Database Systems
Spring 2013

The Relational Model
SL02

- The Relational Model
- Basic Relational Algebra Operators
- Additional Relational Algebra Operators
- Extended Relational Algebra Operators
- Modification of the Database
- Relational Calculus

Literature and Acknowledgments

Reading List for SL02:

- Database Systems, Chapters 3 and 6, Sixth Edition, Ramez Elmasri and Shamkant B. Navathe, Pearson Education, 2010.

These slides were developed by:

- Michael Böhlen, University of Zürich, Switzerland
- Johann Gamper, Free University of Bozen-Bolzano, Italy

The slides are based on the following text books and associated material:

- Fundamentals of Database Systems, Fourth Edition, Ramez Elmasri and Shamkant B. Navathe, Pearson Addison Wesley, 2004.
- A. Silberschatz, H. Korth, and S. Sudarshan: Database System Concepts, McGraw Hill, 2006.

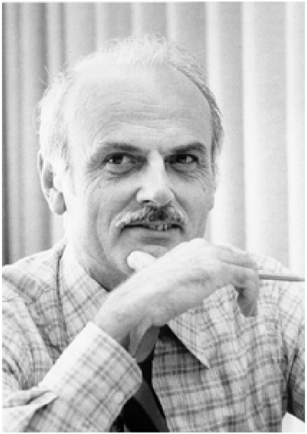
The Relational Model

- schema, attribute, domain, tuple, relation, database
- superkey, candidate key, primary key
- entity constraints, referential integrity

The Relational Model/1

- The **relational model** is based on the concept of a relation.
- A relation is a mathematical concept based on the ideas of sets.
- The relational model was proposed by Codd from IBM Research in the paper:
  - *A Relational Model for Large Shared Data Banks*, Communications of the ACM, June 1970
- The above paper caused a major revolution in the field of database management and earned Codd the coveted ACM Turing Award.
- The strength of the relational approach comes from the formal foundation provided by the theory of relations.
- In practice, there is a standard model based on SQL. There are several important differences between the formal model and the practical model, as we shall see.

The Relational Model/2



- Edgar Codd, a mathematician and IBM Fellow, is best known for creating the relational model for representing data that led to today's 12 billion database industry.
- Codd's basic idea was that relationships between data items should be based on the item's values, and not on separately specified linking or nesting.
- The idea of relying only on value-based relationships was quite a radical concept at that time, and many people were skeptical. They didn't believe that machine-made relational queries would be able to perform as well as hand-tuned programs written by expert human navigators.

`http://www.research.ibm.com/resources/news/20030423_edgarpassaway.shtml`

Relation Schema

- $R(A_1, A_2, \ldots, A_n)$ is a **relation schema**
- R is the **name** of the relation.
- $A_1, A_2, \ldots, A_n$ are **attributes**
- Example:
  $Customer(CustName, CustStreet, CustCity)$

Attribute

- Each attribute of a relation has a name
- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are required to be **atomic**; that is, indivisible
  - The value of an attribute can be an account number, but cannot be a set of account numbers
- The attribute name designates the role played by a domain in a relation:
  - Used to interpret the meaning of the data elements corresponding to that attribute
  - Example: The domain Date may be used to define two attributes named "Invoice-date" and "Payment-date" with different meanings

Domain

- A domain has a logical definition:
  - Example: USA_phone_numbers are the set of 10 digit phone numbers valid in the U.S.
- A domain also has a data-type or a format defined for it.
  - The USA_phone_numbers may have a format: (ddd)ddd-dddd where each d is a decimal digit.
  - Dates have various formats such as year, month, date formatted as yyyy-mm-dd, or as dd mm,yyyy etc.
- The special value *null* is a member of every domain
- The null value causes complications in the definition of many operations
  - We ignore the effect of null values in our main presentation and consider their effect later

Tuple

- A tuple is an *ordered set* (= list) of values
- Angle brackets $\langle \ldots \rangle$ are used as notation; sometimes regular parentheses (...) are used as well
- Each value is derived from an appropriate domain
- A customer tuple is a 3-tuple and would consist of three values, for example:
  - (Adams, Spring, Pittsfield)

Relational Instance

- $r(R)$ denotes a *relation* (or relation instance) r on *relation schema* R
- Example: $customer(Customer)$
- A relation instance is a subset of the Cartesian product of the domains of its attributes. Thus, a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$
- Formally, given sets $D_1, D_2, \ldots, D_n$ a **relation** r is a subset of $D_1 \times D_2 \times \ldots \times D_n$
- Example:

$$\begin{aligned}
D_1 &= CustName = \{Jones, Smith, Curry, Lindsay, \ldots\} \\
D_2 &= CustStreet = \{Main, North, Park, \ldots\} \\
D_3 &= CustCity = \{Harrison, Rye, Pittsfield, \ldots\} \\
r &= \{\ (Jones, Main, Harrison), (Smith, North, Rye), \\
&\qquad (Curry, North, Rye), (Lindsay, Park, Pittsfield)\ \} \\
&\subseteq CustName \times CustStreet \times CustCity
\end{aligned}$$

Example of a Relation

relation name

attributes

account

AccNr	BranchName	Balance
A-101	Downtown	500
A-215	Mianus	700
A-102	Perryridge	400
A-305	Round Hill	350
A-201	Brighton	900
A-222	Redwood	700
A-217	Brighton	750

tuples

The Customer Relation

customer

CustName	CustStreet	CustCity
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sad Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

