Information Systems 2

1. Modelling Information Systems I: Databases

Lars Schmidt-Thieme

Information Systems and Machine Learning Lab (ISMLL)
Institute for Business Economics and Information Systems
& Institute for Computer Science
University of Hildesheim
http://www.ismll.uni-hildesheim.de

Why Databases?

Benefits of databases:

- Data can be shared.
- Redundancy can be reduced.
- Inconsistency can be avoided (to some extent).
- Transaction support can be provided.
- Integrity can be maintained.

[**?**]

Schema and State

- **Schema** (intension):
  describes the structure of a database with so-called **schema constructs**.
  The schema often is represented graphically.
- **State** (extension, contents):
  describes the contents of a database at a given point in time (**snapshot**).
  The state can be described by a set of instances of schema constructs.

Schema and State / Example

Example:
A small company keeps a list of its customers in an OO Calc table. It contains a column each for name, address, phone number and email address. By now, there are just two rows:

name	address	phone	email
Anna Müller	Schuhstaße 3, 31139 Hildesheim	05121 / 123456	mueller@example.com
Bert Meier	Hauptstraße 11, 30300 Hannover	050 / 12480	meier@beispiel.de

Here,

- “name”, “address” etc. are schema constructs.
- “Anna Müller”, “Schuhstraße 3” etc. describe the state.

Data Model

A data model provides a formal description of how data may be structured and accessed.

It covers:

- **data structures**:
  that may be used to define the schema of a database.
- **integrity rules**:
  placed on the data structures to enforce integrity constraints.
- **data manipulation operators**:
  which allow to query and change the data.

Levels of Data Models

Usually one distinguishes 3 different levels of data models:

Conceptual models
(also **logical models**; **high-level models**):
describe data in terms close to the concepts of users, e.g.:

- **Object model**: describes data as **objects** that are instances of **classes**, which have **properties** and **methods**; classes are organized in an **inheritance hierarchy**.
- **Entity Relationship model** (P. Chen 1976): describes data as **entities** with **attributes** and **relations**.

Representation models
(also **Implementation Models**):
describe data in terms that are close to implementations, e.g.:

- **Relational data model** (Edgar Codd, 1969): describes data as **tables** (**relations**).
- **Network data model** (Charles Bachman, 1969): describes data as a **network of records** (example: LDAP).
- **Hierarchical data model** (mainframe era): describes data as a **tree of records**.

Physical models
(also **low-level models**; **storage models**; **internal models**):
describe storage of data in detail.

Sometimes the logical level is split in

- **external models** (also **user logical models**): describes data from the perspective of different users.
- **conceptual models** (also **community logical models**): describes data from the perspective of the community of all users.

Levels of Data Models

Accordingly, one distinguishes three different levels of schemata:

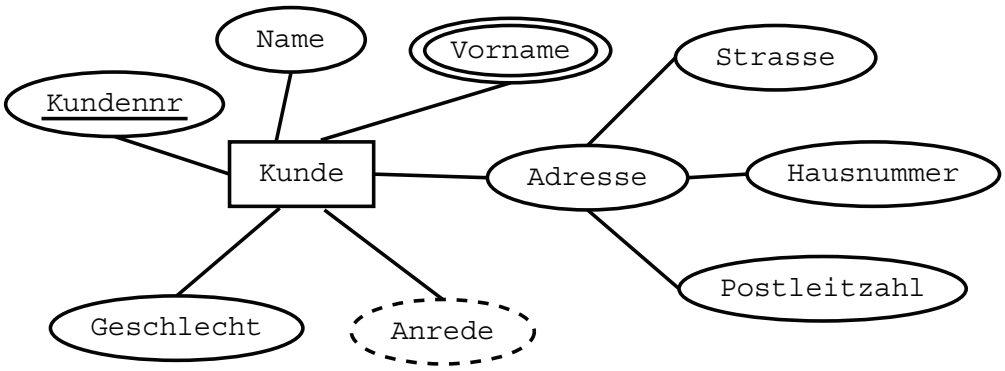
- **Internal schema:**
  describes physical structures in which the data is stored.
- **Conceptual schema**:
  describes the structure of the whole database for users.
- **External schema** (also **user view**):
  describes the structure of a a part of the database for a specific user or user group.

Data is stored accordingly to the internal schema.

There are **mappings** between internal schema and higher schemata.

