Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

CHAI
Humanities-Centered AI

Ralf Möller

Databases
Structured Query Language SQL (Part 1)

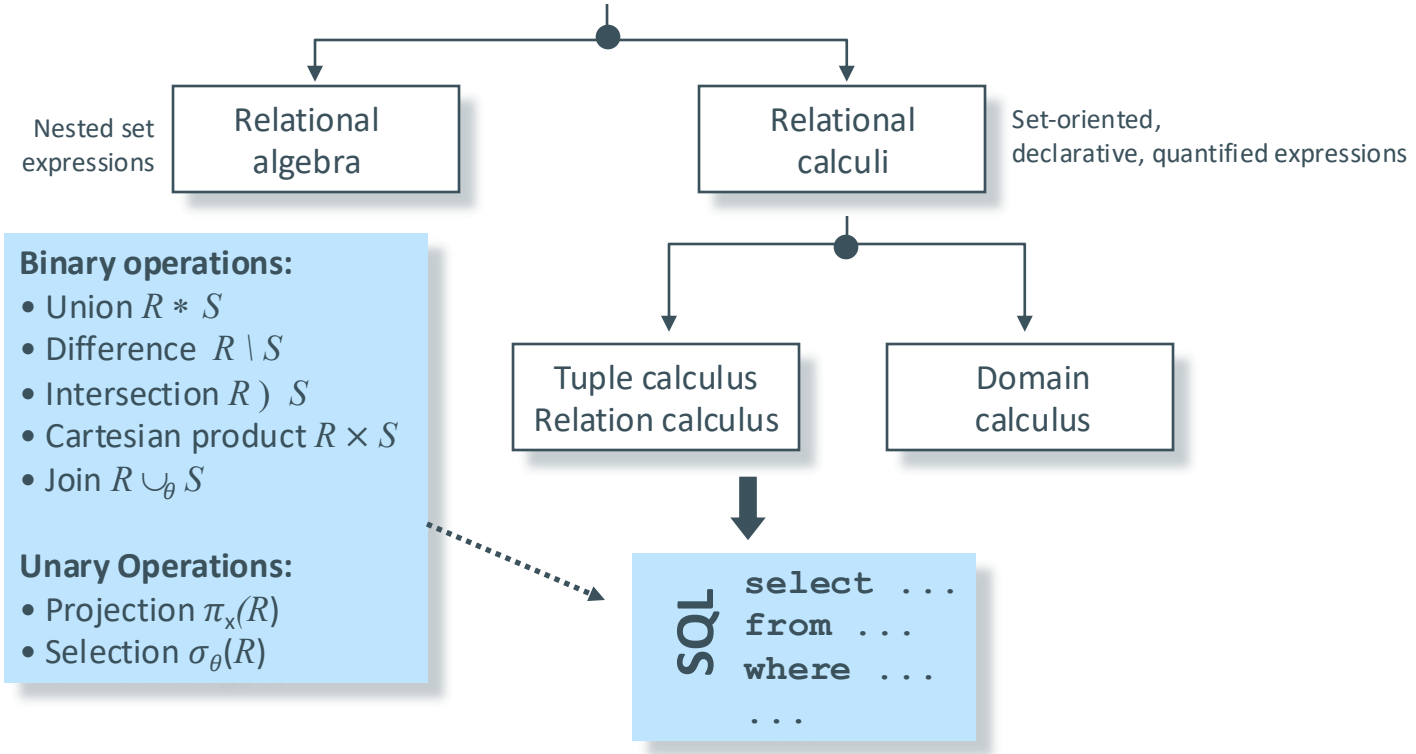
Overview

1. **Introduction**
   - Models and abstraction
   - Database systems
   - Applications and data independence
2. **Entity-relationship model (ER model)**
3. **Relational data model (RDM)**
   - From ER to RDM
4. **Relational schema design theory**
   - Relational algebra as a query language
   - Functional dependencies
   - Normal forms
5. **Structured Query Language (SQL)**
   - Data definition
   - Query specification
   - Data manipulation
6. **Query processing**
   - Architecture
   - Indexing
   - Query plans, optimization
7. **Transactions**
   - Transaction processing, schedules, locks
   - Recovery
8. **Other types of databases**
   - Vector databases
   - Data stream query languages


