$\pi_{R.A,R.B,R.C,S.D} (\sigma_{R.B = S.B \wedge R.C = S.C} (R \times S))$$

- Here is a relational calculus expression for $R \bowtie S$:

$$\{ (x_1,x_2,x_3,x_4): R(x_1,x_2,x_3) \wedge S(x_2,x_3,x_4) \}$$

Note: The natural join is expressible by a quantifier-free formula of relational calculus.

Quotient in Relational Calculus

- Recall that the quotient (or division) R ÷ S of two relations R and S is the relation of arity r – s consisting of all tuples (a_1, …,a_{r-s}) such that for every tuple (b_1,…,b_s) in S, we have that (a_1, …,a_{r-s}, b_1,…,b_s) is in R.

- Assume that R has arity 5 and S has arity 2. Here is R ÷ S in relational calculus (3-ary query):

$$\{ (x_1,x_2,x_3): (\forall x_4)(\forall x_5) (S(x_4,x_5) \rightarrow R(x_1,x_2,x_3,x_4,x_5)) \}$$

  - Much simpler than the relational algebra expression for R ÷ S

Relational Calculus and SQL

- Relational calculus has influenced the design of SQL.
- In particular, existential and universal quantification may occur in the allowable conditions in the WHERE clause of the SELECT … FROM … WHERE construct.
- In addition, sets (or, multisets) are allowed as operands in the WHERE clause

Sets as Operands in SQL

- Sets are allowed as operands in the WHERE clause. Sets are defined
  - by listing their elements, or
  - as the result of a SELECT … FROM … WHERE construct nested inside the WHERE clause of an outer SELECT … FROM .. WHERE

- This is what makes SQL a “structured” language, i.e., we have queries inside queries (subqueries) up to any finite depth of nesting.

- When sets are used as operands in a comparison clause:
  - We must use one of the keywords IN, NOT IN, SOME, ALL.
  - SOME and ALL must be preceded by one of the of comparison operators =, ≠, ≥, ≤, >, <.
  - The use of SOME and ALL is the first form of existential and universal quantification in SQL.

Sets as Operands in SQL: IN and NOT IN

STUDENT(name, major, college)

- Find all CS majors in Crown College or College Eight

  SELECT name
  FROM STUDENT
  WHERE major = ‘CS’ AND college IN (‘Crown’, ‘Eight’)

- Find all CS majors in colleges other than Crown and Eight

  SELECT name
  FROM STUDENT
  WHERE major = ‘CS’ AND college NOT IN (‘Crown’, ‘Eight’)

Sets as Operands in SQL: IN and NOT IN

Example: FACULTY(name,dpt,salary)

- Find the names of faculty who are in a department in which at least one member earns more than $175,000.

```
SELECT F1.name
FROM FACULTY AS F1
WHERE F1.dpt  IN  (SELECT F2.dpt
                   FROM  FACULTY AS F2
                   WHERE F2.salary > 175,000)
```

  that can also be written (since names are implicitly qualified by the table referenced in the FROM list at the same level of nesting):

```
SELECT name
FROM FACULTY
WHERE dpt  IN  (SELECT dpt
                FROM  FACULTY
                WHERE salary > 175,000)
```

Exercise: Express this query without using an SQL subquery.

Sets as Operands in SQL: IN and NOT IN

Example: FACULTY(name,dpt,salary)

- Find the names of faculty who are in a department in which no member earns more than $175,000.

```
SELECT name
FROM FACULTY
WHERE dpt NOT IN (SELECT dpt
                  FROM  FACULTY
                  WHERE salary > 175,000)
```

Exercise: Express this query without using an SQL subquery.

Sets as Operands in SQL: IN and NOT IN

Example: FACULTY(name,dpt,salary), CHAIR(name,dpt)

- Find the names of faculty who are in a department in which the chair earns more than $200,000.

```
SELECT name
FROM FACULTY
WHERE dpt  IN  (SELECT CHAIR.dpt
                FROM  FACULTY, CHAIR
                WHERE FACULTY.name = CHAIR.name
                      AND FACULTY.salary > 200,000)
```

Note: As we said before, attribute names are implicitly qualified by the table referenced in the FROM list at the same level of nesting. This can be overridden via aliasing.

