Database Management Systems INFO 210

SQL Part II

Lecture 9

Franz Wotawa
TU Graz, Institut for Software Technologie
Inffeldgasse 16b/2
wotawa@ist.tugraz.at

Example DB schema

- DB schema “lecturedatabase”:

```
lecture(lnr, lname, hours, lh_name, t_id)
type(id)
lecturehall(name, address)
lecturer(ssnr, name)
student(srnr, name)
participating_students(lnr, srnr)
lecturer_assignment(lnr, ssnr)
```

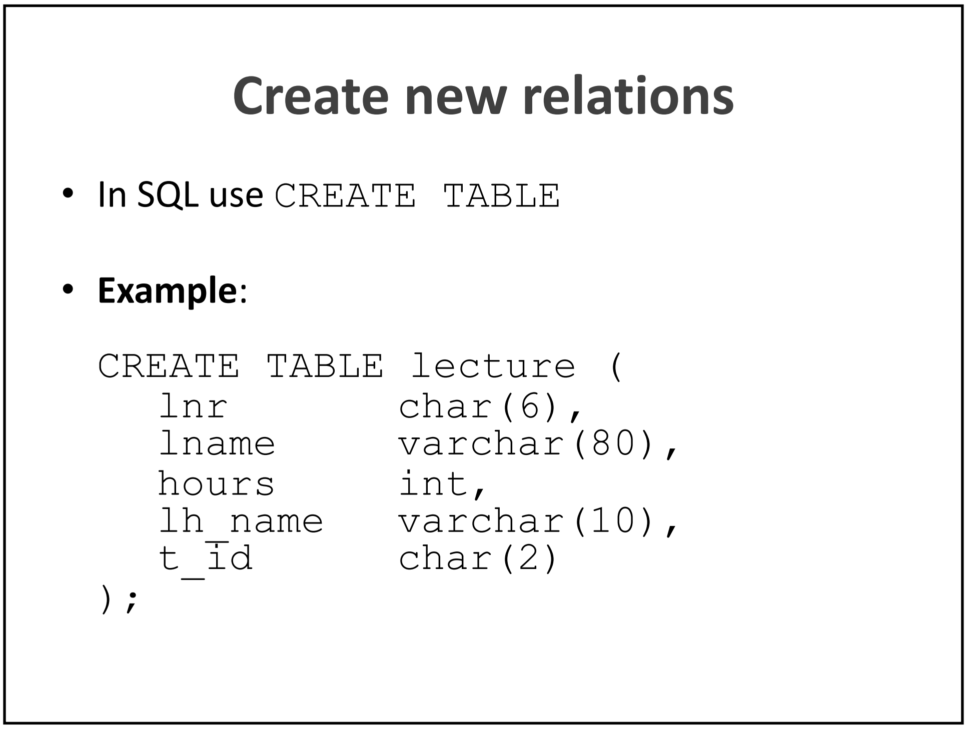
Database implementations

- **Commercial**:
  - Oracle Database
  - Microsoft Access Database
  - ...
- **Open-Source**:
  - MySQL (www.mysql.de)
  - **PostgreSQL (www.postgresql.org)**
    - Latest release: PostgreSQL 12
    - Direct access using the command line tool `psql`
    - Access from `Java` via `JDBC`
  - ...

Create new relations

- Every SQL attribute has a data type
- In SQL there are the following predefined data types available (and even more):

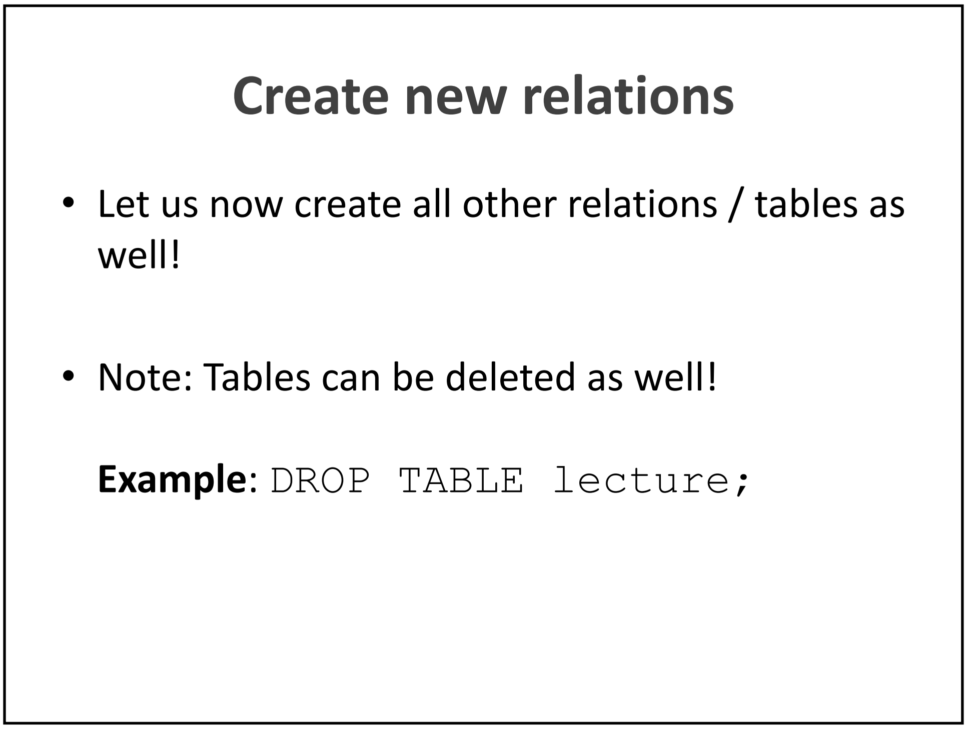
Data type	Explanation
bool	A Boolean value
int	4 byte integer value
real	Single precision real value (4 byte)
float8	Double precision real (8 byte)
char(N)	Fixed length character string of size N
varchar(N)	Variable length character string up to size N
date	The date
time	The time



Create new relations

- In SQL use `CREATE TABLE`
- **Example**:

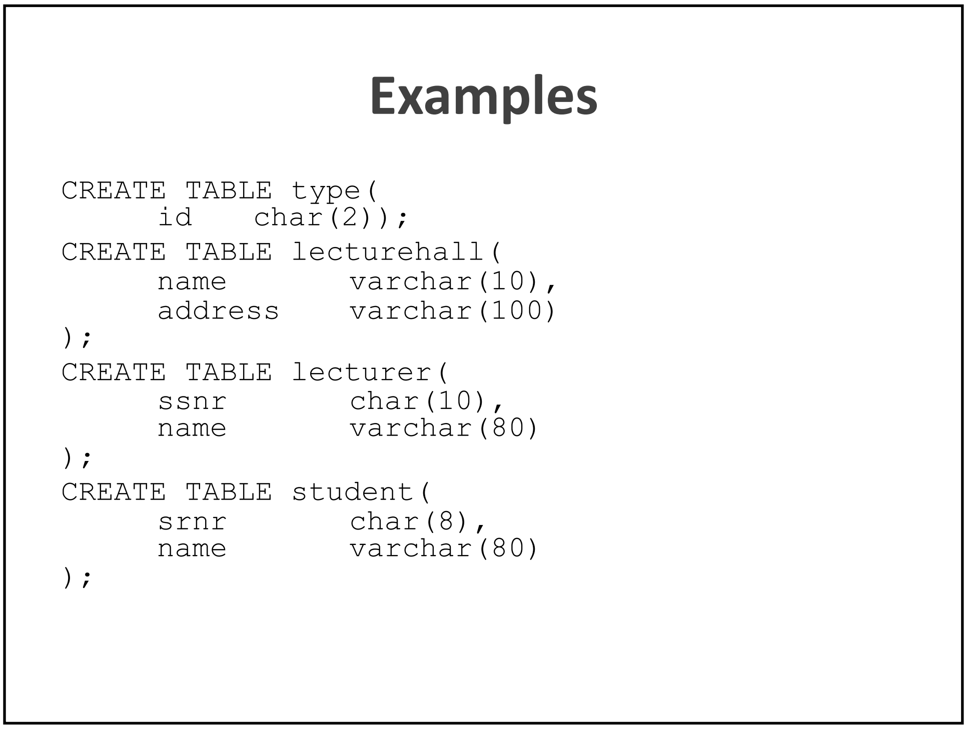
```
CREATE TABLE lecture (
   lnr        char(6),
   lname      varchar(80),
   hours      int,
   lh_name    varchar(10),
   t_id       char(2)
);
```