Characteristics of Relations

- Relations are unordered, i.e., the order of tuples is irrelevant (tuples may be stored and retrieved in an arbitrary order)
- The attributes in $R(A_1, ..., A_n)$ and the values in $t = \langle v_1, ..., v_n \rangle$ are ordered.
- There exist alternative definitions of a relation where attributes in a schema and values in a tuple are not ordered.

depositor

CustName	**AccNr**
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Review 2.1

1. Is $r = \{(Tom, 27, ZH), (Bob, 33, Rome, IT)\}$ a relation?

2. Determine the following objects for $r(X, Y) = \{(1, a), (2, b), (3, c)\}$:

   - the 2nd attribute of relation r?
   - the 3rd tuple of relation r?
   - the tuple in relation r with the smallest value for attribute X?

3. What is the difference between a set and a relation? Illustrate with an example.

Database

- A database consists of multiple relations
- Example: Information about an enterprise is broken up into parts, with each relation storing one part of the information
  - *account*: stores information about accounts
  - *customer*: stores information about customers
  - *depositor*: information about which customer owns which account
- Storing all information as a single relation such as
  - $bank(AccNr, Balance, CustName, \ldots)$

  results in
  - repetition of information: e.g.,if two customers own the same account
  - the need for null values: e.g., to represent a customer without an account

Summary of the Relational Data Model

- A **domain** D is a set of atomic data values.
  - phone numbers, names, grades, birthdates, departments
  - each domain includes the special value `null`
- With each domain a **data type** or format is specified.
  - 5 digit integers, yyyy-mm-dd, characters
- An **attribute** A_i describes the role of a domain in a relation schema.
  - PhoneNr, Age, DeptName
- A **relation schema** $R(A_1, ..., A_n)$ is made up of a relation name R and a list of attributes.
  - $employee(Name, Dept, Salary)$, $department(DName, Manager, Address)$
- A **tuple** t is an ordered list of values $t = (v_1, ..., v_n)$ with $v_i \in dom(A_i)$.
  - $t = (Tom, SE, 23K)$
- A **relation** $r \subseteq D_1 \times ... \times D_n$ over schema $R(A_1, ..., A_n)$ is a set of n-ary tuples.
  - $r = \{(Tom, SE, 23K), (Lene, DB, 33K)\} \subseteq Names \times Departments \times Integer$
- A **database** DB is a set of relations.
  - $DB = \{r, s\}$
  - $r = \{(Tom, SE, 23K), (Lene, DB, 33K)\}$
  - $s = \{(SE, Tom, Boston), (DB, Lena, Tucson)\}$

Review 2.2

1. Illustrate the following relations graphically:
   $r(X, Y) = \{(1, a), (2, b), (3, c)\}$
   $s(A, B, C) = \{(1, 2, 3)\}$

2. What kind of object is $X = \{\{(3)\}\}$ in the relational model?

3. Are DB1 and DB2 identical databases?
   $DB1 = \{\{(1, 5), (2, 3)\}, \{(4, 4)\}\}$
   $DB2 = \{\{(4, 4)\}, \{(2, 3), (1, 5)\}\}$

Constraints

- Constraints are conditions that must be satisfied by all valid relation instances
- There are four main types of constraints in the relational model:
  - Domain constraints: each value in a tuple must be from the domain of its attribute
  - Key constraints
  - Entity constraints
  - Referential integrity constraints

Key Constraints/1

- Let $K \subseteq R$
- K is a **superkey** of R if values for K are sufficient to identify a unique tuple of each possible relation r
  - By "possible" we mean a relation r that could e.g. exist in the enterprise we are modeling.
  - Example: $\{CustName, CustStreet\}$ and $\{CustName\}$ are both superkeys of *Customer*, if no two customers can possibly have the same name.
  - In real life, an attribute such as *CustID* would be used instead of *CustName* to uniquely identify customers, but we omit it to keep our examples small, and instead assume customer names are unique.

CuName	CuStreet
N. Jeff	Binzmühlestr
N. Jeff	Hochstr

CuName cannot be a key

ID	CuName	CuStreet
1	N. Jeff	Binzmühlestr
2	N. Jeff	Hochstr

ID can be a key

Key Constraints/2

- K is a **candidate key** if K is minimal
  Example: $\{CustName\}$ is a candidate key for *Customer*, since it is a superkey and no subset of it is a superkey.
- **Primary key:** a candidate key chosen as the principal means of identifying tuples within a relation
  - Should choose an attribute whose value never, or very rarely, changes.
  - E.g. email address is unique, but may change

Entity Constraints

- The entity constraint requires that the primary key attributes of each relation may not have null values.
- The reason is that primary keys are used to identify the individual tuples.
- If the primary key has several attributes none of these attribute values may be null.
- Other attributes of the relation may also disallow null values although they are not members of the primary key.

