Data Modeling and Databases

Gustavo Alonso, Ce Zhang
Systems Group
Department of Computer Science
ETH Zürich
SQL (structured query language)

- A family of standards
  - Data definition language (DDL) - schemas
  - Data manipulation language (DML) - updates
  - Query language (Query) – reads

- History
  - 1974: first paper by Chamberlin&Boyce
  - SQL 92 (SQL 2): joins, outer-joins, ...
  - SQL 3 (1999): object-relational extensions
  - SQL/XML (2006): domain-specific extensions
  - SQL:2008
  - SQL:2011
### Relational Model of University Schema

#### Professor
<table>
<thead>
<tr>
<th>PersNr</th>
<th>Name</th>
<th>Level</th>
<th>Room</th>
</tr>
</thead>
<tbody>
<tr>
<td>2125</td>
<td>Meyer</td>
<td>FP</td>
<td>226</td>
</tr>
<tr>
<td>2126</td>
<td>Kossmann</td>
<td>FP</td>
<td>232</td>
</tr>
<tr>
<td>2127</td>
<td>Roscoe</td>
<td>AP</td>
<td>310</td>
</tr>
<tr>
<td>2133</td>
<td>Perrig</td>
<td>AP</td>
<td>52</td>
</tr>
<tr>
<td>2134</td>
<td>Sorkine</td>
<td>AP</td>
<td>309</td>
</tr>
<tr>
<td>2136</td>
<td>Welzl</td>
<td>FP</td>
<td>36</td>
</tr>
<tr>
<td>2137</td>
<td>Norrie</td>
<td>FP</td>
<td>7</td>
</tr>
</tbody>
</table>

#### Student
<table>
<thead>
<tr>
<th>Legi</th>
<th>Name</th>
<th>Semester</th>
</tr>
</thead>
<tbody>
<tr>
<td>24002</td>
<td>Gerber</td>
<td>18</td>
</tr>
<tr>
<td>25403</td>
<td>Zollinger</td>
<td>12</td>
</tr>
<tr>
<td>26120</td>
<td>Frey</td>
<td>10</td>
</tr>
<tr>
<td>26830</td>
<td>Küng</td>
<td>8</td>
</tr>
<tr>
<td>27550</td>
<td>Fehr</td>
<td>6</td>
</tr>
<tr>
<td>28106</td>
<td>Lustenberger</td>
<td>3</td>
</tr>
<tr>
<td>29120</td>
<td>Schweizer</td>
<td>2</td>
</tr>
<tr>
<td>29555</td>
<td>Meier</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Lecture
<table>
<thead>
<tr>
<th>Nr</th>
<th>Title</th>
<th>CP</th>
<th>PersNr</th>
</tr>
</thead>
<tbody>
<tr>
<td>5001</td>
<td>Databases</td>
<td>4</td>
<td>2137</td>
</tr>
<tr>
<td>5041</td>
<td>Networks</td>
<td>4</td>
<td>2125</td>
</tr>
<tr>
<td>5043</td>
<td>Operating Systems</td>
<td>3</td>
<td>2126</td>
</tr>
<tr>
<td>5049</td>
<td>Programming</td>
<td>2</td>
<td>2125</td>
</tr>
<tr>
<td>4052</td>
<td>Architecture</td>
<td>4</td>
<td>2125</td>
</tr>
<tr>
<td>5052</td>
<td>Theory</td>
<td>3</td>
<td>2126</td>
</tr>
<tr>
<td>5216</td>
<td>Graphics</td>
<td>2</td>
<td>2126</td>
</tr>
<tr>
<td>5259</td>
<td>Distributed Systems</td>
<td>2</td>
<td>2133</td>
</tr>
<tr>
<td>5022</td>
<td>Formal Methods</td>
<td>2</td>
<td>2134</td>
</tr>
<tr>
<td>4630</td>
<td>Probability</td>
<td>4</td>
<td>2137</td>
</tr>
</tbody>
</table>

#### Assistant
<table>
<thead>
<tr>
<th>Pers Nr</th>
<th>Name</th>
<th>Area</th>
<th>Boss</th>
</tr>
</thead>
<tbody>
<tr>
<td>3002</td>
<td>Heinis</td>
<td>Databases</td>
<td>2125</td>
</tr>
<tr>
<td>3003</td>
<td>Müller</td>
<td>Theory</td>
<td>2125</td>
</tr>
<tr>
<td>3004</td>
<td>Kemme</td>
<td>Networks</td>
<td>2126</td>
</tr>
<tr>
<td>3005</td>
<td>Frey</td>
<td>Graphics</td>
<td>2127</td>
</tr>
<tr>
<td>3006</td>
<td>Peter</td>
<td>Operating Systems</td>
<td>2127</td>
</tr>
<tr>
<td>3007</td>
<td>Kraska</td>
<td>Formal Methods</td>
<td>2126</td>
</tr>
</tbody>
</table>