Create new relations

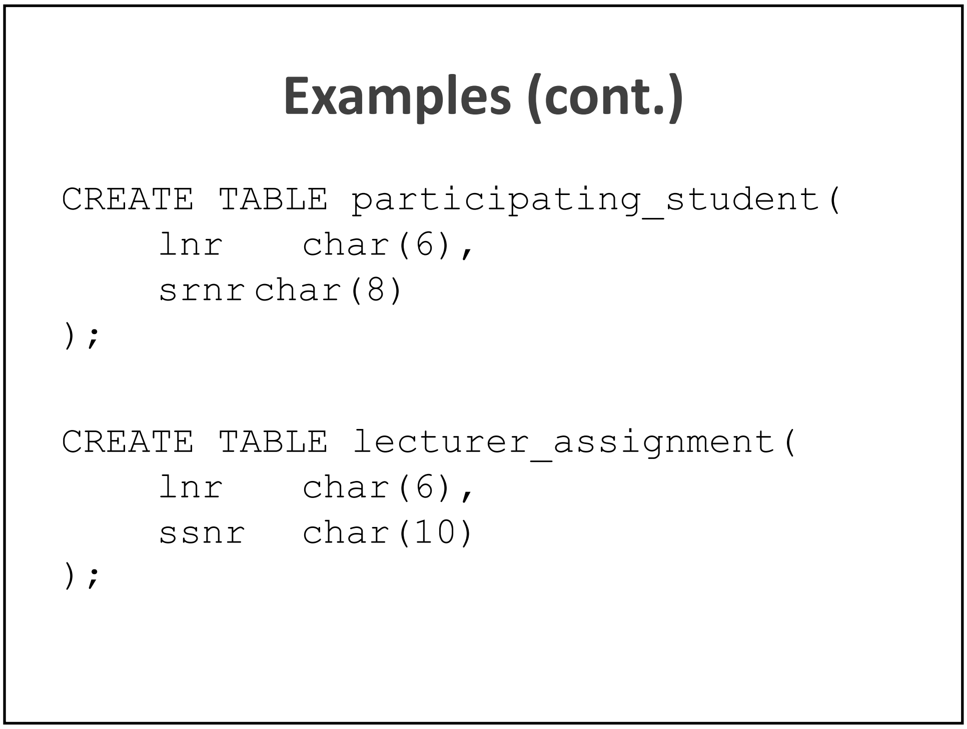
- Let us now create all other relations / tables as well!
- Note: Tables can be deleted as well!

  Example: `DROP TABLE lecture;`



Examples

```
CREATE TABLE type(
     id   char(2));
CREATE TABLE lecturehall(
     name       varchar(10),
     address    varchar(100)
);
CREATE TABLE lecturer(
     ssnr       char(10),
     name       varchar(80)
);
CREATE TABLE student(
     srnr       char(8),
     name       varchar(80)
);
```

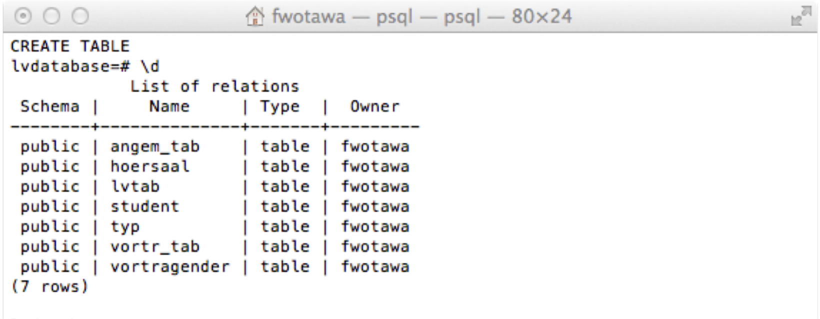


Examples (cont.)

```
CREATE TABLE participating_student(
     lnr    char(6),
     srnr char(8)
);

CREATE TABLE lecturer_assignment(
     lnr    char(6),
     ssnr   char(10)
);
```

Example (cont.)



```
CREATE TABLE
lvdatabase=# \d
             List of relations
 Schema |     Name      | Type  |  Owner
--------+--------------+-------+---------
 public | angem_tab    | table | fwotawa
 public | hoersaal     | table | fwotawa
 public | lvtab        | table | fwotawa
 public | student      | table | fwotawa
 public | typ          | table | fwotawa
 public | vortr_tab    | table | fwotawa
 public | vortragender | table | fwotawa
(7 rows)
```

Insert data into tables

- Use the commands: `INSERT` or `COPY`
- **Examples**:

```
INSERT INTO lecturer
 VALUES ('1234567890', 'Franz
Wotawa');
```

or

```
INSERT INTO lecturer (name, ssnr)
VALUES ('Rui Abreu', '0987654321');
```

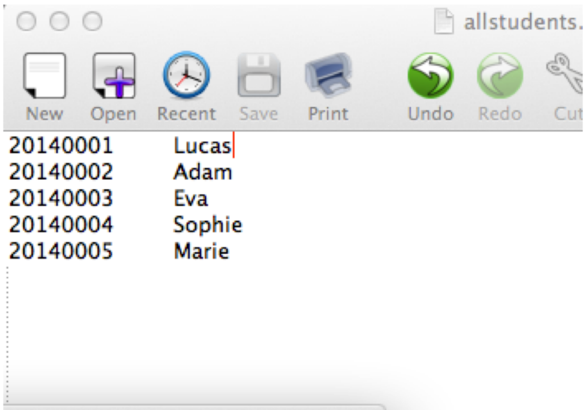
Insert data into tables

- With `COPY` all data stored in a text file can be added to a particular table.

- **Example**:

```
COPY student FROM
'allstudents.txt';
```

Structure of the text file



```
20140001      Lucas
20140002      Adam
20140003      Eva
20140004      Sophie
20140005      Marie
```

Search for data in SQL data bases – The SELECT statement

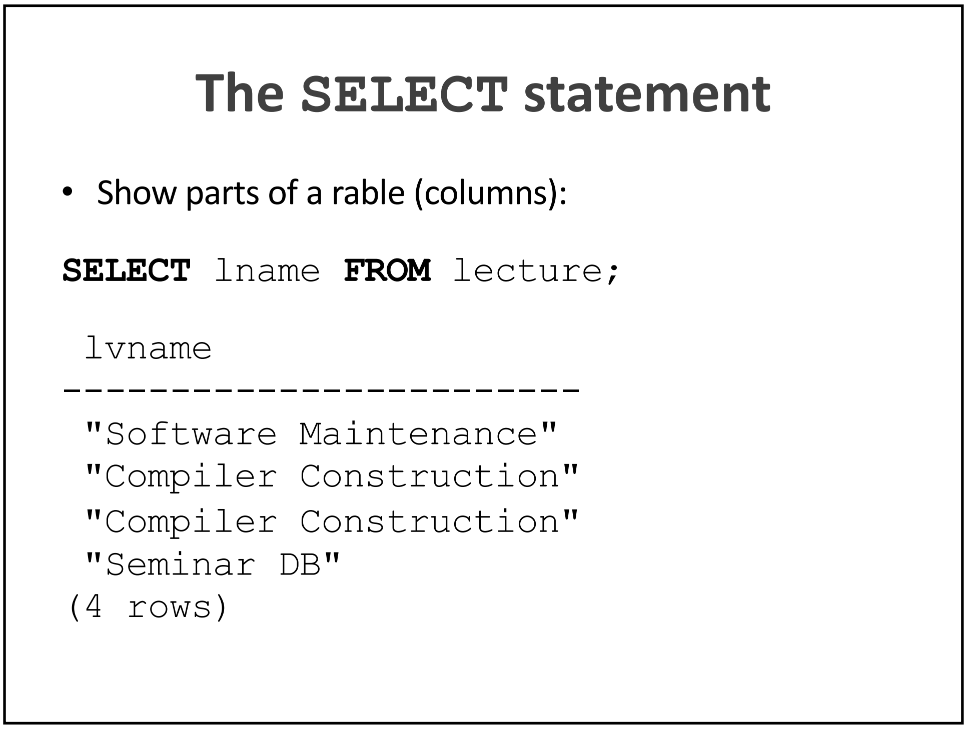
- As already discussed `SELECT` is for searching for data in relational databases using SQL.
- Present all data from a particular table:

```
SELECT * FROM student;

  srnr    |  name
----------+---------
 20140001 |  Lucas
 20140002 |  Adam
 20140003 |  Eva
 20140004 |  Sophie
 20140005 |  Marie
(5 rows)
```

