Part 6: Relational Model I

Elmasri/Navathe:

- Chap. 7, The Relational Data Model, Relational Constraints, and the Relational Algebra
  (except 7.4.7 "The Division Operation", 7.5.1 "Aggregate Functions and Grouping", 7.5.2 "Recursive Closure Operations")

I have also used the following sources:

- Kemper/Eickler: Datenbanksysteme (in German), Ch. 2, Oldenbourg, 1997.
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.
- Silberschatz/Korth/Sudarshan: Database System Concepts, Third Edition, 1999. Chap. 3: Relational Model.
- Codd: A relational model for large shared data banks, CACM, 13(6):377–387, 1970.
  Reprinted in CACM 26(1), 64–69, 1983. See also: `http://www1.acm.org:81/classics/nov95/toc.html`

Objectives

- You should know who proposed the relational model.
- You should be able to explain the basic concepts of the relational model: What is a schema? What is a state for a given schema? What are domains?
- You should be able to enumerate and explain the operations of relational algebra.
- You should be able to write relational algebra queries of the type "join-select-project".
  Plus simple queries involving set difference and union.
- You should be able to discuss correctness and equivalence of relational algebra queries.

Overview

Importance of Relational Model

- Relational database management systems are currently dominant on the market.
- RDBMS-Products, are, e.g. Oracle, IBM DB2, Sybase, Informix, CA Ingres, MS SQL Server.
- Most new database projects use an RDBMS.
- There are still "legacy systems" which are based on a network or hierarchical DBMS.
- New object-oriented database systems are today used mainly for "non-standard applications" (e.g. CAD data).
  But ODBMS lost some of the advantages of the relational products. The current trend goes to object-relational DBMS. All big vendors (especially Informix) have some OR-features.

Relational Model: History

- The relational model was proposed by E. F. Codd (1970).
- First data model which was theoretically defined before it was implemented.
- First Implementations (1976):
  - System R (IBM)
  - Ingres (Stonebraker, UC Berkeley).
- First commercial systems: Oracle (1979), Ingres (1980?) and IBM SQL/DS (1981).
- Today “state of the art” in industry.

Relational Model: Reasons for Success

- Much simpler than earlier data models.
  Only one concept: Finite relation (set of tuples/records).
- Easily understandable: Relations correspond to tables.
- Abstraction of known “files of records”.
- The relational model has set-oriented operations. In earlier models, one had to navigate from one record to the next.
- Declarative query language:
  No need to think about efficient evaluation.
- The relational model has a solid theoretical foundation.
  It is tightly connected to first-order logic.

Overview

1. Relational Model: Importance, History
2. Relational Model Concepts (Schema, State)
3. Relational Algebra I: Selection, Projection
4. Relational Algebra II: Join
5. Relational Algebra III: Set Operations

Example of a Relational DB State

Composer			
CNo	CName	Born	Died
1	Vivaldi	1678	1741
2	Bach	1685	1750
3	Mozart	1756	1791
4	Prokofiev	1891	1953
5	Dvorak	1841	1904

Piece		
PNo	CNo	PName
1000	1	The Four Seasons
1002	2	B-minor Mass
1003	2	Christmas Oratorio
1004	3	Missa solemnis
1006	4	Classical Symphony
1007	4	Cinderella

Data Values (1)

- Table entries are data values taken from some given selection of data types.
- Usually, the possible data types are defined by the RDBMS which we are going to use (or the SQL-92 standard).
- E.g. strings, numbers (of different lengths and precisions), date and time, money, binary data.
- But the relational model itself is independent from any specific selection of data types.
- There are extensible database systems, which allow to define new data types (e.g. multimedia). Feature of ORDBMS.

  “Universal Server/DB”: can store more than numbers and strings (“all kinds of electronic information”).

Application-Specific “Domains” (1)

- Music piece identifiers and years of birth are both 4-digit numbers, but they are used with an entirely different meaning. E.g. it makes little sense to compare them.
- It has been proposed to define application-specific domains of values, e.g. “`domain Year = number(4);`”

Column	→	Domain	→	Data Type
“Born”		“Year”		“number(4)”

- Domains are “shorthands” for standard data types, but with a specific meaning, different from all other domains.