Components

construct	models	diagram
entity	entity, object, thing in the real world	rectangle
attribute	property of a thing	oval
relation	relation between two or more things	diamond



Properties of Attributes

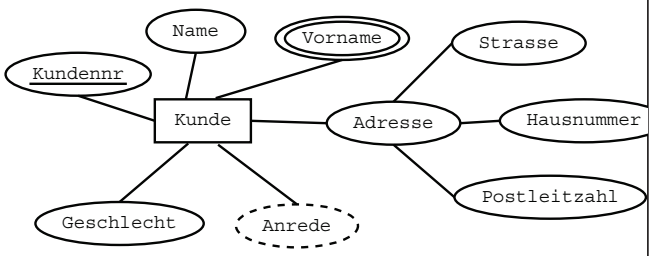
simple vs. **complex**:
simple attributes cannot be decomposed into parts, complex attributes are composed of other attributes.

single-valued vs. **multi-valued**:
for single-valued attributes each entity has at most one value, for multi-valued attributes an entity may have several values.

stored vs. **derived**:
stored attributes are stored explicitly in the database, derived attributes can be computed from other information.

value domain:
the values of an attribute come from a fixed set, e.g., integers, real numbers, strings of a maximal length etc.

null value:
the special value null marks missing values or attributes that do not apply for a given entity.



complex: an oval with attached ovals.

multi-valued: double oval.

derived: dashed border.

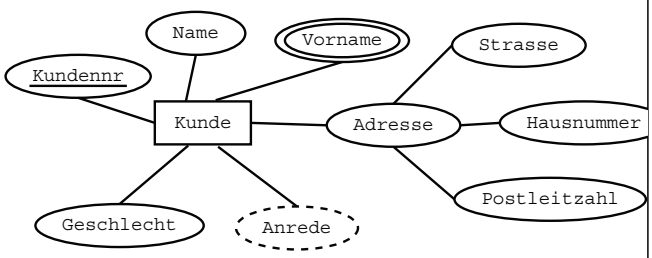
Entities

Things of the same type, i.e., things that can be described by the same attributes and relations, are modeled as instances of an entity.

key attributes: an attribute whichs value occurs among all instances of an entity at most once, i.e., allows to identify an instance, is called a key.

Most entities have exactly one key. But they may also have none or several keys.

regular vs. **weak**:
an entity with at least one key is called regular. An entity without key is called weak.



underlined: key attribute.

Relations

Relations are used to model relationships between entities. The entities involved are called **participating entities**.

Arity:
the number of participating entities of a relation is called its arity.

Role names: the positions at which entities can enter a relation are called roles. E.g., Auftragserteiler, Auftrag, Auftragsbearbeiter.

Role names are especially important if the same entity can participate in several roles in a relationship.

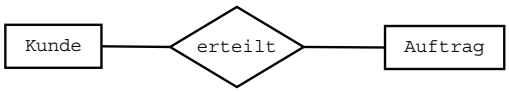


Figure 4: A binary (2-ary) relation: [Kunde] erteilt [Auftrag].

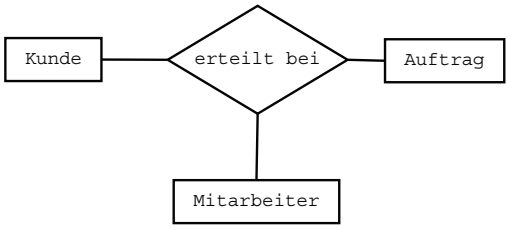


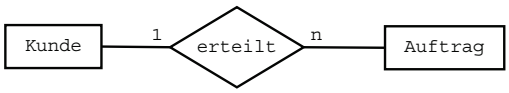
Figure 5: A ternary (3-ary) relation: [Kunde] erteilt [Auftrag] bei [Mitarbeiter].

Relations / Cardinality Constraints

Cardinality constraint:
restricts how often an instance of an entity may participate in a relation.

1:1:
an instance is allowed to participate at most with one other instance.

1:n:
an instance of role 1 may participate at most with n other instances in role 2, but an instance of role 2 may participate at most with one other instances in role 1.

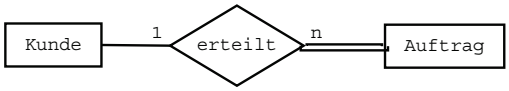


n:m:
an instance of role 1 may participate at most with n other instances in role 2, an instance of role 2 may participate at most with m other instances in role 1.