Example table lecture

```
  lnr   |         lname          | hours     | lh_name |    t_id
--------+------------------------+-----------+---------+------------
 117101 | "Software Maintenance" |         3 | i7      | LP
 117102 | "Compiler Construction"|         2 | i11     | L
 117103 | "Compiler Construction"|         1 | i11     | P
 117104 | "Seminar DB"           |         2 | i12     | SE
(4 rows)
```

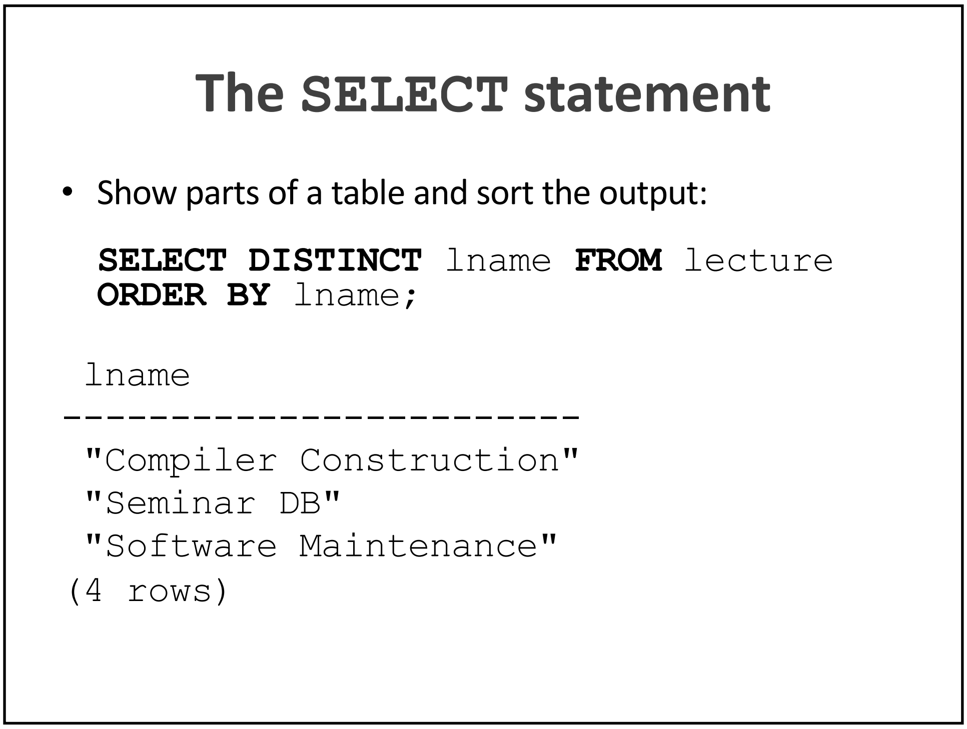


The SELECT statement

- Show parts of a rable (columns):

```
SELECT lname FROM lecture;

 lvname
------------------------
 "Software Maintenance"
 "Compiler Construction"
 "Compiler Construction"
 "Seminar DB"
(4 rows)
```



The SELECT statement

- Show parts of a table and sort the output:

```
  SELECT DISTINCT lname FROM lecture
  ORDER BY lname;

 lname
------------------------
 "Compiler Construction"
 "Seminar DB"
 "Software Maintenance"
(4 rows)
```

The SELECT statement

- Select tuples or rows of a table:

```
SELECT lname, t_id FROM lecture
WHERE hours > 2;
```

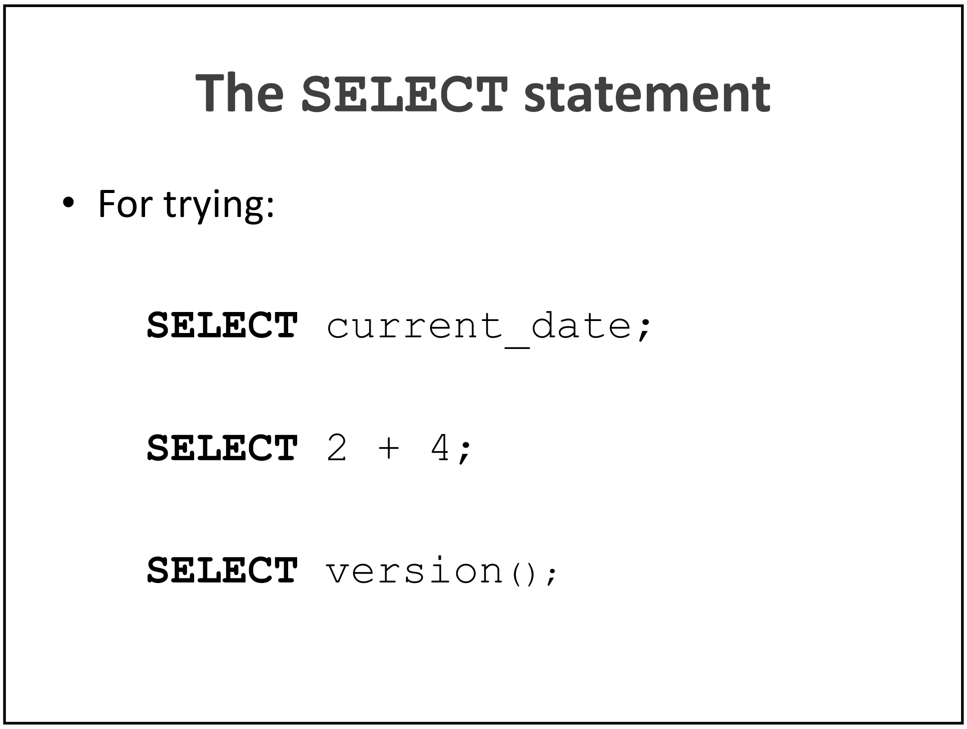
```
        lname           |    t_id
------------------------+------------
 "Software Maintenance" | LP
(1 row)
```

The SELECT statement

- Extended functionality:

```
SELECT * FROM lecture
WHERE hours > 2 AND
lh_name LIKE ’i7’;
```

```
  lnr   |         lname          | hours     | lh_name |    t_id
--------+------------------------+-----------+---------+------------
 117101 | "Software Maintenance" |         3 | i7      | LP
(1 row)
```



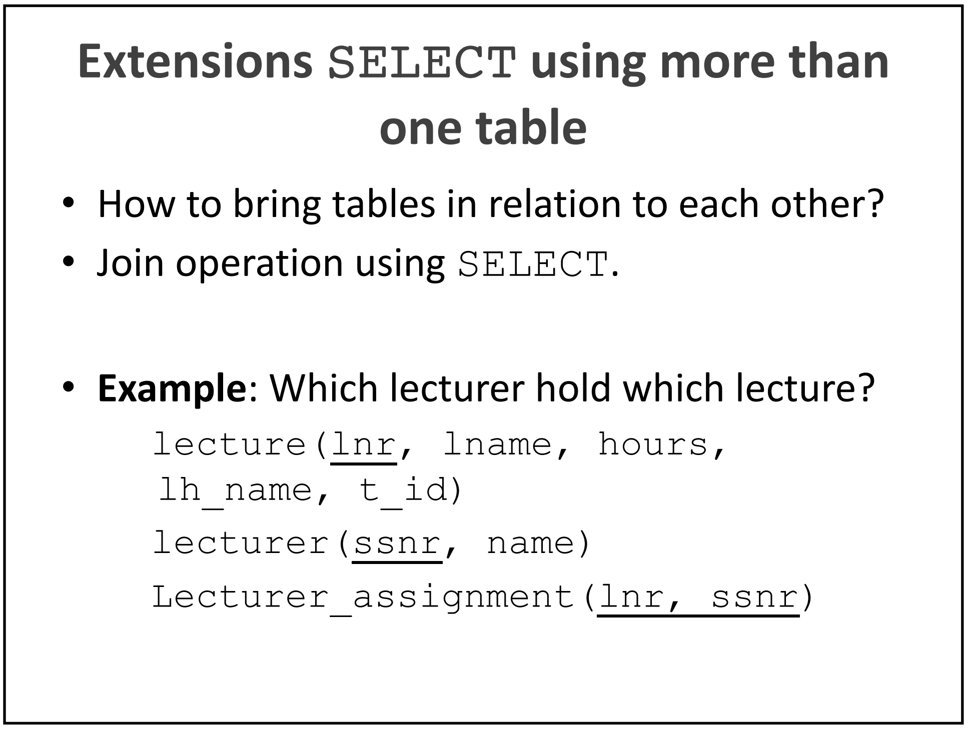
The SELECT statement

- For trying:

```
SELECT current_date;

SELECT 2 + 4;

SELECT version();
```



Extensions SELECT using more than one table

- How to bring tables in relation to each other?
- Join operation using SELECT.