JOIN NOW

$\pi_{persnr,\,name}\left(\sigma_{rang=\text{"C4"}}\left(Professoren\right)\right)$	Rel. Algebra
$\{t.persnr,\, t.name \mid t \in Professoren \wedge t.rang = \text{"C4"}\}$	Tupelkalkül
$\{[p,n] \mid \exists r,o\,([p,n,r,o] \in Professoren \wedge r = \text{"C4"})\}$	Domänenkalkül

RDM: Query languages




Gringotts Databank
SQL

Introduction to SQL

A note on methodology

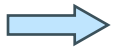
- SQL is huge
- Introduction with examples
- No formal “grammar” of SQL introduced

Example data: Project database

Simple queries

Projection and Selection: SQL query to determine name and ID of all departments, with super department “MSW”

```
select Name, ID
from Departments
where SupDept = 'MSW';
```



Result table

Name	*ID*
Mainframe SW	MFSW
Unix SW	UXSW
PC SW	PCSW

Selection (w/o projection): Transfer of all columns (with * in projection list) in the same sequence

```
select *
from Departments
where SupDept
        = 'MSW';
```

Result table

ID	*Name*	*SupDept*
MFSW	Mainframe SW	MSW
UXSW	Unix SW	MSW
PCSW	PC SW	MSW

More complex queries (w/ variables)

Query language SQL:

Abstraction from iteration with **select from where**:

- **Select** clause: Specification of projection list for result
- **From** clause: Specification of queried tables and variables ranging over tuples
- **Where** clause: Selection specification working on cartesian product or join of queries tables

Join: Multiple tables are combined based on equal values with reference to attributes (especially primary key / foreign key attributes)

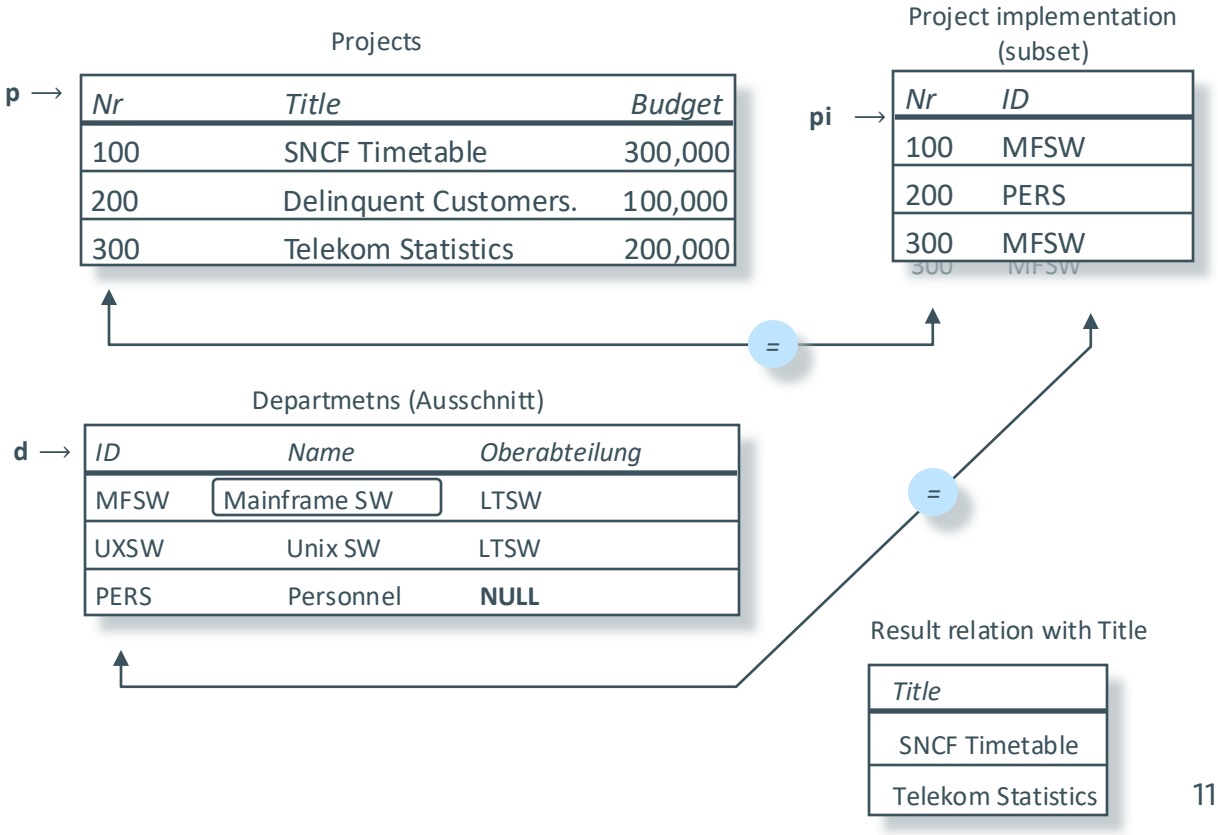
Determination of Title of projects on which the departments of Mainframe Software works:

```
select p.Title
from Projects p,
  Project_implementation pi,
  Departments d
where p.Nr = pi.Nr
and d.ID = pi.Kurz
and d.Name = 'Mainframe SW';
```