Relations / Total vs. Partial Participation

Total Participation:
An entity participates totally in a relation, if each instance must be related to some other instances by that relation. (lower bound cardinaliy restriction ≥ 1).

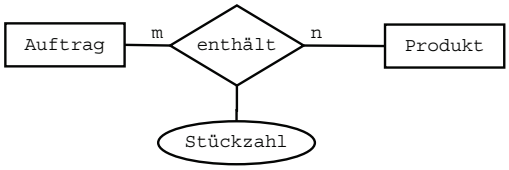
Partial Participation:
An entity participates partially in a relation, otherwise.



Relations / Attributes

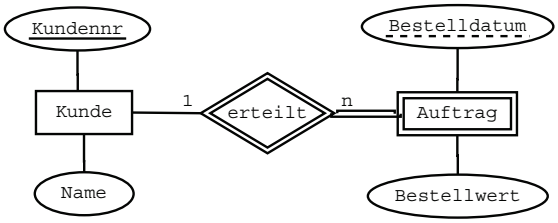
Attributes:
Relations may have attributes, too.

Each instance of the relation, i.e., each tuple of instances of entities between which the relation holds, has a value for each of the attributes of the relation.



Weak Entities

Weak entity:
an entity without a key attribute.



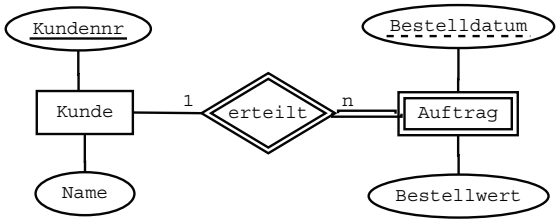
Instances of weak entities are identified indirectly by means of a relation.
An instance of a weak entity is identified as the instance that

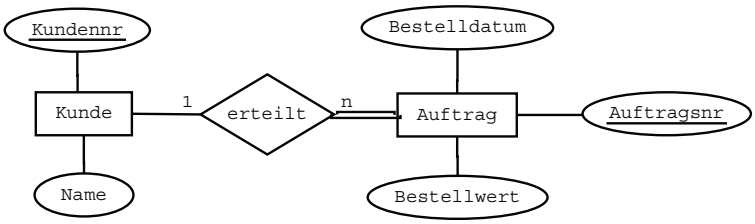
- is related to a given other entity (**identifying entity; owner entity**)
- with respect to a given relation (**identifying relation**) and
- has a given value for a given attribute (**partial key**).

Weak entities always participate completely in their identifying relation.

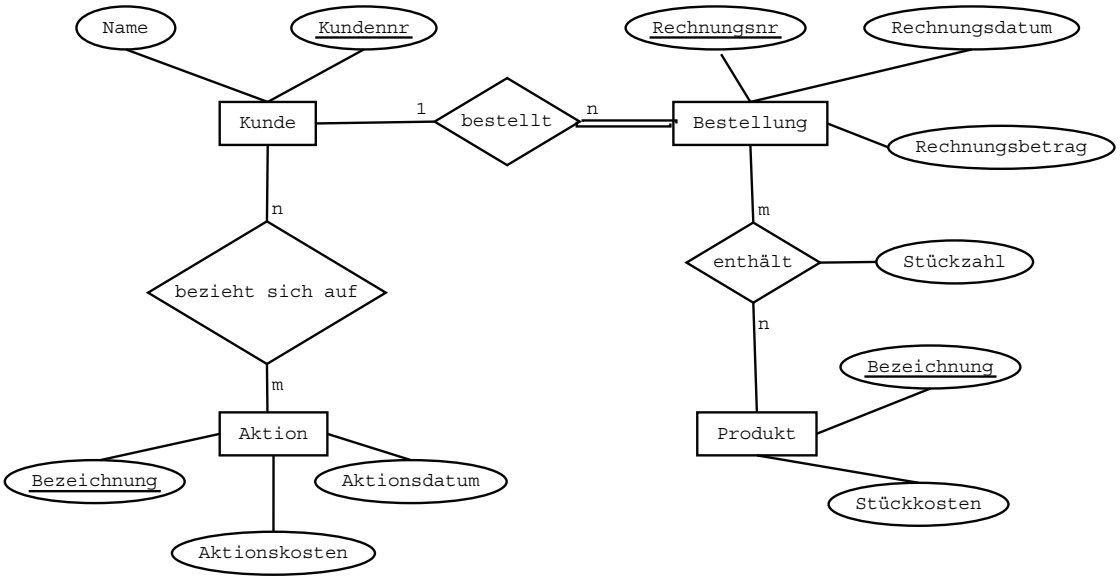
Weak Entities

Weak entities can be converted to regular entities by introducing a key.