ID	Name	CuStreet
1	N. Jeff	Binzmühlestr
	T. Hurd	Hochstr

ID cannot be primary key

ID	Name	CuStreet
1	N. Jeff	Binzmühlestr
2	T. Hurd	Hochstr

ID can be primary key

Referential Integrity

- A relation schema may have an attribute that corresponds to the primary key of another relation. The attribute is called a **foreign key**.
  - E.g. *CustName* and *AccNr* attributes of *depositor* are foreign keys to *Customer* and *Account* respectively.
  - Only values occurring in the primary key attribute of the **referenced relation** (or null values) may occur in the foreign key attribute of the **referencing relation**.
- In a graphical representation of the schema a referential integrity constraint is often displayed as a directed arc from the foreign key attribute to the primary key attribute.

ID	CuName	CuStrNr
1	N. Jeff	2
2	N. Jeff	4

StreetNr	Street
2	Binzmühlestr
3	Hochstr

StreetNr 4 does not exist. CuStrNr = 4 is an invalid reference.

Review 2.3

1. Determine the candidate keys of relation R:

R

X	Y	Z
1	2	3
1	4	5
2	2	2

Review 2.3

2. Determine possible superkeys, candidate keys, primary keys and foreign keys for relations *R* and *S*:

R

A	B	C
a	d	e
b	d	c
c	e	e

S

D	E
d	a
e	a
a	a

possible superkeys:

possible candidate keys:

possible primary keys:

possible foreign keys:

Query Languages

- Language in which user requests information from the database.
- Categories of languages
  - Procedural: specifies **how** to do it; can be used for query optimization
  - Declarative: specifies **what** to do; not suitable for query optimization
- Pure languages:
  - Relational algebra (procedural)
  - Tuple relational calculus (declarative)
  - Domain relational calculus (declarative)
- Pure languages form underlying basis of query languages that people use (such as SQL).

The Basic Relational Algebra

- select σ
- project π
- union $\cup$
- set difference $-$
- Cartesian product $\times$
- rename ρ

Relational Algebra

- The relational algebra is a procedural language
- The relational algebra consists of six basic operators
  - select: σ
  - project: π
  - union: $\cup$
  - set difference: $-$
  - Cartesian product: $\times$
  - rename: ρ
- The operators take one or two relations as inputs and produce a new relation as a result.
- This property makes the algebra **closed** (i.e., all objects in the relational algebra are relations).

Select Operation

- **Notation**: $\sigma_p(r)$
- p is called the **selection predicate**
- **Definition**: $t \in \sigma_p(r) \Leftrightarrow t \in r \wedge p(t)$
- p is a condition in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
- Example: $\sigma_{BranchName=`Perryridge`}(account)$
- Example: $\sigma_{A=B \wedge D>5}(r)$

r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\sigma_{A=B \wedge D>5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Project Operation

- **Notation**: $\pi_{A_1,\ldots,A_k}(r)$
- The result is defined as the relation of k columns obtained by deleting the columns that are not listed
- **Definition**: $t \in \pi_{A_1,\ldots,A_k}(r) \Leftrightarrow \exists x(x \in r \wedge t = x[A_1, \ldots, A_k])$
- There are no duplicate rows in the result since relations are sets
- Example: $\pi_{AccNr,Balance}(account)$
- Example: $\pi_{A,C}(r)$

r

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

$\pi_{A,C}(r)$

A	C
α	1
β	1
β	2

Union Operation

- **Notation**: $r \cup s$
- **Definition**: $t \in (r \cup s) \Leftrightarrow t \in r \vee t \in s$
- For $r \cup s$ to be valid r and s must have the same schema (i.e., attributes).
- Example: $\pi_{CustName}(depositor) \cup \pi_{CustName}(borrower)$
- Example: $r \cup s$

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

$r \cup s$

A	B
α	1
α	2
β	1
β	3

Set Difference Operation

- **Notation**: $r - s$
- **Definition**: $t \in (r - s) \Leftrightarrow t \in r \wedge t \notin s$
- Set differences must be taken between (union) compatible relations.
  - r and s must have the same **arity**
  - attribute domains of r and s must be compatible
- Example: $r - s$

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

$r - s$

A	B
α	1
β	1

Cartesian Product Operation

- **Notation**: $r \times s$
- **Definition**: $t \in (r \times s) \Leftrightarrow x \in r \wedge y \in s \wedge t = x \circ y$
- We assume that the attribute names of r and s are disjoint. If the attribute names are not disjoint, then renaming must be used.
- Example: $r \times s$

r

A	B
α	1
β	2

s

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

$r \times s$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Rename Operation

- Allows us to name the results of relational algebra expressions by setting relation and attribute names.
- The rename operator is also used if there are name clashes.
- Various flavors:
  - $\rho_r(E)$ changes the relation name to r.
  - $\rho_{r(A_1,\dots,A_n)}(E)$ changes the relation name to r and the attribute names to $A_1, \dots, A_k$.
  - $\rho_{(A_1,\dots,A_n)}(E)$ changes attribute names to $A_1, \dots, A_k$.
- Example: $\rho_{s(X,Y,U,V)}(r)$

r

A	B	C	D
α	α	1	7
β	β	23	10

s

X	Y	U	V
α	α	1	7
β	β	23	10

Composition of Operations

- Since the relational algebra is closed, i.e., the result of a relational algebra operator is always a relation, it is possible to nest expressions.
- Example: $\sigma_{A=C}(r \times s)$

r

A	B
α	1
β	2

s

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

$r \times s$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

$\sigma_{A=C}(r \times s)$

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

Review 2.4

1. Identify and correct syntactic mistakes in the following relational algebra expressions. The schema of relation R is $R(A, B)$.

