Introduction to SQL

SQL (structured query language)

- History

  1974: first paper by Chamberlin&Boyce

  SQL 92 (SQL 2): joins, outer-joins, ...

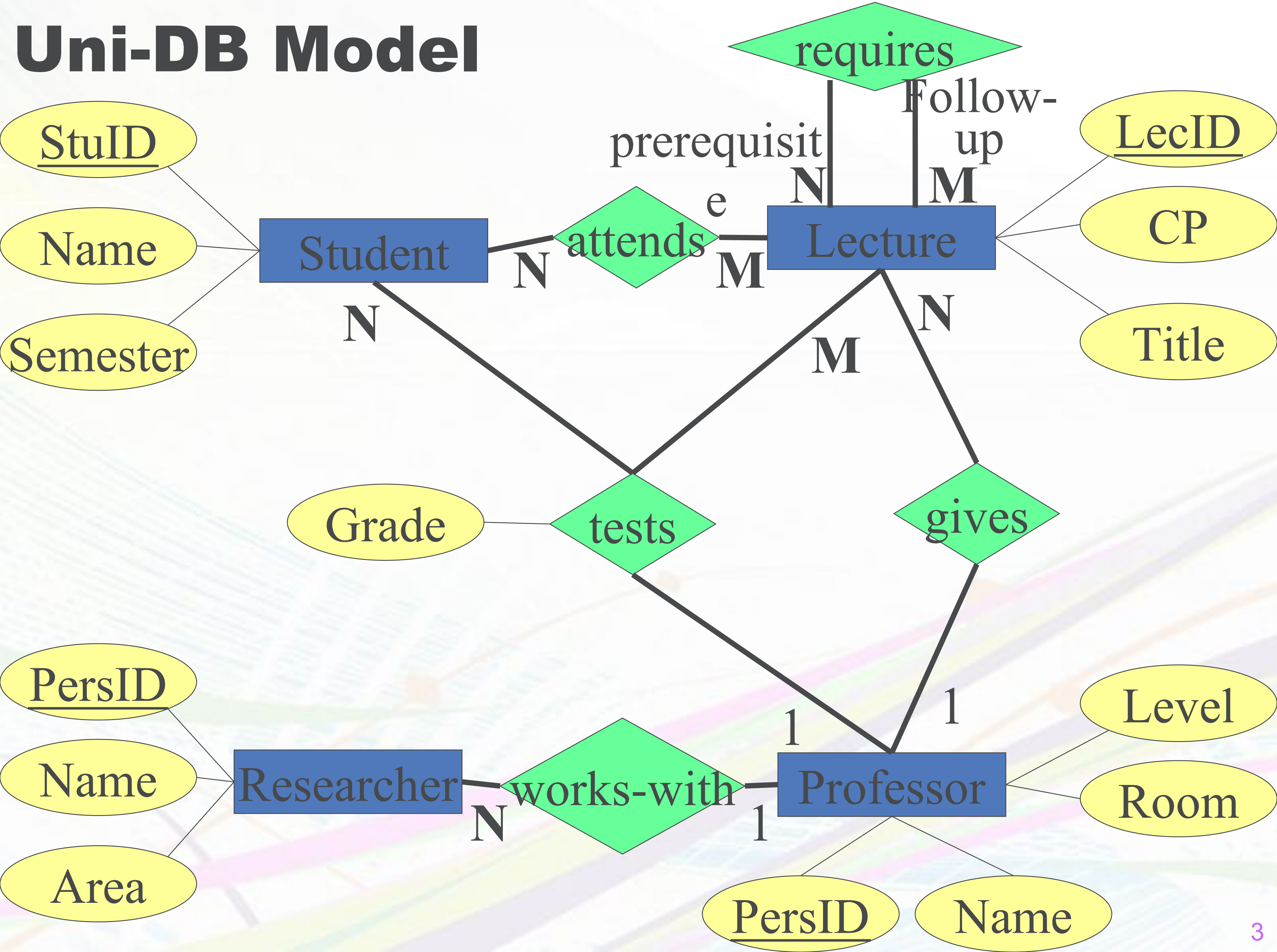
  SQL 3: object-relational extensions

  SQL/XML, etc.: domain-specific extensions

- A family of standards

  - Data definition language (DDL) - schemas
  - Data manipulation language (DML) - updates
  - Query language (Query) - reads

