Databases (LIX022B05)
Logical Database Design

Instructor: Çağrı Çöltekin
c.coltekin@rug.nl

Information science/Informatiekunde

2012-09-17

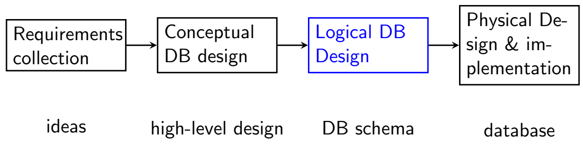
Summary of last week

- A conceptual design using E-R data model allows us to
  - think about the DB requirements systematically and formalize the ideas from the requirement analysis,
  - communicate the overall design of the database using a graphical representation.
- E-R constructs can be reduced to a database schema.
- Conceptual modeling is helpful, however, it does not guarantee correct relational database design.

Conceptual (E-R) design: things to remember

- Entity / entity set
- Relationship / relationship set
- Attribute
  - Simple
  - Composite
  - Multi-valued
- Weak entity
- one-to-one, one-to-many, many-to-one relationships
- total or partial participation
- binary or n-ary relationships
- Converting E-R diagrams to table schemas
- Primary keys, foreign keys
- SQL **create table** statement

Database Design Process



- DB design is generally part of a bigger software design process.
- These steps reflect the idealized case. Typically, you may need to re-iterate over some of the steps multiple times.
- In some cases 'conceptual design' step is skipped.
- This week, we are interested in the third step.

What can go wrong (1)

Is anything wrong with this table?

Name	*Dept.*	Course	*Email*
Ubbo Emmius	History	Frisian Hist.	u.emmius@rug.nl
Frits Zernike	Physics	Optics, Intr. to Physics	f.zernike@rug.nl

Problem 1: The course column is not atomic.

- Try to write an SQL query that finds the instructor(s) that teach a certain course. (possible, but tricky, error prone and likely slow)
- What happens if a typo in an application replaces the separator ',' with another character?
- Solution is to use atomic domains.

What can go wrong (2)

We know that this is bad, too. But why?

Name	*Dept.*	*Email*	*Course*	*ECTS*
Ubbo Emmius	History	u.emmius@rug.nl	Frisian Hist.	5
Frits Zernike	Physics	f.zernike@rug.nl	Optics	3
Frits Zernike	Physics	f.zernike@rug.nl	Intr. to Physics	10

More problems:

- Some fields are repeated unnecessary: waste of storage. (Who cares, disks are cheap?)
- If one updates email address of Zernike on one instance but not on the other: inconsistent database. (Can people really be that careless?)
- What happens to information about 'U. Emmius' if we want to remove the last course he teaches?

Solution is ... to split the table into smaller tables (decomposition)

What do we want to avoid?

- Inconsistent database.
- Redundant repeated storage of information.
- Update anomalies, where we update the same information in one place but not the other.
- Deletion anomalies, where we have to delete unwanted data together with the data we want to delete.
- Insertion anomalies, where it is not possible to store a certain information without inserting additional unnecessary/unrelated data.

The solution

The solution is to make sure that your tables meet certain formal criteria: normal forms.

- The normal forms are achieved by decomposing (splitting) the tables that do not conform into multiple tables such that the new tables are in the desired normal form.
- Being in a certain normal form is not enough, you need to make sure that you keep the same information and same constrains using the new tables.
- There still are some loose ends: for example, which normal form to pick, and whether violate some of the requirements intentionally.

What comes next is a rather 'light' introduction to a highly theoretical subject.

First normal form

First Normal Form (1NF): Domains of all attributes should be atomic.

Name	*Dept.*	Course	*Email*
Ubbo Emmius	History	Frisian Hist.	u.emmius@rug.nl
Frits Zernike	Physics	Optics, Intr. to Physics	f.zernike@rug.nl

⇓

Name	*Dept.*	Course	*Email*
Ubbo Emmius	History	Frisian Hist.	u.emmius@rug.nl
Frits Zernike	Physics	Optics	f.zernike@rug.nl
Frits Zernike	Physics	Intr. to Physics	f.zernike@rug.nl

First normal form (2)

Name	*Dept.*	*Course*	*Email*
Ubbo Emmius	History	Frisian Hist.	u.emmius@rug.nl
Frits Zernike	Physics	Optics	f.zernike@rug.nl
Frits Zernike	Physics	Intr. to Physics	f.zernike@rug.nl

- 1NF does not guarantee a good design, it is just a beginning.
- 1NF is typically assumed by any database design process.
- The definition of 'atomic', and as a result 1NF, is somewhat unclear and application dependent:
  - Is 'name' field above atomic? (probably not)
  - How about email field? (maybe not)
  - How about an integer field, such as course number? (certainly atomic?)