Application-Specific “Domains” (2)

- The SQL-92 standard contains domain definitions, but few database management systems support them.
- Domains are at least a useful comment to better understand the database (which connections exist between columns?).
- Even if the RDBMS does not support domains, they should be defined during DB design.

  Oracle Designer supports domains and replaces them by the defined types when it creates CREATE TABLE statements.
- Often, domain names can be used as attribute names.
- Domains are also advantageous when data types of attributes must later be changed (consistency of types).

Atomic Attribute Values

- The relational model treats the single table entries as atomic.
- I.e. structured and multi-valued attributes are forbidden.
- In contrast, the NF^2 (“Non First Normal Form”) data model allows table entries to be again complete tables:

Composer				
No	Name	Born	Died	Pieces
1	Vivaldi	1678	1741	The Four Seasons
2	Bach	1685	1750	B-minor Mass Christmas Oratorio

- Oracle8 (with “Objects” option) allows any PL/SQL type for the columns, including nested tables. Feature of ORDBMS.

Database Schemas

A relation schema s defines

- a (finite) sequence $A_1 \ldots A_n$ of attribute/column names, satisfying $A_i \neq A_j$ for $i \neq j$ (i.e. the names are distinct), and
- for each attribute A_i a data type (or domain) D_i.

  We write $dom(A_i)$ for the set of possible values of attribute A_i, i.e. $dom(A_i) := |D_i|$.

A relational database schema defines

- a finite set of relation names $\{R_1, \ldots, R_m\}$, and
- for every relation R_i, a relation schema $sch(R_i)$.

  The later defined integrity constraints are also part of the DB schema.

Relational DB Schemas: Notation (1)

There are many possibilities to denote relational schemas:

- Composer(CNo, CName, Born, Died)

  I normally use this concise notation.

- The header or sketch of the table:

Composer			
CNo	CName	Born	Died
⋮	⋮	⋮	⋮

- Composer(CNo: decimal(3), CName: char(40), ...)
- domain CNo = decimal(3), CName = . . . , Year = . . . ;
  relation Composer(CNo, CName, Born: Year, Died: Year);

Relational DB Schemas: Notation (2)

- A table with information about each attribute in one row:

Composer	
Column	Type
CNo	decimal(3)
CName	char(40)
⋮	⋮

- In SQL (see below):

```
CREATE TABLE Composer(CNo decimal(3),
                      CName char(40), ...)
```


Exercise

Define a relational schema for the administration of the homework results of this course. So we want to store information about

- Students: name and email-address,
- Exercises: number, topic, and maximal points, and
- Homework results: which student got how many points for which exercise?

If you should consider further data as necessary, please add it.

Which attributes should have the same domain?

Tuples

- An n-tuple (tuple for short) is a sequence of n values.
- E.g. XY-coordinates are pairs (X, Y) of real numbers. Pairs are tuples of length 2 ("2-tuples").
- The cartesian product $\times$ constructs sets of tuples, e.g.:

$$\mathbb{R} \times \mathbb{R} := \{(X, Y) \mid X \in \mathbb{R},\ Y \in \mathbb{R}\}.$$

- A tuple t for the relation schema $(A_1\colon D_1, \ldots, A_n\colon D_n)$ is a sequence $(x_1, \ldots, x_n)$ of n values with $x_i \in |D_i|$.

$$\text{I.e. } t \in |D_1| \times \cdots \times |D_n|.$$

- Given such a tuple, we write $t.A_i$ for the value x_i in the column corresponding to A_i. Alternative notation: $t[A_i]$.

Database States

Let a database schema $(\{R_1, \ldots, R_m\},\ \text{sch})$ be given.

- A database state $\mathcal{S}$ for this database schema defines for every relation R_i a finite set of tuples with respect to the relation schema $\text{sch}(R_i)$.
- I.e. if $\text{sch}(R_i) = (A_{i,1}\colon D_{i,1},\ \ldots,\ A_{i,n_i}\colon D_{i,n_i})$, then

$$\mathcal{S}(R_i) \subseteq |D_{i,1}| \times \cdots \times |D_{i,n_i}|.$$

Exercise:

- What are differences between such relations and, e.g., "$<$" (an order relation known from mathematics)?

