Principles of Data- and Knowledge-based Systems

Torsten Schaub
University of Potsdam
torsten@cs.uni-potsdam.de

Relational algebra: Overview

Outline

Example Database (1)

STUDENTS			
SID	FIRST	LAST	EMAIL
101	Ann	Smith	...
102	Michael	Jones	(null)
103	Richard	Turner	...
104	Maria	Brown	...

EXERCISES			
CAT	ENO	TOPIC	MAXPT
H	1	Rel. Algeb.	10
H	2	SQL	10
M	1	SQL	14

RESULTS			
SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

Example Database (2)

- `STUDENTS`: one row for each student in the course.
  - `SID`: "Student ID" (primary key).
  - `FIRST`, `LAST`: First and last name.
  - `EMAIL`: Email address (can be null).
- `EXERCISES`: one row for each graded exercise.
  - `CAT`: Exercise category (key together with `ENO`).
  - `ENO`: Exercise number (within category).
  - `TOPIC`: Topic of the exercise.
  - `MAXPT`: Max. no. of points (How many points is it worth?).
- `RESULTS`: one row for each submitted solution to an exercise.
  - `SID`: Student who wrote the solution.
  - `CAT`, `ENO`: Identification of the exercise.
  - `POINTS`: Number of points the student got for the solution.
  - A missing row means that the student did not yet hand in a solution to the exercise.

Relational Algebra (1)

- Relational algebra (RA) is a theoretical query language for the relational model.
- Relational algebra is not used in any commerical system on the user interface level.
- However, variants of it are used to represent queries internally (for query optimization and execution).
- Knowledge of relational algebra will help in understanding SQL and relational database systems.

Relational Algebra (2)

- An algebra is a set together with operations on this set.
- For instance, the set of integers together with the operations $+$ and $*$ forms an algebra.
- In the case of relational algebra, the set is the set of all finite relations.
- One operation of relational algebra is $\cup$ (union).
  This is natural since relations are sets.

Relational Algebra (3)

- Another operation of relational algebra is selection.
- E.g. $\sigma_{\mathtt{SID=101}}$ selects all tuples in the input relation that have the value "101" in column "SID":

$\sigma_{\mathtt{SID=101}}$ (RESULTS)

RESULTS			
SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

=

SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12

Relational Algebra (4)

- Since the output of a relational algebra operation is again a relation, it can be input for another relational algebra operation.
- A query is then a term/expression in this algebra.
- Arithmetic expressions like $(\mathtt{x} + 2) * \mathtt{y}$ are familiar.
- In relational algebra, relations are connected:

$$\pi_{\mathsf{FIRST},\,\mathsf{LAST}}(\mathsf{STUDENTS} \ \bowtie \ \sigma_{\mathsf{CAT}='\mathsf{M}'}(\mathsf{RESULTS})).$$

Relational Algebra (5)

- Null values are usually excluded in the definition of relational algebra, except when operations like the outer join are defined.
- Relational algebra treats relations as sets, i.e. any duplicate tuples are automatically eliminated.
- Relational algebra is much simpler than SQL, it has only five basic operations and can be completely defined on one page.
- Relational algebra is also a yardstick for measuring the expressiveness of query languages.
- E.g., every query that can be formulated in relational algebra can also be formulated in SQL.

Selection (1)

- The operation σ_φ selects a subset of the tuples of a relation, namely those which satisfy the condition φ.
  Selection acts like a filter on the input set.
- Example:

$$\sigma_{A=1}\left(\begin{array}{|c|c|}\hline A & B \\ \hline 1 & 3 \\ 1 & 4 \\ 2 & 5 \\ \hline \end{array}\right) \quad = \quad \begin{array}{|c|c|}\hline A & B \\ \hline 1 & 3 \\ 1 & 4 \\ \hline \end{array}$$

- The selection condition has the following form:

  $\langle$Term$\rangle$ $\langle$Comparison-Operator$\rangle$ $\langle$Term$\rangle$

- The selection condition returns a Boolean value (true or false) for a given input tuple.

Selection (2)

- ⟨Term⟩ (or "expression") is something that can be evaluated to a data type element for a given tuple:
  - an attribute name,
  - a data type constant, or
  - an expression composed from attributes and constants by data type operations like $+$, $-$, $*$, $/$.
- ⟨Comparison-Operator⟩ is
  - $=$ (equals), $\neq$ (not equals),
  - $<$ (less than), $>$ (greater than), $\leq$,
  - or other data type predicates (e.g. `LIKE`).
- Examples for Conditions:
  - `LAST = 'Smith'`
  - `POINTS >= 8`
  - `POINTS = MAXPT`

Selection (3)

- Of course, the attributes used in the selection condition must appear in the input table:

$$\sigma_{C=1}\left(\begin{array}{|c|c|}\hline A & B \\ \hline 1 & 3 \\ 2 & 4 \\ \hline \end{array}\right) \quad = \quad \textbf{Error}$$

- The following is legal, but the selection is superfluous, because the condition is always true:

$$\sigma_{A=A}\left(\begin{array}{|c|c|}\hline A & B \\ \hline 1 & 3 \\ 2 & 4 \\ \hline \end{array}\right) \quad = \quad \begin{array}{|c|c|}\hline A & B \\ \hline 1 & 3 \\ 2 & 4 \\ \hline \end{array}$$

Selection (4)

- It is no error if the result of a relational algebra expression happens to be empty in a specific state:

$$\sigma_{A=3}\left(\begin{array}{|c|c|} \hline A & B \\ \hline 1 & 3 \\ 2 & 4 \\ \hline \end{array} \right) \quad = \quad \emptyset$$

- It is legal, but most probably an error, to use a condition that is always false (inconsistent):

$$\sigma_{1=2}\left(\begin{array}{|c|c|} \hline A & B \\ \hline 1 & 3 \\ 2 & 4 \\ \hline \end{array} \right) \quad = \quad \emptyset$$

Selection (5)

- $\sigma_{\varphi}(\mathtt{R})$ corresponds to the following SQL query:

$$\begin{array}{ll} \texttt{SELECT} & * \\ \texttt{FROM} & R \\ \texttt{WHERE} & \varphi \end{array}$$

- I.e. selection corresponds to the `WHERE`-clause.
- A different relational algebra operation called “projection” corresponds to the `SELECT`-clause in SQL.
  This can be slightly confusing.