Join over tables Projects, Departments and Project_implementations with selection and projection

SQL ← SEQUEL = Structured English Query Language

Join in where part



Explicit joins in SQL

```
select Title
from Projects natural join
     Project_implementation natural join
     Departments
where Name = 'Mainframe SW';
```

Join operators:

- Table **cross join** Table
- Table **natural join** Table
- Table [**inner**] **join Table** [on **cond**]
- Table (**left** | **right** | **full**) [**outer**] **join** Table [on Cond]

RDM: Update operations

Update operations refer to relations or sub-relations (select …):

- **Insert statement:**
  - Inserts into a relation a single tuple whose attribute values are passed as parameters
  - Inserts into a relation a full relation with compatible attributes
- **Update statement:**
  - Selection of all respective tuples to be changed
  - Update values of selected tuples with computed values
- **Delete statement:**
  - Selection and deletion of respective tuples

```
insert into Project_implementation
values (400, 'XYZA')
```

```
insert into Project_implementation
   (Nr, ID)
select p.Nr, d.ID
from Projects p, Departments d
where p.Title = 'Telekom Statistics'
      and d.Name = 'Unix SW'
```

```
update Projects
set Budget = Budget * 1.5
where Budget > 150000
```

```
delete
from Project_implementation
where ID = 'MFSW';
```

Lexical and syntactic rules (1)

SQL has a very large set of syntax rules with lots of optional clauses and keyword-based operators

SQL source code is decomposed into a sequence of symbols (⟶ *lexemes, tokens*)

- Non-printable control characters (e.g., linefeed) and comments are treated as blanks
- Comments begin with "--" and range to end of line
- Lowercase characters are transformed to capital letters if they do not occur in string constants

Lexical and syntactical rules

SQL symbol are:

- **Regular names** bstarting with a letter and followed by further letters, digits, or "_".
- **Keywords**: SQL definies over 210 names as keywords and are interpreted in a special way by the SQL system
- **Delimited names** are character sequences in double quotation marks (e.g., for names that are actually keywords)
- **Literals** denote values from SQL basic datatypes
- **Strings** are sequences of characters in single quotation marks for denoting special literals
- Futher sysmbols (operatoren etc.)

```
Peter, mary33
```

```
create, select
```

```
"intersect", "create"
```

```
'abc'       character(3)
123         smallint
B'101010'   bit(6)
```

```
<, >, =, %, &, (, ),
*, +, ...
```

Schemas and catalogs (1)

- An SQL schema is a namespace containing (local) SQL objects (tables, views, rules, ...)
- In MySQL/MariaDB creating a schema also creates a database (in other systems use **`create database`**)
- Binding for objects to names can be established and dropped explicitly with statements as shown below
- Schemas are in turn contained in visibility areas, the catalogs..
- A catalog is a logical container that groups a set of schemas (and therefore tables, views, procedures, etc.) together
- Catalogs contain additional information such as access rights, storage medium, date of last backup, etc.