- **Example**: Which lecturer hold which lecture?

```
lecture(lnr, lname, hours,
 lh_name, t_id)
lecturer(ssnr, name)
Lecturer_assignment(lnr, ssnr)
```

Extensions SELECT (cont.)

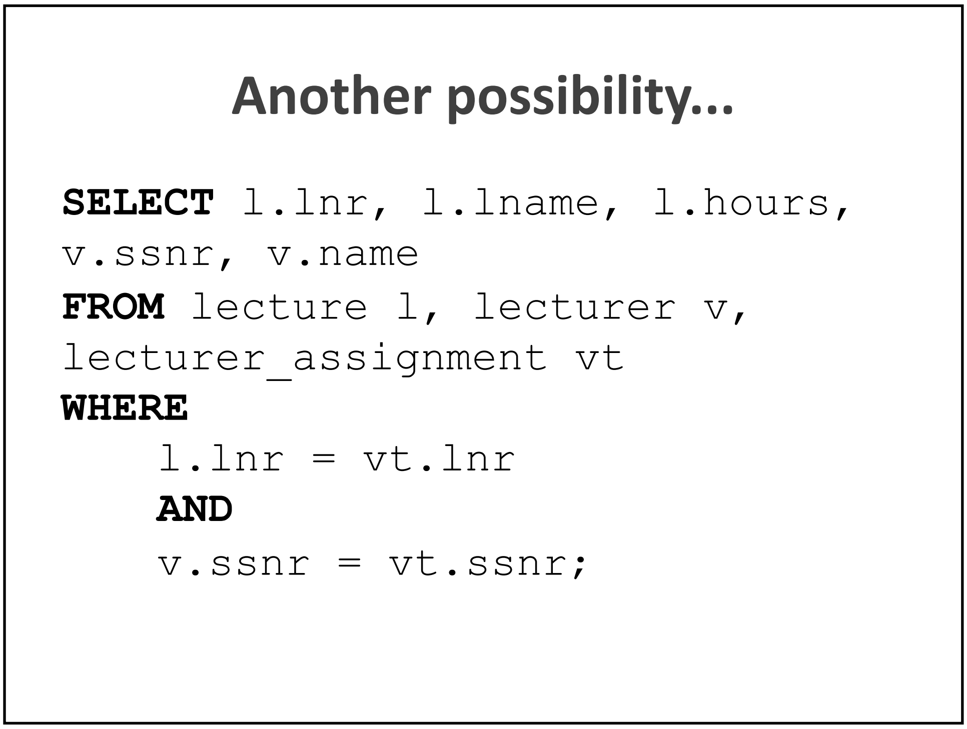
- Solution:

```
SELECT lecture.lnr, lecture.lname,
lecture.hours, lecturer.ssnr, lecturer.name
FROM lecture, lecturer, lecturer_assignment
WHERE
    lecture.lnr = lecturer_assignment.lnr
    AND
   lecturer.ssnr = lecturer_assignment.ssnr;
```

Extensions SELECT (cont.)

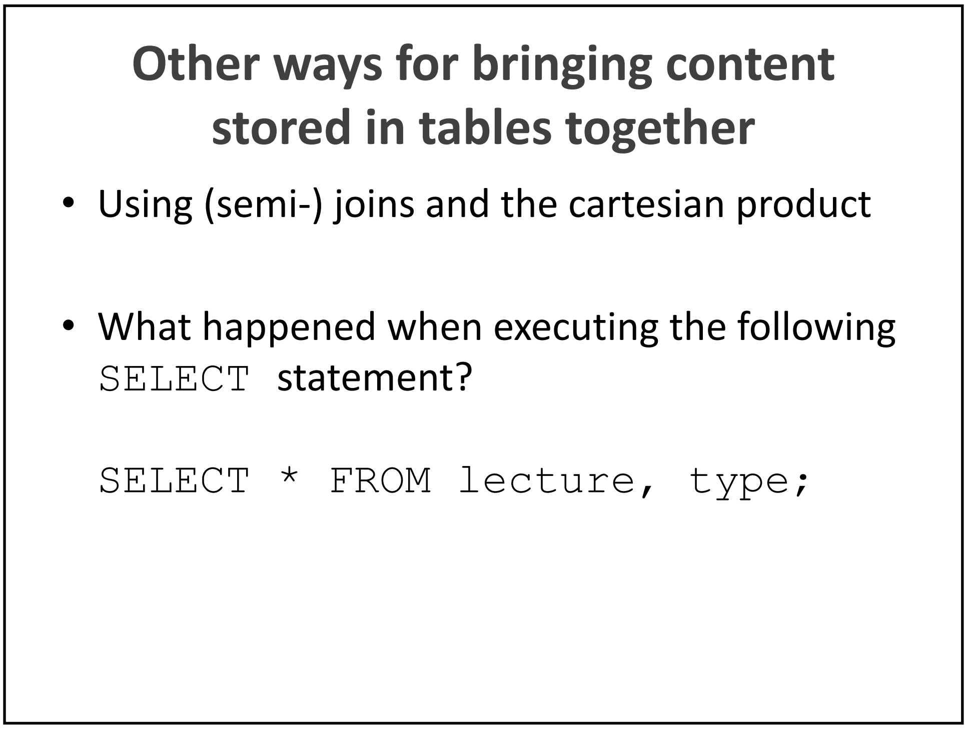
- Final result using our example table:

```
  lnr   |          lname          | hours     |    ssnr    |     name
--------+------------------------+-----------+------------+--------------
 117101 | "Software Maintenance" |         3 | 0987654321 | Rui Abreu
 117101 | "Software Maintenance" |         3 | 1234567890 | Franz Wotawa
 117104 | "Seminar DB”           |         2 | 1234567890 | Franz Wotawa
 117102 | "Compiler Construction"|         2 | 1234567890 | Franz Wotawa
 117103 | "Compiler Construction"|         1 | 0987654321 | Rui Abreu
(5 rows)
```



Another possibility...

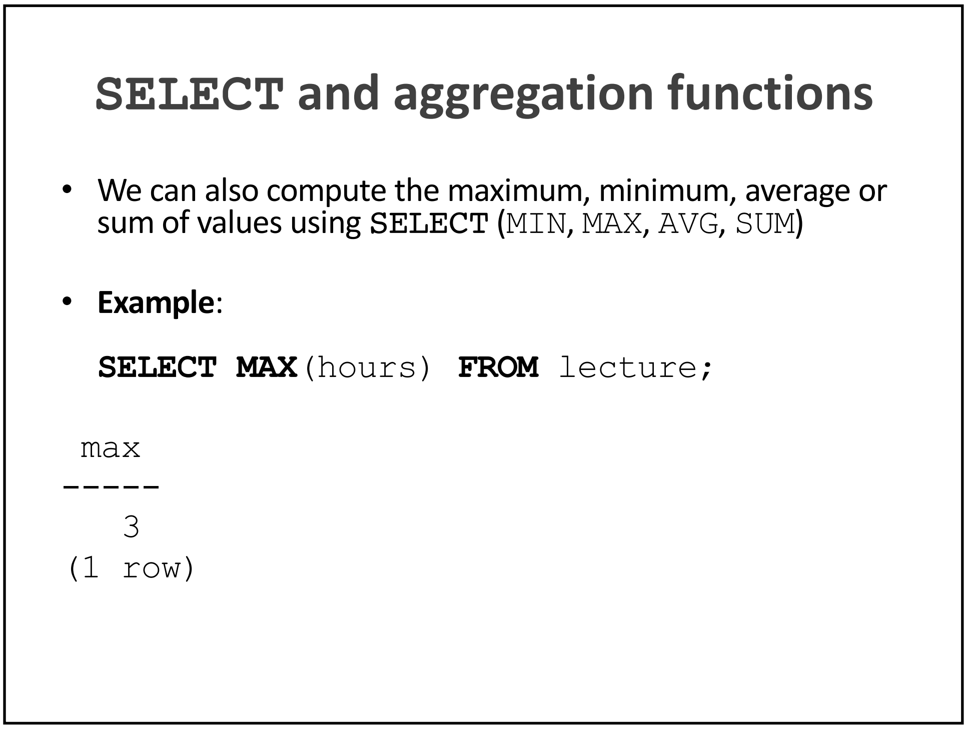
```
SELECT l.lnr, l.lname, l.hours,
v.ssnr, v.name
FROM lecture l, lecturer v,
lecturer_assignment vt
WHERE
    l.lnr = vt.lnr
    AND
    v.ssnr = vt.ssnr;
```



