Database System Concepts for Non-Computer Scientists

Repetition & Exercise

Michael Jungmair

Chair for Database Systems, TUM

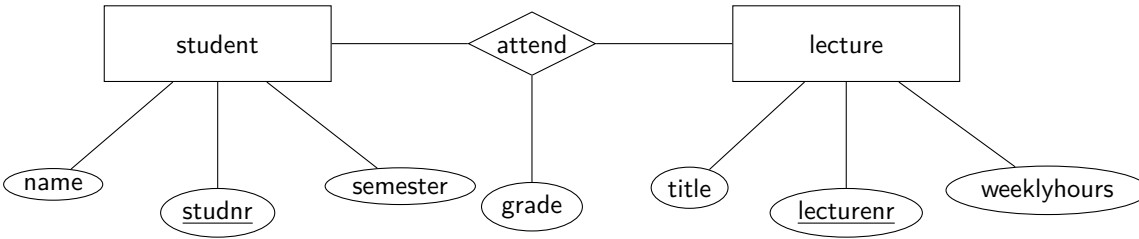
December 18, 2025

Content Overview

- We discuss the most important topics
- Everything discussed in the lecture is exam-relevant!

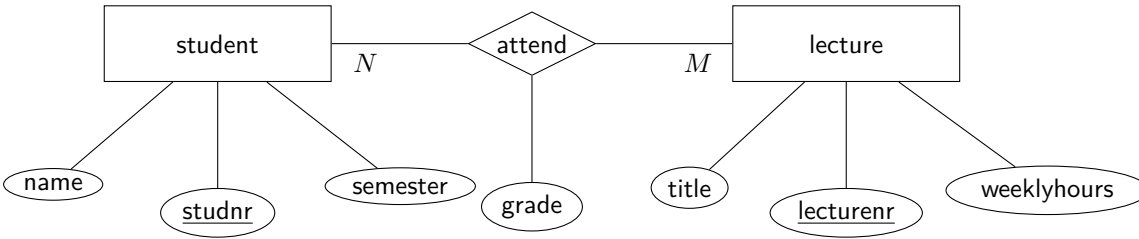
E/R modeling

- Underline keys!