An Example



Basic Concepts

The Relational model organizes data in tables.

Relational model	common sense table
attribute	column
attribute domain	value domain of a column
tuple	row
relation	table
null value	cells without entry (missing values, unappropriate attributes)
key	set of columns which values uniquely identify a row
primary key	key usually used for identifying rows

KUNDE

Kundennr	Name	Geburtstag
1	Frank Müller	20.11.1980
2	Fred Schmidt	6.6.1972
3	Heribert Mayer	11.1.1954
4	Frank Müller	3.7.1978

Basic Concepts / Foreign Keys

Foreign Key:
an attribute (or set of attributes) that contains the key value of another relation.

The value domain of the foreign key must be the same as the value domain of the key of the **referenced relation**.

For each tuple of the **referencing relation** the value of the foreign keys must occur among the values of the key attribute of the referenced relation or be null (**referential integrity**).

KUNDE

Kundennr	Name	Geburtstag
1	Frank Müller	20.11.1980
2	Fred Schmidt	6.6.1972
3	Heribert Mayer	11.1.1954
4	Frank Müller	3.7.1978

BESTELLUNG

Rechnungsnr	Rechnungsdatum	Rechnungsbetrag	Kundennr
1099	12.2.2000	2099,-	2
1100	12.2.2000	589,-	1
1101	13.2.2000	4490,-	3
1102	15.2.2000	3349,-	2
1103	18.2.2000	10500,-	5

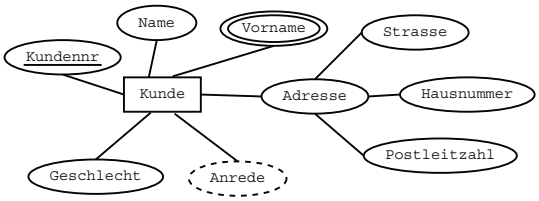
Mapping an ER Model to a Relational Model / Regular Entity

Regular entity with $\longrightarrow$relation (**entity relation**) with
- simple attributes $\longrightarrow$ attribute
- complex attributes $\longrightarrow$ one attribute for each component (neglect structure)
- key $\longrightarrow$ primary key (select one)
- keys $\longrightarrow$ secondary keys (all other)

Multi-valued attributes are mapped to an own relation that contains

- the foreign key of the entity relation and
- the value of the attribute.

Together they define the primary key of the new relation.



KUNDE

Kundennr	Name	Geschlecht	Strasse	Hausnummer	Postleitzah

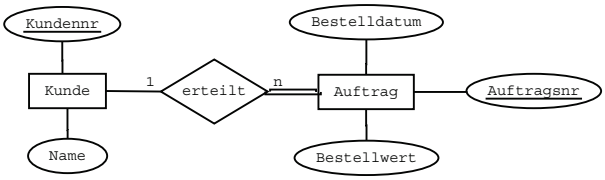
KUNDENVORNAMEN

Kundennr	Vorname

Mapping an ER Model to a Relational Model / Binary Relation 1:n

Binary relation 1:n$\longrightarrow$add a foreign key
- $\longrightarrow$ referencing the entity relationship on the 1-side
- $\longrightarrow$ to the entity relationship on the n-side.

All attributes of the relation are added to the entity relationship on the n-side.



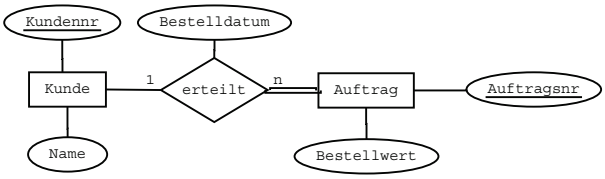
KUNDE

Kundennr	Name

AUFTRAG

Auftragsnr	Bestelldatum	Bestellwert	Kundennr

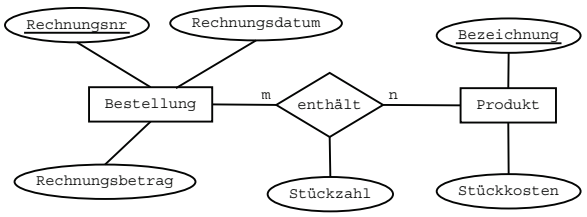
variant:



Mapping an ER Model to a Relational Model / Binary Relation n:m

Binary relation n:m $\longrightarrow$ relation (**relationship relation**) with
- $\longrightarrow$ foreign key to the entity relation on the n-side and
- $\longrightarrow$ foreign key to the entity relation on the m-side
- $\longrightarrow$ (jointly defining the primary key)

All attributes of the relation are mapped to attributes of the relationship relation.



