1 Relational Databases
2 Relational Algebra and Calculus
3 Introduction to SQL
Terminological confusion

- database
- data bank
- information system
- data base
- database system
- database management system
- data model
- database model
Information system: computer science view

- Information system
- Database system
- + External media of communication
- + Application specific methods
example: Geo Information System (GIS)

satellite navigation system (GPS)

database system

virtual map interface
• Every **information system** uses a **database system** to manage its data.

• Thus, an information system is ‘more’ than a database system while the later has to fulfill the following main tasks:

  - schema management
  - query processing
  - transaction management
    - access protection
    - integrity control
    - multi-user synchronisation
    - recovery from errors
  - storage management
Intelligent Analysis of Data Streams

database system: notion

DBMS: database management system

users and application programs

DBMS: database management system
A helpful „formula“ for remembering the principle idea:

\[
\text{DBS} = \text{DBMS} + n \times \text{DB}
\]

„informal formula“:

database system

database management system
database

application independent services

application dependent information

one or many
Thus, the main tasks of a DBS …

- schema management
- query processing
- transaction management
  - access protection
  - integrity control
  - multi-user synchronization
  - recovery from errors
- storage management

… are realized by its DBMS.
• **information system** = database system +
  external media of communication +
  application specific methods

• **database system** DBS = DBMS + n* DB

• **database** DB = a set of data stored according to the concepts of the data model
  supported by the DBMS

• **database management system** DBMS = a system for controlling the access (reading and
  writing access) to the databases

• **data model** = a collection of concepts that determine how a database is structured and
  can be used (e.g., a relational model allows to structure data in "tables").
  The DBMS is responsible for the user being able to see data in structured
  form while the physical representation (in a file) remains hidden.
Relational databases: General remarks

- The DB-market is completely dominated by systems supporting the relational data model today.

- Leading (commercial) manufacturers of relational DB-products:
  
  - Oracle
  - Sybase
  - Microsoft (Access, SQL Server)
  - Postgres (Freeware)
  - IBM (DB2, Informix)
  - MySQL (Freeware)

- The notion "relational" is motivated by the mathematical concept of a relation. Relations in mathematics are sets of tuples.

- Relational databases are collections of one or more relations.

- In practice, relations can be visualized as tables, the rows of which are individual records of data with the same (homogeneous) field structure.

- In science, relational databases have a broad range of theoretical foundations.
• The idea to organize data in tables is quite old and pretty obvious.

• The idea to investigate this representation of data by means of the theory of relations is due to one man, who proposed this view at the end of the 1960s:

   **Edgar F. Codd**

• In 1970, he published his seminal paper

   "**A Relational Model of Data for Large Shared Data Banks**, in which he fixed all foundations of relational databases with amazing precision and clarity.

• Codd died in early 2003.
Europe.mdb is a small database for MS Access.

Just now it contains solely two tables: countries and cities in Europe.
Relational tables are grids, the fields of which are consisting of columns and rows. (There is a specific terminology for such tables in MS Access.)

<table>
<thead>
<tr>
<th>country</th>
<th>code</th>
<th>capital</th>
<th>area</th>
<th>population</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>Italy</td>
<td>I</td>
<td>Rome</td>
<td>301230</td>
<td>57460274</td>
<td>2001</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table name**: countries

- **Attributes**: (field names)
  - country
  - code
  - capital
  - area
  - population
  - year

**Tuple**: (record)

- **Attribute data type**
  - text
  - char
  - text
  - integer
  - integer
  - integer
  - integer
Unfortunately, the basic concepts of the relational model are denoted by different terms depending on the context. There are synonymous, but different terminologies in database theory, the standard DB language SQL and MS Access:

<table>
<thead>
<tr>
<th>theory</th>
<th>SQL</th>
<th>Access</th>
</tr>
</thead>
<tbody>
<tr>
<td>relation</td>
<td>table</td>
<td>datasheet</td>
</tr>
<tr>
<td>tuple</td>
<td>row</td>
<td>record</td>
</tr>
<tr>
<td>attribute</td>
<td>column</td>
<td>field name</td>
</tr>
<tr>
<td>domain</td>
<td>data type</td>
<td>field data type</td>
</tr>
</tbody>
</table>

Be warned of this „Babylonic confusion“ of terms – we urgently recommend that you always stick to a single system of notions in a consistent manner. It doesn‘t matter which system you use – but never mix them up !
Access table: datasheet view

Access table „countries“:

datasheet view
Access table „countries“:

design view

- field size
- format
- input mask
- caption
- default value
- validation rule
- validation text
- Required
- Allow zero length
- Indexed
- Unicode compression
The two different "views" of a table in Access correspond to two fundamental notions of relational databases:

- **Schema** of a relation: definition of name and structure of the relation
- **State** of a relation: all tuples currently contained in the relation
- The structure of each state of a relation is defined by its schema. (States are called **instances** of the schema.)
- In general, the schema remains **fixed** during state changes.
- Sometimes, however, there are schema modifications as well, followed by immediate state adaptations: **schema evolution**
- Plural of schema: **schemas** (not "schemes")!
Schemas and states

schema₁

states

instances of schema₁

schema₂

instances of schema₂

current state
There are two basic **forms of interaction** with a database:

**Reading access:** query mode

**Writing access:** update mode
Query languages

- A fundamental characteristic of each database management system is the support of one or more query language.