Extended Selection (1)

- In the basic selection operation, only simple conditions consisting of a single comparison ("atomic formula") are possible.
- However, one can extend the possible conditions by permitting to combine the single conditions by the logical operators $\wedge$ (and), $\vee$ (or), and $\neg$ (not):

φ_1	φ_2	$\varphi_1 \wedge \varphi_2$	$\varphi_1 \vee \varphi_2$	$\neg\varphi_1$
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

Extended Selection (2)

- $\varphi_1 \wedge \varphi_2$ is called the "conjunction of φ_1 and φ_2"
- $\varphi_1 \vee \varphi_2$ is called the "disjunction of φ_1 and φ_2"
- $\neg\varphi_1$ is called the "negation of φ_1".
- One can write "`and`", "`or`" and "`not`" instead of the symbols "$\wedge$", "$\vee$", "$\neg$" used in logic.

Extended Selection (3)

- The selection condition must permit evaluation for each input tuple in isolation.
- This extended form of selection is not necessary, since it can always be expressed with the basic operations of relational algebra.
  But it is convenient.
- E.g. $\sigma_{\varphi_1 \wedge \varphi_2}(\mathtt{R})$ is equivalent to $\sigma_{\varphi_1}\big(\sigma_{\varphi_2}(\mathtt{R})\big)$.

Projection (1)

- The projection π eliminates attributes (columns) from the input relation.
- Example:

$$\pi_{A,C}\left(\begin{array}{|c|c|c|}\hline A & B & C \\ \hline 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \\ \hline\end{array}\right) = \begin{array}{|c|c|}\hline A & C \\ \hline 1 & 7 \\ 2 & 8 \\ 3 & 9 \\ \hline\end{array}$$

Projection (2)

- In general, the projection $\pi_{\mathtt{A_{i_1},\dots,A_{i_k}}}(\mathtt{R})$ produces for each input tuple $(\mathtt{A_1}: \mathtt{d_1}, \dots, \mathtt{A_n}: \mathtt{d_n})$ an output tuple $(\mathtt{A_{i_1}}: \mathtt{d_{i_1}}, \dots, \mathtt{A_{i_k}}: \mathtt{d_{i_k}})$.
- I.e. the attribute values are not changed, but only the explicitly mentioned attributes are retained.
  All other attributes are "projected away".

Projection (3)

- Normally, there is one output tuple for every input tuple. However, if two input tuples lead to the same output tuple, the duplicate will be eliminated.
- Example:

$$\pi_B\left(\begin{array}{|c|c|}\hline A & B \\ \hline 1 & 4 \\ 2 & 5 \\ 3 & 4 \\ \hline \end{array}\right) \quad = \quad \begin{array}{|c|}\hline B \\ \hline 4 \\ 5 \\ \hline \end{array}$$

Projection (4)

- Attributes can be renamed: $\pi_{\mathtt{B}_1 \leftarrow \mathtt{A}_{\mathtt{i}_1}, \ldots, \mathtt{B}_\mathtt{k} \leftarrow \mathtt{A}_{\mathtt{i}_\mathtt{k}}}(\mathtt{R})$ transforms the input tuple $(\mathtt{A}_1 : \mathtt{d}_1, \ldots, \mathtt{A}_\mathtt{n} : \mathtt{d}_\mathtt{n})$ into the output tuple $(\mathtt{B}_1 : \mathtt{d}_{\mathtt{i}_1}, \ldots, \mathtt{B}_\mathtt{k} : \mathtt{d}_{\mathtt{i}_\mathtt{k}})$.
- Return values can be computed by datatype operations such as $+$ or || (string concatenation):

$$\pi_{\mathsf{SID},\ \mathsf{NAME} \leftarrow \mathsf{FIRST}\ ||\ '\ '\ ||\ \mathsf{LAST}}(\mathsf{STUDENTS}).$$

- Columns can be created with constant values:

$$\pi_{\mathsf{SID}, \mathsf{FIRST}, \mathsf{LAST}, \mathsf{GRADE} \leftarrow '\mathsf{A}'}(\mathsf{STUDENTS}).$$

Projection (5)

- The projection is a mapping, which is applied to every input tuple.
- Each input tuple is mapped locally to an output tuple.
- Only functions which are defined based on single input tuples are allowed.

Projection (6)

- $\pi_{A_1,\ldots,A_n}(R)$ corresponds to the SQL query:

$$\texttt{SELECT}\ A_1,\ \ldots,\ A_n$$
$$\texttt{FROM}\ R$$

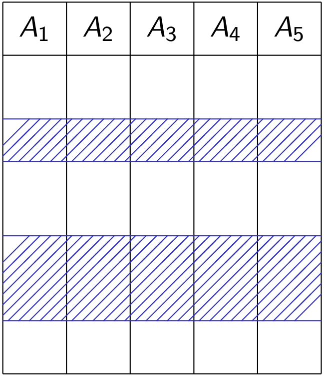
- $\pi_{B_1 \leftarrow A_1,\ldots,B_n \leftarrow A_n}(R)$ is written in SQL as follows:

$$\texttt{SELECT}\ A_1\ \texttt{AS}\ B_1,\ \ldots,\ A_n\ \texttt{AS}\ B_n$$
$$\texttt{FROM}\ R$$

- The keyword `AS` can be left out ("syntactic sugar").

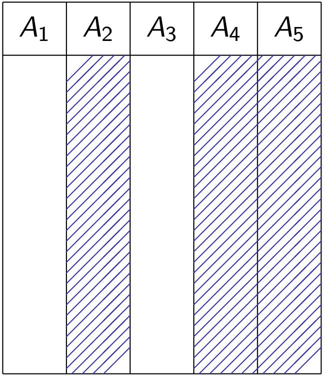
Summary

Selection σ



(Filters some rows)

Projection π



(Maps each row)

Combining Operations (1)

- Since the result of a relational algebra operation is also a relation, it can act as input to another algebra operation.
- For instance, to compute the exercises solved by student 102:

$$\pi_{\mathsf{CAT},\,\mathsf{ENO}}(\sigma_{\mathsf{SID}\,=\,102}(\mathsf{RESULTS}))$$

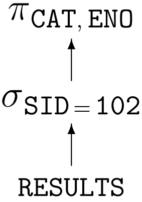
- An intermediate result can be stored in a temporary relation (can also be seen as macro definition):

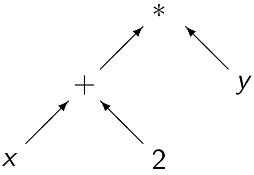
$$\mathtt{S102} \;:=\; \sigma_{\mathtt{SID}\,=\,102}(\mathtt{RESULTS});$$

$$\pi_{\mathtt{CAT},\,\mathtt{ENO}}(\mathtt{S102})$$

Combining Operations (2)

- Expressions of relational algebra may become clearer if depicted as operator tree:





- For comparison, an operator tree for the arithmetic expression $(x+2)*y$ is shown on the right.

