

Summary: last week

Database-enabled web technology
A review of DB design and SQL

Instructor: Çağrı Çöltekin
c.coltekin@rug.nl

Information science/Informatiekunde

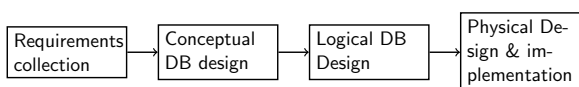
Fall 2011/12

Last week:

Two quick introductions:

- ▶ PHP: the language.
- ▶ Git: basics.

Database Design Process

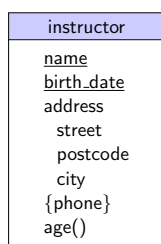


ideas high-level design DB schema database

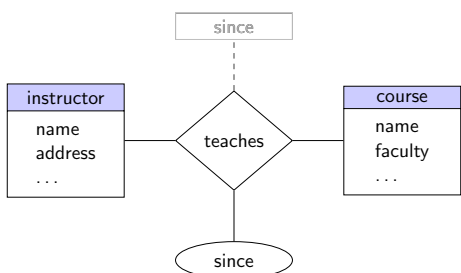
- ▶ DB design is generally part of a bigger software design process.
- ▶ These steps reflect the idealized case. Typically, you may need to re-iterate over some of the steps multiple times.

The entity sets (the textbook notation)

- ▶ Entity sets are represented with split rectangles.
- ▶ The top part is name of the entity set, bottom part lists the attributes.
- ▶ Attributes that form the primary key are underlined.
- ▶ Composite attributes are listed as part of them indented.
- ▶ Multi-valued attributes are listed in curly braces.
- ▶ Derived attributes are suffixed with '()'.



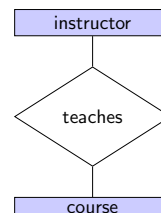
Relationship sets



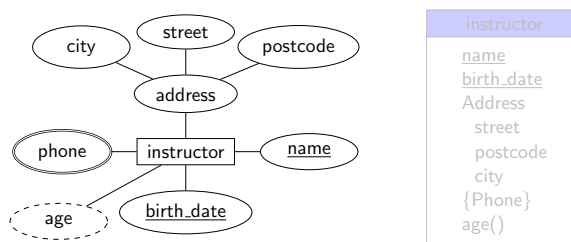
The entity-relationship data model

The entity-relationship (E-R) model is commonly used for specifying high-level database design.

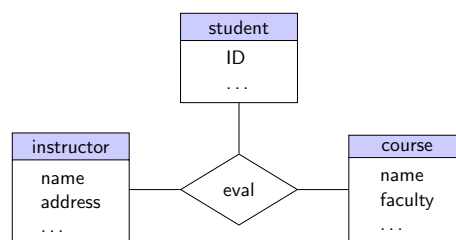
- ▶ An E-R model specifies real-world entities (objects, things) and their relations.
- ▶ It has a diagrammatic representation. That particularly comes handy in communicating your design with non-specialists.



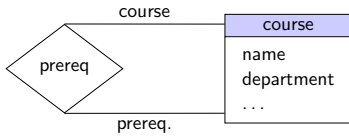
The entity sets (more common notation)