  One can see "$<$" as $\{(X, Y) \in \mathbb{N} \times \mathbb{N} \mid X < Y\}$.

Consequences:

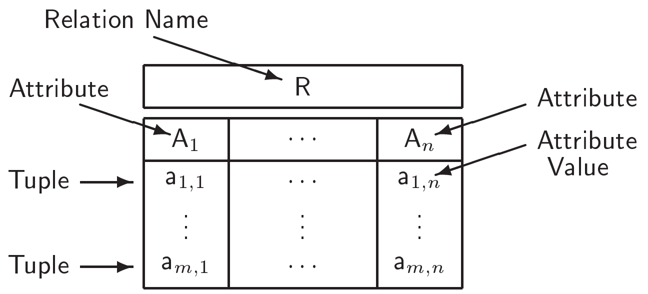
Relations are sets of tuples. Therefore:

- The sequence of the tuples is undefined.
  - So the tabular representation is a bit misleading, there is no first, second, etc. row.

    The file space management strategy defines where a new row is inserted (e.g. reuse space of deleted row).
  - Of course, relations can be sorted on output.
- There are no duplicate tuples.
  - Most current systems in fact allow duplicate tuples as long as no key is defined (see below).

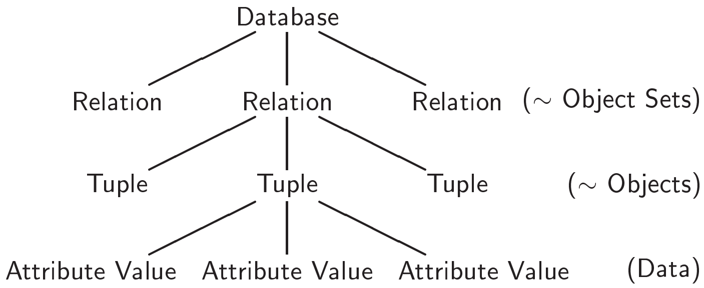
    So a formalization as a bag of tuples would be correct.

Summary (1)



Synonyms: Relation and table.
Tuple, row, and record.
Attribute, column, field.
Attribute value and table entry.

Summary (2)



Comparison with the ER-Model

- Relationships are relations between objects (entities), relations of the relational model relate data values.
- Relationships can be expressed in the relational model via equal attribute values.
- Relations are similar to entity types (tuples correspond to entities). But entities have an “object-identity”.

  There can be different entities with the same attribute values. An entity remains the same object even if you change all attributes.
- When keys are defined, and updates of the primary key are forbidden, entity sets really correspond to relations.

  But relationships are indirectly also expressed in relations.

Storage Structures

- Obviously, a relation can be stored as a file of records. But other data structures can offer a relational interface, too.

  The relational model does not require any specific storage structure. Tables are only the logical view. Other storage structures might allow one to answer certain queries more efficiently. E.g. the `V$*` tables in Oracle are an interface to data structures in the server.

Exercise:

- Define a relational interface to

  ```
  month_names:  array[1..12] of string;
  ```

  What are differences between this array and the standard “file of records” for your relation?

Update Operations

The three basic update operations of the relational model are:

- Insertion: $\mathcal{S}_{\text{new}}(\mathsf{R}) := \mathcal{S}_{\text{old}}(\mathsf{R}) \cup \{(\mathsf{a}_1, \ldots, \mathsf{a}_n)\}$
- Deletion: $\mathcal{S}_{\text{new}}(\mathsf{R}) := \mathcal{S}_{\text{old}}(\mathsf{R}) - \{(\mathsf{a}_1, \ldots, \mathsf{a}_n)\}$
- Modification: $\mathcal{S}_{\text{new}}(\mathsf{R}) := \mathcal{S}_{\text{old}}(\mathsf{R}) - \{(\mathsf{a}_1, \ldots, \mathsf{a}_i, \ldots, \mathsf{a}_n)\} \cup \{(\mathsf{a}_1, \ldots, \mathsf{a}'_i, \ldots, \mathsf{a}_n)\}$

  Modification is also called update (of a tuple).
  It corresponds to a deletion followed by an insertion, but without interrupting the “existence” of the tuple (which might be required by constraints).