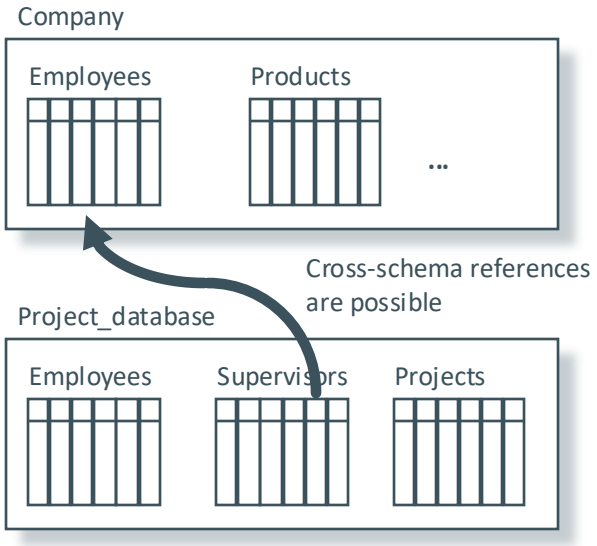
```
create schema Company;
  create table Employees ...;
  create table Products ... ;

create schema Project_schema;
  create table Empoloyees ...;
  create view Supervisors ...;
  create table Projects ...;
  create table Test ...;
  drop table Test;

drop schema Company;
```

Schemas and catalogs (2)

- Schemas and tables are (persistently stored) data
- Schema and table structures can be queried

Schema catolog

Name	User	
Company	matthes	
Project_ database Test	matthes schmidt	



Notation of names:

```
Comany.Employees
Project_database.Employee
```

Basic datatypes and type compatibilities

- Relational data model based on a set of domains for atomic values in attributes of tuples
  - Build-in domains: basic datatypes
  - Semantics of domains independent from programming languages
  - Modern database models also support user-defined data types
- Requirements for a domain D:
  - Existence of an equivalence relation on D (⟶ duplicate elimination, functional dependency checks)
  - Existence of additional Boolean predicates (>, <, >=, substring, odd, ...) on D for selection and join expressions via attributes (optional)
- SQL performs type conversion (details about compatibility skipped here)
  - Implicit (coercing) or explicit (casting)

```
integer, smallint,
numeric(p,s),
decimal(p,s)
```

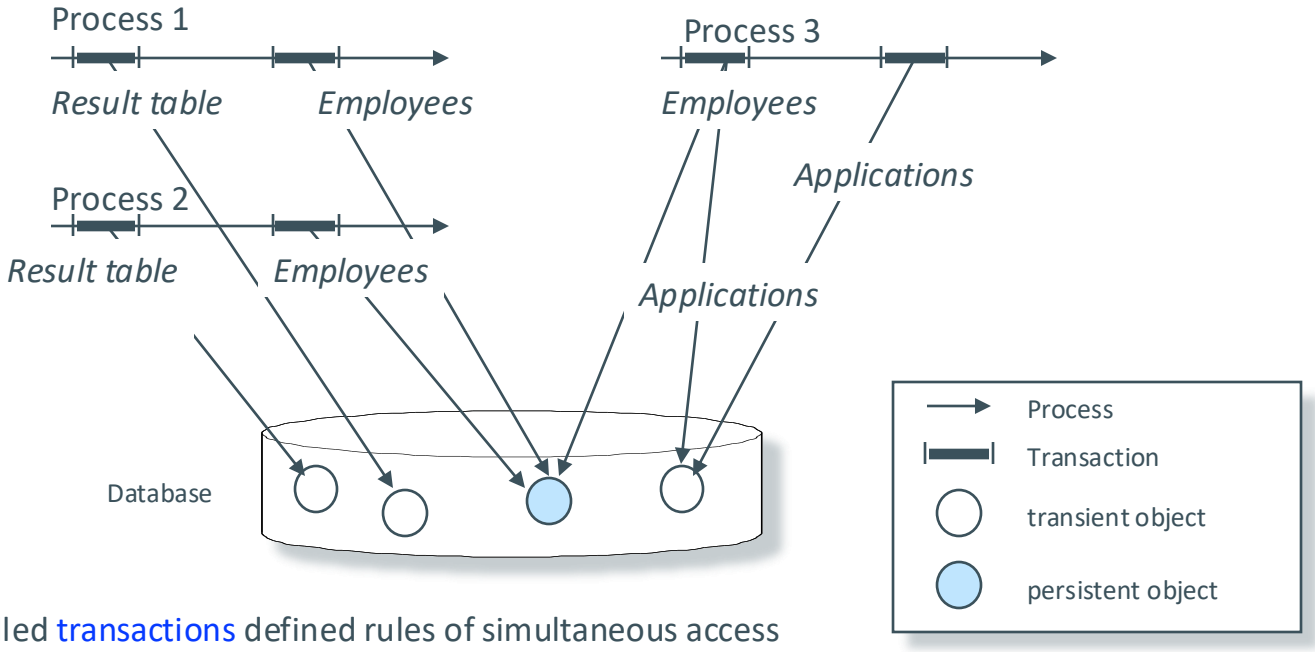
```
real,
double precision,
float(p)
```

```
character(n),
character varying(n)
```

```
bit(n),
bit varying(n)
```