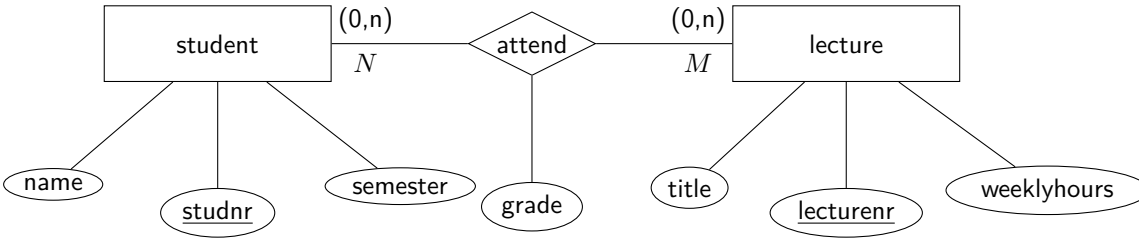
E/R modeling

- Underline keys!
- Functionalities

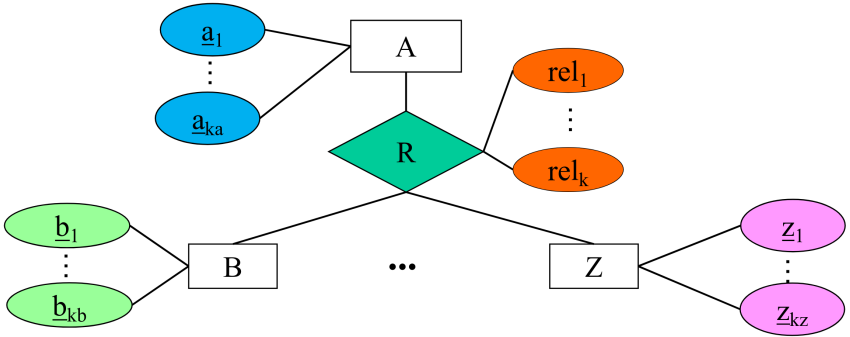


E/R modeling

- Underline keys!
- Functionalities
- Min/Max notation



Derive Relational Schema



$$R: \{[\ \underbrace{a_1,\ldots,a_{ka}}_{\text{Keys of A}}\ ,\ \underbrace{b_1,\ldots,b_{kb}}_{\text{Keys of B}}\ ,\ldots,\ \underbrace{z_1,\ldots,z_{kz}}_{\text{Keys of Z}}\ ,\ \underbrace{rel_1,\ldots,rel_k}_{\text{Attributes of R}}\ ]\}$$

Simplfication

1:N-Relationship (simple):

Professors : {[*PersNr, Name, Level, Room*]}

Lectures : {[*LectureNr, Title, WeeklyHours*]}

give: {[*PersNr , LectureNr*]}

Refinement by subsumption:

Professors : {[*PersNr, Name, Level, Room*]}

Lectures : {[*LectureNr, Title, WeeklyHours, given_by*]}

Create Table

- simple create table statement

```
create table students (
  studnr    integer,
  name      varchar(30),
  semester  integer
);
```

Create Table

- simple create table statement

```
create table students (
  studnr   integer,
  name     varchar(30),
  semester integer
);
```

- constraints: primary key, unique

```
create table professors (
  persnr integer primary key,
  name   varchar(30) not null,
  room   integer unique
);
```

Create Table

- simple create table statement

```
create table students (
  studnr   integer,
  name     varchar(30),
  semester integer
);
```

- constraints: primary key, unique

```
create table professors (
  persnr integer primary key,
  name   varchar(30) not null,
  room   integer unique
);
```

- data types: int, numeric/decimal, char, varchar/text, float, date/timestamp

Create Table: Foreign keys

- constraints: foreign key

```
create table lectures (
  lecturenr integer primary key,
  title     varchar(30),
  givenby   integer,
  foreign key (givenby) references professors(persnr)
);
```

- short form:

```
create table lectures (
  lecturenr integer primary key,
  title     varchar(30),
  givenby   integer  references professors
);
```

Basic structure

- select from where

```
select name, semester
from students
where semester >= 10;
```

Basic structure

- select from where

```
select name, semester
from students
where semester >= 10;
```

- joins

```
select s.name, l.title
from students s, attend a, lectures l
where s.studnr = a.studnr and a.lecturenr = l.lecturenr;
```

Basic structure

- select from where

```
select name, semester
from students
where semester >= 10;
```

- joins

```
select s.name, l.title
from students s, attend a, lectures l
where s.studnr = a.studnr and a.lecturenr = l.lecturenr;
```

- order by

```
select name, semester
from students
order by semester desc, name asc;
```

Expressions

- three valued logic

```
1 -- null comparisons evaluate to unknown
2 select studnr, name
3 from students
4 where semester = null;
5
6 -- correct:
7 -- where semester is null;
```

Expressions

- three valued logic

```
-- null comparisons evaluate to unknown
select studnr, name
from students
where semester = null;

-- correct:
-- where semester is null;
```

- expressions: between, like, not

```
where semester between 2 and 6
and name like 'A%'
and matrnr <> 5
```

Aggregations

- sum(), avg(), max(), ...

```
1 select avg(grade), min(grade), max(grade)
2 from test;
```

Aggregations

- sum(), avg(), max(), ...

```
1 select avg(grade), min(grade), max(grade)
2 from test;
```

- group by, having

```
1 select persnr, avg(grade)
2 from test
3 group by persnr
4 having count(*) > 5
```

Outer Joins

- outer joins

```
1 select l.title, a.studnr
2 from lectures l
3 left outer join attend a
4   on a.lecturenr = l.lecturenr;
```

Outer Joins

- outer joins

```
select l.title, a.studnr
from lectures l
left outer join attend a
  on a.lecturenr = l.lecturenr;
```

- count(*) vs count(col)

```
select count(*) as all_assistants,
       count(area) as not_fresh_assistants
from assistants;
```