- A query is an expression in this language which:
  - is able to express arbitrarily complex search criteria.
  - refers to one or more tables simultaneously.
  - returns one or more records or simply yes/no as an answer.
  - returns records in form of answer tables.

- Every commercial DBMS understands the textual "Structured Query Language" (SQL).

- SQL is the most widely distributed query language for relational DBs.

- SQL is standardized by the American National Standards Institute (ANSI).
Retrieve name, capital and area of all countries larger than 100000 km² in descending order of size!

SQL

QBE
Retrieve name, capital and area of all countries larger than 100000 km² in descending order of size!

**Answers** to relational queries are always returned as tables, too.

Thus, they may be „reused“ as input for further queries.

However, these tables are not stored in the DB! They are „virtual“ tables recomputed each time the query is asked.
There are two basic **forms of interaction** with a database:

- **Reading access**: query mode
- **Writing access**: update mode
State changes: general principles

- "Write" access to a database . . .
  - . . . always results in a state change of the DB.
  - . . . always takes place under control of the DBMS.

- There are three basic forms of write access:
  - insertions of new records into a table
  - deletions of existing records from a table
  - modifications of the value of a particular field in a record of a table

- Insertions and modifications are accepted by the DBMS only if the data types of the resp. fields declared in the schema of the table fit with the values in the new/modified records.

- Caution! The English notion "update" is used in this context with two different meanings – be sure you understand which of them is actually meant:
  - as a synonym for modification
  - as a generalization comprising all three kinds of write access

- Update statements can be expressed in SQL, too.
Some examples of how to formulate update statements in SQL:

- Insert statement with **direct** reference to the rows to be inserted:

  ```sql
  INSERT INTO cities (City, Country, Population, Year)
  VALUES ('Bonn', 'D', 317000, 2008);
  ```

- Insert Statement with **indirect** identification of the rows to be inserted:

  ```sql
  INSERT INTO cities
  SELECT capital,population,year
  FROM countries
  WHERE population >= 120000;
  ```

- Update statement with

  ```sql
  UPDATE countries
  SET capital='Bonn'
  WHERE Code='D';
  ```
Integrity constraints and integrity checking

- Primary key definitions and validation rules are special examples of a very important general concept in database design:

- In general, an integrity constraint (constraint for short) is a logical condition to be satisfied by each state of the database at all times, i.e., integrity constraints are required to be invariably true during the lifetime of the database.

- In SQL, we will find a rather powerful language for expressing nearly arbitrary such conditions.

- Integrity constraint violations – likely to happen during DB modifications – are controlled automatically by the DBMS. Each insertion, deletion or update of a table is checked for possibly violating any constraint prior to the execution of the resp. modification:

- If integrity violations are detected, the DBMS either refuses to perform the desired modification or „repairs“ the semantic mistake causing the violation automatically, if possible.

- Key and validation rule violations cannot be „repaired“!
In SQL, most integrity constraints are defined within a `CREATE TABLE`-command:

```
CREATE TABLE Countries
(
  County text,
  Code char,
  Capital text,
  Area number(15) DEFAULT NULL
    CHECK ( Area > 0 OR IS NULL ),
  Population number(15) NOT NULL
    CHECK ( Population > 0 OR IS NULL ),
  YEAR date,
  PRIMARY KEY (Country),
  FOREIGN KEY (Capital) REFERENCES Cities
) ;
```
**Summary** of the notions/concepts you should know:

<table>
<thead>
<tr>
<th>data model</th>
<th>DB query</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB schema</td>
<td>query language</td>
</tr>
<tr>
<td>DB state</td>
<td>subquery</td>
</tr>
<tr>
<td>relation (table,datasheet)</td>
<td>integrity constraint</td>
</tr>
<tr>
<td>attribute (column,field)</td>
<td>check constraint</td>
</tr>
<tr>
<td>tuple (row,record)</td>
<td>primary key</td>
</tr>
<tr>
<td>domain/(field) data type</td>
<td>foreign key</td>
</tr>
<tr>
<td>null value</td>
<td>referential integrity</td>
</tr>
<tr>
<td>default value</td>
<td></td>
</tr>
<tr>
<td>relationship</td>
<td></td>
</tr>
</tbody>
</table>
Understanding SQL

SQL is based on the languages RA and TRC which are essential for understanding the Semantics of SQL expressions:

\[
\begin{align*}
\text{SELECT} & \quad \text{Capital, Inhabitants} \\
\text{FROM} & \quad \text{city, country} \\
\text{WHERE} & \quad \text{Inhabitants} \geq 1000 \quad \text{AND} \quad \text{Name} = \text{Capital} \\
\end{align*}
\]

RA

\[
\pi_{\text{Capital, Inhabitants}} \left( \sigma_{\text{Inhabitants} \geq 1000 \land \text{Name} = \text{Capital}} \left( \text{city} \times \text{country} \right) \right)
\]

TRC

\[
\{ [y.\text{Capital, x.Inhabitants}] \mid \\
\text{city}(x) \land \text{country}(y) \land \\
x.\text{Inhabitants} \geq 1000 \land x.\text{Name} = y.\text{Capital} \}
\]
1 Relational Databases
2 **Relational Algebra and Calculus**
3 Introduction to SQL
Why additional languages?