BESTELLUNG

Rechnungsnr	Rechnungsdatum	Rechnungsbetrag

PRODUKT

Bezeichnung	Stückkosten

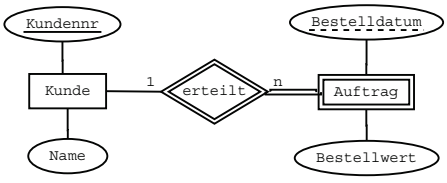
POSTEN

Rechnungsnr	Produktbezeichnung	Stückzahl

Mapping an ER Model to a Relational Model / Binary Relation n:m

Weak entity $\longrightarrow$ relation (**entity relation**) with
- $\longrightarrow$ foreign key to the identifying entity relation
- $\longrightarrow$ (jointly with the partial key defining the primary key)

All attributes are mapped as for regular entities.



KUNDE

Kundennr	Name

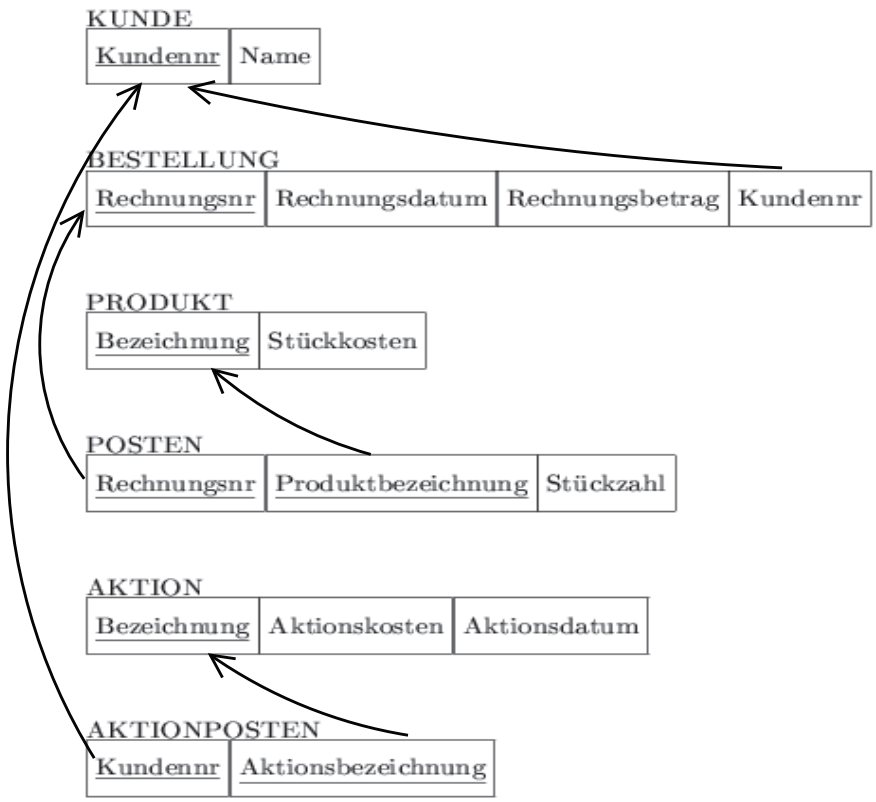
AUFTRAG

Kundennr	Bestelldatum	Bestellwert

Mapping an ER Model to a Relational Model / Summary

ER model	Relational model
entity	relation (**entity relation**)
relation, binary, 1:1	add foreign key to one of the participating entity relations
relation, binary, 1:n	add foreign key to the participating entity relation on the n-side
relation, binary, n:m	relation (**relationship relation**) with 2 foreign keys
relation, n-ary	relation (**relationship relation**) with n foreign keys
attribute, simple	add attribute to relation
attribute, complex	add attributes to relation, one for each component
attribute, multi-valued	relation with foreign key
key	primary or secondary key