#### Student tests
<table>
<thead>
<tr>
<th>Legi</th>
<th>Nr</th>
<th>PersNr</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>28106</td>
<td>5001</td>
<td>2126</td>
<td>1</td>
</tr>
<tr>
<td>25403</td>
<td>5041</td>
<td>2125</td>
<td>2</td>
</tr>
<tr>
<td>27550</td>
<td>4630</td>
<td>2137</td>
<td>2</td>
</tr>
</tbody>
</table>

#### Prerequisite and Follow-up
<table>
<thead>
<tr>
<th>Prerequisite</th>
<th>Follow-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>5001</td>
<td>5041</td>
</tr>
<tr>
<td>5001</td>
<td>5043</td>
</tr>
<tr>
<td>5001</td>
<td>5049</td>
</tr>
<tr>
<td>5041</td>
<td>5216</td>
</tr>
<tr>
<td>5043</td>
<td>5052</td>
</tr>
<tr>
<td>5041</td>
<td>5052</td>
</tr>
<tr>
<td>5052</td>
<td>5259</td>
</tr>
</tbody>
</table>

#### Lecture requires
<table>
<thead>
<tr>
<th>Nr</th>
<th>Title</th>
<th>CP</th>
<th>PersNr</th>
</tr>
</thead>
<tbody>
<tr>
<td>5001</td>
<td>Databases</td>
<td>4</td>
<td>2137</td>
</tr>
<tr>
<td>5041</td>
<td>Networks</td>
<td>4</td>
<td>2125</td>
</tr>
<tr>
<td>5043</td>
<td>Operating Systems</td>
<td>3</td>
<td>2126</td>
</tr>
<tr>
<td>5049</td>
<td>Programming</td>
<td>2</td>
<td>2125</td>
</tr>
<tr>
<td>4052</td>
<td>Architecture</td>
<td>4</td>
<td>2125</td>
</tr>
<tr>
<td>5052</td>
<td>Theory</td>
<td>3</td>
<td>2126</td>
</tr>
<tr>
<td>5216</td>
<td>Graphics</td>
<td>2</td>
<td>2126</td>
</tr>
<tr>
<td>5259</td>
<td>Distributed Systems</td>
<td>2</td>
<td>2133</td>
</tr>
<tr>
<td>5022</td>
<td>Formal Methods</td>
<td>2</td>
<td>2134</td>
</tr>
<tr>
<td>4630</td>
<td>Probability</td>
<td>4</td>
<td>2137</td>
</tr>
</tbody>
</table>
DDL (DATA DEFINITION LANGUAGE)
(Simple) Data Definition in 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 (character large object)
- date

Create Tables
- create table Professor
  (PersNr 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;`
DML (DATA MANIPULATION LANGUAGE)
Updates (DML)

- **Insert Tuples**

  ```sql
  insert into Student (Legi, Name) 
  values (28121, 'Frey');
  ```

- Can also be done with nested queries (the nested one selects what to insert, the outer one inserts)

  ```sql
  insert into attends (Legi, 'Databases') 
  select Legi 
  from Student 
  where semester > 2;
  ```
# Updates (DML)

<table>
<thead>
<tr>
<th>Student</th>
<th>Name</th>
<th>Semester</th>
</tr>
</thead>
<tbody>
<tr>
<td>Legi</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29120</td>
<td>Walter</td>
<td>2</td>
</tr>
<tr>
<td>29555</td>
<td>Gärtner</td>
<td>2</td>
</tr>
<tr>
<td>28121</td>
<td>Frey</td>
<td>-</td>
</tr>
</tbody>
</table>

Null
Sequence Types (Automatic Increment for Surrogates)

create sequence PersNr_seq increment by 1 start with 1;
insert into Professor(PersNr, Name) values (PersNr_seq.nextval, "Roscoe");

- Syntax is vendor dependent
  - E.g., AUTO-INCREMENT Option in MySQL
  - Syntax above was standardized in SQL 2003
Updates (ctd.)

