Relational Query Languages

• A major strength of the relational model: supports simple, powerful querying of data
• Queries can be written intuitively, and DBMS is responsible for efficient evaluation
  • Relational queries have precise semantics
  • The DBMS query optimizer can extensively reorder operations to answer a query, and still ensure that query results are correct
CREATE TABLE Students
(sid CHAR(20),
name CHAR(30),
login CHAR(20),
age INTEGER,
gpa REAL,
PRIMARY KEY(sid),
UNIQUE(login))

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Luis</td>
<td>l@cs</td>
<td>85</td>
<td>3.2</td>
</tr>
<tr>
<td>3</td>
<td>Panos</td>
<td>p@cs</td>
<td>30</td>
<td>4.0</td>
</tr>
<tr>
<td>4</td>
<td>Amelie</td>
<td>a@cs</td>
<td>30</td>
<td>4.1</td>
</tr>
<tr>
<td>5</td>
<td>Hila</td>
<td>h@cs</td>
<td>29</td>
<td>4.1</td>
</tr>
</tbody>
</table>

CREATE TABLE Courses
(cid CHAR(10),
cname CHAR(10),
credits INTEGER,
PRIMARY KEY(cid))

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DB</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>OS</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>PLT</td>
<td>4</td>
</tr>
</tbody>
</table>

CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(10),
grade CHAR(2),
FOREIGN KEY (sid) REFERENCES Students,
FOREIGN KEY (cid) REFERENCES Courses)

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>C-</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>A+</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>A+</td>
</tr>
</tbody>
</table>
The SQL Query Language

• SQL is the most widely used relational query language

• To find all 30-year-old students, with all their attributes, we can write:

```sql
SELECT * 
FROM Students S 
WHERE S.age=30
```

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Panos</td>
<td>p@cs</td>
<td>30</td>
<td>4.0</td>
</tr>
<tr>
<td>4</td>
<td>Amelie</td>
<td>a@cs</td>
<td>30</td>
<td>4.1</td>
</tr>
</tbody>
</table>

• To find just the name and login attributes, change the first line:

```sql
SELECT S.name, S.login 
FROM Students S 
WHERE S.age=30
```

<table>
<thead>
<tr>
<th>name</th>
<th>login</th>
</tr>
</thead>
<tbody>
<tr>
<td>Panos</td>
<td>p@cs</td>
</tr>
<tr>
<td>Amelie</td>
<td>a@cs</td>
</tr>
</tbody>
</table>
Querying Multiple Relations

CREATE TABLE Students
(sid CHAR(20),
 name CHAR(30),
 login CHAR(20),
 age INTEGER,
 gpa REAL,
 PRIMARY KEY(sid),
 UNIQUE(login))

CREATE TABLE Courses
(cid CHAR(10),
cname CHAR(10),
 credits INTEGER,
 PRIMARY KEY(cid))

CREATE TABLE Enrolled
(sid CHAR(20),
cid CHAR(10),
 grade CHAR(2),
 PRIMARY KEY(sid, cid),
 FOREIGN KEY (sid) REFERENCES Students,
 FOREIGN KEY (cid) REFERENCES Courses)

What does the following query request?

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='A'
```
### Students

<table>
<thead>
<tr>
<th>sid</th>
<th>name</th>
<th>login</th>
<th>age</th>
<th>gpa</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Luis</td>
<td>l@cs</td>
<td>85</td>
<td>3.2</td>
</tr>
<tr>
<td>3</td>
<td>Panos</td>
<td>p@cs</td>
<td>30</td>
<td>4.0</td>
</tr>
<tr>
<td>4</td>
<td>Amelie</td>
<td>a@cs</td>
<td>30</td>
<td>4.1</td>
</tr>
<tr>
<td>5</td>
<td>Hila</td>
<td>h@cs</td>
<td>29</td>
<td>4.1</td>
</tr>
</tbody>
</table>

### Courses

<table>
<thead>
<tr>
<th>cid</th>
<th>cname</th>
<th>credits</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>DB</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>OS</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>PLT</td>
<td>4</td>
</tr>
</tbody>
</table>

### Enrolled

<table>
<thead>
<tr>
<th>sid</th>
<th>cid</th>
<th>grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>C-</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
<td>A</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>A+</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>A+</td>
</tr>
</tbody>
</table>

SQL Query:

```sql
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='A'
```
Semantics of a SQL Query