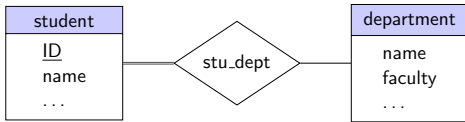
N-ary relations



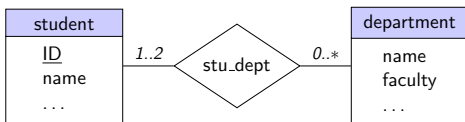
Recursive relationship sets/roles



Participation constraints

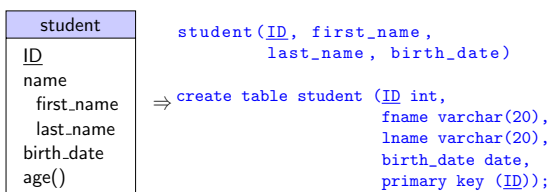


Alternative notation for constraints

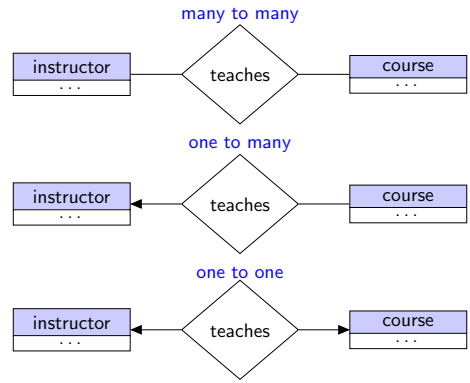


- ▶ Lower limit of 1 means that the participation is total.
- ▶ Upper limit of 1 on both sides means a one-to-one relationship set.
- ▶ Upper limit of 1 only on the right side means a one-to-many relationship set.

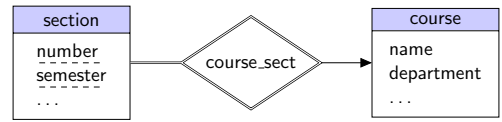
E-R entity to relation schema (example)



Mapping cardinalities

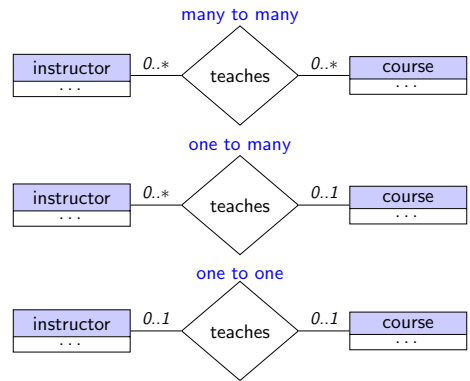


Weak entities

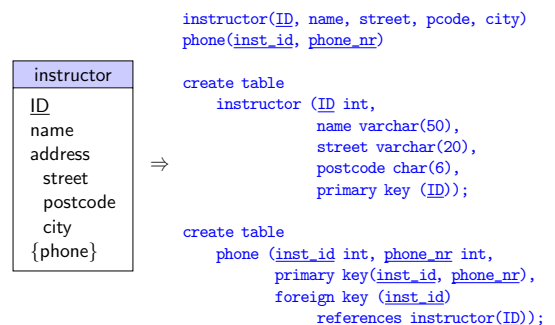


Mapping cardinalities (example)

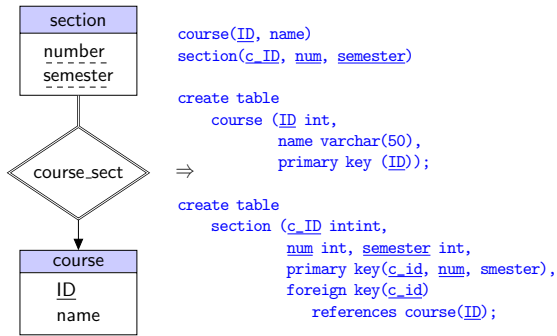
Mapping cardinalities (example)



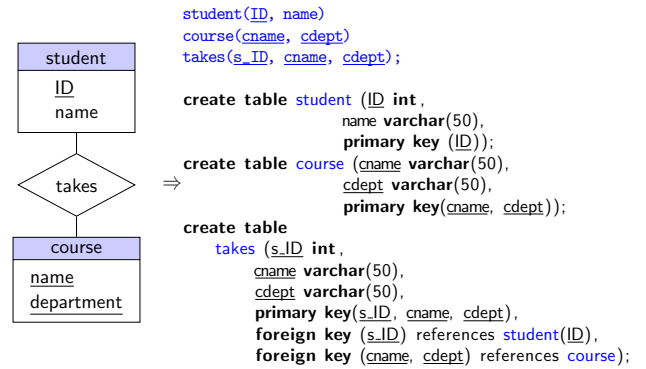
E-R multi-valued attribute to relation schema (example)