Overview

1. Relational Model: Importance, History
2. Relational Model Concepts (Schema, State)
3. Relational Algebra I: Selection, Projection
4. Relational Algebra II: Join
5. Relational Algebra III: Set Operations

Relational Algebra (1)

- Relational algebra is a theoretical query language for the relational model.
- Whereas relational algebra is not used in any commerical system on the user interface level, it is used to represent queries internally (for query optimization and query execution).
- Relational algebra is much simpler than SQL, it has only five basic operations and can be completely defined on one page.
- Knowledge of relational algebra will help you to understand SQL and relational database systems better.

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 our case, the set is the set of all finite relations.
- A query is then a term/expression in this algebra.
- You know expressions like $(x + 2) * y$ connecting numbers.
- In relational algebra, we connect relations:

$$\pi_{\text{PName}}\big(\text{Piece} \bowtie \sigma_{\text{CName}=\text{'Prokofiev'}}(\text{Composer})\big).$$

Relational Algebra (3)

- The relational algebra is so simple because the relational model has only a single construct: The relation.
- So the result of a query is again a relation.
- This means we can combine queries by operations which take relations as inputs and yield relations.
- This also makes view definitions simple: A view is simply a shorthand for a query, and can be used everywhere in place of a stored relation (like a macro).
- For example, in the ER-model there is a mismatch between query results (which are typically also relations, since entities cannot be printed) and the concepts of the model.

Selection (1)

- The operation σ_φ selects a subset of the tuples of a relation, namely those, which satisfy the condition φ.

  σ is the greek letter sigma, φ is the greek letter phi.
  All textbooks use σ for the selection, but φ is not standard.
  In ASCII, write e.g. `SELECT[condition](Relation)`.

- Example:

$$\sigma_{\mathsf{A}=\mathsf{a}_1}\left(\begin{array}{|c|c|}\hline \mathsf{A} & \mathsf{B} \\ \hline \mathsf{a}_1 & \mathsf{b}_1 \\ \mathsf{a}_1 & \mathsf{b}_2 \\ \mathsf{a}_2 & \mathsf{b}_3 \\ \hline \end{array}\right) = \begin{array}{|c|c|}\hline \mathsf{A} & \mathsf{B} \\ \hline \mathsf{a}_1 & \mathsf{b}_1 \\ \mathsf{a}_1 & \mathsf{b}_2 \\ \hline \end{array}$$

Selection (2)

- The selection condition can have one of the following forms:
  - Attribute = Constant, e.g. CName = ’Prokofiev’.
  - Attribute 1 = Attribute 2, e.g. Country_of_Residence = Citizen_of.
  - Predicates of the data types can be used instead of $=$. E.g. numbers and strings support $<$, $\leq$, $\geq$, $>$, $\neq$. So Year > 1964 would be a valid condition.

    Strings also allow `LIKE` as in SQL or “substring_of” etc.
  - You may also use arithmetic expressions involving attributes, constants, and data type operators like $+$, $-$, $*$, $/$ on both sides of the comparison operators.

Selection (3)

- In addition, you can 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

- But the selection condition must be able to be evaluated for each input tuple in isolation.

  So you cannot use “exists” ($\exists$) and “for all” ($\forall$) and nested relational algebra queries in selection conditions.

Selection (4)

- The important point is that the selection condition refers to single tuples only.
- So selection acts like a filter:

  σ_φ(R):

  foreach tuple t **from** input relation R **do**
      Evaluate condition φ for tuple t;
      if true **then output** t;
  od;
- With other data structures (e.g. a B-tree index), it might be possible to compute σ_φ(R) without reading each tuple.