Functional dependencies: formal definition

A set of attributes $B = \beta_1 \cdots \beta_M$ is functionally dependent on another set of attributes $A = \alpha_1 \cdots \alpha_N$ if for all possible tuples in the relation, value of A on a certain tuple determine the value of B in the same tuple.

We note this functional dependency as

$$\alpha_1\, \alpha_2\, \cdots\, \alpha_N \rightarrow \beta_1\, \beta_2\, \cdots\, \beta_M$$

and if this is a valid for a particular relation (table), we say that the functional dependency holds for that particular relation.

In other words: the functional dependency $A \rightarrow B$ means that 'if two tuples (rows) have identical value(s) for A then they have to have identical value(s) for B'.

Functional dependencies by example

Inst	*Dept.*	*Email*	*Course*	*ECTS*
Ubbo Emmius	History	u.emmius@rug.nl	Frisian Hist.	5
Frits Zernike	Physics	f.zernike@rug.nl	Optics	3
Frits Zernike	Physics	f.zernike@rug.nl	Intr. to Physics	10

Do following functional dependencies hold?

- Inst → Email ✔
- Inst → ECTS ✗
- Inst Course → ECTS ✔
- Course → ECTS ✔
- Inst → Department ✗
- Department → Inst ✗
- ECTS → Course ✗
- Inst Course → ECTS Email ✔
- Course → Course ✔
- Inst Course → Course ✔

Trivial functional dependency

A functional dependency is called a trivial functional dependency if all attributes on the right side also appear on the left side. Examples:

- Course → Course
- Course ECTS → Course
- Course ECTS → Course ECTS

Trivial functional dependencies hold regardless of the choice of attributes.

Inferring FDs from others

For normalization, we typically need to consider all possible functional dependencies. Some rules help us reduce the FDs that we need to consider.

Armstrong's axioms: for sets of attributes A, B, and C

1. If B ⊆ A, then A → B holds (consider trivial FDs).
   Example: *Course ECTS* → *ECTS* holds.
2. If A → B holds, A C → B C holds.
   Example: If *Course* → *ECTS*,
   then *Course Department* → *ECTS Department*
3. If A → B and B → C holds, then A → C also holds.
   Example: If *Course* → *ECTS* and *ECTS* → *Course_hours*,
   then *Course* → *Course_hours*

More rules for inferring FDs from others

More rules (can be derived from Armstrong's axioms):

- If A → B and A → C hold, then A → B C also holds.
  Example: If *Inst* → *Dept* and *Inst* → *Email*,
  then *Inst* → *Dept Email*.
- If A → B C then, A → B and A → C also hold.
  Example: If *Inst* → *Dept Email*,
  then *Inst* → *Dept* and *Inst* → *Email*.
- If A → B and B C → D hold, then A C → D also holds.
  Example: If *Inst* → *Dept* and *Dept Course* → *ECTS*, then *Inst Course* → *ECTS*.

Why should you care?

- Certain normal forms (forms that prevent anomalies) depend on functional dependencies.
- The conditions for normal forms typically require you to consider all functional dependencies.
- All functional dependencies for a relation is an exponentially growing set of FDs.
- Knowing rules to infer one FD from others helps us by reducing the number of dependencies that we need to consider.

  For example, if we are looking for functional dependencies that does not hold, we can easily eliminate all trivial FDs (*Course ECTS* → *Course*), or if we know *Inst* → *Email Address*, we do not need to consider *Inst* → *Email* and *Inst* → *Address*.

Keys . . . again

Superkey is a set of attributes, that uniquely identify a record for a given relation.

Candidate key (or simply 'key') is a minimal set of attributes, that uniquely identify a record for a given relation. A key is the set of attributes form a superkey where unnecessary attributes removed.

Primary key is a key which is chosen by the DB designer.

For the schema addres(street_addr, postcode, city):

- Is {*street_addr, postcode, city*} a superkey(✔)/key(✗)
- Is {*street_addr, postcode*} a superkey(✔)/key(✔)
- Is {*street_addr, city*} a superkey(✔)/key(✔)
- Is {*postcode*} a superkey(✗)/key(✗)

Functional dependencies and keys

Where do the keys come from?