- **Delete tuples**
  
  ```sql
  delete Student
  where Semester > 13;
  ```

- **Update tuples**
  
  ```sql
  update Student
  set Semester = Semester + 1;
  ```
ETL

- Extract, Transform, Load (ETL)
- Populating a real database cannot be done manually tuple by tuple: cumbersome, error prone, inaccurate, etc.
- Automatic tools are used to:
  - Extract: get the data from some file
  - Transform: apply transformation to the data when needed (types, conversion, format ...)
  - Load: insert into the database as a bulk operation
QUERY LANGUAGE
Queries

```
select PersNr, Name
from Professor
where Level = 'FP';
```

<table>
<thead>
<tr>
<th>PersNr</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2125</td>
<td>Meyer</td>
</tr>
<tr>
<td>2126</td>
<td>Kossmann</td>
</tr>
<tr>
<td>2136</td>
<td>Welzl</td>
</tr>
<tr>
<td>2137</td>
<td>Norrie</td>
</tr>
</tbody>
</table>

```
select *
from Professor
where Level = 'AP';
```

<table>
<thead>
<tr>
<th>PersNr</th>
<th>Name</th>
<th>Level</th>
<th>Num</th>
</tr>
</thead>
<tbody>
<tr>
<td>2127</td>
<td>Roscoe</td>
<td>AP</td>
<td>310</td>
</tr>
<tr>
<td>2133</td>
<td>Perrig</td>
<td>AP</td>
<td>52</td>
</tr>
<tr>
<td>2134</td>
<td>Sorkine</td>
<td>AP</td>
<td>309</td>
</tr>
</tbody>
</table>
**Queries: Sorting**

```
select PersNr, Name, Level 
from Professor 
order by Level desc, Name asc;
```

<table>
<thead>
<tr>
<th>PersNr</th>
<th>Name</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>2136</td>
<td>Welzl</td>
<td>FP</td>
</tr>
<tr>
<td>2137</td>
<td>Norrie</td>
<td>FP</td>
</tr>
<tr>
<td>2125</td>
<td>Meyer</td>
<td>FP</td>
</tr>
<tr>
<td>2126</td>
<td>Kossmann</td>
<td>FP</td>
</tr>
<tr>
<td>2134</td>
<td>Sorkine</td>
<td>AP</td>
</tr>
<tr>
<td>2127</td>
<td>Roscoe</td>
<td>AP</td>
</tr>
<tr>
<td>2133</td>
<td>Perrig</td>
<td>AP</td>
</tr>
</tbody>
</table>
Duplicate Elimination

```sql
select distinct Level
from Professor
```

<table>
<thead>
<tr>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>AP</td>
</tr>
<tr>
<td>FP</td>
</tr>
</tbody>
</table>

What happens with these ones?:

```sql
select distinct Name, Level
from Professor
```

```sql
select distinct PersNr
from Lecture
```
Who teaches Operating Systems?

```sql
select Name
from Professor, Lecture
where PersNr = ProfNr and Title = 'Operating Systems' ;
```

\[
\Pi_{\text{Name}} (\sigma \text{PersNr = ProfNr \land Title = 'Operating Systems'}(\text{Professor} \times \text{Lecture}))
\]

N.B.: Renamed Lecture.PersNr to ProfNr. Will show later how this can be done as part of a query.
SQL -> Relational Algebra

SQL

select $A_1, \ldots, A_n$
from $R_1, \ldots, R_k$
where $P$;

Relational Algebra

\[
\Pi_{A_1, \ldots, A_n}(\sigma_P (R_1 \times \ldots \times R_k))
\]

D-INFK, ETH Zurich, Data Modeling and Databases
Joins and Tuple Variables

Who attends which lecture?

```sql
select Name, Title
from Student, attends, Lecture
where Student.Legi = attends.Legi and
    attends.Nr = Lecture.Nr;
```

Alternative:

```sql
select s.Name, l.Title
from Student s, attends a, Lecture l
where s.Legi = a.Legi and
    a.Nr = l.Nr;
```
Rename of Attributes

Give title and professor of all lectures?

```sql
select Title, PersNr as ProfNr
from Lecture;
```
Set Operations

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

( select Name  
  from Assistant )
union  
( select Name  
  from Professor);
Grouping, Aggregation

Aggregate functions: \texttt{avg, max, min, count, sum}

\begin{verbatim}
select avg (Semester) from Student;
\end{verbatim}
Grouping, Aggregation

Aggregate functions: \texttt{avg}, \texttt{max}, \texttt{min}, \texttt{count}, \texttt{sum}

\begin{verbatim}
select avg (Semester)
from Student;

select PersNr, sum (CP)
from Lecture
group by PersNr;
\end{verbatim}
Grouping, Aggregation

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

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

select PersNr, sum (CP)
from Lecture
group by PersNr;

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

- Remember modeling!!!