Combining Operations (3)

- In SQL, σ and π (and $\times$, see below) can be combined in a single `SELECT`-expression:

```
SELECT CAT, ENO
FROM   RESULTS
WHERE  SID = 102
```

- Complex queries can be constructed step by step:

```
CREATE VIEW S102
AS SELECT *
   FROM   RESULTS
   WHERE  SID = 102
```

- Then `S102` can be used like a stored table.

Basic Operands

- The leaves of the operator tree are
  - the names of database relations
  - constant relations (explicitly enumerated tuples).
- A relation name R is a legal expression of relational algebra.
  Its value is the entire relation stored under that name.
  It corresponds to the SQL query:

```
SELECT *
FROM   R
```

- It is not necessary to write a projection on all attributes.

Exercises

- Which of the following relational algebra expressions are syntactically correct? What do they mean?
  - ☐ $\mathtt{STUDENTS}$.
  - ☐ $\sigma_{\mathtt{MAXPT} \neq 10}(\mathtt{EXERCISES})$.
  - ☐ $\pi_{\mathtt{FIRST}}(\pi_{\mathtt{LAST}}(\mathtt{STUDENTS}))$.
  - ☐ $\sigma_{\mathtt{POINTS} \leq 5}(\sigma_{\mathtt{POINTS} \geq 1}(\mathtt{RESULTS}))$.
  - ☐ $\sigma_{\mathtt{POINTS}}(\pi_{\mathtt{POINTS} = 10}(\mathtt{RESULTS}))$.

Outline

Cartesian Product (1)

- Often, answer tuples must be computed that are derived from two (or more) input tuples.
- This is done by the "cartesian product" $\times$.
- $R \times S$ concatenates ("glues together") each tuple from R with each tuple from S.

Cartesian Product (2)

- Example:

AA	B
$A1$	2
3	4

$\times$

AC	D
$A6$	7
8	9

$=$

AA	B	C	D
$A1$	2	6	7
1	2	8	9
3	4	6	7
3	4	8	9

- Since attribute names must be unique within a tuple, the cartesian product may only be applied when R and S have no attribute in common.
- This is no real restriction, since we may rename the attributes first (with π) and then apply $\times$.

Cartesian Product (3)

- Some authors define $\times$ such that it automatically renames double attributes:
  - E.g. for relations $R(A, B)$ and $S(B, C)$ the product $R \times S$ has attributes $(R.A, R.B, S.B, S.C)$.
  - As in SQL, one can also use the names A and C, because they uniquely identify the attributes.
- **In this course, this is not permitted!**

Cartesian Product (4)

- If the relation R contains n tuples, and the relation S contains m tuples, then $R \times S$ contains $n * m$ tuples.
- The cartesian product is in itself seldom useful, because it leads to a "blowup" in relation size.
- The problem is that $R \times S$ combines each tuple from R with each tuple from S. Usually, the goal is to combine only selected pairs of tuples.
- Thus, the cartesian product is useful only as input for a following selection.

Cartesian Product (5)

- $R \times S$ is written in SQL as

```
SELECT *
FROM   R, S
```

- In SQL it is no error if the two relations have common attribute names, since one can reference attributes also in the form "$R.A$" or "$S.A$".

Renaming

- An operator $\rho_R(S)$ that prepends "R." to all attribute names is sometimes useful:

$$\rho_R\left(\begin{array}{|r|r|}\hline A & B \\ \hline 1 & 2 \\ 3 & 4 \\ \hline\end{array}\right) \quad = \quad \begin{array}{|r|r|}\hline R.A & R.B \\ \hline 1 & 2 \\ 3 & 4 \\ \hline\end{array}$$

- This is only an abbreviation for an application of the projection: $\pi_{\mathtt{R.A}\leftarrow\mathtt{A},\,\mathtt{R.B}\leftarrow\mathtt{B}}(\mathtt{S})$.
- Otherwise, attribute names in relational algebra do not automatically contain the relation name.

Join (1)

- Since this combination of cartesian product and selection is so common, a special symbol has been introduced for it:

$$R \underset{A=B}{\bowtie} S \quad \text{is an abbreviation for} \quad \sigma_{\mathtt{A=B}}(\mathtt{R} \times \mathtt{S}).$$

- This operation is called "join": It is used to join two tables (i.e. combine their tuples).
- The join is one of the most important and useful operations of the relational algebra.

Join (2)

STUDENTS ⋈ RESULTS						
SID	FIRST	LAST	EMAIL	CAT	ENO	POINTS
101	Ann	Smith	···	H	1	10
101	Ann	Smith	···	H	2	8
101	Ann	Smith	···	M	1	12
102	Michael	Jones	(null)	H	1	9
102	Michael	Jones	(null)	H	2	9
102	Michael	Jones	(null)	M	1	10
103	Richard	Turner	···	H	1	5
103	Richard	Turner	···	M	1	7

- Student Maria Brown does not appear, because she has not submitted any homework and did not participate in the exam.
- What is shown above, is the natural join of the two tables. However, in the following first the standard join is explained.

Join (3)

- The above procedure is called "nested loop join".
- Note that the intermediate result of $R \times S$ is not materialized (explicitly stored).
- Quite a lot of different algorithms have been developed for computing the join.

Join (4)

- The join condition does not have to take the form $A = B$ (although this is most common). It can be an arbitrary condition, for instance also $A < B$.
- A typical application of a join is to combine tuples based on a foreign key, e.g.

$$\mathrm{RESULTS} \underset{\mathrm{SID}=\mathrm{SID}'}{\bowtie} \pi_{\mathrm{SID}' \leftarrow \mathrm{SID},\mathrm{FIRST},\mathrm{LAST},\mathrm{EMAIL}}(\mathrm{STUDENTS})$$

Join (5)

- The join not only combines tuples, but also acts as a filter: It eliminates tuples without join partner.
  (Note: Foreign key ensures that join partner exists.)

AA	B
$A1$	2
3	4

$\underset{B=C}{\bowtie}$

AC	D
$A4$	5
6	7

$=$

AA	B	C	D
$A3$	4	4	5

- A "semijoin" (ı , ı) works only as a filter.
- An "outer join" (see end of this part) does not work as a filter: It preserves all input tuples.

Natural Join (1)

- Another useful abbreviation is the “natural join” $\bowtie$.
- It combines tuples which have equal values in attributes with the same name.