Other ways for bringing content stored in tables together

- Using (semi-) joins and the cartesian product
- What happened when executing the following `SELECT` statement?

```
SELECT * FROM lecture, type;
```

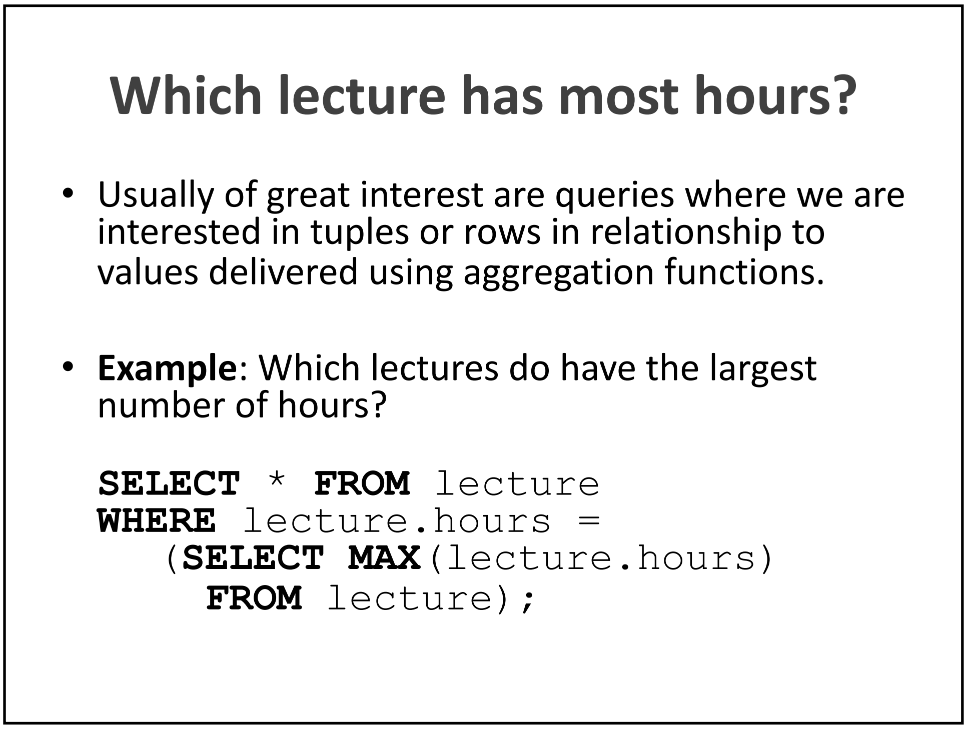


SELECT and aggregation functions

- We can also compute the maximum, minimum, average or sum of values using `SELECT` (`MIN`, `MAX`, `AVG`, `SUM`)
- **Example**:

```
SELECT MAX(hours) FROM lecture;
```

```
 max
-----
   3
(1 row)
```



Which lecture has most hours?

- Usually of great interest are queries where we are interested in tuples or rows in relationship to values delivered using aggregation functions.
- **Example**: Which lectures do have the largest number of hours?

```
SELECT * FROM lecture
WHERE lecture.hours =
   (SELECT MAX(lecture.hours)
     FROM lecture);
```

The GROUP BY and HAVING clauses

- Assume that we only want to use the aggregation function over parts of a table.
- **Example**: A course might comprise a lecture and a practical part having the same name assigned. For the real effort, we are interested in the sum of the hours.

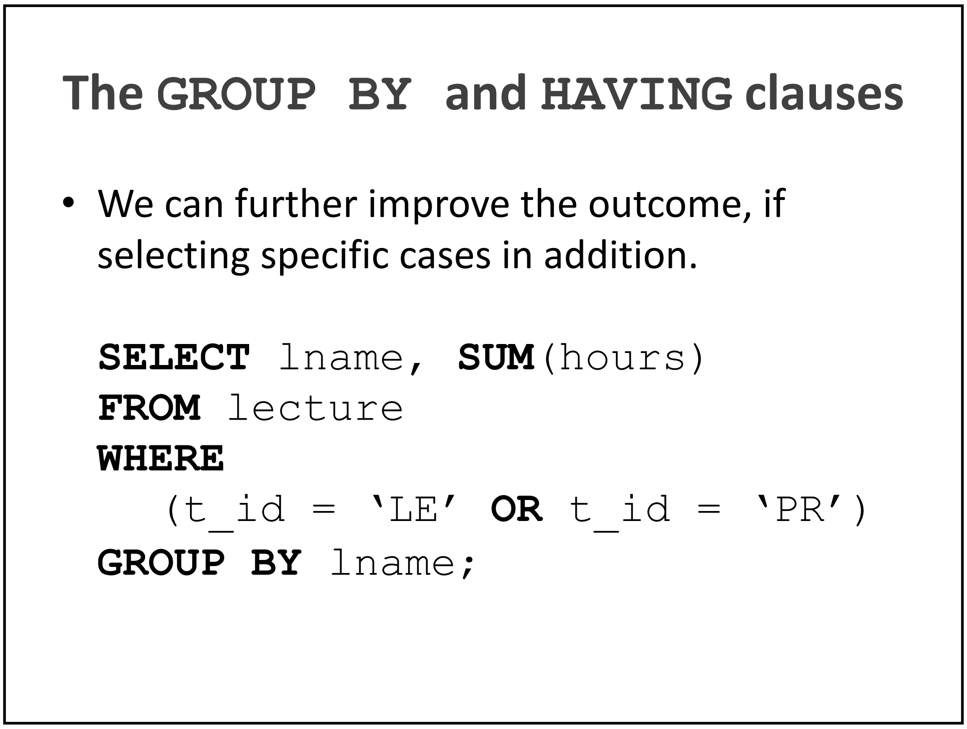
```
SELECT lname, SUM(hours)
FROM lecture
GROUP BY lname;
```

The GROUP BY and HAVING clauses

```
 lnr    |          lname          | hours     | lh_name |    t_id
--------+------------------------+-----------+---------+------------
 117101 | "Software Maintenance" |         3 | i7      | LP
 117102 | "Compiler Construction"|         2 | i11     | LE
 117103 | "Compiler Construction"|         1 | i11     | PR
 117104 | "Seminar DB‟           |         2 | i12     | SE
(4 rows)
```

```
SELECT lname, SUM(hours)
FROM lecture
GROUP BY lname;
```