SOME and ALL in SQL

- Syntax: In the WHERE clause, we can have have subclauses of the form
  - <attribute name> op SOME T
  - <attribute name> op ALL T, where
    - op is one of the comparison operators =, <>, ≥, ≤, >, <
    - T is the result of a nested SELECT … FROM … WHERE clause.
- Semantics:
  - <attribute name> op SOME T means:

    $(\exists x)(x \in T \wedge$ <attribute name> op $x)$
  - <attribute name> op ALL T means:

    $(\forall x)(x \in T \rightarrow$ <attribute name> op $x)$.
- Note:
  - <attribute name> = SOME T is the same as IN T
  - <attribute name> ≠ ALL T is the same as NOT IN T

SOME and ALL in SQL

Example: FACULTY(name,dpt,salary)

- Find the highest paid faculty in CS

```
SELECT name
FROM FACULTY
WHERE dpt = "CS" AND salary ≥ ALL (SELECT salary
                                   FROM FACULTY
                                   WHERE dpt = "CS").
```

Question: What is the result of the following SQL query?

```
SELECT name
FROM FACULTY
WHERE dpt = "CS" AND salary > ALL (SELECT salary
                                   FROM FACULTY
                                   WHERE dpt = "CS").
```

SOME and ALL in SQL

Example: FACULTY(name,dpt,salary)

- Find the highest paid faculty in CS

```
SELECT name
FROM FACULTY
WHERE dpt = "CS" AND salary ≥ ALL (SELECT salary
                                   FROM FACULTY
                                   WHERE dpt = "CS").
```

Question: What is the result of the following SQL query?

```
SELECT name
FROM FACULTY
WHERE dpt = "CS" AND salary > ALL (SELECT salary
                                   FROM FACULTY
                                   WHERE dpt = "CS").
```

Answer: The query returns the empty set.

SOME and ALL in SQL

Question: What are the results of the following two SQL queries?

- SELECT name
  FROM FACULTY
  WHERE dpt = “CS” AND salary > SOME (SELECT salary
  FROM FACULTY
  WHERE dpt = “CS”).
- SELECT name
  FROM FACULTY
  WHERE dpt = “CS” AND salary ≥ SOME (SELECT salary
  FROM FACULTY
  WHERE dpt = “CS”).

- Answer:
  - The first returns all CS faculty who are not the lowest paid ones.
  - The second returns all CS faculty.

SOME and ALL in SQL

- Find the names of the lowest paid faculty in the university

```
SELECT name
FROM FACULTY
WHERE salary <= ALL (SELECT salary
                     FROM FACULTY).
```

- Find the names of all CS faculty who earn less than some Philosophy faculty

```
SELECT name
FROM FACULTY
WHERE dpt = 'CS' AND salary <
                     SOME (SELECT salary
                           FROM FACULTY
                           WHERE dpt = 'Philosophy').
```

EXISTS and NOT EXISTS in SQL

- Syntax:
  - SELECT … FROM … WHERE EXISTS (SELECT … FROM … WHERE)
- Semantics: The subquery (SELECT … FROM … WHERE) is evaluated and the resulting set is tested for emptiness:
  - If it is non-empty, then the condition in WHERE evaluates to “true”; otherwise, it evaluates to “false”.
- Syntax:
  - SELECT … FROM … WHERE NOT EXISTS (SELECT … FROM … WHERE)
- Semantics: The subquery (SELECT … FROM … WHERE) is evaluated and the resulting set is tested for emptiness:
  - If it is empty, then the condition in WHERE evaluates to “true”; otherwise, it evaluates to “false”.

EXISTS and NOT EXISTS in SQL

- Example: FACULTY(name,dpt,salary)

  Find the faculty in the CS dpt who are not the lowest paid ones.

```
SELECT R.name
FROM FACULTY as R
WHERE R.dpt = `CS' AND
                    EXISTS (SELECT *
                            FROM FACULTY AS T
                            WHERE T.dpt = `CS' AND
                                  R.salary > T.salary)
```

Note: This is an example of a correlated subquery:

- The subquery has to be evaluated separately for each tuple in the FROM list of the outer query.
- The tuple is kept or removed depending on the result of the EXISTS test.

EXISTS and NOT EXISTS in SQL

ENROLS(student,course), TEACHES(instructor,course)

- Find all students who take a course taught by Mateas, but no course taught by Mackey

```
SELECT R.student
FROM ENROLS AS R, TEACHES AS S
WHERE R.course = S.course AND S.instructor= 'Mateas' AND
        NOT EXISTS (SELECT *
                    FROM ENROLS AS T, TEACHES AS W
                    WHERE R.student = T.student AND
                          T.course =  W.course AND
                          W.instructor = 'Mackey')
```

EXISTS and NOT EXISTS in SQL

ENROLS(student,course), TEACHES(instructor,course)

- Find the names of all students who are enrolled in every course taught by Mackey.

```
SELECT R.student
FROM ENROLS AS R
WHERE there is no course taught by Mackey and not taken by  R.student

SELECT R.student
FROM ENROLS AS R
WHERE NOT EXISTS (SELECT S.course
                  FROM TEACHES AS S
                  WHERE S.instructor = 'Mackey' AND
                      S.course NOT IN (SELECT T.course
                                       FROM ENROLS AS T
                                       WHERE R.student = T.student))
```