```
SELECT S.name, E.cid
FROM Students S, Enrolled E
WHERE S.sid=E.sid AND E.grade='A'
```

A conceptual evaluation method for this query:

1. **FROM clause:** Compute cross-product of Students and Enrolled
2. **WHERE clause:** Check conditions, discard tuples that fail
3. **SELECT clause:** Delete unwanted attributes (i.e., all attributes except for name and cid)

This is only conceptual, to specify the meaning of the query and what its results should be; an actual evaluation will typically be much more efficient, but must produce the same answers as the conceptual evaluation.
<table>
<thead>
<tr>
<th>S.sid</th>
<th>S.name</th>
<th>S.login</th>
<th>S.age</th>
<th>S.gpa</th>
<th>E.sid</th>
<th>E.cid</th>
<th>E.grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Luis</td>
<td>l@cs</td>
<td>85</td>
<td>3.2</td>
<td>1</td>
<td>1</td>
<td>C-</td>
</tr>
<tr>
<td>1</td>
<td><strong>Luis</strong></td>
<td>l@cs</td>
<td>85</td>
<td>3.2</td>
<td>1</td>
<td><strong>5</strong></td>
<td><strong>A</strong></td>
</tr>
<tr>
<td>1</td>
<td>Luis</td>
<td>l@cs</td>
<td>85</td>
<td>3.2</td>
<td>4</td>
<td>1</td>
<td><strong>A+</strong></td>
</tr>
<tr>
<td>1</td>
<td>Luis</td>
<td>l@cs</td>
<td>85</td>
<td>3.2</td>
<td>3</td>
<td>1</td>
<td><strong>A+</strong></td>
</tr>
<tr>
<td>3</td>
<td>Panos</td>
<td>p@cs</td>
<td>30</td>
<td>4.0</td>
<td>1</td>
<td>1</td>
<td><strong>C-</strong></td>
</tr>
<tr>
<td>3</td>
<td>Panos</td>
<td>p@cs</td>
<td>30</td>
<td>4.0</td>
<td>1</td>
<td><strong>5</strong></td>
<td><strong>A</strong></td>
</tr>
<tr>
<td>3</td>
<td>Panos</td>
<td>p@cs</td>
<td>30</td>
<td>4.0</td>
<td>4</td>
<td>1</td>
<td><strong>A+</strong></td>
</tr>
<tr>
<td>3</td>
<td>Panos</td>
<td>p@cs</td>
<td>30</td>
<td>4.0</td>
<td>3</td>
<td>1</td>
<td><strong>A+</strong></td>
</tr>
<tr>
<td>4</td>
<td>Amelie</td>
<td>a@cs</td>
<td>30</td>
<td>4.1</td>
<td>1</td>
<td>1</td>
<td><strong>C-</strong></td>
</tr>
<tr>
<td>4</td>
<td>Amelie</td>
<td>a@cs</td>
<td>30</td>
<td>4.1</td>
<td>1</td>
<td><strong>5</strong></td>
<td><strong>A</strong></td>
</tr>
<tr>
<td>4</td>
<td>Amelie</td>
<td>a@cs</td>
<td>30</td>
<td>4.1</td>
<td>4</td>
<td>1</td>
<td><strong>A+</strong></td>
</tr>
<tr>
<td>4</td>
<td>Amelie</td>
<td>a@cs</td>
<td>30</td>
<td>4.1</td>
<td>3</td>
<td>1</td>
<td><strong>A+</strong></td>
</tr>
<tr>
<td>5</td>
<td>Hila</td>
<td>h@cs</td>
<td>29</td>
<td>4.1</td>
<td>1</td>
<td>1</td>
<td><strong>C-</strong></td>
</tr>
<tr>
<td>5</td>
<td>Hila</td>
<td>h@cs</td>
<td>29</td>
<td>4.1</td>
<td>1</td>
<td><strong>5</strong></td>
<td><strong>A</strong></td>
</tr>
<tr>
<td>5</td>
<td>Hila</td>
<td>h@cs</td>
<td>29</td>
<td>4.1</td>
<td>4</td>
<td>1</td>
<td><strong>A+</strong></td>
</tr>
<tr>
<td>5</td>
<td>Hila</td>
<td>h@cs</td>
<td>29</td>
<td>4.1</td>
<td>3</td>
<td>1</td>
<td><strong>A+</strong></td>
</tr>
</tbody>
</table>
Mapping ER Diagrams to Relational Schemas

Entity sets get mapped to tables

CREATE TABLE Employees
(ssn CHAR(11),
 name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))

CREATE TABLE Departments
(did INTEGER,
dname CHAR(20),
budget REAL,
PRIMARY KEY (did))
Mapping ER Diagrams to Relational Schemas