AA	B
$A1$	2
3	4

$\bowtie$

AB	C
$A4$	5
4	8
6	7

$=$

AA	B	C
$A3$	4	5
3	4	8

Natural Join (2)

- The natural join of two relations
  - $\mathtt{R}(\mathtt{A}_1, \dots, \mathtt{A}_n, \mathtt{B}_1, \dots, \mathtt{B}_k)$ and
  - $\mathtt{S}(\mathtt{B}_1, \dots, \mathtt{B}_k, \mathtt{C}_1, \dots, \mathtt{C}_m)$

  produces in database state $\mathcal{I}$ all tuples of the form

  $$(\mathtt{a}_1, \dots, \mathtt{a}_n, \mathtt{b}_1, \dots, \mathtt{b}_k, \mathtt{c}_1, \dots, \mathtt{c}_m)$$

  such that
  - $(\mathtt{a}_1, \dots, \mathtt{a}_n, \mathtt{b}_1, \dots, \mathtt{b}_k) \in \mathcal{I}(\mathtt{R})$ and
  - $(\mathtt{b}_1, \dots, \mathtt{b}_k, \mathtt{c}_1, \dots, \mathtt{c}_m) \in \mathcal{I}(\mathtt{S})$.

Natural Join (3)

- The natural join not only corresponds to a cartesian product followed by a selection, but also
  - automatically renames one copy of each common attribute before the cartesian product, and
  - uses a projection to eliminate these double attributes at the end.
- E.g., given $\mathtt{R}(\mathtt{A},\mathtt{B})$, and $\mathtt{S}(\mathtt{B},\mathtt{C})$, then $\mathtt{R} \bowtie \mathtt{S}$ is an abbreviation for

$$\pi_{A,B,C}\big(\sigma_{B=B'}(R \times \pi_{B'\leftarrow B,\, C}(S))\big).$$

Natural Join (4)

A Note on Relational Database Design

- In order to support the natural join, it is beneficial to give attributes from different relations, which are typically joined together, the same name.
- Even if the utilized query language does not have a natural join, this provides good documentation.
- If domain names are used as attribute names, this will happen automatically.
- Try to avoid giving the same name to attributes which will probably not be joined.

Joins in SQL (1)

- $\mathtt{R} \underset{\mathtt{A=B}}{\bowtie} \mathtt{S}$ is normally written in SQL like $\sigma_{\mathtt{A=B}}(\mathtt{R} \times \mathtt{S})$:

```
SELECT *
FROM   R, S
WHERE  A = B
```

- Attributes can also referenced with explicit relation name (required if the attribute name appears in both relations):

```
SELECT *
FROM   R, S
WHERE  R.A = S.B
```

Joins in SQL (2)

- In SQL-92, one can also write:

```
SELECT *
FROM   R JOIN S ON R.A = S.B
```

- This shows the influence of relational algebra, but it is not really in the spirit of SQL.
- E.g. in Oracle 8i, the new alternative syntax for the join is not supported.

Algebraic Laws (1)

- The join satisfies the associativity condition:

$$(R \bowtie S) \bowtie T \;=\; R \bowtie (S \bowtie T).$$

- Therefore, the parentheses are not needed:

$$R \;\bowtie\; S \;\bowtie\; T.$$

- The join is not quite commutative: The sequence of columns (from left to right) will be different.
- However, if a projection follows later, this does not matter (one can also introduce π for this purpose):

$$\pi_{\dots}(R \bowtie S) \;=\; \pi_{\dots}(S \bowtie R).$$

Algebraic Laws (2)

- Further algebraic laws hold, which are utilized in the query optimizer of a relational DBMS.
- E.g., if the condition φ refers only to S, then

$$\sigma_{\varphi}(R \bowtie S) \;=\; R \bowtie \sigma_{\varphi}(S).$$

  The right hand side can often be evaluated more efficiently (depending on relation sizes, indexes).
- But for this course, efficiency is not important.

A Common Query Pattern (1)

- The following query structure is very common:

$$\pi_{A_1,\ldots,A_k}\big(\sigma_\varphi(R_1 \bowtie \cdots \bowtie R_n)\big).$$

  - First join all tables which are needed to answer the query.
  - Second, select the relevant tuples.
  - Third, project on the attributes which should appear in the output.

A Common Query Pattern (1)

- Patterns are often useful conventions of thought.
- But relational algebra operations can be combined in any way. It is not necessary to adhere to this pattern.
- In contrast, in SQL the keywords `SELECT` and `FROM` are required, and the sequence must always be

```
SELECT ... FROM ... WHERE ...
```

A Common Query Pattern (1)

- $\pi_{\mathtt{A_1,\ldots,A_k}}(\sigma_\varphi(\mathtt{R_1} \bowtie \cdots \bowtie \mathtt{R_n}))$ is written in SQL as:

$$\begin{array}{ll}\texttt{SELECT DISTINCT} & A_1\texttt{, ..., } A_k \\ \texttt{FROM} & R_1\texttt{, ..., } R_n \\ \texttt{WHERE} & \varphi \ \texttt{AND} \ \langle\text{Join Conditions}\rangle\end{array}$$

- It is a common mistake to forget a join condition.
- Usually, every two relations are linked (directly or indirectly) by equations, e.g. $\mathtt{R_1.B_1} = \mathtt{R_2.B_2}$.
- "`DISTINCT`" is not always necessary (see above).

A Common Query Pattern (1)

- To formulate a query, think first about the relations needed:
  - Usually, the natural language version of the query names certain attributes.
  - Each such attribute requires at least one relation which contains this attribute.

A Common Query Pattern (1)

- Query Formulation, continued:
  - Finally, sometimes intermediate relations are required in order to make the join meaningful.
  - E.g., suppose that relations $\mathtt{R}(\mathtt{A}, \mathtt{B})$, $\mathtt{S}(\mathtt{B}, \mathtt{C})$, $\mathtt{T}(\mathtt{C}, \mathtt{D})$ are given and attributes $\mathtt{A}$ and $\mathtt{D}$ are needed. Then $\mathtt{R} \bowtie \mathtt{T}$ would not be correct. Why?
  - Instead, the join must be $\mathtt{R} \bowtie \mathtt{S} \bowtie \mathtt{T}$.
  - It often helps to have a graphical representation of the foreign key links between the tables (which correspond to typical joins).

Self Joins (1)

- Sometimes, it is necessary to refer to more than one tuple from one relation at the same time.
- E.g. who got more points than student 101 for any exercise?
- In this case, two tuples of the relation `RESULTS` are needed in order to compute one result tuple:
  - One tuple for the student 101.
  - One tuple for the same exercise, in which `POINTS` is greater than in the first tuple.