Mapping an ER Model to a Relational Model / Example



SQL

SQL (**Structured Query Language**) is the industrial standard for

- the defintition of relation schemata (**schema definition language**, DDL),
- the manipulation of the contents of a relational database (**data manipulation language**, DML) and
- the manipulation of access rights to a relational database (**data control language**, DCL)

History

SQL is an ISO/ANSI standard:

- based on SEQUEL (Structured English Query Language) by Donald D. Chamberlin and Raymond F. Boyce (IBM) in the early 1970s
- 1986 standardized by ISO/ANSI (SQL/1, SQL-86)
- 1992 update to SQL/2, SQL-92; 1999 update to SQL/3, SQL:1999
- 2003 update to SQL:2003; 2006 update to SQL:2006 (XML features)
- 2008 update to SQL:2008.

DBMS Implementations

SQL is supported by nearly all relational database management systems:

- Many simpler DBMS (such as mysql) do not implement parts of the standard.
- Most DBMS provide (mutually incompatible) non-standard extensions.

Queries and Result Sets

SQL defines a text format for database queries.

Every SQL query returns

- an error code or
- a result set.

Creating a Database

DBMS can manage several databases at the same time.

One can create a new database via:

create database ⟨*database-name*⟩;
use ⟨*database-name*⟩;

All following operations will be applied to this database.

One can destroy a database via:

drop database ⟨*database-name*⟩;

Creating a Table

One can create a table via

create table ⟨*table-name*⟩ (⟨*table-spec*⟩) ;

where ⟨*table-spec*⟩ is a comma-separated list of table specifications, i.e., of

- columns:

  ⟨*column-name*⟩ ⟨*column-type*⟩ [not null] [primary key]
  [references ⟨*table-name*⟩ [(⟨*column-commalist*⟩)]]

- primary keys:

  primary key (⟨*column-commalist*⟩)

- foreign keys:

  foreign key (⟨*column-commalist*⟩)
  references ⟨*table-name*⟩ [(⟨*column-commalist*⟩)]

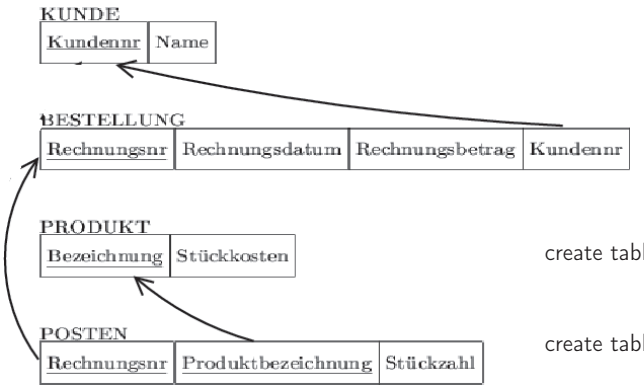
One can destroy a table via:

drop table ⟨*table-name*⟩;

Types (SQL:99)

type	description
int	integer (4 byte)
smallint	integer (2 byte)
float(p)	floating point number (accuracy p: number of valid digits)
decimal(p,q)	formatted floating point number (accuracy p: number of valid digits; scale q: number of post-comma digits)
char(n)	string of fixed length n
varchar(n)	string of maximal length n
bit(n)	bit sequence of fixed length n
varbit(n)	bit sequence of maximal length n
date	date
time	time
timestamp	combination of date and time
blob	binary large object
clob, nclob	character large object (of variable or fixed length)

Creating a Table / Example




```
create table kunde (kundennr integer not null primary key,
                    name varchar(30)) ;

create table bestellung (rechnungsnr integer not null primary key,
                         rechnungsdatum date,
                         rechnungsbetrag integer,
                         kundennr integer references kunde);

create table produkt (bezeichnung varchar(30) not null primary key,
                      stueckkosten integer);

create table posten (rechnungsnr integer not null references bestellung,
                     produktbezeichnung varchar(30) not null
                       references produkt (bezeichnung),
                     stueckzahl integer,
                     primary key (rechnungsnr, produktbezeichnung) );
```


Modifying the Contents of a Table / Insert

Insert rows into a table:

insert into ⟨*table-name*⟩ [⟨*column-commalist*⟩] ⟨*table-expr*⟩ ;

where ⟨*table-expr*⟩ in the simplest case is as

values (⟨*scalar-expr-commalist*⟩)

Example:

```
insert into kunde values (1, "Frank Mueller");
insert into produkt values ("Spark II", 400);
insert into bestellung values (2001, "2000-06-31", 2000, 1);
insert into posten values (2001, "Spark II", 1);
```

Modifying the Contents of a Table / Delete

Delete rows in a table:

delete from ⟨*table-name*⟩ [where ⟨*cond-expr*⟩];

where ⟨*cond-expr*⟩ in the simplest case is as

⟨*column-name*⟩ ⟨*comparision-operator*⟩ ⟨*value*⟩

or a combination of such expressions with the boolean operators “and,” “or” and “not”.