Uni-DB Model



requires
prerequisite
Follow-up
N
M
StuID
LecID
Name
CP
Student
attends
Lecture
N
M
Semester
Title
N
N
M
Grade
tests
gives
PersID
1
1
Level
Name
Researcher
works-with
Professor
Room
N
1
Area
PersID
Name

(Simple) Data Definition with SQL

Data types

- **character** *(n)*, ***char*** *(n)*
- **character varying** *(n)*, **varchar** *(n)*
- **numeric** *(p,s)*, **integer**
- **blob** or **raw** for large binaries
- **clob** for large string values
- **date**

Create Tables

create table Professor
(PersID **integer not null,**
Name **varchar** (30) **not null**
Level **character** (2) **default** 'AP');

DDL (ctd.)

Delete a Table

- **drop table** Professor;

Modify the structure of a Table

- **alter table** Professor **add column(**age **integer);**

Management of Indexes (Performance tuning)

- **create index** myIndex **on** Professor(name, age)**;**
- **drop index** myIndex**;**

Updates (DML)

Insert Tuples

```
insert into attends
      select StuID, LecID
      from Student, Lecture
      where Title= 'DMD' ;


insert into Student (StuID, Name)
      values (28121,  `Archimedes‘);
```

Student		
StuID	Name	Semester
⋮	⋮	⋮
29120	Theophrastos	2
29555	Feuerbach	2
28121	Archimedes	-

Null

Sequence Types (Automatic Increment for Surrogates)

- **create sequence** PersID_seq **increment by** 1 **start with** 1;
- **insert into** Professor(PersID, Name)
  values(*PersNr_seq.nextval*, ‘Roscoe’);
- Syntax is vendor dependent
  - E.g., AUTO-INCREMENT Option in MySQL
  - Syntax above was standardized in SQL 2003
  - *what about PostgreSQL??*

Updates (ctd.)

Delete tuples

delete Student

where Semester > 13;

Update tuples

update Student

set Semester= Semester + 1;

Queries

select PersID, Name

from Professor

where Level = ´FP´;

PersID	Name
2125	Qiang
2126	Sadegh
2136	Joo
2137	Someone

Queries: Sorting

select PersID, Name, Level

from Professor

order by Level **desc**, Name **asc**;

PersID	Name	Level
2136	Curie	FP
2137	sadegh	FP
2126	someone	FP
2127	Kopernikus	AP
2133	Popper	AP

Duplicate Elimination

select distinct Level

from Professor

Level
AP
FP

Queries: Joins

Who teaches DMD?

select Name

from Professor, Lecture

where PersID = ProfID **and** Title = 'DMD' ;

$$\prod_{\text{Name}}(\sigma_{\text{PersID} = \text{ProfID} \wedge \text{Title} = \text{'DMD'}}(\text{Professor} \times \text{Lecture}))$$

N.B.: Renamed Lecture.PersID to ProfID.
Will show later how this can be done as part of a query.

Joins

Professor			
PersID	Name	Level	Room
2125	Qiang	AP	226
2126	Sadegh	FP	232
⋮	⋮	⋮	⋮
2137	someother	FP	7

Lecture			
LecID	Title	CP	ProfID
5001	DSP	4	2137
5041	OOP	4	2125
⋮	⋮	⋮	⋮
5049	DMD	2	2125
⋮	⋮	⋮	⋮
4630	whatever	4	2137