Self Joins (2)

- This requires a generalization of the above query pattern, where two copies of a relation are joined (at least one must be renamed first).

$$S := \rho_{\mathtt{X}}(\mathtt{RESULTS}) \underset{\substack{\mathtt{X.CAT} = \mathtt{Y.CAT} \\ \wedge\, \mathtt{X.ENO} = \mathtt{Y.ENO}}}{\bowtie} \rho_{\mathtt{Y}}(\mathtt{RESULTS});$$

$$\pi_{\mathtt{X.SID}}(\sigma_{\mathtt{X.POINTS} > \mathtt{Y.POINTS} \wedge \mathtt{Y.SID} = \mathtt{101}}(S))$$

- Such joins of a table with itself are sometimes called "self joins".

Outline

Set Operations (1)

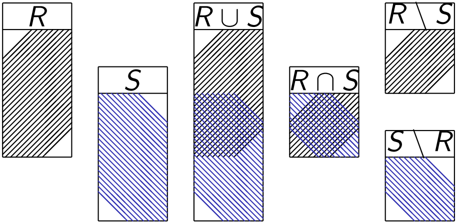
- Since relations are sets (of tuples), the usual set operations $\cup$, $\cap$, $\setminus$ can also be applied to relations.
- However, both input relations must have the same schema.
- $\mathtt{R} \cup \mathtt{S}$ contains all tuples which are contained in R, in S, or in both relations (Union).

Set Operations (2)

- $\mathtt{R} \setminus \mathtt{S}$ contains all tuples which are contained in R, but not in S (Set Difference).
- $\mathtt{R} \cap \mathtt{S}$ contains all tuples which are contained in both, R and S (Intersection).
- Intersection is (like the join) a derived operation: It can be expressed in terms of $\setminus$:

$$\mathtt{R} \cap \mathtt{S} = \mathtt{R} \setminus (\mathtt{R} \setminus \mathtt{S}).$$

Set Operations (3)



Union (1)

- Without $\cup$, every result column can contain only values from a single column of the stored tables.
- E.g. suppose that besides the registered students, who submit homeworks and write exams, there are also guests that attend the course:

$$\texttt{GUESTS}(\underline{\texttt{FIRST}}, \underline{\texttt{LAST}}, \texttt{EMAIL}^{o}).$$

- The task is to produce a list of email addresses of registered students and guests in one query.

Union (2)

- With $\cup$, this is simple:

$$\pi_{\texttt{EMAIL}}(\texttt{STUDENTS}) \cup \pi_{\texttt{EMAIL}}(\texttt{GUESTS}).$$

- This query cannot be formulated without $\cup$.
- Another typical application of $\cup$ is a case analysis:

$$\texttt{MPOINTS} := \pi_{\texttt{SID,POINTS}}\big(\sigma_{\texttt{CAT='M'} \wedge \texttt{ENO}=\texttt{1}}(\texttt{RESULTS})\big);$$

$$\begin{array}{l}
\quad\pi_{\texttt{SID, GRADE}\leftarrow\texttt{'A'}}\big(\sigma_{\texttt{POINTS}\geq\texttt{12}}(\texttt{MPOINTS})\big) \\
\cup\,\pi_{\texttt{SID, GRADE}\leftarrow\texttt{'B'}}\big(\sigma_{\texttt{POINTS}\geq\texttt{10}\,\wedge\,\texttt{POINTS}<\texttt{12}}(\texttt{MPOINTS})\big) \\
\cup\,\pi_{\texttt{SID, GRADE}\leftarrow\texttt{'C'}}\big(\sigma_{\texttt{POINTS}\geq\texttt{7}\,\wedge\,\texttt{POINTS}<\texttt{10}}(\texttt{MPOINTS})\big) \\
\cup\,\pi_{\texttt{SID, GRADE}\leftarrow\texttt{'F'}}\big(\sigma_{\texttt{POINTS}<\texttt{7}}(\texttt{MPOINTS})\big)
\end{array}$$

Union (3)

- In SQL, `UNION` can be written between two `SELECT`-expressions:

```
SELECT SID, 'A' AS GRADE
FROM   RESULTS
WHERE  CAT = 'M' AND ENO = 1 AND POINTS >= 12
UNION
SELECT SID, 'B' AS GRADE
FROM   RESULTS
WHERE  CAT = 'M' AND ENO = 1
AND    POINTS >= 10 AND POINTS < 12
UNION
...
```

Union (4)

- `UNION` was already contained in the first SQL standard (SQL-86) and is supported in all DBMS.
- There is no other way to formulate a union in SQL.
- `UNION`, an algebra operator, is a bit strange in SQL.

Set Difference (1)

- The operators σ, π, $\times$, $\bowtie$, $\cup$ have a monotonic behaviour, e.g.

$$R \subseteq S \implies \sigma_\varphi(R) \subseteq \sigma_\varphi(S)$$

- Then it follows that also every query Q that uses only the above operators behaves monotonically:
  - Let $\mathcal{I}_1$ be a database state, and let $\mathcal{I}_2$ result from $\mathcal{I}_1$ by the insertion of one or more tuples.
  - Then every tuple t contained in the answer to Q in $\mathcal{I}_1$ is also contained in the answer to Q in $\mathcal{I}_2$.

Set Difference (2)

- If the query must behave nonmonotonically, it is clear that the previous operations are not sufficient, and one must use set difference "$\setminus$". E.g.
  - Which student has not solved any exercise?
  - Who got the most points in Homework 1?
  - Who has solved all exercises in the database?

Set Difference (3)

STUDENTS			
SID	FIRST	LAST	EMAIL
101	Ann	Smith	...
102	Michael	Jones	(null)
103	Richard	Turner	...
104	Maria	Brown	...

EXERCISES			
CAT	ENO	TOPIC	MAXPT
H	1	Rel. Algeb.	10
H	2	SQL	10
M	1	SQL	14

RESULTS			
SID	CAT	ENO	POINTS
101	H	1	10
101	H	2	8
101	M	1	12
102	H	1	9
102	H	2	9
102	M	1	10
103	H	1	5
103	M	1	7

Set Difference (4)

- E.g. which student has not solved any exercise?

$$\texttt{NO\_SOL} := \pi_{\texttt{SID}}(\texttt{STUDENTS}) \setminus \pi_{\texttt{SID}}(\texttt{RESULTS});$$

$$\pi_{\texttt{FIRST},\,\texttt{LAST}}(\texttt{STUDENTS} \bowtie \texttt{NO\_SOL})$$

- What is the error in this query?

$$\pi_{\texttt{SID},\,\texttt{FIRST},\,\texttt{LAST}}(\texttt{STUDENTS}) \setminus \pi_{\texttt{SID}}(\texttt{RESULTS})$$

- Is this a correct solution?

$$\pi_{\texttt{FIRST},\,\texttt{LAST}}\big(\texttt{STUDENTS} \underset{\texttt{SID}\neq\texttt{SID2}}{\bowtie} \pi_{\texttt{SID2}\leftarrow\texttt{SID}}(\texttt{RESULTS})\big)$$

Set Difference (5)

- When using $\setminus$, a typical pattern is the anti-join.
- E.g. given $\mathtt{R}(\mathtt{A}, \mathtt{B})$ and $\mathtt{S}(\mathtt{B}, \mathtt{C})$, the tuples from R that do not have a join partner in S can be computed as follows:

$$R \bowtie \big(\pi_B(R) \setminus \pi_B(S)\big).$$

- The following is equivalent: $R \setminus \pi_{A,B}(R \bowtie S)$.
- A symbol for the anti-join is not common, but one could use $R \,\overline{\ltimes}\, S$ (a complemented semi-join).