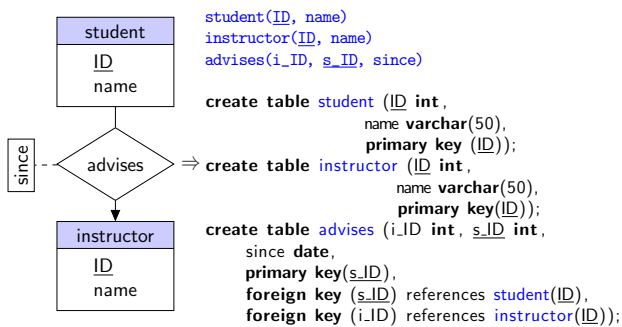
E-R weak entity to relation schema (example)



E-R relationship to relation schema (example)



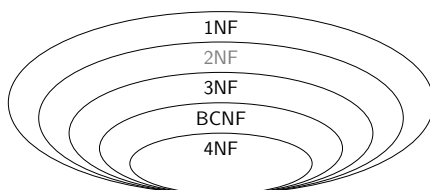
E-R relationship to relation schema (example 2)



An overall summary

- ▶ 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 notation is varied and non-standard, one can alternatively use another representation like UML.
- ▶ E-R constructs can be reduced to a database schema.
- ▶ Conceptual modeling is helpful, however, it does not guarantee correct relational database design.

Normal forms



- ▶ **Normal forms** are a set of formal rules that ensure certain types of anomalies do not occur.
- ▶ Except Boyce-Codd normal form (BCNF), they are named as **first normal form**, **second normal form**, ...
- ▶ Higher level normal forms are more strict, and require all lower level forms.

Rules of thumb

Conceptual database design is an art as much as it is a science. Generally, there is no single correct solution.

- ▶ Do try to represent the real-world as faithful as possible.
- ▶ Don't model same thing twice: avoid redundancy.
- ▶ Don't model things that are not needed: look for simpler solutions.

What do we want to avoid?

In DB design we want to avoid:

- 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.

Main cause of these anomalies are redundant/repeated storage of the same information.

Normal forms: a summary

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

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 '.

Boyce-Codd Normal form (BCNF)

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 **decomposed** into multiple tables to achieve BCNF.

Multi valued dependencies

For set of attributes that A, B , the multivalued dependency (MVD)

$$A \twoheadrightarrow B$$

holds if

- ▶ given the value of A , B may have multiple values
- ▶ but the set of values only depend on A ,
- ▶ and only A (no other attribute in the table affects the dependency)

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

Normal forms: a summary

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

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 the 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.

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 **BCNF 3NF**:

Every **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.

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.

- ▶ The definition of 4NF is similar to definition of BCNF, except we require MVDs to hold instead of FDs.
- ▶ 4NF eliminates most causes of anomalies, but (still) not all.

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

Some guidelines on DB design

- ▶ **Clear semantics**: it should be easy to explain the meaning of your DB schema.
 - ▶ use understandable names
 - ▶ do not overload your tables
- ▶ Avoid **redundancy and inconsistency**: a good DB design should prevent all anomalies. Follow the most strict normal form you can.
- ▶ Avoid **null** values.
- ▶ Joins should reference only keys.

SQL basics: summary (1)

- ▶ SQL includes a data definition language (DDL) and data manipulation language (DML) statements as well as being a database query language.
- ▶ The DDL statements include `create table`, `alter table` and `drop table`
- ▶ The DML statements include `insert into`, `update` and `delete from` statements.
- ▶ The SQL query language is closely related to formal query language [relational algebra](#).
- ▶ Relational algebra operations include, `selection` (σ), `projection` (π), `Cartesian product` (\times), `natural join` (\bowtie), other join operations such as `outer joins`, and set operations `union`, `intersection`, `set difference`.

