The Relational Model

The Relational Model

General Remarks

- Developed in the seventies by E.F.Codd (he received the Turing Award for it)
- Currently the most widely used data model
- Has replaced the hierarchical and network models
- The relational model is gradually being extended with object-oriented concepts

Definitions

- A relational database contains a set of relations
- A relation R consists of two parts:
  - An instance R: a table with rows and columns; the current content of the relation
  - A schema $\mathcal{R}$: specifies the name of the relation and the names and data types of the columns; defines the structure of the relation

Example of an Instance

- Suppose we store the data of students in the relation `Students`:

StudNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
...	...	...

- Each row is called a *tuple*
- Each column is called an *attribute*

Schema of the Relation Students

- `Students` has three attributes: `StudNr`, `Name`, and `Semester`
- Associated with each attribute is a domain (set of values)
  - D_{StudNr} = integer
  - D_{Name} = string
  - $D_{Semester}$ = integer
- The complete schema looks like this:

Students(StudNr:integer,
Name:string,
Semester:integer)

Formal Definition

- A relation (instance) is a subset of the Cartesian product of the domains
- $R \subseteq D_1 \times D_2 \times ... \times D_n$
- Example:
  Students $\subseteq D_{StudNr} \times D_{Name} \times D_{Semester}$ =
  Students $\subseteq$ integer $\times$ string $\times$ integer

Formal Definition(2)

- The size of a relation (number of tuples) is called the *cardinality* of the relation
- The minimal set of attributes whose values uniquely identify a tuple is called a *key*
- The *primary key* is underlined:

Students(<u>StudNr</u>, Name, Semester)

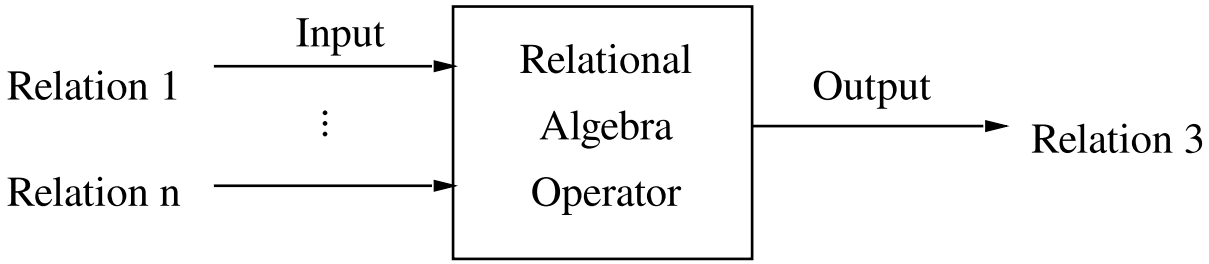
Query Languages

- So far only the structure of the data has been described; to retrieve information from the database, a query language is needed
- We first cover
  - the relational algebra
  - the relational calculus

Query Languages(2)

- The relational calculus
  - is a purely declarative language
  - was originally developed together with the relational model and forms the basis for SQL
- The relational algebra
  - is more procedurally oriented
  - A variant of it is used on the physical level to build query plans (the DBMS itself does not work declaratively)

- All operators of the relational algebra are set-oriented and closed:
  - Each operator receives one (or more) sets of tuples as input …
  - …and outputs a set of tuples



Projection

- Has the following syntax: $\pi_{A_1,\ldots,A_n}(R)$
- Selects the attributes (columns) $A_1, \ldots, A_n$ from the relation R
- Filters out all other attributes

Example

Students

StudNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Query: "Output the names of all students"

$\pi_{Name}(\text{Students})$

Name
Xenokrates
Jonas
Fichte
Aristoxenos
Schopenhauer
Carnap
Theophrastos
Feuerbach

Projection(2)

- What happens to duplicates?
- The relational model is set-oriented (a relation is a set of tuples)
- Duplicates are eliminated

Example

Students

StudNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Query: "Output the semesters of all students"