• Query languages for databases, such as SQL and Datalog – to be introduced in next chapters – are formal languages, relying on a rigorously defined syntax and semantics.

• Furthermore, we will introduce two variants of logics and set theory, resp., tailored particularly for the manipulation of relations, being special sets and thus requiring special operators and special syntax:
  - Relational algebra is the basis of relational query processing.
  - Relational calculus is the logical counterpart.

• SQL is based on both, relational algebra as well as relational calculus.

• Datalog is purely based on relational calculus (domain calculus).
Relational Algebra

\[ \pi \]

\[ \sigma \]

\[ \emptyset \]

\[ \times \]

\[ \cup \]
- The (mathematical) concept of a *set* is of fundamental importance for almost every area of computer science.

“A set is a collection into a whole of definite, distinct objects of our perception or our thought.”

Georg Cantor (1845-1918), originator of set theory

- The order and number of occurrences of a member in a set is irrelevant (in contrast to SQL):

\[
\begin{align*}
\{2, 4, 6, 8\} & \\
\{6, 4, 8, 2\} & \\
\{2, 2, 4, 8, 8, 8, 6\} &
\end{align*}
\]

three denotations of the same set
In set theory, there are three basic operations by which two sets can be combined:

- **Union**
  \[ A \cup B = \{ e \mid e \in A \text{ or } e \in B \} \]

- **Intersection**
  \[ A \cap B = \{ e \mid e \in A \text{ and } e \in B \} \]

- **Difference**
  \[ A \setminus B = \{ e \mid e \in A \text{ and } e \notin B \} \]

**Examples:**

\[ \{1, 2\} \cup \{2, 3\} = \{1, 2, 3\} \]
\[ \{1, 2\} \cap \{2, 3\} = \{2\} \]
\[ \{1, 2\} \setminus \{2, 3\} = \{1\} \]
Cartesian product and tuples

- more binary basic operations of set theory:

\[ A \times B = \{ (a, b) \mid a \in A \text{ und } b \in B \} \]

(Cartesian) product

- generalized construction of products for \( n \) sets (\( n \geq 2 \)):

\[ A_1 \times \ldots \times A_n = \{ (a_1, \ldots, a_n) \mid a_i \in A_i \} \]

- The members of a product of \( n \) sets are called \((n)\)-tuples.

- special denominations for tuples:
  - \( n = 1 \): singleton
  - \( n = 2 \): pair
  - \( n = 3 \): triple
  - \( n = 4 \): quadruple
  - \( n = 5 \): quintuple
Example of a product of two sets

$A \times B$

$\{(\bullet, 1), (\circ, 1), (\bullet, 1), (\circ, 1), (\bullet, 1), (\circ, 1), (\bullet, 5), (\circ, 5), (\bullet, 5), (\circ, 5), (\bullet, 5), (\circ, 5), (\bullet, 7), (\circ, 7), (\bullet, 7), (\circ, 7)\}$
In computer science, sets of tuples are very important for modeling relationships between objects.

Sets of tuples are called relations.

Relations are usually denoted in the form of a table.

Every subset $R$ of a product $A_1 \times \ldots \times A_n$ is called a relation over $A_1, \ldots, A_n$.

Relations are usually denoted in the form of a table.

### Example

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${a, b, c}$</td>
<td>${1, 2}$</td>
<td>${%, $}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$R$</th>
<th>$a$</th>
<th>1</th>
<th>$%$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$a$</td>
<td>1</td>
<td>$$</td>
</tr>
<tr>
<td></td>
<td>$b$</td>
<td>2</td>
<td>$$</td>
</tr>
</tbody>
</table>
Relational algebra: Overview

- We will see that the operators of set theory are a good basis for manipulating relations (as they are sets indeed), but that they have deficiencies and thus have to be amended and extended.

- Already in his seminal paper introducing relational databases Codd introduced a choice of operators particularly tailored for dealing with relations. This was the basis of the formal language called the relational algebra today.

- Relational algebra is a mathematical language and thus not particularly user-friendly. But its operators have been incorporated into most of the query languages for relational databases in use today (e.g., in SQL). Thus, it is important to know about them.

- Moreover, relational algebra is used internally by a DBMS for evaluating queries written in SQL (or other languages). SQL queries are compiled into relational algebra expressions and then transformed into equivalent formulations which can be evaluated more efficiently ("query optimization").
What is an algebra?

- An algebra is a system of operators manipulating objects in a particular carrier set, i.e.
  - all input parameters are taken from this set, and
  - the result after applying the operators is contained in the carrier as well.

- consequence: Operators can be applied to results of previous operator applications, i.e., nesting of operators is possible.