Selection (5)

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

$$\sigma_{\mathsf{C}=\mathsf{a}_1}\left(\begin{array}{|c|c|}\hline \mathsf{A} & \mathsf{B} \\ \hline \mathsf{a}_1 & \mathsf{b}_1 \\ \mathsf{a}_2 & \mathsf{b}_2 \\ \hline \end{array}\right) = \textbf{Error}$$

- The following is legal, but the selection is superfluous:

$$\sigma_{\mathsf{A}=\mathsf{A}}\left(\begin{array}{|c|c|}\hline \mathsf{A} & \mathsf{B} \\ \hline \mathsf{a}_1 & \mathsf{b}_1 \\ \mathsf{a}_2 & \mathsf{b}_2 \\ \hline \end{array}\right) = \begin{array}{|c|c|}\hline \mathsf{A} & \mathsf{B} \\ \hline \mathsf{a}_1 & \mathsf{b}_1 \\ \mathsf{a}_2 & \mathsf{b}_2 \\ \hline \end{array}$$

Selection (6)

Exercise:

Write relational algebra expressions for the following queries:

- Give me information about “Mozart”.
- Which componists were born in the 17th century?
- Which componists lived less than 50 years?
- List all compositions of composer number 2.

This refers to the schema on slide 6-8:

- Composer(CNo, CName, Born, Died)
- Piece(PNo, CNo, PName)

Projection (1)

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

  π is the greek letter “pi”. In databases it always means “projection”, and not 3.14.... If you must use ASCII characters, you can write `PROJECT[Columns](Relation)`.

- Example:

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

Projection (2)

- While σ selects certain rows from the input relation, and discards the others, π selects certain columns from the input relation, and discards the others.
- The projection $\pi_{\mathsf{A}_{i_1},\ldots,\mathsf{A}_{i_k}}(\mathsf{R})$ produces for each input tuple $(\mathsf{A}_1\colon \mathsf{d}_1, \ldots, \mathsf{A}_n\colon \mathsf{d}_n)$ an output tuple $(\mathsf{A}_{i_1}\colon \mathsf{d}_{i_1}, \ldots, \mathsf{A}_{i_k}\colon \mathsf{d}_{i_k})$.
- I.e. the attribute values are not changed, but only the explicitly mentioned attributes are retained. All other attributes are “projected away”.

  Note that “to project a column away” is database slang. Normally, things are projected onto or into something else.

Projection (3)

- In general, 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.

  DBMS use an explicit duplicate elimination when needed. But in theory, relations are sets.

- Example:

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

Projection (4)

- So the projection is a mapping, which is applied to every input tuple:

  $\pi_{\mathsf{A}_{i_1},\ldots,\mathsf{A}_{i_k}}(\mathsf{R})$:

  foreach tuple $\mathsf{t} = (\mathsf{A}_1\colon \mathsf{d}_1, \ldots, \mathsf{A}_n\colon \mathsf{d}_n)$ **in** R **do**

  Compute tuple $\mathsf{u} = (\mathsf{A}_{i_1}\colon \mathsf{d}_{i_1}, \ldots, \mathsf{A}_{i_k}\colon \mathsf{d}_{i_k})$;

  output u;

  od;

- This program fragment does not handle the possible duplicate elimination.

Projection (5)

- The mapping can also be more general:
  - First, we need a renaming of attributes:

    $\pi_{\mathsf{B}_1\leftarrow\mathsf{A}_{i_1},\ldots,\mathsf{B}_k\leftarrow\mathsf{A}_{i_k}}(\mathsf{R})$ transforms the input tuple $(\mathsf{A}_1\colon \mathsf{d}_1, \ldots, \mathsf{A}_n\colon \mathsf{d}_n)$ into $(\mathsf{B}_1\colon \mathsf{d}_{i_1}, \ldots, \mathsf{B}_k\colon \mathsf{d}_{i_k})$.
  - Second, we can also return values computed by datatype operations such as $+$, $-$, or string contatenation:

    $$\pi_{\mathsf{CName},\ \mathsf{Age}\leftarrow\mathsf{Died}-\mathsf{Born}}(\mathsf{Composer}).$$
  - Third, you can also create columns with constant values:

    $\pi_{\text{'Name:'},\mathsf{CName}}(\mathsf{Composer})$.

  So each input tuple is mapped locally to an output tuple. Only functions are allowed which are defined based on single input tuples.