$\pi_{Semester}(\text{Students})$

Semester
18
12
10
8
6
3
2

Selection

- Syntax: $\sigma_p(R)$
- Selects all tuples from R that satisfy the predicate p
- Predicates can be combined with logical operators: $\wedge$, $\vee$, $\neg$
- The usual comparison operators are allowed: $=, <, \leq, >, \geq$

Example

Students

StudNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Query: "Output all students with a StudNr less than 26000 and a semester greater than 8"

$$\sigma_{StudNr<26000 \wedge Semester>8}(\text{Students})$$

StudNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12

Cross Product

- So far all operators worked on only one relation; what if information from several relations is needed?
- Two relations can be combined with the cross product: $R_1 \times R_2$
- Also called the Cartesian product

Example

Lectures

LectureNr	...	GivenBy
5041	...	2125
5049	...	2125
5001	...	2137
4630	...	2137

Professors

PersNr	Name	...
2125	Sokrates	...
2137	Kant	...

Query: “Output all lectures and professors”

Lectures $\times$ Professors

Result

LectureNr	...	GivenBy	PersNr	Name	...
5041	...	2125	2125	Sokrates	...
5041	...	2125	2137	Kant	...
5049	...	2125	2125	Sokrates	...
5049	...	2125	2137	Kant	...
5001	...	2137	2125	Sokrates	...
5001	...	2137	2137	Kant	...
4630	...	2137	2125	Sokrates	...
4630	...	2137	2137	Kant	...

Many of these combinations make no sense!

- Nonsensical combinations of the cross product can be filtered out with a subsequent selection
- For the query above:
  $\sigma_{GivenBy=PersNr}(\text{Lectures} \times \text{Professors})$
- Since this occurs very frequently, there is a dedicated operator for it: a join operator
- $R_1 \bowtie_{R_1.A_i=R_2.A_j} R_2 = \sigma_{R_1.A_i=R_2.A_j}(R_1 \times R_2)$
- Joins can be implemented more efficiently than cross products

Example

Query: “Output all lectures together with the responsible professors”

$$\text{Lectures} \bowtie_{GivenBy=PersNr} \text{Professors}$$

LectureNr	...	GivenBy	PersNr	Name	...
5041	...	2125	2125	Sokrates	...
5049	...	2125	2125	Sokrates	...
5001	...	2137	2137	Kant	...
4630	...	2137	2137	Kant	...

Joins(2)

- All attribute names must be unique, so one has to be careful with attributes of the same name in different relations
- Example:
  Assistants(PersNr, Name, Area, Boss)
  Professors(PersNr, Name, Rank, Room)
  $\text{Assistants} \bowtie_{Boss=PersNr} \text{Professors}$
- Here one uses the renaming operator ρ:
  $\rho_{APersNr \leftarrow PersNr, AName \leftarrow Name}(\text{Assistants})$
  $\bowtie_{Boss=PPersNr}$
  $\rho_{PPersNr \leftarrow PersNr, PName \leftarrow Name}(\text{Professors})$

Joins(3)

- Join operators are classified by their join predicate
  - Theta join (θ-join): the most general kind, the join predicate may contain arbitrary comparison operators: $=, \neq, <, \leq, >, \geq$
  - Equi-join: the join predicate may only check for equality (=)
  - Natural join: a special kind of equi-join that only compares attributes with the same name (and projects away redundant columns)

Query: “Output all lectures together with the responsible professors”

$$\rho_{PersNr \leftarrow GivenBy}(\text{Lectures}) \bowtie \text{Professors}$$

(join predicate is not specified, it is given implicitly)

LectureNr	...	PersNr	Name	...
5041	...	2125	Sokrates	...
5049	...	2125	Sokrates	...
5001	...	2137	Kant	...
4630	...	2137	Kant	...

Joins(4)

- Any number of relations can be joined in an algebra expression
- The order in which this is done does not matter for the correctness of the result (joins are commutative and associative)

Example

- Combine students with the lectures they attend

Students

StudNr	Name	...
26120	Fichte	...
27550	Schopenhauer	...
...	...	...