- e.g.: arithmetic (numbers, + / * / −),
  propositional logic (truth values, or / and / not ),
  set algebra (power set of a set, ∪ / ∩ / −)

- The relational algebra (RA) is a special variant of set algebra, the carrier set of which consists in particular of relations rather than arbitrary objects. Arguments of RA-operators as well as their results are relations.
Set operators for relations (1)

- Relations are (special) sets, and thus operators of set algebra are applicable to relations, too:

<table>
<thead>
<tr>
<th>Union</th>
<th>R ∪ S</th>
<th>Difference</th>
<th>R − S</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intersection</td>
<td>R ∩ S</td>
<td>Product</td>
<td>R × S</td>
</tr>
</tbody>
</table>

- Intersection can be expressed via difference:
  \[ R \cap S = R - (R - S) \]

- Attention! Even if all input parameters of one of these operators are relations, it is by no means guaranteed that the results are relations, too. It may well be that applying a set operator to relations returns „just“ an ordinary set, but not a relation!

- Thus, not every application of set operators in RA is defined!
Set operators for relations (2)

Union of two **inhomogeneous** relations . . .

\[ R \cup S \]

\[ \begin{array}{ccc}
1 & 2 & 3 \\
2 & 4 & 5 \\
3 & 6 & 9 \\
\end{array} \quad \begin{array}{cc}
a & b \\
c & d \\
x & y \\
\end{array} \]

. . . results in a set, . . .

- (1,2,3)
- (a,b)
- (2,4,5)
- (c,d)
- (3,6,9)
- (x,y)

. . . but not in a relation!
• Only „similar“ relations can be united, intersected or subtracted. For product, however, similarity is not required.

• Relations the union of which is a relation again, are called union compatible.

• „Similarity“ of relations can be defined in various ways, a minimal requirement being
  • identical arity
  • identity of types of all columns.

• In addition, identity of names of all columns in both relations is often required.

• Identity of names can be reached by systematic renaming of columns. RA has an „auxiliary“ operator $\rho$ (griech. rho) for denoting renamings, e.g.:

$$\rho_{A \leftarrow B}(R)$$

[In relation R, column A is renamed into B.]
• In set theory, the product of two relations is always a binary relation, the elements of which are pairs of tuples.

\[ A \times B = \{ (a, b) \mid a \in A \text{ and } b \in B \} \]

• If e.g. tuple \((a,b)\) is an element of the binary relation \(A\) and tuple \((1,2,3)\) is an element of the ternary relation, then the product of \(A\) and \(B\) contains the pair \((a,b,1,2,3)\).

• In relational algebra, however, the product operator is defined in a slightly (but distinctively) different manner: Tuples from both operand relations are concatenated into a single tuple before being entered into the product relation:

\(\text{(a,b)} \rightarrow \text{(a,b,1,2,3)}\)

\(\text{(1,2,3)} \rightarrow \text{(a,b,1,2,3)}\)

• Thus, in RA the product of an \(n\)-ary and an \(m\)-ary relation is an \((n+m)\)-ary (but not a binary relation)!

© 2013 Manthey, Behrend
Intelligent Analysis of Data Streams
Renaming while building products

- If constructing the product of two relations, renaming of columns may be necessary in order to ensure that all columns of the result relation have different names.

- In order to resolve ambiguities, one often uses the name of the origin relation of a column as prefix for attributes in the result relation:
Special operators for sets of tuples

- In addition to the set operators (adapted to relations) RA offers another selection of special operators, defined for sets of tuples only.

- **Basic operators** for tuple sets (i.e. relations) are two unary operators for extracting . . .
  - ... certain columns:
    - projection
    - $\pi$
  - ... certain rows (tuples):
    - selection
    - $\sigma$

- Apart from these, there are various **derived operators** based upon projection and selection (in combination with set operators):
  - The various forms of the **join operator** are variants of product:
    - inner/outer join, natural join
  - A very special form of difference is very helpful for formalising certain variants of universal quantification in set theory: **division**
The projection operator $\pi$ ,,officially“ has one relational parameter only, but additionally needs one or more columns of the operand relation as a kind of ,,auxiliary parameters“ indicating on which columns to project.

In principle, one ought to say that there are very many different projection operators instead of just one: Per combination of columns on which to project there should be one such operator. For simplicity‘s sake, however, a ,,compromise notation“ is used:

All columns not appearing as an index of $\pi$ are eliminated by projection.
Projection and duplicate elimination

- While applying projection it may happen that the result relation contains duplicates – tuples occurring more than once..

- This may even be the case if the input relation itself was free of duplicates (which ought to be the case for each proper relation, as a set, anyway!).

\[
\pi_{A,B}:
\begin{array}{ccc}
A & B & C \\
1 & 2 & 3 \\
2 & 3 & 4 \\
1 & 2 & 5 \\
\end{array}
\rightarrow
\begin{array}{cc}
A & B \\
1 & 2 \\
2 & 3 \\
\end{array}
\]

- In order to be able to return a relation again, it may be necessary to eliminate duplicates (which may be an expensive task for large relations).