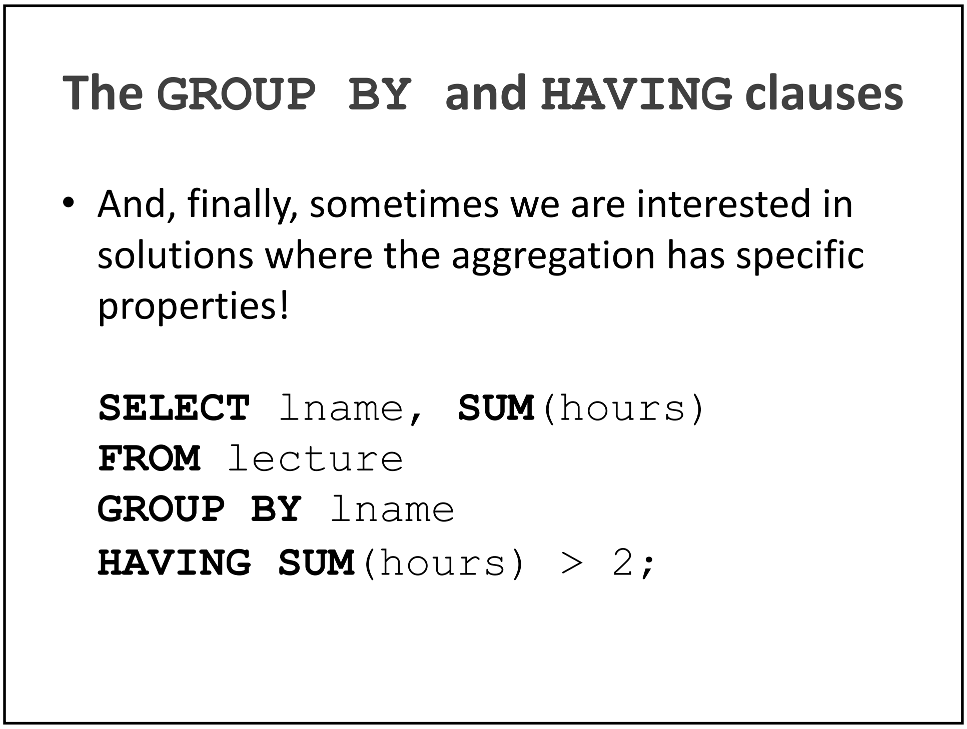
```
 lname                    | sum
-------------------------+-----
 "Seminar DB Dipl"       |   2
 "Software Maintenance"  |   3
 "Compiler Construction "|   3
(3 rows)
```



The GROUP BY and HAVING clauses

- We can further improve the outcome, if selecting specific cases in addition.

```
SELECT lname, SUM(hours)
FROM lecture
WHERE
   (t_id = ‘LE’ OR t_id = ‘PR’)
GROUP BY lname;
```



The GROUP BY and HAVING clauses

- And, finally, sometimes we are interested in solutions where the aggregation has specific properties!

```
SELECT lname, SUM(hours)
FROM lecture
GROUP BY lname
HAVING SUM(hours) > 2;
```

Change data in tables using SQL

- Use the `UPDATE` function!
- **Example**: Every lecture of type LP (lecture including a practical part) receives one hour more.

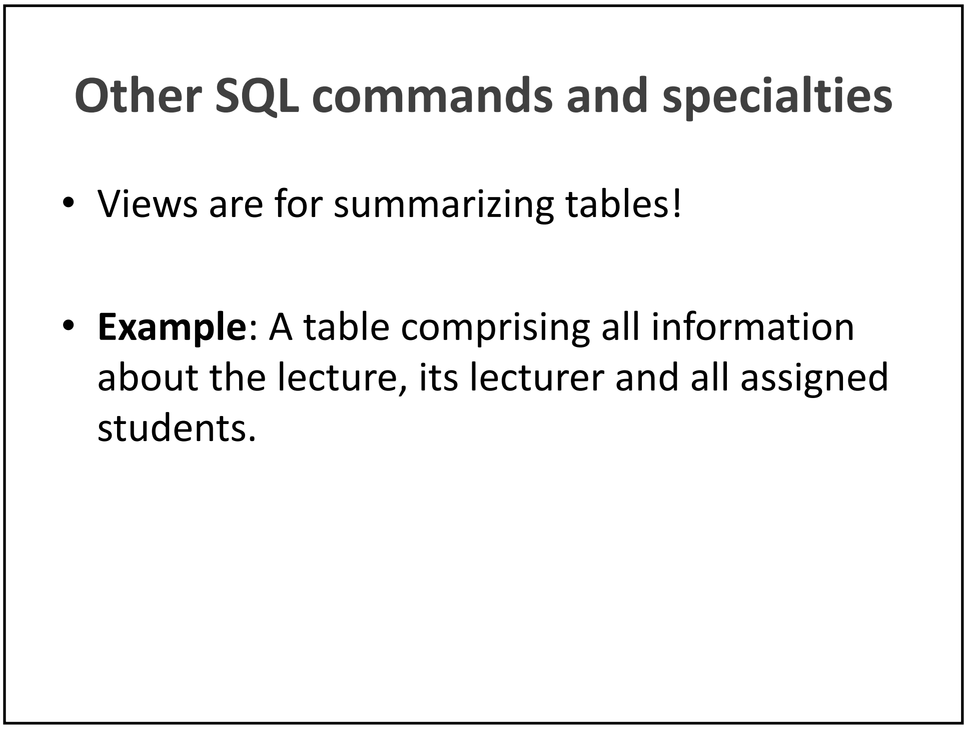
```
UPDATE lecture
SET hours = hours + 1
WHERE t_id = ’LP’;
```

Delete data in SQL

- Make use of the DELETE function!
- **Example**: Delete all lectures of type ’P’ (only practical parts).

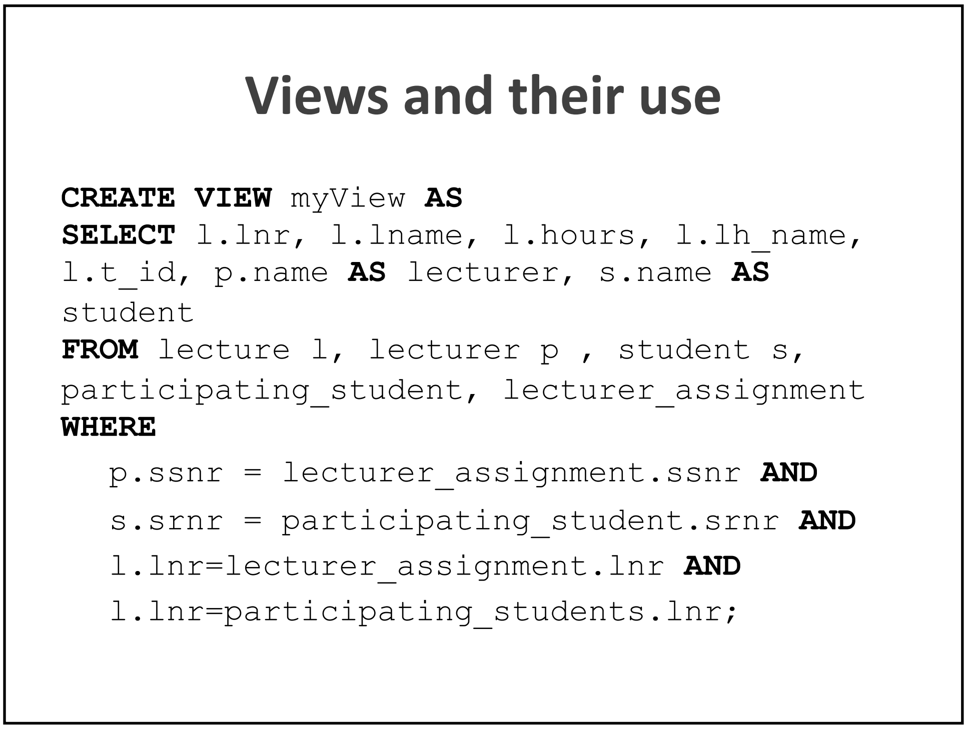
```
DELETE FROM lecture
WHERE t_id= ’PR’;
```

- **ATTENTION:** `DELETE FROM lecture;` removes all entries!



Other SQL commands and specialties

- Views are for summarizing tables!
- **Example**: A table comprising all information about the lecture, its lecturer and all assigned students.