A set of attributes K is a key, if for all attributes A, functional dependency $K \rightarrow A$ holds, and K is a minimal set of attributes with this property.

Where do the functional dependencies come from?

We assert them based on our knowledge about entities/relationships that we are modeling.

Boyce-Codd normal form (BCNF)

A relation is in Boyce-Codd normal form if for any non-trivial functional dependency $A \rightarrow B$, A is a superkey.

- In other words, the BCNF says that all functional dependencies should involve keys.
- If you have functional dependencies that violate BCNF, then there is a sub-structure in the relation.
- It does not solve all problems of DB design, but the BCNF eliminates most sources of redundancy.
- The BCNF is one of the most common normal forms DB design practice aims to achieve.

Is this table in BCNF?

Inst	*Dept.*	*Email*	*Course*	*ECTS*
Ubbo Emmius	History	u.emmius@rug.nl	Frisian Hist.	5
Frits Zernike	Physics	f.zernike@rug.nl	Optics	3
Frits Zernike	Physics	f.zernike@rug.nl	Intr. to Physics	10

Let's consider a few functional dependencies.

- Inst Dept Course ECTS Email $\rightarrow$ Email . . . holds, but says nothing: left side is a superkey.
- Inst Dept Course $\rightarrow$ Email . . . holds, but says nothing: left side is a superkey.
- Course $\rightarrow$ ECTS . . . holds, and proves that table is not in BCNF: left side is not a superkey.

BCNF: decomposition

Is this decomposition good?

Instructor	*Dept.*	*Email*
Ubbo Emmius	History	u.emmius@rug.nl
Frits Zernike	Physics	f.zernike@rug.nl

- *Inst* $\rightarrow$ *Dept Email* holds, Inst is a (super)key.
- *Email* $\rightarrow$ *Inst Dept* holds, Email is a (super)key.
- *Dept* $\rightarrow$ *Inst* doesn't hold.
- *Dept* $\rightarrow$ *Email* doesn't hold.

$\rightarrow$ relation is in BCNF.

Instructor	*Course*	*ECTS*
Ubbo Emmius	Frisian Hist.	5
Frits Zernike	Optics	3
Frits Zernike	Intr. to Physics	10

- *Inst Course* $\rightarrow$ *ECTS* holds, {Inst, Course} is a (super)key.
- *Course* $\rightarrow$ *ECTS* holds, but Course is not a superkey.

$\rightarrow$ relation is **not** in BCNF.

BCNF: a trivia

Any relation (table) with two attributes (columns) is in BCNF.

Consider a relation *r(A,B)*, all possible non-trivial functional dependencies of concern are: $A \rightarrow B$, $B \rightarrow A$ (why?)

	Key			
	AB	A	B	A & B
$A \rightarrow B$	cannot hold	has to hold	cannot hold	has to hold
$B \rightarrow A$	cannot hold	cannot hold	has to holds	has to hold

Rules for decomposition

Why not decomposing everything to two-row tables?

Aside form not upsetting people who use the database, we want our decomposition to,

- produce tables that are in the desired normal form, for example, BCNF.
- allow lossless join: we should be able to recover the original table by joining the new tables.
- be dependency preserving: the functional dependencies that exist in the original table should be present in the resulting tables.

An example decomposition (bad)

Inst	*Dept.*	*Email*	*Course*	*ECTS*
Ubbo Emmius	History	u.emmius@rug.nl	Frisian Hist.	5
Frits Zernike	Physics	f.zernike@rug.nl	Optics	3
Frits Zernike	Physics	f.zernike@rug.nl	Intr. to Physics	10

⇓

Instructor	*Dept.*	*Email*
Ubbo Emmius	History	u.emmius@rug.nl
Frits Zernike	Physics	f.zernike@rug.nl

Course	*ECTS*
Frisian Hist.	5
Optics	3
Intr. to Physics	10

Both tables are in BCNF, what is wrong with this decomposition?

Another example decomposition

Is this table in BCNF?

`address(street_addr, postcode, city)`

We identify the following FDs:
street_addr city ⟶ *postcode*   *postcode* ⟶ *city*
! *postcode is not a superkey*

Let's decompose:

`addr1(street_addr, postcode) & addr2(postcode, city)`

This is (trivially) in BCNF, but what happened to FD: *street_addr city* ⟶ *postcode* ?