- Projection and union are the only basic operations of RA requiring duplicate elimination.
The selection operator $\sigma$ – which is unary in principle, too – needs an „auxiliary parameter“ as well.

A condition in the syntax of propositional logic, composed of comparisons of column values, called selection condition is added to $\sigma$. All tuples of the input relation not satisfying this condition are eliminated.

Selection conditions consist of column names of $R$, constants, comparison operators $(\equiv, \neq, <, \leq, >, \geq)$ and logical connectives $(\land, \lor, \neg)$.
Example for using the selection operator:

Find all tuples in relation $R$, the $A$-field of which is bigger than the $C$-field, and the $B$-field of which is not 'b'!

Formulation of this query in relational algebra:

$$\sigma_{A > C \land B \neq 'b'}(R)$$
The "full" product of two relations is not very useful in most situations. Very often a product is immediately reduced by eliminating rows and columns.

The most frequently used such variant results in two tables being connected via one or more of their columns on identical values in each of these columns, e.g.:

\[
\begin{array}{ccc}
A & B & C \\
1 & 2 & 3 \\
\end{array}
\]  \hspace{1cm}  \begin{array}{ccc}
B & C & D \\
2 & 3 & 4 \\
2 & 3 & 5 \\
\end{array}
\]

The most natural way of building a table containing all such "connections" is by looking for identical values in columns with identical name and then to concatenate the tuples thus linked (similarly to building a product relation). Due to the identical values, however, it is sufficient to keep only one copy of the joined columns (rather than two as in a product):
Example of a natural join:

Instead of the 9 tuples of a full product, only 3 "meaningful" combinations of tuples are kept!

### student

<table>
<thead>
<tr>
<th>MatrNr</th>
<th>Name</th>
<th>Semester</th>
</tr>
</thead>
<tbody>
<tr>
<td>26120</td>
<td>John</td>
<td>10</td>
</tr>
<tr>
<td>27550</td>
<td>Eve</td>
<td>12</td>
</tr>
<tr>
<td>28117</td>
<td>Bill</td>
<td>27</td>
</tr>
</tbody>
</table>

### course

<table>
<thead>
<tr>
<th>MatrNr</th>
<th>CourseNr</th>
</tr>
</thead>
<tbody>
<tr>
<td>26120</td>
<td>5001</td>
</tr>
<tr>
<td>27550</td>
<td>5001</td>
</tr>
<tr>
<td>27550</td>
<td>4052</td>
</tr>
</tbody>
</table>

**common join column**

[Diagram showing the natural join operation and the resulting table]
• The natural join is a derived operator in RA as its effect could as well be reached by combining projection, selection and product:

\[ R \bowtie S = \pi_{A_1, \ldots, A_m, R.B_1, \ldots, R.B_k, C_1, \ldots, C_n} (\sigma_{R.B_1 = S.B_1 \land \ldots \land R.B_k = S.B_k} (R \times S)) \]

<table>
<thead>
<tr>
<th>attr(R) – attr(S)</th>
<th>attr(R) \cap attr(S)</th>
<th>attr(S) – attr(R)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A_1 \ldots A_m</td>
<td>B_1 \ldots B_k</td>
<td>C_1 \ldots C_n</td>
</tr>
</tbody>
</table>

• Here, attr(R) denotes the set of all column names (attributes) in R.

• Obviously, using the special operator results in much higher readability of the expression.
Inner join

- It is not always clear that concatenation of tuples based on identity is indeed intended. For expressing explicit join conditions there is the inner join:
  - no automatic selection of tuples with identical fields
  - no automatic projection on "relevant" columns

- Example of an inner join: \( R \landmit S \mid \Theta = (R.A \leq S.C \land S.B > 0) \)

- The join condition \( \Theta \) is syntactically constructed like a selection condition.
- In Access, only the inner join is supported (no natural join!).
There is an **outer join** as well, which extends the inner join by maintaining the information about non-matching tuples in both input relations by „joining“ them with special „null values“ representing the fact that there is no match.

**Example** of an outer join: \( R \bowtie_{\Theta} S \) mit \( \Theta = (R.A \leq S.C \land S.B > 0) \)

If only non-matching tuples from one of the partner relations are to be filled up with Null, **left or right outer join** is to be used: \( \leftarrow \) resp. \( \rightarrow \)
• Most sophisticated, but also quite useful operator of RA: **division**

• **formal notation** like in arithmetics: $R \div S$

• **general idea**: algebraic counterpart to universal quantification in logic (for all)

• **principle** of division: Which A-values appear in R combined with all S-tuples?

\[
\begin{array}{c|c}
R & A & B \\
\hline
a & 1 \\
\hline
a & 2 \\
\hline
a & 3 \\
\hline
b & 2 \\
\hline
b & 3
\end{array}
\quad
\begin{array}{c|c}
S & B \\
\hline
1 & 2
\end{array}
\quad
\begin{array}{c|c}
R \div S & A \\
\hline
a
\end{array}
\]

Only 'a' appears in R combined with all S-tuples!