Combining Operations (1)

- Since the result of a relational algebra operation is again a relation, it can be input to another algebra operation.
- For instance, we can compute the names of all composers born before 1700:

$$\pi_{\mathsf{CName}}\big(\sigma_{\mathsf{Born}\,<\,1700}(\mathsf{Composer})\big)$$

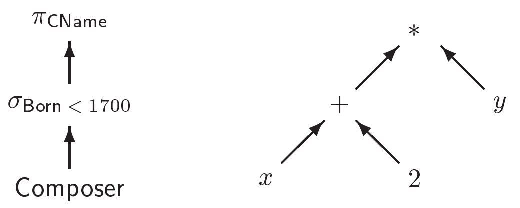
- If you like, you can also store an intermediate result in a temporary relation:

$$\mathsf{R} := \sigma_{\mathsf{Born}\,<\,1700}(\mathsf{Composer});$$

$$\pi_{\mathsf{CName}}(\mathsf{R})$$

Combining Operations (2)

- Often, expressions of relational algebra become clearer if we draw an operator tree:



- For comparison, an operator tree for the arithmetic expression $(x + 2) * y$ is shown on the right.
- Intermediate results flow along the lines from bottom to top.

Combining Operations (3)

Exercise:

- Please write a relational algebra expression for this query: What is the number of the composer who wrote “The Four Seasons”. Write it as a tree and nested with parenthesis.
- Which of the following algebra expressions are syntactically correct? What do they mean?
  - ☐ Composer.
  - ☐ $\sigma_{\text{Born} > 1700}(\text{Piece})$.
  - ☐ $\pi_{\text{CName}}\big(\pi_{\text{Born}}(\text{Composer})\big)$.
  - ☐ $\sigma_{\text{Born} < 1800}\big(\sigma_{\text{Born} \geq 1700}(\text{Composer})\big)$.
  - ☐ $\sigma_{\text{Born}}\big(\pi_{\text{Born} \geq 1700}(\text{Composer})\big)$.

Overview

1. Relational Model: Importance, History
2. Relational Model Concepts (Schema, State)
3. Relational Algebra I: Selection, Projection
4. Relational Algebra II: Join
5. Relational Algebra III: Set Operations

Cartesian Product (1)

- The operations σ and π were defined on single tuples.
- But sometimes we need to combine tuples in order to compute result tuples which are derived from two (or more) input tuples.
- This is done by the operation “cartesian product” ($\times$). R $\times$ S concatenates (“glues together”) each tuple from R with each tuple from S.
- 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 and then apply $\times$.

Cartesian Product (2)

- Example:

A	B
1	2
3	4

$\times$

C	D
6	7
8	9

$=$

A	B	C	D
1	2	6	7
1	2	8	9
3	4	6	7
3	4	8	9

- In ASCII, write "`R PRODUCT S`" or "`R x S`" for $\mathsf{R} \times \mathsf{S}$.

  If necessary, use `(...)` around the input relations.

Cartesian Product (3)

- The cartesian product can be computed by two nested loops:

  $\mathsf{R} \times \mathsf{S}$:

  foreach tuple $\mathsf{t} = (\mathsf{A}_1\colon \mathsf{d}_1, \ldots, \mathsf{A}_n\colon \mathsf{d}_n)$ **in** R **do**

      foreach tuple $\mathsf{u} = (\mathsf{B}_1\colon \mathsf{e}_1, \ldots, \mathsf{B}_m\colon \mathsf{e}_m)$ **in** S **do**

          output $\mathsf{t} \circ \mathsf{u} = (\mathsf{A}_1\colon \mathsf{d}_1, \ldots, \mathsf{A}_n\colon \mathsf{d}_n,$

                  $\mathsf{B}_1\colon \mathsf{e}_1, \ldots, \mathsf{B}_m\colon \mathsf{e}_m)$;

      od;

  od;

Cartesian Product (4)

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

Join (1)

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

  $$\mathsf{R} \underset{\mathsf{A=B}}{\bowtie} \mathsf{S} \text{ is a shorthand for } \sigma_{\mathsf{A=B}}(\mathsf{R} \times \mathsf{S}).$$

  The combined operation can often be executed more efficiently than first computing the complete cartesian product and then evaluating the selection.
- In ASCII write "`R JOIN[A=B] S`".
- This operation is called "join": it is used to join two tables.
- The join is one of the most important and useful operations of the relational algebra (immediately after the selection).