Nested Queries

- direct comparison with scalar results

```
select * from test
where grade = (select min(grade) from test);
```

Nested Queries

- direct comparison with scalar results

```
select * from test
where grade = (select min(grade) from test);
```

- in

```
select name from students
where studnr in ( ... )
```

Nested Queries

- direct comparison with scalar results

```
select * from test
where grade = (select min(grade) from test);
```

- in

```
select name from students
where studnr in ( ... )
```

- exists/not exists

```
select title from lectures l
where not exists (
  select * from attend a where a.lecturenr = l.lecturenr
);
```

WITH Statement

- with x as ...

```
with attcounts as (
  select lecturenr, count(*) as cnt
  from attend
  group by lecturenr
)
select l.title, attcounts.cnt
from lectures l
left join attcounts
  on attcounts.lecturenr = l.lecturenr;
```

Set Operations

- union / intersect / except

```
select studnr
from attend
where lecturenr = 5001
union
select studnr
from attend
where lecturenr = 5041;
```

Set Operations

- union / intersect / except

```
select studnr
from attend
where lecturenr = 5001
union
select studnr
from attend
where lecturenr = 5041;
```

- matching schema!

Set Operations

- union / intersect / except

```
select studnr
from attend
where lecturenr = 5001
union
select studnr
from attend
where lecturenr = 5041;
```

- matching schema!
- union all vs union

insert statements

- insert values

```
1 insert into students (studnr, name, semester)
2 values (30000, 'ada', 1);
```

insert statements

- insert values

```
insert into students (studnr, name, semester)
values (30000, 'ada', 1);
```

- insert from queries

```
-- insert all students in semester >= 10 into a new table
insert into senior_students (studnr, name)
select studnr, name
from students
where semester >= 10;
```

delete and update

- delete

```
1 -- delete all attendances of a student
2 delete from attend
3 where studnr = 30000;
```

delete and update

- delete

```
-- delete all attendances of a student
delete from attend
where studnr = 30000;
```

- update

```
-- increase semester by 1 for all students
update students
set semester = semester + 1;
```

Index Structures

- Why do we need index structures?

Index Structures

- Why do we need index structures?
- How to create index for data

```
1 CREATE INDEX full_name ON Person (Last_Name, First_Name)
```

Security

- Access rights can be managed

Security

- Access rights can be managed
- Views: Give access to subset/aggregated data, while protect raw data

Security

- Access rights can be managed
- Views: Give access to subset/aggregated data, while protect raw data
- SQL Injections: Avoid with prepared statements

Exercise 1

Consider the following requirements for a database for a movie streaming web site:

- Movies have a unique title and a production year.
- Actors have a unique id, a name and a gender.
- An actor can star in any number of movies, in different roles.
- A Genre is defined by its name.
- A movie has between 1 and 5 genres.
- Each movie is age-rated with regards to multiple systems (e.g., FSK, MPAA)

a) Create an E/R diagram for these requirements
b) Annotate functionalities and min/max
c) Derive a relational schema, apply simplifications
d) Write SQL-DDL statements to create database tables

Exercise 2

Evaluate the following query manually on the attached instantiation of the university schema. Please give attribute/column names and values in form of a table

```
select p.name as name,
       count(l2.lectureNr) as lec_cnt,
       sum(l2.weeklyhours) as hours_sum
from Professors p
  left outer join lectures l1 on p.persNr = l1.given_by
  left outer join require r on l1.lectureNr = r.successor
  left outer join lectures l2 on r.predecessor = l2.lectureNr
where p.level = 'C4'
  and p.name <> 'Curie' and p.name <> 'Russel'
group by p.persNr, p.name
order by p.name
```

Exercise 3

Calculate the average grade students received in the ‘Ethik’ lecture. Expected column in result: average grade

Exercise 4

Find the student (or students) with the best grade. (Expected columns in result: student name, student number, grade, the title of the lecture, and the name of the professor; one student may occur multiple times)

Exercise 5

a) Name one famous relational database system.

b) Give two good reasons for using a database system and briefly explain why.