• **precise definition** of division (nearly unreadable for normal users again):

\[
R \div S := \pi_{\text{attr}(R)} - \text{attr}(S)(R) - \pi_{\text{attr}(R)} - \text{attr}(S)((\pi_{\text{attr}(R)} - \text{attr}(S)(R) \times S) - R)
\]
### Relational algebra: Summary

- The following operators are comprised by the relational algebra:

<table>
<thead>
<tr>
<th>base operators</th>
<th>derivable operators</th>
</tr>
</thead>
<tbody>
<tr>
<td>union</td>
<td>intersection</td>
</tr>
<tr>
<td>difference</td>
<td>join</td>
</tr>
<tr>
<td>product</td>
<td>division</td>
</tr>
<tr>
<td>projection</td>
<td></td>
</tr>
<tr>
<td>selection</td>
<td></td>
</tr>
</tbody>
</table>

- Query languages able to express at least (the effect of) each RA operator are called **relationally complete**. Thus, RA serves as a measure for the expressive power of DB query languages.

- SQL – to be presented in the next chapter – is a relationally complete language, but exceeds RA in expressivity.
The schema of a university database:

- **Lectures**: `{[LecID: integer, Title: string, Credits: integer, Held_By: integer]}`
- **Professors**: `{[PersID: integer, Name: string, Position: string, Room: integer]}`
- **Assistents**: `{[PersID: integer, Name: string, Research: string, ProfID: integer]}`
- **Students**: `{[StudID: integer, Name: string, Semester: integer]}`
- **attended_by**: `{[StudID: integer, LecID: integer]}`
- **PreLecture**: `{[PredID: integer, SuccID: integer]}`
Schema State

<table>
<thead>
<tr>
<th>attended_by</th>
<th>StudID</th>
<th>LecID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>26120</td>
<td>4001</td>
</tr>
<tr>
<td></td>
<td>27550</td>
<td>4001</td>
</tr>
<tr>
<td></td>
<td>27550</td>
<td>4002</td>
</tr>
<tr>
<td></td>
<td>28106</td>
<td>4001</td>
</tr>
<tr>
<td></td>
<td>28106</td>
<td>4002</td>
</tr>
<tr>
<td></td>
<td>28106</td>
<td>4003</td>
</tr>
<tr>
<td></td>
<td>29120</td>
<td>4001</td>
</tr>
<tr>
<td></td>
<td>29120</td>
<td>4002</td>
</tr>
<tr>
<td></td>
<td>29555</td>
<td>4003</td>
</tr>
<tr>
<td></td>
<td>30112</td>
<td>4001</td>
</tr>
<tr>
<td></td>
<td>30112</td>
<td>4002</td>
</tr>
<tr>
<td></td>
<td>30112</td>
<td>4003</td>
</tr>
<tr>
<td></td>
<td>31403</td>
<td>4002</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Students</th>
<th>StudID</th>
<th>Name</th>
<th>Semester</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>24002</td>
<td>Xenokrates</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>25403</td>
<td>Jonas</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>26120</td>
<td>Fichte</td>
<td>10</td>
</tr>
<tr>
<td></td>
<td>26830</td>
<td>Aristoxenos</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>27550</td>
<td>Schopenhauer</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>28106</td>
<td>Carnap</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>29120</td>
<td>Theophrastos</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>29555</td>
<td>Feuerbach</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lectures</th>
<th>LecID</th>
<th>Title</th>
<th>Credits</th>
<th>Held_By</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4001</td>
<td>logic</td>
<td>4</td>
<td>2125</td>
</tr>
<tr>
<td></td>
<td>4002</td>
<td>knowledge theory</td>
<td>3</td>
<td>2126</td>
</tr>
<tr>
<td></td>
<td>4003</td>
<td>database systems</td>
<td>4</td>
<td>2137</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Professors</th>
<th>PersID</th>
<th>Name</th>
<th>Position</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2125</td>
<td>Sokrates</td>
<td>C4</td>
<td>226</td>
</tr>
<tr>
<td></td>
<td>2126</td>
<td>Russel</td>
<td>C4</td>
<td>232</td>
</tr>
<tr>
<td></td>
<td>2127</td>
<td>Kopernikus</td>
<td>C3</td>
<td>310</td>
</tr>
<tr>
<td></td>
<td>2133</td>
<td>Popper</td>
<td>C3</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>2134</td>
<td>Augustinus</td>
<td>C3</td>
<td>309</td>
</tr>
<tr>
<td></td>
<td>2136</td>
<td>Curie</td>
<td>C4</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>2137</td>
<td>Kant</td>
<td>C4</td>
<td>7</td>
</tr>
</tbody>
</table>
Possible queries:

1. Which students have been studying for more than 9 semesters?
2. What position may be occupied by a professor?
3. Which professors and assistants exists in the database?
4. Which student under those who attend all the lectures in this semester has the smallest student ID?
5. ....
Relational algebra: Example

Which students have already studied more than 9 semesters?

\[ \sigma_{\text{Semester} > 9} (\text{Students}) \]