Views and their use

```
CREATE VIEW myView AS
SELECT l.lnr, l.lname, l.hours, l.lh_name,
l.t_id, p.name AS lecturer, s.name AS
student
FROM lecture l, lecturer p , student s,
participating_student, lecturer_assignment
WHERE
   p.ssnr = lecturer_assignment.ssnr AND
   s.srnr = participating_student.srnr AND
   l.lnr=lecturer_assignment.lnr AND
   l.lnr=participating_students.lnr;
```

Example VIEW

- `SELECT * FROM myView;` returns:

```
 lnr    |         lname          | hours     | lh_name |    t_id    | lecturer     | student
--------+------------------------+-----------+---------+------------+--------------+---------
 117101 | "Software Maintenance" |         3 | i7      | LP         | Birgit Hofer | Lucas
 117101 | "Software Maintenance" |         3 | i7      | LP         | Birgit Hofer | Sophie
 117101 | "Software Maintenance" |         3 | i7      | LP         | Birgit Hofer | Marie
 117101 | "Software Maintenance" |         3 | i7      | LP         | Birgit Hofer | Adam
 117101 | "Software Maintenance" |         3 | i7      | LP         | Franz Wotawa | Lucas
 117101 | "Software Maintenance" |         3 | i7      | LP         | Franz Wotawa | Sophie
 117101 | "Software Maintenance" |         3 | i7      | LP         | Franz Wotawa | Marie
 117101 | "Software Maintenance" |         3 | i7      | LP         | Franz Wotawa | Adam
 117102 | "Compiler Construction"|         2 | i11     | LE         | Franz Wotawa | Eva
 117102 | "Compiler Construction"|         2 | i11     | LE         | Franz Wotawa | Adam
 117102 | "Compiler Construction"|         2 | i11     | LE         | Franz Wotawa | Sophie
 117103 | "Compiler Construction"|         1 | i11     | PR         | Birgit Hofer | Adam
 117103 | "Compiler Construction"|         1 | i11     | PR         | Birgit Hofer | Sophie
 117104 | "Seminar DB"           |         2 | i12     | SE         | Franz Wotawa | Eva
 117104 | "Seminar DB"           |         2 | i12     | SE         | Franz Wotawa | Adam
(15 rows)
```

Views (cont.)

- Views can be used like tables.
- It is often reasonable to introduce Views whenever we want to summarize data.
- With Views details of the database are hidden from the user.

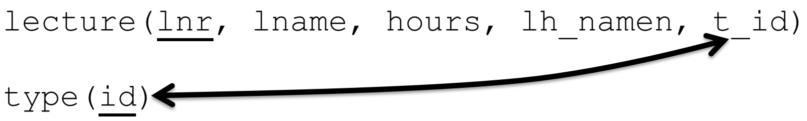
How to assure referential integrity?

- In table `lecture` only lecture types are allowed that are in `type`. This can be achieved via querying the database whenever we want to add information. Hence, whenever we want to add a tuple to `lvtab` we have to ask whether the type already exists.

- **Better**: Use the database directly for assuring referential integrity!

Using the primary key

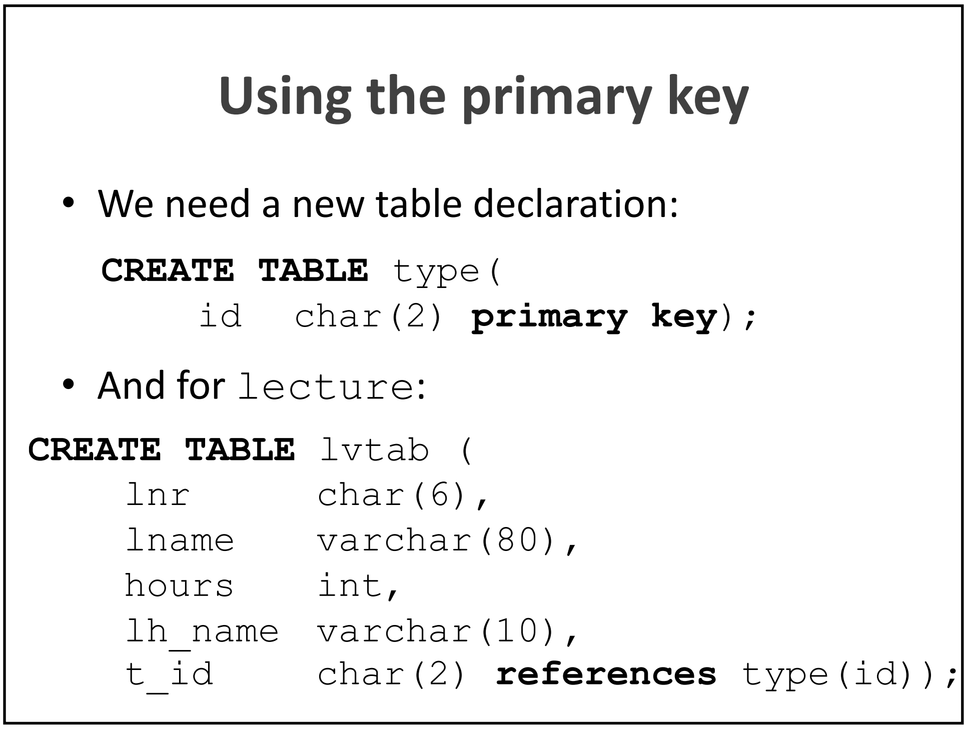
- Current situation:



```
lecture(lnr, lname, hours, lh_namen, t_id)

type(id)
```

Because `id` in `type` is a key, we can use it directly!



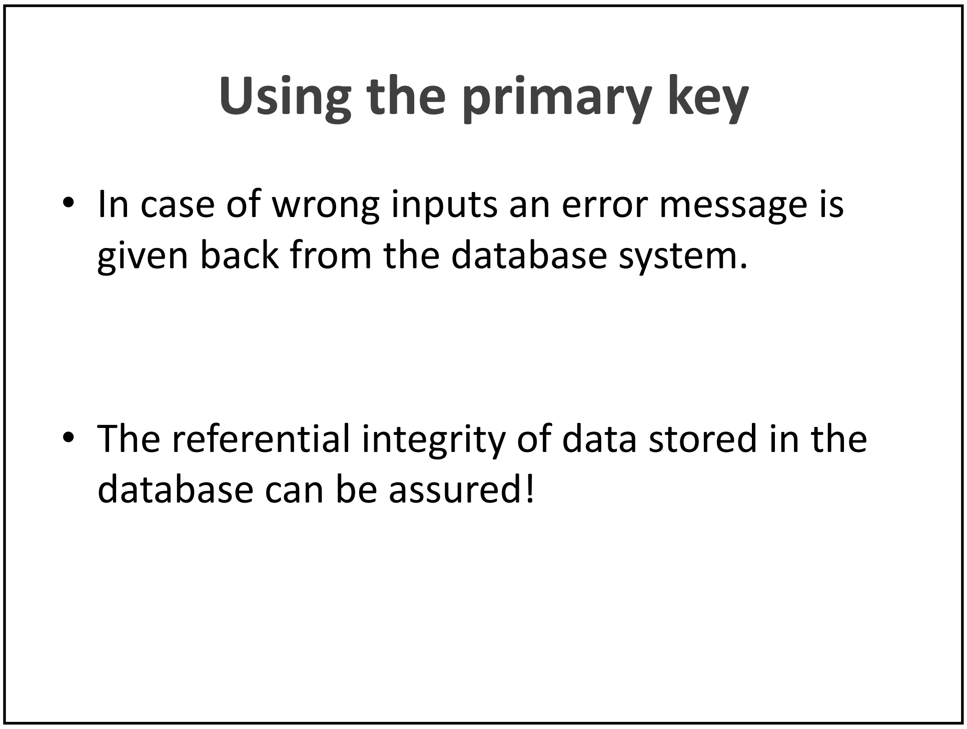
Using the primary key

- We need a new table declaration:

```
CREATE TABLE type(
    id  char(2) primary key);
```

- And for `lecture`:

```
CREATE TABLE lvtab (
    lnr       char(6),
    lname     varchar(80),
    hours     int,
    lh_name   varchar(10),
    t_id      char(2) references type(id));
```



Using the primary key

- In case of wrong inputs an error message is given back from the database system.
- The referential integrity of data stored in the database can be assured!

Database transactions

- In practice a database is usually accessed from different (remote) places at the same time!
- For `SELECT` statements a concurrent access is usually not problematic.
- However, in case of changes we have to assure that no data integrity problems arise.

Database transactions

- **Example**: Let us assume that we have a database in a bank storing the account information of customers.
  - If we withdraw money from the account of Julia and put it into the account of Romeo, we want to assure that both operations are carried out finally!
  - We can assure this using SQL transaction.