$\sigma_{R.A>5}(R)$

$\sigma_{A,B}(R)$

$R \times R$

Review 2.4

2. Identify and correct syntactic mistakes in the following relational algebra expressions. Relation *Pers* has schema $Pers(Name, Age, City)$.

$\sigma_{Name='Name'}(Pers)$

$\sigma_{City=Zuerich}(Pers)$

$\sigma_{Age>'20'}$

Banking Example

- *branch(BranchName, BranchCity, Assets)*
- *customer(CustName, CustStreet, CustCity)*
- *account(AccNr, BranchName, Balance)*
- *loan(LoanNr, BranchName, Amount)*
- *depositor(CustName, AccNr)*
- *borrower(CustName, LoanNr)*

Review 2.5

branch(BranchName, BranchCity, Assets)
customer(CustName, CustStreet, CustCity)
account(AccNr, BranchName, Balance)
loan(LoanNr, BranchName, Amount)
depositor(CustName, AccNr)
borrower(CustName, LoanNr)

- Find all loans of over $1200.
- Find the loan number for each loan that is greater than $1200.
- Find the names of all customers who have a loan, an account, or both, from the bank.

Review 2.6

branch(BranchName, BranchCity, Assets)
customer(CustName, CustStreet, CustCity)
account(AccNr, BranchName, Balance)
loan(LoanNr, BranchName, Amount)
depositor(CustName, AccNr)
borrower(CustName, LoanNr)

- Find the names of all customers who have a loan at the Perryridge branch.

- Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

Review 2.6

branch(BranchName, BranchCity, Assets)
customer(CustName, CustStreet, CustCity)
account(AccNr, BranchName, Balance)
loan(LoanNr, BranchName, Amount)
depositor(CustName, AccNr)
borrower(CustName, LoanNr)

- Give a different relational algebra expressions that determines the names of all customers who have a loan at the Perryridge branch. Compare it to the solution in Review 2.6.

Review 2.7

branch(BranchName, BranchCity, Assets)
customer(CustName, CustStreet, CustCity)
account(AccNr, BranchName, Balance)
loan(LoanNr, BranchName, Amount)
depositor(CustName, AccNr)
borrower(CustName, LoanNr)

- Determine the largest account balance.

Formal Definition of Relational Algebra Expressions

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation (e.g., $\{(1,2),(5,3)\}$)
- Let E_1 and E_2 be relational algebra expressions; the following are all relational algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p(E_1)$, p is a predicate on attributes in E_1
  - $\pi_s(E_1)$, s is a list consisting of some of the attributes in E_1
  - $\rho_x(E_1)$, x is the new name for the result of E_1

Review 2.8

Assume the following schemas:

$train(TrainNr, StartStat, EndStat)$
$link(FromStat, ToStat, TrainNr, Departure, Arrival)$

1. Sketch an instance of the database.

Review 2.8

2. Determine all direct connections (no change of train) from Zürich to Olten.

Additional Relational Algebra Operators

We define additional operations that do not add expressive power to the relational algebra, but that simplify common queries. Thus, these are redundant relational algebra operators.

- Set intersection $\cap$
- Join $\bowtie$
- Division $\div$
- Assignment $\leftarrow$

Set Intersection Operation

- **Notation**: $r \cap s$
- **Definition**: $t \in (r \cap s) \Leftrightarrow t \in r \wedge t \in s$
- Precondition: union compatible
  - r, s have the *same arity*
  - attributes of r and s are compatible
- Note: $r \cap s = r - (r - s)$
- Example: $r \cap s$

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

$r \cap s$

A	B
α	2

Theta Join

- **Notation**: $r \bowtie_\theta s$
- Let r and s be relations on schemas R and S, respectively. θ is a boolean condition on the attributes of r and s.
- $r \bowtie_\theta s$ is a relation on schema that includes all attributes from schema R and all attributes from schema S.
- Example:
  - $R(A, B, C, D)$ and $S(B, D, E)$
  - $r \bowtie_{B<X \land D=Y} \rho_{(X,Y,Z)}(s)$
  - Schema of result is (A, B, C, D, X, Y, Z)
  - Equivalent to: $\sigma_{B<X \land D=Y}(r \times \rho_{(X,Y,Z)}(s))$

r

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

s

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ϵ

$\sigma_{B<X \land D=Y}(r \times \rho_{(X,Y,Z)}(s))$

A	B	C	D	X	Y	Z
α	1	α	a	3	a	β
β	2	γ	a	3	a	β
α	1	γ	a	3	a	β
δ	2	β	b	3	b	ϵ

Natural Join

- **Notation**: $r \bowtie s$
- Let r and s be relations on schemas R and S, respectively.
- Attributes that occur in r and s must be identical.
- $r \bowtie s$ is a relation on a schema that includes all attributes from schema R and all attributes from schema S that do not occur in schema R.
- Example:
  - $r \bowtie s$ with $R(A, B, C, D)$ and $S(E, B, D)$
  - Schema of result is (A, B, C, D, E)
  - Equivalent to: $\pi_{A,B,C,D,E}(\sigma_{B=Y \wedge D=Z}(r \times \rho_{(E,Y,Z)}(s)))$

r

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

s

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ϵ

$r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Division Operation

- **Notation**: $r \div s$
- Suited for queries that include the phrase "for all".
- Let r and s be relations on schemas $R(A_1, \ldots, A_m, B_1, \ldots, B_n)$ and $S(B_1, \ldots, Bn)$, respectively
- The result of $r \div s$ is a relation with schema $R - S = (A_1, \ldots, A_m)$
- **Definition**: $t \in (r \div s) \Leftrightarrow t \in \pi_{R-S}(r) \wedge \forall u \in s(t \circ u \in r)$
- $t \circ u$ is the concatenation of tuples t and u