X

PerID	Name	Level	Room	LecID	Title	CP	ProfID
2125	Qiang	AP	226	5001	DSP	4	2137
1225	Manuel	FP	226	5041	OOP	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2125	Qiang	FP	226	5049	DMD	2	2125
⋮	⋮		⋮	⋮	⋮	⋮	⋮
2126	Sadegh	FP	232	5001	ML	4	2137
2126	Sadegh	FP	232	5041	IR	4	2125
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
2137	someother	FP	7	4630	whatever	4	2137

↓ σ

PersID	Name	Level	Room	LecID	Title	CP	ProfID
2125	Qiang	AP	226	5049	DMD	2	2125

↓ π

Name
Qiang

SQL -> Relational Algebra

SQL

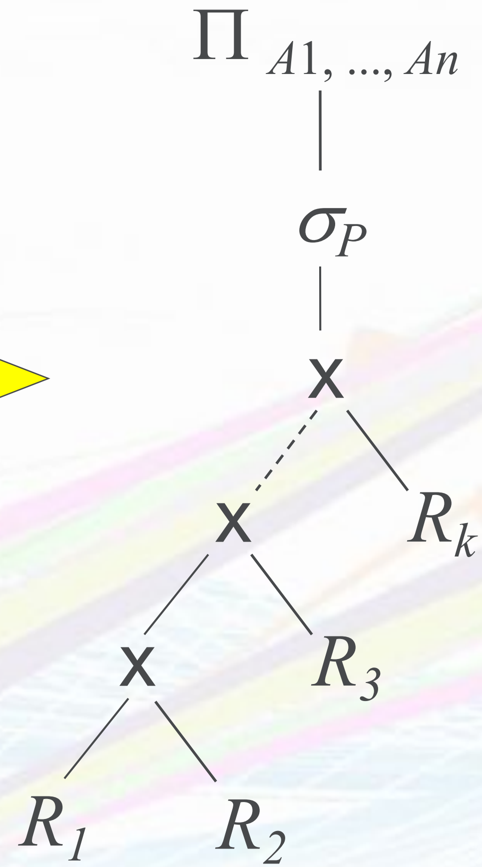
Relational Algebra

$\Pi_{A1, \ldots, An}(\sigma_P (R_1 \times \ldots \times R_k))$

select $A_1, \ldots, A_n$

from $R_1, \ldots, R_k$

where P;



Joins and Tuple Variables

Who attends which lecture?

```
select Name, Title
from Student, attends, Lecture
where Student.StuID = attends.StuID and
      attends.LecID = Lecture.LecID;
```

Alternative:

```
select s.Name, l.Title
from Student s, attends a, Lecture l
where s.StuID= a.StuID and
      a.LecID = l.LecID;
```

Rename of Attributes

Give title and professor of all lectures?

select Title, PersID **as** ProfID

from Lecture;

select attributes or expressions of attributes

Set Operations

SQL supports: **union, intersect, minus**

```
( select Name
  from Researcher)
union
( select Name
  from Professor);
```

```
( select Name
  from Researcher)
union all
( select Name
  from Professor);
```

Grouping, Aggregation

Aggregate functions: **avg, max, min, count, sum**

select avg (Semester)

from Student;

Grouping, Aggregation

Aggregate functions: **avg, max, min, count, sum**

```
select avg (Semester)
from Student;
```

```
select PersID, sum (CP)
from Lecture
group by PersID;
```

Grouping, Aggregation

Aggregate functions: **avg, max, min, count, sum**

select avg (Semester)

from Student;

having clause following group by

select PersID, **sum** (CP)

from Lecture

group by PersID;

select p.PersID, Name, **sum** (CP)
from Lecture l, Professor p
where l.PersID= p.PersID **and** level = ´FP´
group by p.PersID, Name
having avg (CP) >= 3;

Imperative Processing in SQL

- Step 1:

  from Lecture l, Professor p

  where l.PersID= p.PersID **and** level = ´FP´
- Step 2:

  group by p.PersID, Name
- Step 3:

  having avg (CP) >= 3;
- Step 4:

  select p.PersID, Name, **sum** (CP)