Join (2)

- $\mathsf{R} \underset{\mathsf{A}_i=\mathsf{B}_j}{\bowtie} \mathsf{S}$ can be evaluated similarly to $\sigma_{\mathsf{A}_i=\mathsf{B}_j}(\mathsf{R} \times \mathsf{S})$ but without materializing the intermediate result of $\times$:

  foreach tuple $t = (A_1\colon d_1, \ldots, A_n\colon d_n)$ **in** R **do**
      foreach tuple $u = (B_1\colon e_1, \ldots, B_m\colon e_m)$ **in** S **do**
          if $d_i = e_j$ **then**
              output $t \circ u = (A_1\colon d_1, \ldots, A_n\colon d_n,$
                  $B_1\colon e_1, \ldots, B_m\colon e_m)$;
          fi;
      od;
  od;

Join (3)

- The above procedure is called “the nested loop join”:
- There are other, more efficient procedures.
  E.g. “merge join”, “index join”, “hash join”.
- The join condition does not necessarily have to be of the form A = B (although this is the most common case).
  It can be an arbitrary condition, for instance also A < B.
  A join with condition A = B is called an “equijoin”.
- Note that the join not only combines tuples, but also acts as a filter: It eliminates tuples which do not have a join partner.
  A “semijoin” is a version of the join which is used only as a filter. An “outer join” (see below under Null values) does not work as a filter (preserves all input tuples).

Join (4)

- Another useful shorthand is the “natural join” $\bowtie$.
  If you don’t have the fancy symbol, you can use “*”.
- It combines tuples which have equal values in attributes with the same name.
  Whereas the cartesian product as well as the general join require that the attributes of both relations have distinct names, the natural join uses the equal names to derive a join condition.

A	B
1	2
3	4

$\bowtie$

B	C
4	5
4	8
6	7

$=$

A	B	C
3	4	5
3	4	8

Join (5)

- In general, the natural join of two relations $R(A_1, \ldots, A_n, B_1, \ldots, B_k)$ and $S(B_1, \ldots, B_k, C_1, \ldots, C_m)$ produces all tuples of the form $(a_1, \ldots, a_n, b_1, \ldots, b_k, c_1, \ldots, c_m)$ such that $(a_1, \ldots, a_n, b_1, \ldots, b_k) \in R$ and $(b_1, \ldots, b_k, c_1, \ldots, c_m) \in S$.
- So the natural join not only corresponds to a cartesian product followed by a selection, but also uses a projection to eliminate one copy of the double attributes.
- So given R(A, B), and S(B, C), then $R \bowtie S$ is a shorthand for

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

Join (6)

A Note on Relational Database Design:

- In order to support the natural join, it is good to give attributes from different relations, which are typically joined together, the same name.
- Even if your query language does not have a natural join, this is a good documentation.
- If you use the defined domain names as attribute names, this will happen automatically.
- Try to avoid giving the same name to attributes which you do not intend to join.

Renaming

- The input relations of the cartesian product and general joins (not the natural join) must have disjoint attribute names.
- Thus we introduce an operator $\rho_{\mathsf{S}}(\mathsf{R})$ which prepends S. to all attribute names:

$$\rho_{\mathsf{S}}\left(\begin{array}{|c|c|}\hline \mathsf{A} & \mathsf{B} \\ \hline \mathsf{a}_1 & \mathsf{b}_1 \\ \mathsf{a}_2 & \mathsf{b}_2 \\ \hline \end{array}\right) = \begin{array}{|c|c|}\hline \mathsf{S.A} & \mathsf{S.B} \\ \hline \mathsf{a}_1 & \mathsf{b}_1 \\ \mathsf{a}_2 & \mathsf{b}_2 \\ \hline \end{array}$$

- This is only a shorthand for the corresponding application of the projection to rename the attributes: $\pi_{\mathsf{S.A} \leftarrow \mathsf{A},\, \mathsf{S.B} \leftarrow \mathsf{B}}(\mathsf{R})$.
- Otherwise, attribute names in relational algebra do not automatically contain the relation name.