Lifespace, Visibility, share usages (1)

The same database can be used sequentially or simultaneously by multiple data processing tasks



So-called transactions defined rules of simultaneous access

Default values for attributes

When inserting tuples into a table, values for certain attributes can be left unspecified

```
insert into Employees
  (Name, Salary, Vacation)
values ("Peter", 3000, null)
```

```
insert into Employee
  (Name, Salary)
values ("Peter", 3000)
```

Missing values filled with null or default values specified at table creating time

- Defaults values are given by basic datatypes (or computed at inserting time with specified functions)
- Data independence and schema evolution support with defaults values
  - What should happen to old tuples if attributes were added to a table on the fly?

Null values

In SQL, every basic datatype is extended by a specific value called **`null`** (different from other values of the type)

Null values can be excluded with an extended type specification

Example for a type with null excluded

```
integer not null
```

Null values and truth values

Three-values SQL logic

OR	true	false	null
true	*true*	*true*	*true*
false	*true*	*false*	*null*
null	*true*	*null*	*null*

AND	true	false	null
true	*true*	*false*	*null*
false	*false*	*false*	*false*
null	*null*	*false*	*null*

x	not x	x is null	x is not null
true	*false*	*false*	*true*
false	*true*	*false*	*true*
null	*null*	*true*	*false*

Difficulties in consistently extending a domain to include null values are already apparent in the simple example of Boolean values and the basic logical equivalence x and not x = false, which is violated when the domain is extended to include null values (null and not null = null).

Three-valued semantics of truth values

Pros:

- Explicit and consistent handling of zero values by all applications (as opposed to ad hoc solutions where, for example, the value -1, -MaxInt, or an empty string is used as a zero value)
- Precise definition of the semantics of database operators (assignment, comparison, arithmetic) on zero values

Cons:

- Extending a data type to include null values often conflicts with the algebraic properties (existence of null elements, associativity, commutativity, order, etc.) of the non-extended data type
- Algebraic properties are often exploited for query optimization. A query that uses values of a data type T' (where null∈T') generally offers less scope for optimization than a query with values of the data type T (with null ∈ T)

Algebraic table operations (1)

Union *R* U *S*:

- All tuples of two relations are combined into a result relation.
- The result does not contain any duplicates.
- Avoidance of duplicate elimination through union all.
- Compatibility of the column structure of the tables involved. (⟶ same column names and data types)
- The column names of the result correspond to the first relation (if necessary, rename them with AS)
- Special case: Create corresponding (same-named) columns and perform UNION

```
Rel1
union
Rel2;
```

```
select *
from R
union
select *
from S;
```

```
Professors
union corresponding
Students;
```

Algebraic table operations (2)

Difference *R* \ *S*:

- The tuples of two relations are compared with each other
- The tuples in the first relation but not in the second are included in the result relation

```
Rel1
except
Rel2;
```

Intersection *R* ∩ *S*:

- The tuples of two relations are compared with each other
- The tuples that are contained in the first relation but not in the second relation are included in the result relation.

```
Rel1
intersect
Rel2;
```

- **Cartesian product *R* × *S*:**

```
select *
from R cross join S;
```

Overview

1. **Introduction**
   - Models and abstraction
   - Database systems
   - Applications and data independence
2. **Entity-relationship model (ER model)**
3. **Relational data model (RDM)**
   - From ER to RDM
4. **Relational schema design theory**
   - Relational algebra as a query language
   - Functional dependencies
   - Normal forms
5. **Structured Query Language (SQL)**
   - Data definition
   - Query specification
   - Data manipulation
6. **Query processing**
   - Architecture
   - Indexing
   - Query plans, optimization
7. **Transactions**
   - Transaction processing, schedules, locks
   - Recovery
8. **Other types of databases**
   - Vector databases
   - Data stream query languages