Database transactions

```
BEGIN;
  UPDATE accounts
  SET balance = balance - 100.00
  WHERE name = ’Julia’;
  UPDATE accounts
  SET balance = balance + 100.00
  WHERE name = ’Romeo’
COMMIT;
```

Database transactions

- Using `BEGIN;` and `COMMIT;` we define an transactions
- In the transactions it is assured that all SQL operations are carried out together. In case of a problem in one operation, the effect of all previous operations is removed.
- Transactions are atomic operations.
- **Extensions**:
  - `SAVEPOINT my_savepoint;`
  - `ROLLBACK TO my_savepoint;`

Example

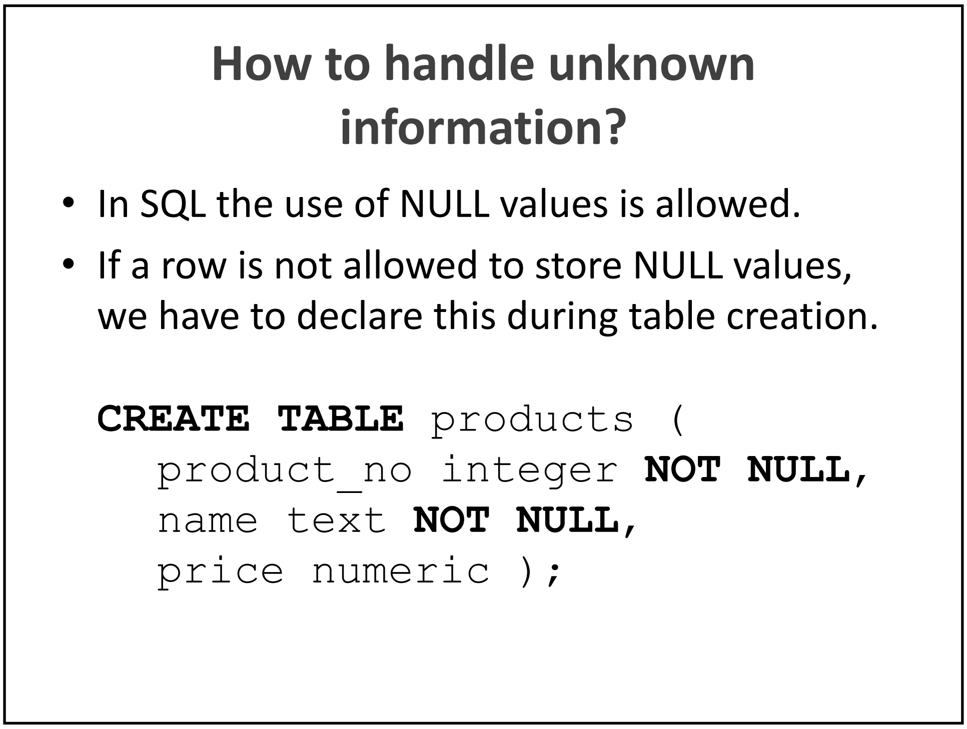
```
SQL> SAVEPOINT SP1;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=1;
1 row deleted.
SQL> SAVEPOINT SP2;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=2; 1
row deleted.
SQL> SAVEPOINT SP3;
Savepoint created.
SQL> DELETE FROM CUSTOMERS WHERE ID=3;
1 row deleted.
```

Now that the three deletions have taken place, let us assume that you have changed your mind and decided to ROLLBACK to the SAVEPOINT that you identified as SP2. Because SP2 was created after the first deletion, the last two deletions are undone

```
SQL> ROLLBACK TO SP2;
Rollback complete.
```

Database transactions

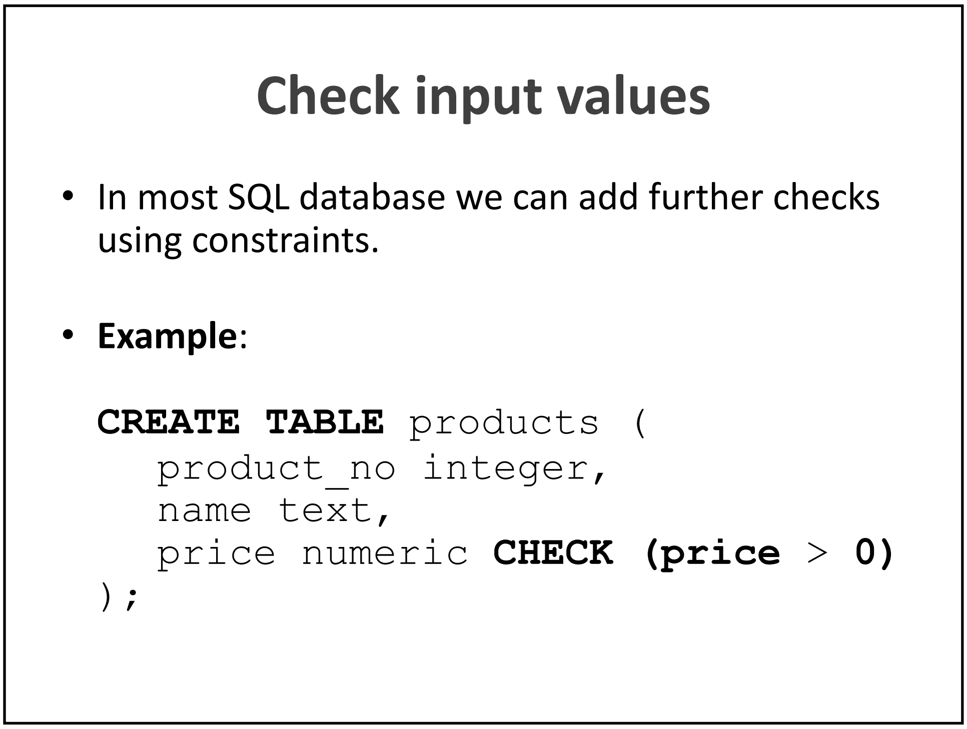
- The use of transactions is absolutely requested in practice!
- Transaction handling is often at least partically implemented in libraries. This should be checked using the library documentation.
- Testing of database applications has also to check the behavior in case of concurrent database access and changes!



How to handle unknown information?

- In SQL the use of NULL values is allowed.
- If a row is not allowed to store NULL values, we have to declare this during table creation.

```
CREATE TABLE products (
  product_no integer NOT NULL,
  name text NOT NULL,
  price numeric );
```



Check input values

- In most SQL database we can add further checks using constraints.
- **Example**:

```
CREATE TABLE products (
  product_no integer,
  name text,
  price numeric CHECK (price > 0)
);
```

Using primary keys with more than one attribute

- We can define primary keys comprising more than one attribute.
- **Example**:

```
CREATE TABLE example (
   a integer,
   b integer,
   c integer,
   PRIMARY KEY (a, c)
);
```

Making attributes unique

- Use the `UNIQUE` keyword (similar to primary key or not null).
- Makes a given attribute unique, i.e., it can only store different values in each row.
- Similar to primary key.
- Of use in cases where you have a primary key and another attribute that is also unique but only for information, e.g.:

```
EMPLOYEE(ssn, passportNr,…)
```

SQL Constraints Summary

- NOT NULL Constraint − Ensures that a column cannot have NULL value.
- DEFAULT Constraint − Provides a default value for a column when none is specified.
- UNIQUE Constraint − Ensures that all values in a column are different.
- PRIMARY Key − Uniquely identifies each row/record in a database table.
- FOREIGN Key − Uniquely identifies a row/record in any of the given database table.
- CHECK Constraint − The CHECK constraint ensures that all the values in a column satisfies certain conditions.
- INDEX − Used to create and retrieve data from the database very quickly.

We can also change the database schemes

- Using SQL we can change database schemes after creation as well!
- **Example**:

```
ALTER TABLE products
ADD COLUMN description text;
```

Other possibilities...

- Bring together query results:

```
query1 UNION [ALL] query2
query1 INTERSECT [ALL] query2
query1 EXCEPT [ALL] query2
```

- If `ALL` is not used, duplicates were automatically removed.

What we have discussed today...

- Implementing databases using SQL:
  - CREATE
  - DROP
  - SELECT
  - UPDATE
  - INSERT

… And there is more

- Inheritance
- Schemes
- User management
  - Access control using SQL databases directly
- Optimizing performance (of queries)
- ….

Have a look at

`https://www.tutorialspoint.com/sql`

Next Class

SQL- Part III (ODBC, JDBC,…)