- All students who have not completed the requisites to get the bachelor
  - Do we have all information we need?
  - How is it organized?
  - Watch the negation

- Try it at home ...
Existential Quantification:

exists sub-queries

select p.Name
from Professor p
where not exists ( select *
from Lecture l
where l.PersNr = p.PersNr
);
Correlated Sub-queries

```
select p.Name
from Professor p
where not exists ( select *
    from Lecture l
    where l.PersNr = p.PersNr );
```
Uncorrelated Sub-query

```sql
select Name
from Professor
where PersNr not in ( select PersNr
                        from Lecture);
```
Sub-queries with all

Not as powerful as relational division!

```sql
select Name
from Student
where Semester >= all ( select Semester
  from Student);
```
From the Oracle manual

- Assuming subqueries don't return zero rows, the following statements can be made for both list and subquery versions:
  - "x = ALL (...)": The value must match all the values in the list to evaluate to TRUE.
  - "x != ALL (...)": The value must not match any values in the list to evaluate to TRUE.
  - "x > ALL (...)": The value must be greater than the biggest value in the list to evaluate to TRUE.
  - "x < ALL (...)": The value must be smaller than the smallest value in the list to evaluate to TRUE.
  - "x >= ALL (...)": The value must be greater than or equal to the biggest value in the list to evaluate to TRUE.
  - "x <= ALL (...)": The value must be smaller than or equal to the smallest value in the list to evaluate to TRUE.

- If a subquery returns zero rows, the condition evaluates to TRUE.
Query rewrites (ORACLE)

SELECT empno, sal
FROM emp
WHERE sal > ALL (2000, 3000, 4000);

Transformed to equivalent statement without ALL:

SELECT empno, sal
FROM emp
WHERE sal > 2000 AND sal > 3000 AND sal > 4000;
Query rewrites (Oracle)

SELECT e1.empno, e1.sal
FROM emp e1
WHERE e1.sal > ALL (SELECT e2.sal
    FROM emp e2
    WHERE e2.deptno = 20);

Transformed to equivalent statement without ALL:

SELECT e1.empno, e1.sal
FROM emp e1
WHERE NOT EXISTS (SELECT e2.sal
    FROM emp e2
    WHERE e2.deptno = 20
    AND e1.sal <= e2.sal);
Subqueries in SELECT, FROM

```sql
select PersNr, Name, ( select sum (CP) as load
from Lecture l
where p.PersNr=l.PersNr )
from Professor p;
```

```sql
select p.PersNr, Name, l.load
from Professor p, ( select PersNr, sum (CP) as load
from Lecture
group by PersNr ) l
where p.PersNr = l.PersNr;
```
Query Rewrite

```sql
select *
from Assistant a
where exists
  ( select *
      from Professor p
      where a.Boss = p.PersNr and p.age < a.age);
```

Equivalent Join Query: Why is this better?

```sql
select a.*
from Assistant a, Professor p
```
Universal Quantification

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

\[
\{ s \mid s \in \text{Student} \land \\
\forall l \in \text{Lecture} \ (l.CP=4 \Rightarrow \\
\exists a \in \text{attends} \ (a.Nr=l.Nr \land a.Legi=s.Legi)) \}
\]

- Approach: Elimination of \( \forall \) and \( \Rightarrow \)

\[
\forall t \in R \ (P(t)) = \neg (\exists t \in R(\neg P(t))) \\
R \Rightarrow T = \neg R \lor T
\]
Applying these rules:
\[
\{ s \mid s \in \text{Student} \land \neg (\exists l \in \text{Lecture} \neg (l.\text{CP}=4) \lor \\
\exists a \in \text{attends}(a.\text{Nr}=l.\text{Nr} \land a.\text{Legi}=s.\text{Legi})) \}
\]

Applying DeMorgan rules:
\[
\{ s \mid s \in \text{Student} \land \neg (\exists l \in \text{Lecture}(l.\text{CP}=4 \land \\
\neg (\exists a \in \text{attends}(a.\text{Nr}=l.\text{Nr} \land a.\text{Legi}=s.\text{Legi})))) \}
\]
This can be implemented in SQL:

```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.Nr = l.Nr and a.Legi=s.Legi)
  );
```
Or do it this way

```
select a.Legi
from attends a
group by a.Legi
having count(*) = (select count(*) from Lecture);
```
Considering only 4 CP lectures

```sql
select a.Legi
from attends a, Lecture l
where a.Nr = l.Nr and l.CP = 4
group by a.Legi
having count(*) = (select count(*) from Lecture
    where CP = 4);
```