Set Difference (6)

- Note that in order for the set difference $R \setminus S$ to be applicable, it is not (!) required that $S \subseteq R$.
- E.g. this query computes the SIDs of students that have solved Homework 2, but not Homework 1:

$$\begin{array}{rl} & \pi_{\texttt{SID}}\big(\sigma_{\texttt{CAT}=\texttt{'H'}\,\wedge\,\texttt{ENO}=2}(\texttt{RESULTS})\big) \\ \setminus & \pi_{\texttt{SID}}\big(\sigma_{\texttt{CAT}=\texttt{'H'}\,\wedge\,\texttt{ENO}=1}(\texttt{RESULTS})\big) \end{array}$$

- It is no problem that there might also be students that have solved Homework 1, but not Homework 2.

Set Difference (7)

- Suppose that R and S are represented in SQL as
  - SELECT A_1, ..., A_n FROM R_1, ..., R_m WHERE φ_1
  - SELECT B_1, ..., B_n FROM S_1, ..., S_k WHERE φ_2
- Then $R \setminus S$ can be represented as

```
SELECT A1, ..., An
FROM   R1, ..., Rm
WHERE  φ1 AND NOT EXISTS
       (SELECT * FROM S1, ..., Sk
        WHERE  φ2
        AND    B1 = A1 AND ... AND Bn = An)
```

Union vs. Join

- Two alternative representations of the homework, midterm, and final totals of the students are:

Results_1			
STUDENT	H	M	F
Jim Ford	95	60	75
Ann Lloyd	80	90	95

Results_2		
STUDENT	CAT	PCT
Jim Ford	H	95
Jim Ford	M	60
Jim Ford	F	75
Ann Lloyd	H	80
Ann Lloyd	M	90
Ann Lloyd	F	95

- Give algebra expressions to translate between them.

Outline

Outer Join (1)

- The usual join eliminates tuples without partner:

A	B
a_1	b_1
a_2	b_2

$\bowtie$

B	C
b_2	c_2
b_3	c_3

$=$

A	B	C
a_2	b_2	c_2

- The left outer join guarantees that tuples from the left table will appear in the result:

A	B
a_1	b_1
a_2	b_2

$⟕$

B	C
b_2	c_2
b_3	c_3

$=$

A	B	C
a_1	b_1	
a_2	b_2	c_2

Outer Join (2)

- The right outer join preserves tuples from the right table:

A	B
a_1	b_1
a_2	b_2

$⟖$

B	C
b_2	c_2
b_3	c_3

$=$

A	B	C
a_2	b_2	c_2
	b_3	c_3

- The full outer join does not eliminate any tuples:

A	B
a_1	b_1
a_2	b_2

$⟗$

B	C
b_2	c_2
b_3	c_3

$=$

A	B	C
a_1	b_1	
a_2	b_2	c_2
	b_3	c_3

Outer Join (3)

- E.g. students with their homework results, students without homework result are listed with null values:

$$\texttt{STUDENTS} ⟕ \pi_{\texttt{SID,ENO,POINTS}}(\sigma_{\texttt{CAT='H'}}(\texttt{RESULTS}))$$

SID	FIRST	LAST	EMAIL	ENO	POINTS
101	Ann	Smith	···	1	10
101	Ann	Smith	···	2	8
102	Michael	Jones	(null)	1	9
102	Michael	Jones	(null)	2	9
103	Richard	Turner	···	1	5
104	Maria	Brown	···	(null)	(null)

Outer Join (4)

- Is there any difference between
  - STUDENTS $\bowtie$ RESULTS and
  - STUDENTS ⟗ RESULTS?
- The outer join is especially useful together with aggregation functions (e.g. `count`, `sum`), see below.
- With a selection on the outer join result, one can use it like a set difference: But questionable style.

Outer Join (5)

- The outer join is a derived operation (like $\bowtie$, $\cap$), i.e. it can be simulated with the five basic relational algebra operations.
- E.g. consider relations $R(A, B)$ and $S(B, C)$.
- The left outer join $R ⟕ S$ is an abbreviation for

$$R \bowtie S \quad \cup \quad (R \setminus \pi_{A,B}(R \bowtie S)) \times \{(C: \mathit{null})\}$$

(where $\bowtie$ can be further replaced by $\times$, σ, π).

Outline

Definitions: Syntax (1)

- A set $\mathcal{S}_\mathcal{D}$ of data type names, and for each $D \in \mathcal{S}_\mathcal{D}$ a set $val(D)$ of values.
- A set $\mathcal{A}$ of possible attribute names (identifiers).
- A relational database schema $\mathcal{S}$ that consists of
  - a finite set of relation names $\mathcal{R}$, and
  - for every $R \in \mathcal{R}$, a relation schema $sch(R)$.

  (Constraints are not important here.)

Definitions: Syntax (2)

- One recursively defines the set of relational algebra (RA) expressions (queries) together with the relation schema of each RA expression.
  Base Cases:
  - $\boxed{R}$: For every $R \in \mathcal{R}$, the relation name R is an RA expression with schema $sch(R)$.
  - $\boxed{\{(A_1\colon d_1, \ldots, A_n\colon d_n)\}}$ ("relation constant") is an RA expression if $A_1, \ldots, A_n \in \mathcal{A}$, and $d_i \in val(D_i)$ for $1 \leq i \leq n$ with $D_1, \ldots, D_n \in \mathcal{S}_{\mathcal{D}}$. The schema of this RA expression is $(A_1\colon D_1, \ldots, A_n\colon D_n)$.

Definitions: Syntax (3)

- Recursive cases, Part 1: Let Q be an RA expression with schema $\rho = (A_1 \colon D_1, \ldots, A_n \colon D_n)$. Then also the following are RA expressions:
  - $\boxed{\sigma_{A_i = A_j}(Q)}$ for $i, j \in \{1, \ldots, n\}$. It has schema ρ.
  - $\boxed{\sigma_{A_i = d}(Q)}$ for $i \in \{1, \ldots, n\}$ and $d \in val(D_i)$. It has schema ρ.
  - $\boxed{\pi_{B_1 \leftarrow A_{i_1}, \ldots, B_m \leftarrow A_{i_m}}(Q)}$ for $i_1, \ldots, i_k \in \{1, \ldots, n\}$ and $B_1, \ldots, B_m \in \mathcal{A}$ such that $B_j \neq B_k$ for $j \neq k$. It has schema $(B_1 \colon D_{i_1}, \ldots, B_m \colon D_{i_m})$.