SQL basics: summary (3)

- ▶ We can sort the output of an SQL query by adding an `order by` clause at the end of our queries.
- ▶ A set of aggregate functions, `count`, `sum`, `avg`, `max` and `min` can be used to gather statistics about certain column(s) of a query.
- ▶ The results of aggregate functions can be grouped together by `group by` clause.
- ▶ Set operations `union`, `intersection` and difference (`except`), can be used to combine the results of two queries.
- ▶ Sub-queries can be used in the `from` clause, or as an argument to `in`.

SQL and null values

Null values create a number of difficult cases in relational database theory.

- ▶ Arithmetic expressions involving `null` are `null` (`1 + null = null`).
- ▶ Any comparison (like `1 = null`, `1 < null`) involving nulls results in a third truth value: `unknown`.
- ▶ This includes the comparison `null = null`, except for set operations and for `distinct`.
- ▶ Expressions `is null` or `is not null` can be used to test if a value is null or not.
- ▶ Logical operations with unknown values:
 - ▶ true `and` unknown = unknown, false `and` unknown = false
 - ▶ true `or` unknown = true, false `or` unknown = unknown
 - ▶ `not` unknown = unknown
- ▶ Except `count(*)` aggregate functions ignore null values.

Natural Join

student				advisor		
sID	s_dept	year	aID	aID	a_dept	phone
1	IK	1	1	1	CIW	1111
2	CIW	2	2	2	IK	2222
3	IK	3	1	3	IK	3333
4	CIW	2	3	4	CIW	4444

`natural join` joins two or more tables using matching column names.

```
select sID, s_dept, aID, phone
from student natural join advisor;
```

sID	s_dept	aID	phone
1	IK	1	1111
2	CIW	2	2222
3	IK	1	1111
4	CIW	3	3333

You can join more than two tables using the same syntax:

```
t1 natural join t2 natural join t3 ...
```

SQL basics: summary (2)

- ▶ Basic form of SQL queries is:

```
select attribute1, ..., attributeN
from table_name1, ..., table_nameM
where condition;
```

- ▶ `from` clause lists the tables used in the query.
- ▶ `where` statement picks the rows we are interested in using predicates containing
 - ▶ comparisons: `=`, `<>`, `>`, `<`, `>=` and `<=`.
 - ▶ sub-strings match operator `like`.
 - ▶ logical operators `and`, `or` and `not`.
- ▶ `select` clause picks the columns we are interested in.
- ▶ `select` and `where` may include arithmetic operations, and string operations, `upper`, `lower`, and `concat`

SQL and distinct values

- ▶ By default, SQL does not eliminate duplicate records.
- ▶ Except in set operations `union`, `intersect` and `except`.
- ▶ For an SQL query to return only the distinct values, we need to add `distinct` keyword to select clause. For example, `select distinct name from student;`
- ▶ `distinct` keyword can also be used in aggregate functions. For example, `select count(distinct name) from student;`

Joins: summary

- ▶ A join is a combination of rows from multiple tables according to one or more related columns on each table.
- ▶ In SQL a join can either be specified implicitly in `where` clause, or explicitly in `from` clause.
- ▶ `Inner joins` are join operations which select only the tuples that meet the join condition.
- ▶ `Outer joins` allow tuples that do not meet the join condition to be included from one or both tables being joined.
- ▶ A `natural join` uses matching attribute names from each table.
- ▶ Joins can be restricted to certain columns with a `using` clause, or full join conditions can be specified using `on`.