Existential Quantification: exists sub-queries

```
select p.Name
from Professor p
where not exists ( select *
                   from Lecture l
                   where l.PersID = p.PersID );
```

EXIST R means R is not NULL

Correlated Sub-queries

Correlation

```
select p.Name
from Professor p
where not exists ( select *
                   from Lecture l
                   where l.PersID = p.PersID );
```

cannot run without outer query

Uncorrelated Sub-query

```
select Name
from Professor
where PersID not in ( select PersID
                      from Lecture);
```

can run without outer query

Differences: Correlated v.s uncorrelated, comparing the above examples, which one is better?
More: https://my.vertica.com/docs/4.1/HTML/Master/10374.htm

Sub-queries with all

select Name

from Student

where Semester >= **all** (**select** Semester

from Student);

Not as powerful as relational division!

Subqueries in SELECT, FROM

select PersID, Name, (**select sum** (CP) **as** load **from** Lecture l

where p.PersID=l.PersID)

from Professor p;

select p.PersID, Name, l.load

from Professor p, (**select** PersID, **sum** (CP) **as** load **from** Lecture

group by PersID) l

where p.PersID = l.PersID;

Is this better than the simple Group By Query from before?

Query Rewrite

```
select *
from Researcher a
where exists
   (select *
    from Professor p
    where a.Supervisor= p.PersID and p.age < a.age);
```

- Equivalent Join Query: Why is this better?

```
select a.*
from Researcher a, Professor p
where a.supervisor=p.PersID and p.age < a.age;
```

Universal Quantifiaction

- SQL does not support relational division directly
- Need to play tricks

$$\{s \mid s \in \text{Student} \wedge$$
$$\forall l \in \text{Lecture}\ (l.CP=4 \Rightarrow$$
$$\exists a \in \text{attends}\ (a.LecID=l.LecID \wedge a.StuID=s.StuID))\}$$

- Approach: Eliminate of $\forall$ and $\Rightarrow$

$$\forall t \in R\ (P(t)) = \neg(\exists t \in R(\neg P(t)))$$

$$R \Rightarrow T = \neg R \vee T$$

- Applying these rules:

$\{s \mid s \in Student \land \neg (\exists l \in Lecture\ \neg(\neg(l.CP=4) \lor$

$\exists a \in attends(a.LecID=l.LecID \land a.StuID=s.StuID))\}$

- Applying DeMorgan rules:

$\{s \mid s \in Student \land \neg (\exists l \in Lecture(l.CP=4 \land$

$\neg(\exists a \in attends(a.LecID=l.LecID \land a.StuID=s.StuID))))\}$

- This can be implemented in SQL:

```
select *
from Student s
where not exists
   (select *
     from Lecture l
     where l.CP = 4 and not exists
        (select *
          from attends a
          where a.LecID = l.LecID and a.StuID=s.StuID) );
```

- This can be implemented in SQL:

```
select *
from Student s
where not exists
   (select *
    from Lecture l
    where l.CP = 4 and not exists
       (select *
        from attends a
        where a.LecID = l.LecID and a.StuID=s.StuID) );
```

Find students for which there is no four-hour lecture that student does not attend!

Or do it this way

select a.StuID

from attends a

group by a.StuID

having count (*) = (**select** count (*) **from** Lecture);

Considering only 4 CP lectures

select a.StuID

from attends a, Lecture l

where a.LecID = l.LecID and l.CP = 4

group by a.StuID

having count (*) = (**select** count (*) **from** Lecture
where CP = 4);

Null Values (NULL = UNKNOWN)

select count (*)

from Student

where Semester < 13 **or** Semester > =13;

vs.

select count (*)
from Student;

Are those two queries equivalent?