Outline

1. The notion of database
2. The relational model of data
3. The relational algebra
4. SQL
5. The relational calculus
6. Datalog
7. Conclusions

Limitations of Relational Algebra & Relational Calculus

Outline:

- Relational Algebra and Relational Calculus have substantial expressive power. In particular, they **can** express
  - Natural Join
  - μ-join, for various conditions μ
  - Quotient
  - …
- However, they **cannot** express recursive queries.
- Datalog is a declarative database query language that augments relational algebra/calculus with a recursion mechanism.
  - Datalog = “Conjunctive Queries + Recursion”

Conjunctive Queries

- Definition: A conjunctive query is a query expressible by a relational calculus formula built from atomic formulas, $\wedge$, and $\exists$ only, i.e., it is an expression of the form

$$\{(x_1,\ldots,x_k): \exists z_1 \ldots \exists z_m \chi(x_1, \ldots,x_k, z_1,\ldots,z_k)\},$$

where $\chi(x_1, \ldots,x_k, z_1,\ldots,z_k)$ is a conjunction of atomic formulas, each of the form $R(y_1,\ldots,y_m)$.

- A conjunctive query can be written as a logic-programming rule

$$Q(x_1,\ldots,x_k) :- R_1(\mathbf{u}_1), \ldots, R_n(\mathbf{u}_n)$$

where

  - Each variable x_i occurs in the right-hand side of the rule.
  - Each $\mathbf{u}_i$ is a tuple of variables (not necessarily distinct)
  - The variables occurring in the right-hand side (the body), but not in the left-hand side (the head) of the rule are existentially quantified (but the quantifiers are not displayed).
  - "," stands for conjunction.

More Examples:

– Path of Length 2: (Binary query)

$$\{ (x,y): \exists z\, (E(x,z) \wedge E(z,y)) \}$$

- As a relational algebra expression,

$$\pi_{1,4}\, (\sigma_{\$2=\$3}(E \times E)$$

- As a rule:

$$q(x,y) :- E(x,z), E(z,y)$$

– Node on a Cycle of Length 3: (Unary query)

$$\{x: \exists y \exists z\, (E(x,y) \wedge E(y,z) \wedge E(z,x))$$

- As a rule:

$$Q(x) :- E(x,z), E(z,y), E(z,x)$$

Parents, Grandparents, and Greatgrandparents

- Let PARENT be a binary relational schema such that if (a,b) $\in$ PARENT in some database instance, then a is a parent of b.

- Using PARENT, we can define GRANDPARENT and GREATGRANPARENT as follows:

$$\text{GRANDPARENT}(x,y) :- \text{PARENT}(x,z), \text{PARENT}(z,y)$$

$$\text{GREATGRANDPARENT}(x,y) :- \text{PARENT}(x,z), \text{PARENT}(z,w), \text{PARENT}(w,y)$$

- Similarly, we can define GREATGREATGRANPARENT using a conjunctive query, and so on up to any fixed level of ancestry.

Parents and Ancestors

- Question: Is there a relational algebra (relational calculus) expression that defines ANCESTOR from PARENT?

- Note: This type of question occurs in other related concepts:
  - Given a binary relation MANAGES(manager, employee), is there a relational algebra (relational calculus) expression that defines HIGHER-MANAGER
  - Given a binary relation DIRECT(from,to) about flights, is there a relational algebra (relational calculus) expression that defines CAN-FLY(from,to)?
  - More abstractly, given a binary relation E, is there a relational algebra (relational calculus) expression that defines the Transitive Closure TC of E?

Edges and Paths

Definition: Let E be a binary relation

- For every n ≥ 1, let $PATH_n$ be the binary query:

  "given a and b, is there a path of length n from a to b along edges from E?"

- PATH is the binary query:

  "given a and b, is there a path from a to b along edges from E?"

Fact:

- For every n ≥ 1, the query $PATH_n$ is expressible by a conjunctive query. (Why?)
- Hence, PATH is expressible by an infinite union of conjunctive queries:

$$PATH \doteq PATH_1 \cup PATH_2 \cup \ldots \cup PATH_n \cup \ldots$$

Edges, Paths, and Transitive Closure

Facts: Let E be a binary relation

- PATH is the Transitive Closure of E, i.e., the smallest binary relation T such that
  - $E \subseteq T$
  - T is transitive (if $(a,b) \in T$ and $(b,c) \in T$, then $(a,c) \in T$).