Inner join:
accidental column match

student				advisor		
sID	dept	year	aID	aID	dept	phone
1	IK	1	1	1	CIW	1111
2	CIW	2	2	2	IK	2222
3	IK	3	1	3	IK	3333
4	CIW	2	3	4	CIW	4444

```
select sID, student.dept, aID, phone
from student join advisor using (aID);
```

sID	student.dept	aID	phone
1	IK	1	1111
2	CIW	2	2222
3	IK	1	1111
4	CIW	3	5555

Inner join: arbitrary column expressions

SQL: a short summary

student				advisor		
ID	s_dept	year	aID	ID	a_dept	phone
1	IK	1	1	1	CIW	1111
2	CIW	2	2	2	IK	2222
3	IK	3	1	3	IK	3333
4	CIW	2	3	4	CIW	4444

```
select student.ID, s_dept, advisor.ID, phone
from student join advisor on student.aID = advisor.ID;
```

student.ID	s_dept	advisor.ID	phone
1	IK	1	1111
2	CIW	2	2222
3	IK	1	1111
4	CIW	3	3333

on clause can take any expression allowed in a where clause.

Join conditions with an equation are sometimes called **equi-join** and join conditions with arbitrary comparisons are called **θ -join** (theta-join).

Left outer join

SQL: a short summary

student				advisor		
sID	s_dept	year	aID	aID	a_dept	phone
1	IK	1	1	1	CIW	1111
2	CIW	2	2	2	IK	2222
3	IK	3	1	3	IK	3333
4	CIW	2	null	4	CIW	4444

```
select * from student natural left outer join advisor;
```

sID	s_dept	year	aID	a_dept	phone
1	IK	1	1	CIW	1111
2	CIW	2	2	IK	2222
3	IK	3	1	CIW	1111
4	CIW	2	null	null	null

Right outer join

SQL: a short summary

student				advisor		
sID	s_dept	year	aID	aID	a_dept	phone
1	IK	1	1	1	CIW	1111
2	CIW	2	2	2	IK	2222
3	IK	3	1	3	IK	3333
4	CIW	2	null	4	CIW	4444

```
select * from student right outer join advisor using (aID);
```

sID	s_dept	year	aID	a_dept	phone
1	IK	1	1	CIW	1111
2	CIW	2	2	IK	2222
3	IK	3	1	CIW	1111
null	null	null	3	IK	3333
null	null	null	4	CIW	4444

Full outer join

SQL: a short summary

student				advisor		
sID	s_dept	year	aID	aID	a_dept	phone
1	IK	1	1	1	CIW	1111
2	CIW	2	2	2	IK	2222
3	IK	3	1	3	IK	3333
4	CIW	2	null	4	CIW	4444

```
select * from student full outer join advisor on student.aID = advisor.aID;
```

sID	s_dept	year	aID	a_dept	phone
1	IK	1	1	CIW	1111
2	CIW	2	2	IK	2222
3	IK	3	1	CIW	1111
4	CIW	2	null	null	null
null	null	null	3	IK	3333
null	null	null	4	CIW	4444

SQL and programming

- SQL has limited use unless combined with a general purpose programming language.
- SQL has the advantage that it abstracts away the way data is stored from the application.
- However, it cannot do many things that a typical application program would require. Just to list a few:
 - arbitrary computation
 - flexible I/O, user interaction
 - formatted input output
 - graphical presentation of data
- There are a number of ways to combine SQL and general purpose programming
 - On DB side: [stored procedures](#).
 - On application side: [embedded SQL](#), or [call-level interfaces](#)
- We will be using call-level interfaces in this course.

A first PHP/MySQL example

```
1 <?php
2     $host="hostname";
3     $user="username";
4     $pass="password";
5     $db="dbname";
6     mysql_connect($host,$user,$pass);
7     mysql_select_db($db);
8     $q = 'select * from book';
9
10    $res = mysql_query($q);
11    echo "<table border='1'>";
12    echo "<tr><th>ISBN</th><th>title</th></tr>";
13    while ($row = mysql_fetch_assoc($res)) {
14        echo "<tr><td>{$row['ISBN']}</td>";
15        echo "<td>{$row['title']}</td></tr>";
16    }
17    echo "</table>";
18    mysql_close();
19 >?
```

Summary

This week:

- A review of past course: DB design and SQL.
- A simple PHP/MySQL example.

Next week:

- What happens 'under the hood' when you do web-based programming?
- PHP and user interaction.