Working with Null Values

1. Arithmetics: Propagate **null**: If an operand is null, the result is **null**.

    null + 1 -> null

    null * 0 -> null

2. Comparisons: New Boolean value **unknown**. All comparisons that involve a **null** value, evaluate to **unknown**.

    null = null -> unknown

    null < 13 -> unknown

    null > null -> unknown

3. Logic: Boolean operators are evaluated using the following tables (next slide):

not	
true	false
unknown	unknown
false	true

and	*true*	*unknown*	*false*
true	true	unknown	false
unknown	unknown	unknown	false
false	false	false	false

or	*true*	*unknown*	*false*
true	true	true	true
unknown	true	unknown	unknown
false	true	unknown	false

4. **where**: Only tuples which evaluate to **true** are part of the query result. (**unknown** and **false** are equivalent here):

select count (*)
from Student
where Semester < 13 **or** Semester > =13;

5. **group by:** If exists, then there is a group for **null**.

select count (*)

from Student
group by Semester;

Predicates with null:

select count (*) **from** Student
where Semester **is null**;

Syntactic Sugar

select *

from Student

where Semester > = 1 **and** Semester < = 6;

select *

from Student

where Semester **between** 1 **and** 6;

select *

from Student

where Semester **in** (2,4,6);

case

```
select StuID, ( case when Grade >= 5.5 then ´A´
                when Grade >= 5.0 then ´B´
                when Grade >= 4.5 then ´C´
                when Grade >= 4.0 then ´D´
               else ´F´end)
from tests;
```

- Behaves like a switch: evaluate from top to bottom
- No "break" needed because at most one clause executed.

Comparisons with like

- "%" represents any sequence of characters (0 to n)
- "_" represents exactly one character
- N.B.: For comparisons with **=** , % and _ are normal chars.

```
select *
from Student
where Name like ´T%eophrastos´;
```

```
select distinct Name
from Lecture l, attends a, Student s
where s.StuID= a.StuID and a.LecID = l.LecID
      and  l.Title like ´%thik%´;
```

Joins in SQL-92

- **cross join:** Cartesian product
- **natural join:**
- **join** or **inner join**: Theta-Join
- **left, right** or **full outer join**: outer join variants
- (union join: not discussed here)

select *
from R_1, R_2
where $R_1.A = R_2.B$;
select *
from R_1 **join** R_2 **on** $R_1.A = R_2.B$;

Left Outer Joins

select p.PersID, p.Name, t.PersID, t.Grade, t.StuID, s.StuID, s.Name

from Professor p **left outer join**

(tests t **left outer join** Student s

on t.StuID= s.StuID)

on p.PersID=t.PersID;

PersID	p.Name	t.PersID	t.Grade	t.StuID	s.StuID	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopen-hauer
2136	Curie	-	-	-	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Right Outer Joins

select p.PersID, p.Name, t.PersID, t.Grade, t.StuID, s.StuID, s.Name

from Professor p **right outer join**

(tests t **right outer join** Student s **on**

t.StuID= s.StuID)

on p.PersID=t.PersID;

PersID	p.Name	t.PersID	t.Grade	t.StuID	s.StuID	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopen-hauer
-	-	-	-	-	26120	Fichte
⋮	⋮	⋮	⋮	⋮	⋮	⋮

Full Outer Joins

select p.PersID, p.Name, t.PersID, t.Grade, t.StuID, s.StuID, s.Name

from Professor p **full outer join**

(tests t **full outer join** Student s **on**

t.StuID= s.StuID)

on p.PersID=t.PersID;

p.PersID	p.Name	t.PersID	t.Grade	t.StuID	s.StuID	s.Name
2126	Russel	2126	1	28106	28106	Carnap
2125	Sokrates	2125	2	25403	25403	Jonas
2137	Kant	2137	2	27550	27550	Schopen-hauer
-	-	-	-	-	26120	Fichte
⋮	⋮	⋮	⋮	⋮	⋮	⋮
2136	Curie	-	-	-	-	-
⋮	⋮	⋮	⋮	⋮	⋮	⋮

connect by Clause (Oracle)

```
select Title
from Lecture
where LecID in (select prerequisite
                from requires
                connect by follow-up = prior prerequisite
                start with follow-up = (select LecID
                                        from Lecture
                                        where Title = ... ));
```