- There are several well-known efficient algorithms for computing the Transitive Closure of a given binary relation E
  - Floyd–Warshall Algorithm

- ANCESTOR, HIGHER-MANAGER, CAN-FLY are all different instantiations of PATH.

Transitive Closure and Relational Calculus

- Question: Is there a relational algebra (relational calculus) expression that defines ANCESTOR from PARENT? In other words, is there a relational algebra (relational calculus) expression that defines the Transitive Closure of a given binary relation E?

- Theorem: A. Aho and J. Ullman – 1979

  There is **no** relational algebra (or relational calculus) expression that defines the Transitive Closure of a given binary relation E.

  Note:

  The proof of this result requires methods from mathematical logic.

Overcoming the Limitations of Relational Calculus

- Question: What is to be done to overcome the limitations of the expressive power of relational calculus?

- Answer 1: Embedded Relational Calculus (Embedded SQL):
  - Allow SQL commands inside a conventional programming language, such as C, Java, etc.
  - This is an inferior solution, as it destroys the high-level character of SQL.

- Answer 2:
  - Augment relational calculus with a high-level declarative mechanism for recursion.
  - Conceptually, this a superior solution as it maintains the high-level declarative character of relational calculus.

Datalog

- Datalog = “Conjunctive Queries + Recursion”

- Datalog was introduced by Chandra and Harel in 1982 and has been studied by the research community in depth since that time:
  - Hundreds of research papers in major database conferences
  - Numerous doctoral dissertations
  - Recent applications, even outside databases, such as:
    - Specification of network properties (Network Datalog)
    - Access control languages
    - Static program analysis (trace recursive calls)

- SQL:1999 and subsequent versions of the SQL standard provide support for a sublanguage of Datalog, called linear Datalog.

Datalog Syntax

- Definition: A Datalog program *p* is a finite set of rules each expressing a conjunctive query

$$T(x_1,\ldots,x_k) \text{ :- } R_1(\mathbf{u}_1), \ldots, R_n(\mathbf{u}_n),$$

where each variable x_i occurs in the body of the rule (this way, every rule is safe).

- Some relational symbols occurring in the heads of the rules may also occur in the bodies of the rules of *p*

  (unlike the rules for conjunctive queries).
  - These relational symbols are the recursive relational symbols; they are also known as intensional database predicates (IDBs).

- The remaining relational symbols in the rules are known as the extensional database predicates (EDBs).

- Note: The data complexity of Datalog is in PTIME.

Datalog

- Example: Datalog program for Transitive Closure

$$T(x,y) :- E(x,y)$$

$$T(x,y) :- E(x,z), T(z,y)$$

  - E is the EDB predicate and T is the IDB predicate
  - The intuition is that the Datalog program gives a recursive specification of the IDB predicate T in terms of the EDB E.

- Example: Another Datalog program for Transitive Closure

$$T(x,y) :- E(x,y)$$

$$T(x,y) :- T(x,z), T(z,y)$$

("divide and conquer" algorithm for Transitive Closure)

Datalog and SQL

- SQL:99 and subsequent versions of the SQL standard provide support for linear Datalog programs (but **not** for non-linear ones)

- Syntax:

  WITH RECURSIVE R, S, T, … AS

  <Datalog program for R, S, T, …>

  <query involving R, S, T, … >

- Semantics:
  - Compute R, S, T, … as the semantics of

    <Datalog program for R, S, T, …>
  - The result of the previous step are temporary relation that are then used, together with other EDBS, as if they were stored relation (EDBs) in <query involving R, S, T, …>.

Datalog and SQL

Example: Give an SQL query that computes all descendants of Noah

```
WITH RECURSIVE ANCESTOR(anc,desc)
(SELECT parent, child
 FROM PARENT
 UNION
 SELECT ANCESTOR.anc, PARENT.child
 FROM   PARENT, ANCESTOR
 WHERE PARENT.child = ANCESTOR.anc
)
SELECT desc
FROM ANCESTOR
WHERE anc = 'Noah'
```

Outline

1. The notion of database
2. The relational model of data
3. The relational algebra
4. SQL
5. The relational calculus
6. Datalog
7. Conclusions

What we did address

- What is a database
- Organizing data according to the relational model
- Extracting data from relational databases through queries
- Relationship between logic and databases ("preciseness" is the basis for databases)

What we did not address

- Database design
- Transaction management
- Concurrency control
- Recovery
- Physical database organization
- Query processing and optimization
- Security and privacy
- Database programming
- Distributed and parallel databases
- Data integration and exchange
- Database management in business intelligence (warehousing, mining, ecc.)
- Other data models (XML, graph-based, …)
- Management of special data (temporal, spatial, scientific, …)
- New paradigms (probabilistic data, streaming data, …)
- New architectures for data management (NO-SQL, …)
- ……..

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