<table>
<thead>
<tr>
<th>StudID</th>
<th>Name</th>
<th>Semester</th>
</tr>
</thead>
<tbody>
<tr>
<td>24002</td>
<td>Xenokrates</td>
<td>18</td>
</tr>
<tr>
<td>25403</td>
<td>Jonas</td>
<td>12</td>
</tr>
<tr>
<td>26120</td>
<td>Fichte</td>
<td>10</td>
</tr>
</tbody>
</table>
What position may be occupied by a professor?

\[ \pi_{\text{Position}} (\text{Professors}) \]

<table>
<thead>
<tr>
<th>Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>C3</td>
</tr>
<tr>
<td>C4</td>
</tr>
</tbody>
</table>

Duplicates are eliminated!
Relational algebra: Example

Which professors and assistants exists in the database?

\[ \pi_{Name}(Professors) \cup \pi_{Name}(Assistants) \]

<table>
<thead>
<tr>
<th>Name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Sokrates</td>
<td></td>
</tr>
<tr>
<td>Russel</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Kant</td>
<td></td>
</tr>
<tr>
<td>Platon</td>
<td></td>
</tr>
<tr>
<td>Aristoteles</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Spinoza</td>
<td></td>
</tr>
</tbody>
</table>
What lectures are attended by which students?

<table>
<thead>
<tr>
<th>Students</th>
<th>attended_by</th>
</tr>
</thead>
<tbody>
<tr>
<td>StudID</td>
<td>LecID</td>
</tr>
<tr>
<td>24002</td>
<td>4001</td>
</tr>
<tr>
<td>25403</td>
<td>4001</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>LecID</td>
</tr>
<tr>
<td>4001</td>
</tr>
<tr>
<td>4002</td>
</tr>
<tr>
<td>4003</td>
</tr>
</tbody>
</table>
### Relational algebra: Example

What lectures are attended by which students?

<table>
<thead>
<tr>
<th>StudID</th>
<th>Name</th>
<th>Semester</th>
<th>LecID</th>
<th>Title</th>
<th>Credits</th>
<th>Held_By</th>
</tr>
</thead>
<tbody>
<tr>
<td>27550</td>
<td>Schopenhauer</td>
<td>6</td>
<td>4001</td>
<td>logic</td>
<td>4</td>
<td>2125</td>
</tr>
<tr>
<td>27550</td>
<td>Schopenhauer</td>
<td>6</td>
<td>4002</td>
<td>knowledge theory</td>
<td>3</td>
<td>2126</td>
</tr>
<tr>
<td>28106</td>
<td>Carnap</td>
<td>3</td>
<td>4001</td>
<td>logic</td>
<td>4</td>
<td>2125</td>
</tr>
<tr>
<td>28106</td>
<td>Carnap</td>
<td>3</td>
<td>4002</td>
<td>knowledge theory</td>
<td>3</td>
<td>2126</td>
</tr>
<tr>
<td>28106</td>
<td>Carnap</td>
<td>3</td>
<td>4003</td>
<td>database systems</td>
<td>4</td>
<td>2137</td>
</tr>
<tr>
<td>29120</td>
<td>Theophrastos</td>
<td>2</td>
<td>4001</td>
<td>logic</td>
<td>4</td>
<td>2125</td>
</tr>
<tr>
<td>29120</td>
<td>Theophrastos</td>
<td>2</td>
<td>4002</td>
<td>knowledge theory</td>
<td>3</td>
<td>2126</td>
</tr>
<tr>
<td>29120</td>
<td>Theophrastos</td>
<td>2</td>
<td>4003</td>
<td>database systems</td>
<td>4</td>
<td>2137</td>
</tr>
<tr>
<td>29555</td>
<td>Feuerbach</td>
<td>2</td>
<td>4003</td>
<td>database systems</td>
<td>4</td>
<td>2137</td>
</tr>
</tbody>
</table>
Relational algebra: Example

What is the difference to the result of the following expression?

\[ \sigma_{\text{Students.StudID} = \text{attended_by.StudID} \land \text{attended_by.LecID} = \text{Lectures.LecID}} \left( \text{Students} \times \text{attended_by} \times \text{Lectures} \right) \]
1st problem: How to find those students who attend all lectures?

• Which relations are needed for this subquery?
  \[ \Rightarrow \text{attended}_\text{by}: \quad \{[\text{StudID}: \text{integer}, \text{LecID}: \text{integer}]\} \]
  \[ \Rightarrow \text{Lectures}: \quad \{[\text{LecID}: \text{integer}, \text{Title}: \text{string}, \text{Credits}: \text{integer}, \text{Held}_\text{by}: \text{integer}]\} \]

• Which relational algebra operators are needed?
  \[ \Rightarrow \text{Division} \div \quad \text{(universal quantifier)}, \]
  \[ \Rightarrow \text{Projection} \pi \quad \text{(to find all LecIDs)} \]

\[ \text{omnistud} = \text{attended}_\text{by} \div \pi_{\text{LecID}}(\text{Lectures}) \]
2nd problem: How to find now the student with the smallest ID?