Recursion in DB2/SQL99

```
with TransLecture (First, Next)
as (select prerequisite, follow-up from requires
        union all
      select t.First, r.follow-up
      from TransLecture t, requires r
      where t.Next= r.prerequisite)

select Title from Lecture where LecID in
     (select First from TransLecture where Next in
          (select LecID from Lecture
           where Title = `DMD´) )
```

Data Manipulation Language

Insert tuples

```
insert into attends
    select StuID, LecID
    from Student, Lecture
    where Title= `DMD´;

insert into Student (StuID, Name)
    values (28121,  `Archimedes´);
```

Deletion of tuples, Update

delete Student

where Semester > 13;

update Student

set Semester= Semester + 1;

Snapshot Semantics

1. Phase 1: mark tuples which are affected by the update
2. Phase 2: implement update on marked tuples

Otherwise, indeterministic execution of updates:

```
delete from requires
    where prerequisite in (select follow-up
                           from requires);
```

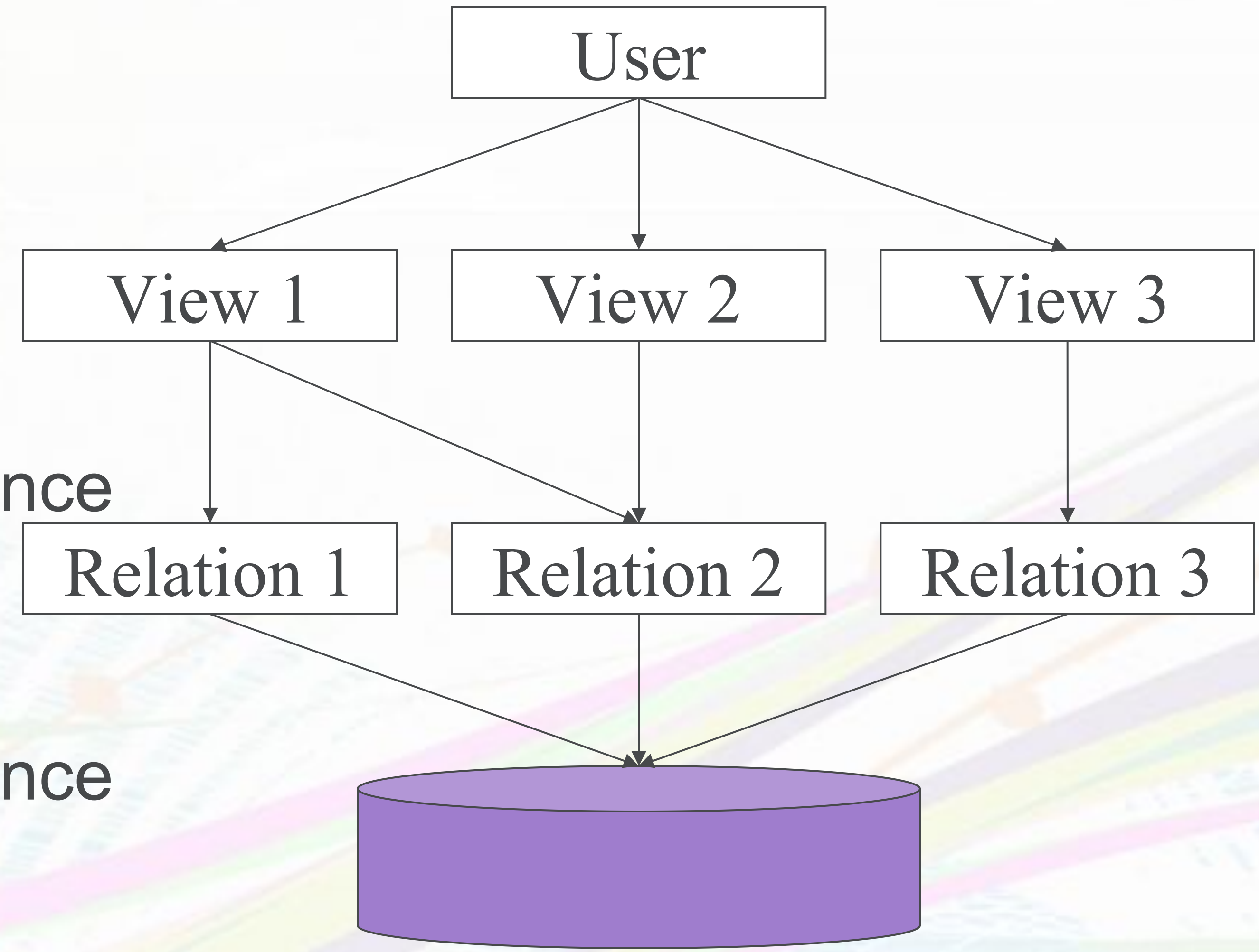
requires	
Prerequisite	Follow-up
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5229