BCNF decomposition

It is possible to decompose any table into multiple tables such that the resulting tables are in BCNF, and decomposition lossless.

An algorithm:

1. Find an FD $A \rightarrow \beta$ that violates BCNF, where A is a set of attributes and β is a single attribute.
2. Decompose the relation into $A\beta$ and C, where C consists of the all attributes except β.
3. Test the resulting tables for BCNF, repeat the above steps for the new tables that are not in BCNF.

Note that there is no guarantee that the result will be dependency preserving.

Dependency preservation is problematic if there are multiple overlapping keys.

BCNF: a summary

The formal definition:

A relation is in Boyce-Codd normal form if for any non-trivial functional dependency $A \rightarrow B$, A is a superkey.

A more intuitive definition (due to Bill Kent/Chris Date):

Each attribute must represent a fact about the key, the complete key, and nothing but the key.

- BCNF eliminates most (but not all!) causes of redundancy/inconsistency.
- A table can be split into multiple tables in a lossless way. However, there is no guarantee of dependency preservation.

Third normal form: one step back

A relation is in third normal form (3NF) if for any non-trivial FD $A \rightarrow B$ one of the following holds.

1. A is a superkey.
2. B-A (attributes in B but not in A) is part of a key.

The intuitive definition of BCNF3NF:

Each non-key attribute must represent a fact about the key, the complete key, and nothing but the key.

- 3NF is less strict than BCNF.
- It may be desirable in cases where BCNF decomposition is not dependency preserving.

3NF example

Is this table in BCNF/3NF?

`address(street_addr, postcode, city)`

Note that {`street_addr, postcode`} and {`street_addr, city`} are keys.

FD	BCNF	3NF
street_addr city ⟶ *postcode*	OK	OK
postcode ⟶ *city*	not OK	OK

Normal forms: an interim summary

- We studied three normal forms, 1NF, BCNF and 3NF, that set rules about good database design.
- 1NF: domains of attributes should be atomic.
- 3NF: Each non-key attribute must represent a fact about the key, the complete key, and nothing but the key.
- BCNF: Each attribute must represent a fact about the key, the complete key, and nothing but the key.
- What happened to 2NF? It is a relaxed form of 3NF, it is mostly considered to be a historical artifact.
- Are we done? No, even BCNF does not prevent all forms of redundancy/inconsistencies.

Do we need more than BCNF?

Is there anything wrong with this table?

Instructor	*Dept*	*Phone*
Ubbo Emmius	History	1111
Ubbo Emmius	Greek	1111
Frits Zernike	Physics	1111
Frits Zernike	Physics	2222

- This schema is in BCNF? (why?)
- It clearly replicates data, we solve the problem by decomposing it into `ins1(name, dept)` and `ins1(name, phone)`.
- But we do not have a principled way of detecting the anomaly.
- Fourth normal form (4NF) is the principled solution we are looking for.

Multi valued dependencies

For set of attributes that X, Y, and Z that form a relation, A multivalued dependency (MVD)

$$X \twoheadrightarrow Y$$

means that given a set of values for X, Y can have multiple values, but the values of Y are independent of values of Z. (see the textbook for the formal definition) Given the relation

`ins_dept(name, dept, phone)`

- `name` ⟶ `dept` ✗
- `name` ⟶ `phone` ✗
- `dept` ⟶ `phone` ✗

- `name` ↠ `dept` ✓
- `name` ↠ `phone` ✓
- `dept` ↠ `phone` ✗

Note: every FD is an MVD, but not every MVD is an FD.

Fourth normal form (4NF)

A relation is in fourth normal form (4NF) if for any non-trivial multivalued dependency $A \twoheadrightarrow B$, A is a superkey.

Back to

`ins_dept(name, dept, phone)`

- We know that it is in BCNF.
- Is it in 4NF?
  - `name` $\twoheadrightarrow$ `dept`
  - but `name` is not a superkey

  ⇒ the relation is not in 4NF.

Note: every relation that is in 4NF is also in BCNF, but the reverse is (obviously) not true.

Normal forms: the list so far

- 1NF domains of attributes should be atomic.
- 3NF Each non-key attribute must represent a fact about the key, the complete key, and nothing but the key.
- BCNF Each attribute must represent a fact about the key, the complete key, and nothing but the key.
- 4NF is a further restriction over the BCNF which eliminates more cases of redundancy/inconsistency.