- R-S: all attributes of schema R that are not in schema S
- Example: $R = (A, B, C, D), S = (E, B, D), R - S = (A, C)$

Division Operation - Examples

- r

A	B
α	1
α	2
α	3
β	1
γ	1
ϵ	6
ϵ	1
β	2

s

B
1
2

$r \div s$

A
α
β

- r

A	B	C	D	E
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

s

D	E
a	1
b	1

$r \div s$

A	B	C
α	a	γ
γ	a	γ

Properties of the Division Operation

- Property
  - Let $q = r \div s$
  - Then q is the largest relation satisfying $q \times s \subseteq r$
- Definition in terms of the basic algebra operation
  Let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$

  $$r \div s = \pi_{R-S}(r) - \pi_{R-S}((\pi_{R-S}(r) \times s) - \pi_{R-S,S}(r))$$

  To see why
  - $\pi_{R-S,S}(r)$ simply reorders attributes of r
  - $\pi_{R-S}(\pi_{R-S}(r) \times s) - \pi_{R-S,S}(r))$ gives those tuples t in $\pi_{R-S}(r)$ such that for some tuple $u \in s, t \circ u \notin r$.

Assignment Operation

- The assignment operation ($\leftarrow$) provides a convenient way to express complex queries by breaking them up into smaller pieces.
  - Write query as a sequential program consisting of
    - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.
  - Assignment must always be made to a temporary relation variable.
- Example: Write $r \div s$ as

$$\begin{aligned} &temp1 \leftarrow \pi_{R-S}(r) \\ &temp2 \leftarrow \pi_{R-S}((temp1 \times s) - \pi_{R-S,S}(r)) \\ &result = temp1 - temp2 \end{aligned}$$

  - The result to the right of the $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$.

Bank Example Queries/1

branch(BranchName, BranchCity, Assets)
customer(CustName, CustStreet, CustCity)
account(AccNr, BranchName, Balance)
loan(LoanNr, BranchName, Amount)
depositor(CustName, AccNr)
borrower(CustName, LoanNr)

- Find all customers who have an account and a loan.

  $\pi_{CustName}(borrower) \cap \pi_{CustName}(depositor)$

- Find the name of all customers who have a loan at the bank and the loan amount

  $\pi_{CustName,Amount}(borrower \bowtie loan)$

Bank Example Queries/2

branch(BranchName, BranchCity, Assets)
customer(CustName, CustStreet, CustCity)
account(AccNr, BranchName, Balance)
loan(LoanNr, BranchName, Amount)
depositor(CustName, AccNr)
borrower(CustName, LoanNr)

- Find all customers who have an account from at least the "Downtown" and the "Uptown" branches.
  - Solution 1

$$\pi_{CustName}(\sigma_{BranchName='Downtown'}(depositor \bowtie account)) \cap$$
$$\pi_{CustName}(\sigma_{BranchName='Uptown'}(depositor \bowtie account))$$

  - Solution 2

$$r \leftarrow \pi_{CustName,BranchName}(depositor \bowtie account))$$
$$s \leftarrow \pi_{BranchName}(\sigma_{BranchName=`Downtown` \vee BranchName=`Uptown`})(account)$$
$$Res \leftarrow r \div s$$

Review 2.9

branch(BranchName, BranchCity, Assets)
customer(CustName, CustStreet, CustCity)
account(AccNr, BranchName, Balance)
loan(LoanNr, BranchName, Amount)
depositor(CustName, AccNr)
borrower(CustName, LoanNr)

- Find all customers who have an account at all branches located in Brooklyn city.

Extended Relational Algebra Operators

Extended relational algebra operators add expressive power to the basic relational algebra.

- Generalized Projection π
- Aggregate Functions ϑ
- Outer Join $⟕$, $⟖$, $⟗$

Generalized Projection

- Extends the projection operation by allowing arithmetic functions to be used in the projection list: $\pi_{F_1, F_2, \ldots, F_n}(E)$
- E is a relational algebra expression
- Each of $F_1, F_2, \ldots, F_n$ are arithmetic expressions involving constants and attributes in the schema of E.
- Example: Given relation $credit\_info(CustName, Limit, CredBal)$, find how much more each person can spend:

  $\pi_{CustName, Limit - CreditBal}(credit\_info)$

Aggregate Functions and Operations

- **Aggregation function** takes a collection of values and returns a single value as a result.

  avg: average value
  min: minimum value
  max: maximum value
  sum: sum of values
  count: number of values