Example:

```
delete from kunde where kundennr = 2000;
delete from bestellung where rechnungsdatum < "2000-07-01";
```


Modifying the Contents of a Table / Update

Update rows in a table:

update ⟨*table-name*⟩ set ⟨*column-name*⟩ = ⟨*scalar-expr*⟩
[where ⟨*cond-expr*⟩];

Example:

```
update produkt set stueckkosten = 1.2 * stueckkosten ;
```

SQL Queries

select ⟨*select-item-commalist*⟩
from ⟨*table-ref-commalist*⟩
[where ⟨*cond-expr*⟩]
[group by ⟨*column-ref-commalist*⟩]
[having ⟨*cond-expr*⟩]

where ⟨*select-item*⟩ is as

⟨*scalar-expr*⟩ [as ⟨*column-name*⟩] | [range-variable .] *

and ⟨*table-ref*⟩ is as

⟨*table-name*⟩ [AS ⟨*range-variable*⟩ [(column-commalist)]]
| ⟨*table-expr*⟩ [AS ⟨*range-variable*⟩ [(column-commalist)]]
| ⟨*join-table-expr*⟩

SQL Queries / Examples

Examples:

select * from bestellung;

rechnungsnr	rechnungsdatum	rechnungsbetrag	kundennr
2001	2000-06-31	2000	1
2002	2000-07-01	6000	3
2003	2000-07-04	1600	1

select kundennr as knr, rechnungsbetrag from bestellung;

knr	rechnungsbetrag
1	2000
3	6000
1	1600

SQL Queries / Joins

More complex queries combine several tables.

The **join operator** (represented as comma or by "join") builds the cartesian product of two tables.

Usually, one is not interested in all combinations of the rows of two tables, but just the ones that are joined by a foreign key. This can be accomplished by:

- filtering by a "where" clause,
- a "left join" or "right join" operator with "on" clause or
- a "natural left join" or "natural right join" operator (join on all attributes with the same name).

SQL Queries / Joins / Example

```
select name,rechnungsbetrag from kunde,bestellung
    where kunde.kundennr = bestellung.kundennr;

select name,rechnungsbetrag from bestellung left join kunde
    on kunde.kundennr = bestellung.kundennr;

select name,rechnungsbetrag from bestellung natural left join kunde;
```

name	rechnungsbetrag
Frank Mueller	2000
Heribert Mayer	6000
Frank Mueller	1600

SQL Queries / Aggregation

One can aggregate the values of a column groupwise by

- defining groups of rows by a “group by” clause and
- use an aggregation function such as
  sum, count, max, min, avg
  in the “select-expr”.

Example:

```
select name,sum(rechnungsbetrag)
   from bestellung natural left join kunde
   group by name;
```

name	sum(rechnungsbetrag)
Frank Mueller	3600
Heribert Mayer	6000

SQL Queries / Aggregation

With the “having” clause one can filter those aggregated rows that meet some specified criteria.

Example:

```
select name,sum(rechnungsbetrag)
   from bestellung natural left join kunde
   group by name
   having count(*) > 1;
```

name	sum(rechnungsbetrag)
Frank Mueller	3600

SQL Queries / Sorting

With the “order by” clause one can sort the rows of the result set.
One has to provide

- the name of the columns to sort by and
- “asc” for ascending or “desc” for descending sorting.

Example:

```
select kundennr,rechnungsbetrag
   from bestellung
   order by rechnungsbetrag desc;
```

kundennr	rechnungsbetrag
3	6000
1	2000
1	1600

SQL Queries / Nested Queries

One can use “select” expressions in “where” clauses.

Example:

```
select name
   from posten natural left join bestellung natural left join kunde
   where produktbezeichnung in
      (select produktbezeichnung
         from posten natural left join bestellung natural left join kunde
         where name="Frank Mueller");
```

name
Frank Müller
Heribert Mayer

Further SQL Concepts

- transactions
- views
- access rights
- trigger
- cursor
- stored procedures