Lectures

LectureNr	Title	...
4052	Logic	...
5001	Fundamentals	...
...	...	...

attend

StudNr	LectureNr
26120	5001
27550	4052
27550	5001
...	...

Other Kinds of Joins

- If a tuple finds no join partner in the other relation, it is lost
- In the outer join ($\bowtie$) these tuples are preserved

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

$=$

Result				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	–	–
–	–	c_3	d_2	e_2

Left Outer Join

- For the left outer join this holds only for the tuples from the left relation

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Result				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	–	–

Right Outer Join

- For the right outer join this holds only for those from the right relation

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\bowtie$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Result				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
–	–	c_3	d_2	e_2

- A semi-join checks the join condition, but keeps only tuples from one of the two relations (those that satisfy the condition)

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\ltimes$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

$=$

Result		
A	B	C
a_1	b_1	c_1

Semi-Join(2)

- And here the second variant

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\rtimes$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

=

Result		
C	D	E
c_1	d_1	e_1

Anti-Join

- An anti-join checks the join condition, but keeps only tuples from one of the two relations (those that do **not** satisfy the condition)

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

$\triangleright$

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

$=$

Result		
A	B	C
a_2	b_2	c_2

Set Operations

- The usual set operations such as union, intersection, and difference can be applied to relations
- However, this requires that both relations have the same schema:
  - Same number of attributes
  - Corresponding attributes have the same type

Union

Prof1

PersNr	Name
2125	Sokrates
2126	Russel

Prof2

PersNr	Name
2126	Russel
2137	Kant

Query: “Combine both lists”

Prof1 ∪ Prof2

PersNr	Name
2125	Sokrates
2126	Russel
2137	Kant

Intersection

Prof1

PersNr	Name
2125	Sokrates
2126	Russel

Prof2

PersNr	Name
2126	Russel
2137	Kant

Query: “Which professors are on both lists?”

Prof1 ∩ Prof2

PersNr	Name
2126	Russel

Set Difference

Prof1

PersNr	Name
2125	Sokrates
2126	Russel

Prof2

PersNr	Name
2126	Russel
2137	Kant

Query: “Which professors are on the first but not on the second list?”

$$\text{Prof1} \setminus \text{Prof2}$$

PersNr	Name
2125	Sokrates

- Used for universally quantified queries
- Formal definition:
  $R_1 \div R_2 = \pi_{(\mathcal{R}_1 \setminus \mathcal{R}_2)}(R_1) \setminus \pi_{(\mathcal{R}_1 \setminus \mathcal{R}_2)}\Big(\big(\pi_{(\mathcal{R}_1 \setminus \mathcal{R}_2)}(R_1) \times R_2\big) \setminus R_1\Big)$
- ...or:

  $t \in R_1 \div R_2$ if for every $v \in R_2$ there exists a $u \in R_1$ such that:

$$u.\mathcal{R}_2 = v.\mathcal{R}_2$$
$$u.(\mathcal{R}_1 \setminus \mathcal{R}_2) = t.(\mathcal{R}_1 \setminus \mathcal{R}_2)$$

- Precondition $\mathcal{R}_2 \subseteq \mathcal{R}_1$

Example

attend

StudNr	LectureNr
25403	5022
26120	5001
27550	4052
27550	5001
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5001
29555	5022

Lect1	Lect2	Lect3
LectureNr	LectureNr	LectureNr
5001	5001	5001
	5041	4052

Example

Query: “List the `StudNr` of *all* students who attend *all* lectures in list 1,2,3”

Result

attend ÷ Lect1

StudNr
26120
27550
29120
29555

attend ÷ Lect2

StudNr
29120

attend ÷ Lect3

StudNr
27550

- The relational calculus is more declaratively oriented, i.e., result tuples are described without a rule for deriving them
- Based on mathematical first-order predicate logic
- There are two different variants:
  - Tuple relational calculus
  - Domain relational calculus

- A query in the relational calculus has the form
  $\{t \mid P(t)\}$
  where t is a tuple variable and P is a formula
- Examples:

All C4 professors

$$\{p \mid p \in \text{Professors} \wedge p.\text{Rank} = \text{'C4'}\}$$

Students with at least one Pythagoras lecture

$$\begin{aligned}\{s \mid s \in \text{Students}\\ \wedge\ \exists h \in \text{attend}\ (s.\text{StudNr} = h.\text{StudNr}\\ \wedge\ \exists v \in \text{Lectures}\ (h.\text{LectureNr} = v.\text{LectureNr}\\ \wedge\ \exists p \in \text{Professors}\ (p.\text{PersNr} = v.\text{GivenBy}\\ \wedge\ p.\text{Name} = \text{'Pythagoras'})))\}\end{aligned}$$

Atoms

- $s \in R$, with s a tuple variable and R a relation name
- $s.A \phi t.B$, with s and t tuple variables, A and B attribute names, and ϕ a comparison operator ($=$, $\neq$, $\leq$, ...)
- $s.A \phi c$ with c a constant

Formulas

- All atoms are formulas
- If P is a formula, then so are $\neg P$ and (P)
- If P_1 and P_2 are formulas, then so are $P_1 \wedge P_2$, $P_1 \vee P_2$, and $P_1 \Rightarrow P_2$
- If $P(t)$ is a formula with free variable t, then so are

$$\forall t \in R(P(t)) \quad \text{and} \quad \exists t \in R(P(t))$$

Safety

- $\{n \mid \neg(n \in \text{Professors})\}$, e.g., is not safe, since the result is infinite
- Condition for safety: the result of the expression must be a subset of the *domain* of the formula.
- The domain of a formula contains
  - all constants occurring in the formula
  - all attribute values of relations that are referenced in the formula

- A query in the domain calculus has the form:
  $\{[v_1, v_2, ..., v_n] \mid P(v_1, ..., v_n)\}$
  where $v_1, ..., v_n$ are domain variables and P is a formula
- Example:

StudNr and Name of the examinees of Sokrates

$$\begin{aligned}\{[m, n] \mid\ & \exists s([m, n, s] \in \text{Students} \\ & \wedge \exists p, v, g([m, p, v, g] \in \text{test} \\ & \wedge \exists a, r, b([p, a, r, b] \in \text{Professors} \\ & \wedge a = \text{'Sokrates'}))\}\end{aligned}$$

Atoms

- $[w_1, w_2, ..., w_m] \in R$, with m-ary relation R and domain variables $w_1, ..., w_m$
- $x \phi y$, with x and y domain variables, ϕ a comparison operator
- $x \phi c$, with constant c

Formal Definition(2)

Formulas

- All atoms are formulas
- If P is a formula, then so are $\neg P$ and (P)
- If P_1 and P_2 are formulas, then so are $P_1 \vee P_2$, $P_1 \wedge P_2$, and $P_1 \Rightarrow P_2$
- If $P(v)$ is a formula with free variable v, then so are $\exists v(P(v))$ and $\forall v(P(v))$

- $\{[p, n, r, o] \mid \neg([p, n, r, o] \in \text{Professors})\}$ unsafe
- $\{[x_1, x_2, ..., x_n] \mid P(x_1, x_2, ..., x_n)\}$ safe if
  - If tuple $[c_1, c_2, ..., c_n]$ is in the result, then each c_i $(1 \leq i \leq n)$ must be contained in the domain of P.
  - For each subformula $\exists x(P_1(x))$ it must hold that P_1 is satisfiable only for elements from the domain of P_1 – or possibly for none at all.
  - For each subformula $\forall x(P_1(x))$ it must hold that it is satisfied iff $P_1(x)$ is satisfied for all values of the domain of P_1. For all other values it must be satisfied anyway.

Expressive Power

- The three languages
  - relational algebra
  - tuple relational calculus, restricted to safe expressions
  - domain relational calculus, restricted to safe expressions

  are equally expressive

Summary

- The relational model is the most widely used data model in today's DBMSs
- This chapter explained the theoretical foundations of the model:
  - Definition of the model
  - Query languages