- **Aggregation operation** in relational algebra

  $$_{G_1,G_2,\ldots,G_n}\vartheta_{F_1(A_1),F_2(A_2),\ldots,F_n(A_n)}(E)$$

  E is any relational-algebra expression

  - $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name

Aggregate Operation - Example

- Relation r, $Res \leftarrow \rho_{Res(SumC)}(\vartheta_{sum(C)}(r))$

r

A	B	C
α	α	7
α	β	7
β	β	3
β	β	10

Res

sumC
27

- Balance per branch:
  $Res \leftarrow \rho_{Res(BName,SumBal)}({}_{BranchName}\vartheta_{sum(Balance)}(account))$

account

BranchName	*AccNr*	*Balance*
Perryridge	A-102	400
Perryridge	A-201	900
Brighton	A-217	750
Brighton	A-215	750
Redwood	A-222	700

Res

BName	*SumBal*
Perryridge	1300
Brighton	1500
Redwood	700

Outer Join

- An extension of the join operation that avoids loss of information.
- Computes the join and then adds tuples from one relation that do not match tuples in the other relation to the result of the join.
- Uses *null* values:
  - *null* signifies that the value is unknown or does not exist
  - All comparisons involving *null* are (roughly speaking) **false** by definition.
    - We shall study precise meaning of comparisons with nulls later

Outer Join Example/1

- Example relations:

loan

LoanNr	*BranchName*	*Amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

borrower

CustName	*LoanNr*
Jones	L-170
Smith	L-230
Hayes	L-155

- Join

$loan \bowtie borrower$

LoanNr	*BranchName*	*Amount*	*CustName*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith

Outer Join Example/2

- Example relations:

loan

LoanNr	*BranchName*	*Amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

borrower

CustName	*LoanNr*
Jones	L-170
Smith	L-230
Hayes	L-155

- Left Outer Join (preserves tuples from left)

$loan ⟕ borrower$

LoanNr	*BranchName*	*Amount*	*CustName*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*

Outer Join Example/3

- Example relations:

loan

LoanNr	*BranchName*	*Amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

borrower

CustName	*LoanNr*
Jones	L-170
Smith	L-230
Hayes	L-155

- Right Outer Join (preserves tuples from right)

loan ⟖ *borrower*

LoanNr	*BranchName*	*Amount*	*CustName*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-155	*null*	*null*	Hayes

Outer Join Example/4

- Example relations:

loan

LoanNr	*BranchName*	*Amount*
L-170	Downtown	3000
L-230	Redwood	4000
L-260	Perryridge	1700

borrower

CustName	*LoanNr*
Jones	L-170
Smith	L-230
Hayes	L-155

- Full Outer Join (preserves all tuples)

loan ⟗ *borrower*

LoanNr	*BranchName*	*Amount*	*CustName*
L-170	Downtown	3000	Jones
L-230	Redwood	4000	Smith
L-260	Perryridge	1700	*null*
L-155	*null*	*null*	Hayes

Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating
- All these operations are expressed using the assignment operator.

Deletion

- A delete request is expressed similarly to a query, except instead of displaying tuples to the user, the selected tuples are removed from the database.
- Can delete only entire tuples; cannot delete values of particular attributes only.
- A deletion is expressed in relational algebra by:

  $r \leftarrow r - E$

  where r is a relation and E is a relational algebra query.

Deletion Examples

- Delete all account records in the Perryridge branch.

  $accout \leftarrow account - \sigma_{BranchName='Perryridge'}(account)$

- Delete all loan records with amount in the range of 10 to 50

  $loan \leftarrow loan - \sigma_{Amount \geq 10 \wedge Amount \leq 50}(loan)$

- Delete all accounts at branches located in Needham.

  $r_1 \leftarrow \sigma_{branch\_city='Needham'}(accout \bowtie branch)$

  $r_2 \leftarrow \pi_{AccNr,BranchName,Balance}(r_1)$

  $r_3 \leftarrow \pi_{CustName,AccNr}(r_2 \bowtie depositor)$

  $account \leftarrow account - r_2$

  $depositor \leftarrow depositor - r_3$

Insertion

- To insert data into a relation, we either:
  - specify a tuple to be inserted
  - write a query whose result is a set of tuples to be inserted
- In relational algebra, an insertion is expressed by:

  $$r \leftarrow r \cup E$$

  where r is a relation and E is a relational algebra expression.
- The insertion of a single tuple is expressed by letting E be a constant relation containing one tuple.

Insertion Examples

- Insert information into the database specifying that Smith has $1200 in account A-973 at the Perryridge branch.

  $$account \leftarrow account \cup \{(\text{`A-973`}, \text{`}Perryridge\text{`}, 1200)\}$$
  $$depositor \leftarrow depositor \cup \{(\text{`}Smith\text{`}, \text{`A-973`})\}$$

- Provide as a gift for all loan customers in the Perryridge branch, a $200 savings account. Let the loan number serve as the account number for the new savings account.

  $$r_1 \leftarrow \sigma_{BranchName='Perryridge'}(borrower \bowtie loan)$$
  $$account \leftarrow account \cup \pi_{LoanNr,BranchName,200}(r_1)$$
  $$depositor \leftarrow depositor \cup \pi_{CustName,LoanNr}(r_1)$$

Updating

- A mechanism to change a value in a tuple without changing *all* values in the tuple; logically this can be expressed by an insertion and deletion; in actual systems updating is much faster than inserting and deleting.
- In relational algebra this can be expressed by replacing r by the result computed by the relational algebra expression E; often the expression is the generalized projection.

$$r \leftarrow E$$

$$r \leftarrow \pi_{F_1,F_2 \ldots, F_i, \ldots}(r)$$

- Each F_i is either
  - the i^{th} attribute of r, if the i^{th} attribute is not updated, or,
  - if the attribute is to be updated F_i is an expression, which defines the new value for the attribute

Update Examples

- Make interest payments by increasing all balances by 5%.