Definitions: Syntax (4)

- Recursive cases, continued: Let Q_1 and Q_2 be RA expressions with the same schema ρ. Then also the following are RA expressions with schema ρ:
  - $\boxed{(Q_1) \cup (Q_2)}$ and $\diamond$ $\boxed{(Q_1) \setminus (Q_2)}$
- Let Q_1 and Q_2 be RA expressions with the schemas $(A_1\colon D_1, \ldots, A_n\colon D_n)$ and $(B_1\colon E_1, \ldots, B_n\colon E_m)$, resp. If $\{A_1, \ldots, A_n\} \cap \{B_1, \ldots, B_m\} = \emptyset$, then also the following is an RA expression:
  - $\boxed{(Q_1) \times (Q_2)}$
    Schema: $(A_1 : D_1, \ldots, A_n : D_n, B_1 : E_1, \ldots, B_m : E_m)$.

Definitions: Syntax (5)

- Nothing else is a relational algebra expression.

Abbreviations

- Parentheses can be left out if the structure is clear (or the possible structures are equivalent).
- As explained above, additional algebra operations (like the join) can be introduced as abbreviations.

Definitions: Semantics (1)

- A database state $\mathcal{I}$ defines a finite relation $\mathcal{I}(R)$ for every relation name R in the database schema.
- The result of a query Q, i.e. an RA expression, in a database state $\mathcal{I}$ is a relation. The query result is written $\mathcal{I}[Q]$ and defined recursively corresponding to the structure of Q:
  - If Q is a relation name R, then $\mathcal{I}[Q] := \mathcal{I}(R)$.
  - If Q is a constant relation $\{(A_1\colon d_1, \ldots, A_n\colon d_n)\}$, then $\mathcal{I}[Q] := \{(d_1, \ldots, d_n)\}$.

Definitions: Semantics (2)

- Definition of the result $\mathcal{I}[Q]$ of an RA expression Q in state $\mathcal{I}$, continued:
  - If Q has the form $\sigma_{A_i=A_j}(Q_1)$, then

    $$\mathcal{I}[Q] := \{(d_1, \ldots, d_n) \in \mathcal{I}[Q_1] \mid d_i = d_j\}.$$

  - If Q has the form $\sigma_{A_i=d}(Q_1)$, then

    $$\mathcal{I}[Q] := \{(d_1, \ldots, d_n) \in \mathcal{I}[Q_1] \mid d_i = d\}.$$

  - If Q has the form $\pi_{B_1 \leftarrow A_{i_1}, \ldots, B_m \leftarrow A_{i_m}}(Q_1)$, then

    $$\mathcal{I}[Q] := \{(d_{i_1}, \ldots, d_{i_m}) \mid (d_1, \ldots, d_n) \in \mathcal{I}[Q_1]\}.$$

Definitions: Semantics (3)

- Definition of $\mathcal{I}[Q]$, continued:
  - If Q has the form $(Q_1) \cup (Q_2)$ then
    $$\mathcal{I}[Q] := \mathcal{I}[Q_1] \cup \mathcal{I}[Q_2].$$
  - If Q has the form $(Q_1) \setminus (Q_2)$ then
    $$\mathcal{I}[Q] := \mathcal{I}[Q_1] \setminus \mathcal{I}[Q_2].$$
  - If Q has the form $(Q_1) \times (Q_2)$, then
    $$\begin{aligned}\mathcal{I}[Q] := \{(d_1, \dots, d_n, e_1, \dots, e_m) \mid & \\ & (d_1, \dots, d_n) \in \mathcal{I}[Q_1], \\ & (e_1, \dots, e_m) \in \mathcal{I}[Q_2]\}.\end{aligned}$$

Monotonicity

- Definition A database state $\mathcal{I}_1$ is smaller than (or equal to) a database state $\mathcal{I}_2$, written $\mathcal{I}_1 \subseteq \mathcal{I}_2$, if and only if $\mathcal{I}_1(R) \subseteq \mathcal{I}_2(R)$ for all relation names R in the schema.
- Theorem If an RA expression Q does not contain the $\setminus$ (set difference) operator, then the following holds for all database states $\mathcal{I}_1, \mathcal{I}_2$:

$$\mathcal{I}_1 \subseteq \mathcal{I}_2 \implies \mathcal{I}_1[Q] \subseteq \mathcal{I}_2[Q].$$

Equivalence (1)

- Definition Two RA expressions Q_1 and Q_2 are equivalent if and only if they have the same schema and for all database states $\mathcal{I}$ the following holds:

  $$\mathcal{I}[Q_1] = \mathcal{I}[Q_2].$$

- There are two notions of equivalence, depending on whether one considers all structurally possible states or only states that satisfy the constraints.

Equivalence (2)

- Examples for equivalences
  - $\sigma_{\varphi_1}(\sigma_{\varphi_2}(Q))$ is equivalent to $\sigma_{\varphi_2}(\sigma_{\varphi_1}(Q))$.
  - $(Q_1 \times Q_2) \times Q_3$ is equivalent to $Q_1 \times (Q_2 \times Q_3)$.
  - If A is an attribute in the schema of Q_1: $\sigma_{A=d}(Q_1 \times Q_2)$ is equivalent to $(\sigma_{A=d}(Q_1)) \times Q_2$
- Theorem The equivalence of relational algebra expressions is undecidable.

Limitations of RA (1)

- Let R be a relation name with schema $(A\colon D, B\colon D)$ and let $val(D)$ be infinite.
- The transitive closure of $\mathcal{I}(R)$ is the set of all $(d, e) \in val(D) \times val(D)$ such that there are $n \in \mathbb{N}$ $(n \geq 1)$ and $d_0, \ldots, d_n \in val(D)$ with $d = d_0$, $e = d_n$ and $(d_{i-1}, d_i) \in \mathcal{I}(R)$ for $i = 1, \ldots, n$.
- E.g. R could be the relation "PARENT", then the transitive closure are all ancestor-relationships (parents, grandparents, great-grandparents, ...).

Limitations of RA (2)

- Theorem There is no RA expression Q such that $\mathcal{I}[Q]$ is the transitive closure of $\mathcal{I}(R)$ for all database states $\mathcal{I}$.
- E.g. in the ancestor example, one would need an additional join for every additional generation.
- Therefore, if one does not know, how many generations the database contains, one cannot write a query that works for all possible database states.