Are we done with the normal forms?
We are, but there are higher (more strict) normal forms that we will not study.

Normal forms: a summary

- Normal forms state (formal) rules that a DB table meet so that it is guarded against certain forms of redundancy/inconsistency.
- Once we detect a violation of a normal form, we decompose tables into smaller tables until all conform to the normal form.
- While splitting the tables, we seek lossless and dependency preserving decomposition.
- Trying to achieve 3NF, BCNF or 4NF is common in practice.
- Sometimes normal forms are intentionally violated for reasons of performance, which is called denormalization. (We will return to this in our discussion of SQL.)
- Higher forms exist, but they cover rather rare problematic cases and complicated to understand and apply.

Normal forms . . . but why?

- A good conceptual (E-R) design eliminates most problems of redundancy/inconsistency, but not all.
  - There are many subjective decision in E-R design that can go wrong. Checking result of E-R design for normal forms may discover poor E-R choices.
  - Even a good E-R design may result in a poor DB schema. Things to watch out: many-to-many relationship sets and multivalued attributes.
- Sometimes you need to start from the data, a design process from the start is not available.

Normal forms: what do you need to know

- Identify functional dependencies and multivalued dependencies that exist in a table schema.
- Identify whether a table is in 1NF, 3NF, BCNF or 4NF.
- Be able to do simple decompositions to meet the requirements of one of these normal forms.

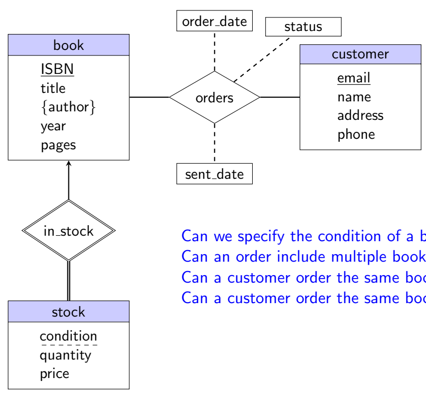
What is next?

- Reading for next week: Introduction to SQL (Chapter 3).
- Homework: online, due next Monday before the course.
- Lab: second part of the homework, mostly SQL exercises.

HW1: requirements

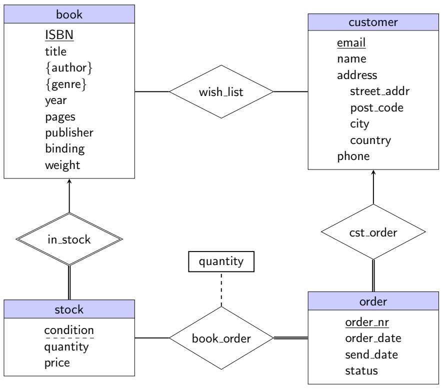
- Database should store bibliographical information on books.
- The catalog should be browsable by author, genre, and alphabetical order.
- Customer can order multiple books at once.
- A single order is shipped in a single package.
- Posting price is determined by the weight.
- Customer information should be kept.
- Information on past orders orders should be kept.
- Customers should be able to create 'wish lists'.

E-R design: a first try

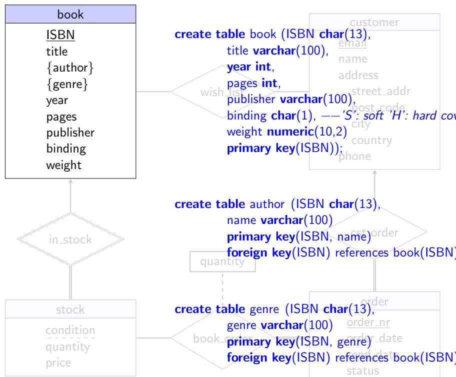


Can we specify the condition of a book in an order?
Can an order include multiple books?
Can a customer order the same book twice?
Can a customer order the same book twice same day?