  $account \leftarrow \pi_{AccNr, BranchName, Balance*1.05}(account)$

- Pay all accounts with balances over $10,000 6% interest and pay all others 5%.

  $account \leftarrow$

  $\pi_{AccNr, BranchName, Balance*1.06}(\sigma_{Balance>100000}(account))$

  $\cup$

  $\pi_{AccNr, BranchName, Balance*1.05}(\sigma_{Balance\leq 100000}(account))$

Relational Calculus

- First Order Predicate Logic
- Tuple Relational Calculus
- Domain Relational Calculus

Relational Calculus

- A **relational calculus** expression creates a new relation, which is specified in terms of variables that range over
  - tuples of the stored database relations (in **tuple calculus**)
  - attributes of the stored relations (in **domain calculus**).
- In a relational calculus expression, there is *no order of operations* to specify how to compute the query result.
- A calculus expression specifies only what information the result should contain; hence relational calculus is a **non-procedural** or **declarative** language.
- In relational algebra we must write a *sequence of operations* to specify a retrieval request; hence relational algebra is a **procedural** way of stating a query.
- Relational calculus is closely related to and a subset of first order predicate logic.

First Order Predicate Logic

Syntax:

- **logical symbols**: $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\exists$, $\forall$, ...
- **constant**: string, number, ...; 'abc', 14, ...
- **identifier**: character sequence starting with a letter
- **variable**: identifier starting with capital letter; X, Y, ...
- **predicate symbol**: identifier starting with lower case letter
- **build-in predicate symbol**: $=, <, >, \leq, \geq$, $\neq$, ...
- **term**: constant, variable
- **atom**: predicate, built-in predicate; $p(t_1, ..., t_n)$, $t_1 < t_2$, ...
  with terms $t_1, ..., t_n$; predicate symbol p
- **formula**: atom, $A \wedge B$, $A \vee B$, $\neg A$, $A \Rightarrow B$, $\exists XA$, $\forall XA$, (A), ...
  with formulas A, B; variable X

Review 2.10

Decide which of the following formulas are syntactically correct first order predicate logic formulas.

- $less\_than(99, 27)$
- $loves(mother('hans'), france \vee italy)$
- $\forall X(danish(X) \Rightarrow danish('bill\_clinton'))$
- $\forall P(P('hans'))$
- $\forall C(neighbour('england', C))$
- $\exists C(neighbour('italien', C))$
- $\forall P(smart(P) \wedge \neg alive(P) \Rightarrow famous(P))$

Selected Properties and Terminology

FOPL Equivalences

- $\forall X(A) = \neg\exists X(\neg A)$
- $A \Rightarrow B = \neg A \vee B$

Set theory

- $A - B = A - (A \cap B)$

Terminology

- A variable is free if it is not quantified
- A variable is bound if it is quantified

Domain Independence

- Relational calculus only permits expressions that are *domain independent*, i.e., expressions that permit sensible answers (e.g., no infinite results).
- Domain independence is not decidable. There exist various syntactic criteria that ensure domain independence, e.g., safe expressions, range restricted expressions, etc.
- Examples:
  - $emp(X)$ is domain independent
  - $\neg emp(X)$ is not domain independent
  - $stud(X) \land \neg emp(X)$ is domain independent
  - $X > 6$ is not domain independent

Review 2.11

Use first order predicate calculus expressions to express the following natural language statements:

- Anyone who is dedicated can learn databases.
- No man is independent.
- Dogs that bark do not bite.
- Not all men can walk.
- Every person owns a computer.
- Lars likes everyone who does not like himself.

Tuple Relational Calculus/1

Syntax:

- **logical symbols**: $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\exists$, $\forall$, ...
- **constant**: string, number, ...; 'abc', 14, ...
- **identifier**: character sequence starting with a letter
- **variable**: identifier starting with lower case letter; t, d, ...
- **predicate symbol**: identifier starting with lower case letter
- **build-in predicate symbol**: $=, <, >, \leq, \geq, \neq$, ...
- **term**: constant, attribute of a tuple; $t.Name$, ...
- **atom**: predicate, built-in predicate; $p(t)$, $t.Sal < 5000$, ...
- **formula**: atom, $A \wedge B$, $A \vee B$, $\neg A$, $A \Rightarrow B$, $\exists tA$, $\forall tA$, (A), ...

- A tuple relational calculus query is of the form

  $\{\ t_1.A_j, t_2.A_k, ..., t_n.A_m \mid formula\ \}$

Tuple Relational Calculus/2

- The tuple relational calculus is based on specifying a number of tuple variables.
- Each tuple variable usually ranges over a particular database relation, meaning that the variable may take as its value any individual tuple from that relation.
- For example, to indicate that tuple variable t ranges over all tuples of *emp* we write $emp(t)$.

- Example: To determine first and last names of all employees whose salary is above $50,000, we write the following TRC expression:

  $\{\ t.FName, t.LName \mid emp(t) \wedge t.Sal > 50000\ \}$

Tuple Relational Calculus/3

- Each free tuple variable is bound successively to each tuple of the relation it ranges over.
- Each combination of bound tuples variables that makes the formula true produces a result tuple according to the specification to the left of the bar $|$.