delete from requries

where Prerequisite **in** (**select** Follow-up

from requires);

Views for Logical Data Independence



Logical

data independence

Physical

data independence

Views ...

for privacy

```
create view testView as
    select StuID, LecID, PersID
    from tests;
```

Views ...

for simpler queries

```
create view StudProf (Sname, Semester, Title, Pname)
as
    select s.Name, s.Semester, l.Title, p.Name
    from Student s, attends a, Lecture l, Professor p
    where s.StuID=a.StuID and a.LecID=l.LecID and
              l.PersID= p.PersID;

select distinct Semester
from StudProf
where PName='Someone';
```

Views for is-a relationships

```
create table Employee
    (PersID  integer not null,
    Name     varchar (30) not null);
create table ProfData
    (PersID  integer not null,
    Level    character(2),
    Room     integer);
create table ResearcherData
    (PersID       integer not null,
    area          varchar(30),
    Supervisor    integer);
```

```
create view Professor as
        select *
        from Employee e, ProfData d
        where e.PersID=d.PersID;
create view Researcher as
        select *
        from Employee e, ResearcherData d
        where e.PersID=d.PersID;
```

➔ Subtypes implemented as a view

```
create table Professor
	(PersID		integer not null,
	Name		varchar (30) not null,
	Level		character (2),
	Room		integer);
create table Researcher
	(PersID		integer not null,
	Name		varchar (30) not null,
	area		varchar (30),
	Supervisor	integer);
create table OtherEmps
	(PersID		integer not null,
	Name			varchar (30) not null);
```

```
create view Employee as
    (select PersID, Name
    from Professor)
      union
    (select PersID, Name
    from Researcher)
     union
    (select*
    from OtherEmps);
```

➔ Supertypes implemented as a view

Updatable Views

Example view which is not updatable

create view ToughProf (PersID, AvgGrade) **as**

select PersID, **avg**(Grade)

from tests

group by PersID;

update ToughProf **set** AvgGrade= 6.0

where PersID = 4711;

insert into ToughProf

values (4711, 6.0);

SQL tries to avoid indeterminisms.

What about this?

create view ToughProf (PersID, AvgGrade) **as**

select PersID, **avg**(Grade)

from tests

group by PersID;

delete ToughProf

where PersID = 4711;

Views and Updates

Example view which is not updatable

```
create view LectureView as
    select Title, CP, Name
    from Lecture l, Professor p
    where l.PersID = p.PersID;
insert into LectureView
    values (`Nihilismus‘, 2, `Nobody‘);
```

There are scenarios in which the "insert" is meaningful.
There are scenarios in which SQL would have to guess.
SQL is conservative and does not allow any scenario.

Views and Updates in SQL

- A SQL view is updatable iff
  - The view involves only one base relation
  - The view involves the key of that base relation
  - The view does NOT involve aggregates, group by, or duplicate-elmination