E-R design: a better solution

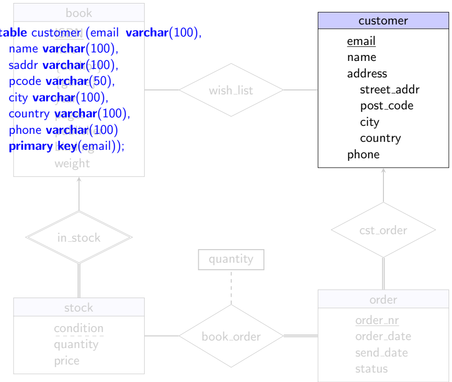


E-R to SQL: the book entity



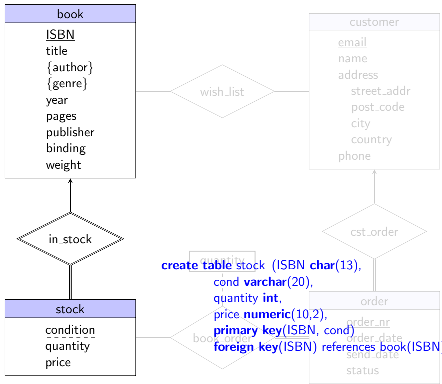

```
create table book (ISBN char(13),
    title varchar(100),
    year int,
    pages int,
    publisher varchar(100),
    binding char(1), -- 'S': soft 'H': hard cover
    weight numeric(10,2)
    primary key(ISBN));

create table author (ISBN char(13),
    name varchar(100)
    primary key(ISBN, name)
    foreign key(ISBN) references book(ISBN));

create table genre (ISBN char(13),
    genre varchar(100)
    primary key(ISBN, genre)
    foreign key(ISBN) references book(ISBN));
```


E-R to SQL: the customer entity



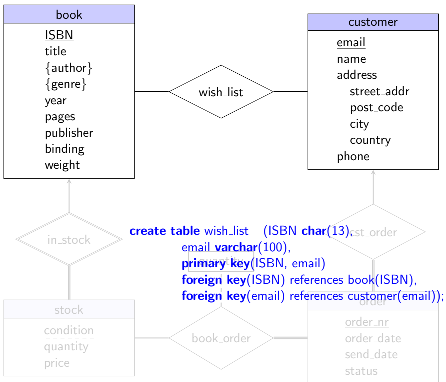

```
create table customer (email varchar(100),
    name varchar(100),
    saddr varchar(100),
    pcode varchar(50),
    city varchar(100),
    country varchar(100),
    phone varchar(100)
    primary key(email));
```


E-R to SQL: the stock entity set



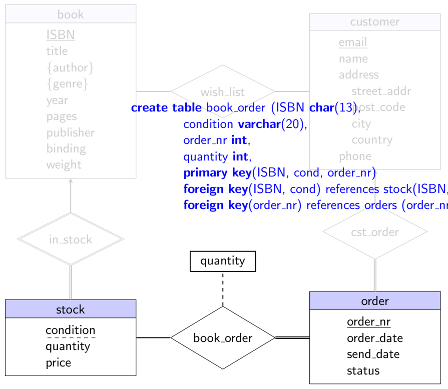

```
create table stock (ISBN char(13),
    cond varchar(20),
    quantity int,
    price numeric(10,2),
    primary key(ISBN, cond)
    foreign key(ISBN) references book(ISBN));
```


E-R to SQL: the wish_list relationship set



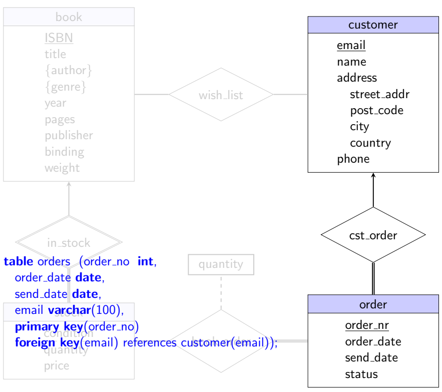

```
create table wish_list (ISBN char(13),
    email varchar(100),
    primary key(ISBN, email)
    foreign key(ISBN) references book(ISBN),
    foreign key(email) references customer(email));
```


E-R to SQL: the book_order relationship set




```
create table book_order (ISBN char(13),
    condition varchar(20),
    order_nr int,
    quantity int,
    primary key(ISBN, cond, order_nr)
    foreign key(ISBN, cond) references stock(ISBN,cond),
    foreign key(order_nr) references orders (order_nr));
```


E-R to SQL: the order entity and cst_order relationship set




```
create table orders (order_no int,
    order_date date,
    send_date date,
    email varchar(100),
    primary key(order_no)
    foreign key(email) references customer(email));
```