Limitations of RA (3)

- This of course implies that relational algebra is not computationally complete:
  - Not every function from database states to relations for which a C program exists can be formulated in relational algebra.
  - However, this can also not be expected, since one wants to be sure that query evaluation always terminates. This is guaranteed for RA.

Limitations of RA (4)

- All RA queries can be computed in time that is polynomically in the size of the database.
- Thus, also very complex functions cannot be formulated in relational algebra.
- As the transitive closure shows, not all problems of polynomial complexity can be formulated in RA.

Expressive Power (1)

- A query language $\mathcal{L}$ for the relational model is called strong relationally complete if for every database schema $\mathcal{S}$ and for every RA expression Q_1 with respect to $\mathcal{S}$ there is a query $Q_2 \in \mathcal{L}_{\mathcal{S}}$ such that for all database states $\mathcal{I}$ with respect to $\mathcal{S}$ the two queries produce the same result: $\mathcal{I}[Q_1] = \mathcal{I}[Q_2]$.
- I.e. the requirement is that every relational algebra expression can be translated into an equivalent query in that language.

Expressive Power (2)

- E.g. SQL is strong relationally complete.
- If the translation of queries is possible in both directions, the two query languages have the same expressive power.
- "Relationally complete" (without "strong") permits to use a sequence of queries and to store intermediate results in temporary relations.

Expressive Power (3)

- The following languages have the same expressive power (queries can be translated between them):
  - Relational algebra
  - SQL without aggregations and with mandatory duplicate elimination.
  - Tuple relational calculus (first order logic with variables for tuples, see below), Domain RC
  - Datalog (a Prolog variant) without recursion
- Thus, the set of functions that can be expressed in RA is at least not arbitrary.

Outline

Summary

The five basic operations of relational algebra are:

- σ_{φ}: Selection
- $\pi_{\mathtt{A}_1,\ldots,\mathtt{A}_k}$: Projection
- $\times$: Cartesian Product
- $\cup$: Union
- $\setminus$: Set Difference

Derived operations: The general join $\underset{\varphi}{\bowtie}$, the natural join $\bowtie$, the renaming operator ρ, the intersection $\cap$.

Bibliography

- The following list of references is compiled from the open source bibliography available at

  `https://github.com/krr-up/bibliography`

- Feel free to submit corrections via pull requests!

[1] S. Abiteboul, R. Hull, and V. Vianu.
Foundations of Databases.
Addison-Wesley, 1995.

[2] C. Aggarwal, editor.
Data Streams — Models and Algorithms, volume 31 of *Advances in Database Systems*.
Springer-Verlag, 2007.

[3] K. Apt, H. Blair, and A. Walker.
Towards a theory of declarative knowledge.
In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, chapter 2, pages 89–148. Morgan Kaufmann Publishers, 1987.

[4] M. Arenas, L. Bertossi, and J. Chomicki.
Consistent query answers in inconsistent databases.

In *Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'99)*, pages 68–79. ACM Press, 1999.

[5] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors.
The Description Logic Handbook: Theory, Implementation, and Applications.
Cambridge University Press, 2003.

[6] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom.
Models and issues in data stream systems.
In L. Popa, editor, *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'02)*, pages 1–16. ACM Press, 2002.

[7] C. Baral.
Knowledge Representation, Reasoning and Declarative Problem Solving.
Cambridge University Press, 2003.

[8] S. Ceri, G. Gottlob, and L. Tanca.
Logic Programming and Databases.
Springer-Verlag, 1990.

[9] R. Elmasri and S. Navathe.
Fundamentals of database systems.
Addison-Wesley, 1994.

[10] R. Fagin, J. Ullman, and M. Vardi.
On the semantics of updates in databases. preliminary report.
In *Proceedings of the Second ACM Conference SIGACT-SIGMOD*, pages 352–365, 1983.

[11] H. Gallaire, J. Minker, and J. Nicolas.
Logic and databases: A deductive approach.
Computing Surveys, 16(2):153–185, 1984.

[12] M. Gebser, R. Kaminski, B. Kaufmann, M. Lindauer, M. Ostrowski, J. Romero, T. Schaub, and S. Thiele.
Potassco User Guide.

University of Potsdam, 2 edition, 2015.

[13] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub.
Answer Set Solving in Practice.
Synthesis Lectures on Artificial Intelligence and Machine Learning.
Morgan and Claypool Publishers, 2012.

[14] M. Gelfond and Y. Kahl.
Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach.
Cambridge University Press, 2014.

[15] M. Gelfond and V. Lifschitz.
Classical negation in logic programs and disjunctive databases.
New Generation Computing, 9:365–385, 1991.

[16] H. Katsuno and A. Mendelzon.
On the difference between updating a knowledge database and revising it.

In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 387–394. Morgan Kaufmann Publishers, 1991.

[17] V. Lifschitz.
Closed-world databases and circumscription.
Artificial Intelligence, 27:229–235, 1985.

[18] V. Lifschitz.
Nonmonotonic databases and epistemic queries.
In J. Myopoulos and R. Reiter, editors, *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 381–386. Morgan Kaufmann Publishers, 1991.

[19] V. Lifschitz.
Introduction to answer set programming.
Unpublished draft, 2004.

[20] V. Lifschitz, F. van Harmelen, and B. Porter, editors.

Handbook of Knowledge Representation.
Elsevier Science, 2008.

[21] L. Liu and M. Özsu, editors.
Encyclopedia of Database Systems.
Springer-Verlag, 2009.

[22] V. Marek and M. Truszczyński.
Nonmonotonic logic: context-dependent reasoning.
Artifical Intelligence. Springer-Verlag, 1993.

[23] J. Minker, editor.
Foundations of Deductive Databases and Logic Programming.
Morgan Kaufmann Publishers, 1988.

[24] R. Reiter.
On closed world data bases.
In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 55–76. Plenum Press, New York, 1978.

[25] R. Reiter.

Towards a logical reconstruction of relational database theory.
In M. Brodie, J. Myopoulos, and J. Schmidt, editors, *On conceptual modeling: Perspectives from Artificial Intelligence, Datbases and Programming Languages*, pages 191–233. Springer-Verlag, 1984.

[26] R. Reiter.
On asking what a database knows.
In J. Lloyd, editor, *Computational Logic*, pages 96–113. Springer-Verlag, 1990.

[27] J. Ullman.
Principles of Database Systems.
Computer Science Press, Rockville MD, 1982.

[28] J. Ullman.
Principles of Database and Knowledge-Base Systems.
Computer Science Press, 1988.