- Join the omnistuds relation with itself:

$$R2 = \text{Omnistuds} \bowtie_{\text{StudID} > \text{ID}_1} \rho_{\text{StudID} \leftarrow \text{ID}_1}(\text{Omnistuds})$$

- Determine now the StudID for which no join partner (with MatrNr > Nr1) could be found:

$$\pi_{\text{StudID}}(\text{Omnistuds}) - \pi_{\text{StudID}}(R2)$$

In this way, the min- and max-function can be simulated but not the aggregate functions sum, avg, etc…!
Which student under those who attend all the lectures in this semester has the smallest student ID?

**last step:** put all sub-expressions together for getting the entire query:

\[
\pi_{\text{StudID}}(\text{attended\_by} \div \pi_{\text{LecID}}(\text{Lectures})) - \\
\pi_{\text{StudID}}((\text{attended\_by} \div \pi_{\text{LecID}}(\text{Lectures})) \bowtie_{\text{StudID} > \text{ID1}} \\
\rho_{\text{StudID} < \text{ID1}}(\text{attended\_by} \div \pi_{\text{LecID}}(\text{Lectures}))))
\]

How to evaluate such a complex relational algebra query? 
⇒ first the most inner expression!

Potential for further optimization by evaluating common subexpressions only once!
1) Determination of the auxiliary relation Omnistuds:

$$\pi_{\text{LecID}}(\text{Lectures})$$

<table>
<thead>
<tr>
<th>LecID</th>
<th>Title</th>
<th>Credits</th>
<th>Held_by</th>
</tr>
</thead>
<tbody>
<tr>
<td>4001</td>
<td>logic</td>
<td>4</td>
<td>2125</td>
</tr>
<tr>
<td>4002</td>
<td>knowledge theory</td>
<td>3</td>
<td>2126</td>
</tr>
<tr>
<td>4003</td>
<td>database systems</td>
<td>4</td>
<td>2137</td>
</tr>
</tbody>
</table>

The auxiliary relations are determined as intermediate results, but the relation name is no longer used.
Determination of the auxiliary relation Omnistuds:

\[
\text{attended_by} \div \pi_{\text{LecID}}(\text{Lectures})
\]

<table>
<thead>
<tr>
<th>StudID</th>
<th>LecID</th>
</tr>
</thead>
<tbody>
<tr>
<td>26120</td>
<td>4001</td>
</tr>
<tr>
<td>27550</td>
<td>4001</td>
</tr>
<tr>
<td>27550</td>
<td>4002</td>
</tr>
<tr>
<td>28106</td>
<td>4001</td>
</tr>
<tr>
<td>28106</td>
<td>4002</td>
</tr>
<tr>
<td>28106</td>
<td>4003</td>
</tr>
<tr>
<td>29120</td>
<td>4001</td>
</tr>
<tr>
<td>29120</td>
<td>4002</td>
</tr>
<tr>
<td>29120</td>
<td>4003</td>
</tr>
<tr>
<td>29555</td>
<td>4003</td>
</tr>
<tr>
<td>30112</td>
<td>4001</td>
</tr>
<tr>
<td>30112</td>
<td>4002</td>
</tr>
<tr>
<td>30112</td>
<td>4003</td>
</tr>
<tr>
<td>31403</td>
<td>4002</td>
</tr>
</tbody>
</table>

\[
\pi_{\text{LecID}}(\text{Lectures})
\]

<table>
<thead>
<tr>
<th>LecID</th>
</tr>
</thead>
<tbody>
<tr>
<td>4001</td>
</tr>
<tr>
<td>4002</td>
</tr>
<tr>
<td>4003</td>
</tr>
</tbody>
</table>

\[
\ldots \div \ldots
\]

<table>
<thead>
<tr>
<th>StudID</th>
</tr>
</thead>
<tbody>
<tr>
<td>28106</td>
</tr>
<tr>
<td>29120</td>
</tr>
<tr>
<td>30112</td>
</tr>
</tbody>
</table>

auxiliary relation which we have called Omnistuds in our example
3) Determination of the relation with the smallest student ID:

\[
\text{Omnistuds} \Join \rho_{\text{StudID} \leftarrow \text{ID1}}(\text{Omnistuds})
\]

Since the StudID 28106 is the smallest one, it does not satisfy the join condition and no 2-tuples with 28106 as value in the first row are in the resulting relation.
4) Determination of the relation with the smallest student ID:

\[ \pi_{\text{StudID}}(R2) \]

Elimination of duplicates is necessary!
5) Determination of the relation with the smallest student ID:

\[ \pi_{\text{StudID}}(\text{Omnistuds}) - \pi_{\text{StudID}}(R2) \]

Projection is redundant!

<table>
<thead>
<tr>
<th>StudID</th>
</tr>
</thead>
<tbody>
<tr>
<td>28106</td>
</tr>
<tr>
<td>29120</td>
</tr>
<tr>
<td>30112</td>
</tr>
</tbody>
</table>

\[ \ldots \div \ldots \]

\[ \pi_{\text{StudID}}(R2) \]

\[ \begin{array}{c}
\text{StudID} \\
29120 \\
30112 \\
\end{array} \]

\[ = \]

\[ \begin{array}{c}
\text{StudID} \\
28106 \\
\end{array} \]

our result for the initial query!