A Common Query Pattern (1)

- The following query structure is very common:

$$\pi_{\mathsf{A}_1,\ldots,\mathsf{A}_k}\big(\sigma_\varphi(\mathsf{R}_1 \bowtie \cdots \bowtie \mathsf{R}_n)\big).$$

  - First join all tables which you need to answer your query.
  - Second, select the relevant tuples. Because you have first joined the tables, the selection condition may refer to attributes from all relations.
  - Third, project on the attributes which you want to appear in the output.
- But relational algebra operations can be combined in any way. You do not have to adhere to this pattern.

A Common Query Pattern (2)

- So when you are going to formulate a query, think first about the relations which you need:
  - Usually, the natural language version of the query names certain attributes.
  - For each such attribute, you need at least one relation which contains this attribute (so that you can refer to it in the selection condition or the projection list).
  - However, sometimes you also need intermediate relations in order to make the join meaningful. For instance, if you have relations $\mathsf{R}(\mathsf{A},\mathsf{B})$, $\mathsf{S}(\mathsf{B},\mathsf{C})$, $\mathsf{T}(\mathsf{C},\mathsf{D})$, and you need attributes A and D, $\mathsf{R} \bowtie \mathsf{T}$ would not be correct. Why?

A Common Query Pattern (3)

Exercise:

- Compute the name of the composer of "The Four Seasons".
- Which are the names of music pieces from composers born before 1700?

Remark:

- Sometimes, you must refer to more than one tuple from one relation at the same time.
  This needs a generalization of the above query pattern, where two copies of a relation are joined (one must be renamed first).
- Which composer was born during the lifetime of Vivaldi?

Overview

1. Relational Model: Importance, History
2. Relational Model Concepts (Schema, State)
3. Relational Algebra I: Selection, Projection
4. Relational Algebra II: Join
5. Relational Algebra III: Set Operations

Set Operations (1)

- Since relations are sets (of tuples), the usual set operations $\cup$, $\cap$, $-$ can also be applied to relations.
- However, both input relations must have the same schema.
  For instance, it is not possible to take the union of two relations R(A) and S(B, C), because there is no common schema for the output relation.
- $R \cup S$ contains all tuples which are contained in R, in S, or in both relations (Union).
- $R - S$ contains all tuples which are contained in R, but not in S (Set Difference).

Set Operations (2)

- $R \cap S$ 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 $-$:

$$R \cap S = R - (R - S).$$

- Also, the generalized selection (with $\wedge$, $\vee$, $\neg$ in the condition) is derived: It can be composed from selections with only atomic formulas and $\cap$, $\cup$, and $-$.

Set Operations (3)

- With these set operations, we can also handle cases where the above σ, π, $\bowtie$ -queries are not sufficient:
  - Queries where the result depends on the non-existence of certain tuples: E.g., customers without orders, or the order for the maximal amount (i.e. there is no order for a higher amount), etc.

    In contrast, σ, π, $\bowtie$ -queries are monotonic: An answer tuple will still be contained in the result when more tuples are added to the database.
  - Queries where attribute values in the result originate from different tables (this needs $\cup$).

Set Operations (4)

Exercises:

- Which composers are in the database without any piece of theirs being stored? (Print their number.)
- What is the difference of your (correct) solution and the following solution:

$$\pi_{\mathrm{CName}}\Big(\sigma_{\mathrm{CNo}\neq\mathrm{X}}\big(\mathrm{Composer} \times \pi_{\mathrm{X}\leftarrow\mathrm{CNo}}(\mathrm{Piece})\big)\Big)$$

- Print the names of all composers without pieces.
- How could you write $\sigma_{\varphi_1 \vee \varphi_2}(\mathrm{R})$ without the $\vee$ (or)? Is it also possible to avoid $\wedge$ and $\neg$ in selection conditions?

Union vs. Join

- Two alternatives for storing the homework, midterm, and final results 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 the two.

Summary

The five basic operations of relational algebra are:

- σ_φ: Selection
- $\pi_{\mathrm{A}_1,\ldots,\mathrm{A}_k}$: Projection
- $\times$: Cartesian Product
- $\cup$: Union
- $-$: Set Difference

We have also discussed the general join $\bowtie\varphi$, the natural join $\bowtie$, the renaming operator ρ, and the intersection $\cap$. These operations are very useful, but they are not strictly necessary, since they can be simulated with the above basic operations.