- Example: Determine last names and department of all employees:
  $\{\ t.LName, d.DName \mid emp(t) \land dept(d) \land t.DNo = d.DNo\ \}$

Review 2.12

branch(BranchName, BranchCity, Assets)
customer(CustName, CustStreet, CustCity)
account(AccNr, BranchName, Balance)
loan(LoanNr, BranchName, Amount)
depositor(CustName, AccNr)
borrower(CustName, LoanNr)

- Find all loans of over $1200.

- Find the loan number for each loan of an amount greater than $1200,

- Find the names of all customers who have a loan, an account, or both, from the bank.

Review 2.13

branch(BranchName, BranchCity, Assets)
customer(CustName, CustStreet, CustCity)
account(AccNr, BranchName, Balance)
loan(LoanNr, BranchName, Amount)
depositor(CustName, AccNr)
borrower(CustName, LoanNr)

- Find the names of all customers who have a loan at the Perryridge branch.

- Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

Review 2.14

branch(BranchName, BranchCity, Assets)
customer(CustName, CustStreet, CustCity)
account(AccNr, BranchName, Balance)
loan(LoanNr, BranchName, Amount)
depositor(CustName, AccNr)
borrower(CustName, LoanNr)

- Determine the largest account balance.

Domain Relational Calculus/1

Syntax:

- **logical symbols**: $\wedge$, $\vee$, $\neg$, $\Rightarrow$, $\exists$, $\forall$, ...
- **constant**: string, number, ...; 'abc', 14, ...
- **identifier**: character sequence starting with a letter
- **variable**: identifier starting with capital letter; X, Y, ...
- **predicate symbol**: identifier starting with lower case letter
- **build-in predicate symbol**: $=, <, >, \leq, \geq, \neq$, ...
- **term**: constant, variable
- **atom**: predicate, built-in predicate; $p(X, ..., 22)$, $X < 5000$, ...
- **formula**: atom, $A \wedge B$, $A \vee B$, $\neg A$, $A \Rightarrow B$, $\exists X(A)$, $\forall X(A)$, (A), ...

- A domain relational calculus query is of the form

  $\{ X_1, ..., X_n \mid formula \}$

Domain Relational Calculus/2

- The domain relational calculus is based on specifying a number of variables that range over single values from domains of attributes.
- In the domain calculus the position of attributes is relevant. Attribute names are not used.
- Often the anonymous variable _ is used to shorten notation: $r(\_, \_, X, \_) = \exists U, V, W(r(U, V, X, W))$
- Example: To determine first and last names of all employees whose salary is above $50,000, we write the following DRC expression:

  $\{\ FN, LN \mid emp(FN, \_, LN, \_, Sal) \land Sal > 50000\ \}$

Review 2.15

branch(BranchName, BranchCity, Assets)
customer(CustName, CustStreet, CustCity)
account(AccNr, BranchName, Balance)
loan(LoanNr, BranchName, Amount)
depositor(CustName, AccNr)
borrower(CustName, LoanNr)

- Find all loans of over $1200.
- Find the loan number for each loan of an amount greater than $1200.
- Find the names of all customers who have a loan, an account, or both, from the bank.

Review 2.16

branch(BranchName, BranchCity, Assets)
customer(CustName, CustStreet, CustCity)
account(AccNr, BranchName, Balance)
loan(LoanNr, BranchName, Amount)
depositor(CustName, AccNr)
borrower(CustName, LoanNr)

- Find the names of all customers who have a loan at the Perryridge branch.

- Find the names of all customers who have a loan at the Perryridge branch but do not have an account at any branch of the bank.

Review 2.17

branch(BranchName, BranchCity, Assets)
customer(CustName, CustStreet, CustCity)
account(AccNr, BranchName, Balance)
loan(LoanNr, BranchName, Amount)
depositor(CustName, AccNr)
borrower(CustName, LoanNr)

- Determine the largest account balance.

Review 2.18

- Consider the following DRC expressions. Formulate equivalent relational algebra expressions. Assume column i of relation r has name rci.
  - $\{X, Y \mid p(X) \land q(X, Y)\}$
  - $\{X \mid p(X, 2) \land X > 7\}$
  - $\{X \mid p(X) \land \neg\exists Y(q(X, Y))\}$
  - $\{X \mid p(X) \land \neg\exists Y(p(Y) \land Y > X)\}$

Summary/1

- The Relational Model
  - attribute, domain, tuple, relation, database, schema
- Basic Relational Algebra Operators
  - Selection σ
  - Projection π
  - Union $\cup$
  - Difference $-$
  - Cartesian product $\times$
  - Rename ρ
- Additional Relational Algebra Operators
  - Join (theta, natural) $\bowtie$
  - Division $\div$
  - Assignment $\leftarrow$

Summary/2

- Extended Relational Algebra Operators
  - Generalized projection π
  - Aggregate function ϑ
  - Outer joins ⟕, ⟖, ⟗
- Modification of the database
  - insert, delete, update
- Relational Calculus
  - tuple relational calculus
  - domain relational calculus
- Know syntax of RA, TRC and DRC expressions.
- Be able to translate natural language queries into RA, TRC and DRC expressions.
- Be able to freely move between RA, TRC and DRC.