```
CREATE TABLE Employees
(ssn CHAR(11),
 name CHAR(20),
 lot INTEGER,
 PRIMARY KEY (ssn))

CREATE TABLE Departments
(did INTEGER,
 dname CHAR(20),
 budget REAL,
 PRIMARY KEY (did))
```

Employees

<table>
<thead>
<tr>
<th>ssn</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string</td>
</tr>
<tr>
<td>lot</td>
<td>integer</td>
</tr>
</tbody>
</table>

Deparments

<table>
<thead>
<tr>
<th>did</th>
<th>integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>dname</td>
<td>string</td>
</tr>
<tr>
<td>budget</td>
<td>real</td>
</tr>
</tbody>
</table>

Working In

| since | date    |
Mapping ER Diagrams to Relational Schemas

Relationship sets **without constraints** get mapped to a relation, with:

- Attributes for the primary key of each participating entity set, specified as foreign keys
- All descriptive attributes for the relationship set

```sql
CREATE TABLE Works_In(
    ssn CHAR(11),
    did INTEGER,
    since DATE,
    PRIMARY KEY (ssn, did),
    FOREIGN KEY (ssn) REFERENCES Employees,
    FOREIGN KEY (did) REFERENCES Departments
)
```
CREATE TABLE Employees
(ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))

CREATE TABLE Departments
(did INTEGER,
dname CHAR(20),
budget REAL,
PRIMARY KEY (did))
CREATE TABLE Employees
(ssn CHAR(11),
 name CHAR(20),
 lot INTEGER,
 PRIMARY KEY (ssn))

CREATE TABLE Departments
(did INTEGER,
 dname CHAR(20),
 budget REAL,
 PRIMARY KEY (did))
Translating ER Diagrams with Key Constraints

Option 1: Separate tables for Employees and Departments, plus table for Manages, with did as key (3 tables total)

CREATE TABLE Manages(
    ssn CHAR(11),
    did INTEGER,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees,
    FOREIGN KEY (did) REFERENCES Departments)
Option 2: Table for Employees plus table for Departments+Manages together, with did as key (2 tables total): possible only because each department has at most one manager

CREATE TABLE Dept_Mgr(
did INTEGER,
dname CHAR(20),
budget REAL,
ssn CHAR(11),
since DATE,
PRIMARY KEY (did),
FOREIGN KEY (ssn) REFERENCES Employees)
Modifying Mapping For 2 Key Constraints?

Option 1:

```sql
CREATE TABLE Manages(
  ssn CHAR(11),
  did INTEGER,
  since DATE,
  PRIMARY KEY (did),

  FOREIGN KEY (ssn) REFERENCES Employees,
  FOREIGN KEY (did) REFERENCES Departments)
```
Modifying Mapping For 2 Key Constraints?

Option 2:
CREATE TABLE Dept_Mgr(  
did INTEGER,  
dname CHAR(20),  
budget REAL,  
ssn CHAR(11),  
since DATE,  
PRIMARY KEY (did),  
FOREIGN KEY (ssn) REFERENCES Employees)
Now let’s introduce a participation constraint …

CREATE TABLE Dept_Mgr(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    ssn CHAR(11),
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees)

CREATE TABLE Employees(
    ssn CHAR(11),
    name CHAR(20),
    lot INTEGER,
    PRIMARY KEY (ssn))
Participation Constraints in SQL

We can capture participation constraints involving one entity set in a binary relationship if that entity set also has a key constraint, as in the previous slide, but little else (without resorting to CHECK constraints, which we will see much later).

```sql
CREATE TABLE Dept_Mgr(
    did INTEGER,
    dname CHAR(20),
    budget REAL,
    ssn CHAR(11) NOT NULL,
    since DATE,
    PRIMARY KEY (did),
    FOREIGN KEY (ssn) REFERENCES Employees,
    ON DELETE NO ACTION)
```
Participation Constraints Without Key Constraints?

Now Departments has total participation in Works_In (without a key constraint):

```
CREATE TABLE Employees
(ssn CHAR(11),
 name CHAR(20),
 lot INTEGER,
 PRIMARY KEY (ssn))

CREATE TABLE Departments
(did INTEGER,
 dname CHAR(20),
 budget REAL,
 PRIMARY KEY (did))

CREATE TABLE Works_In
(ssn CHAR(11),
 did INTEGER,
 since DATE,
 FOREIGN KEY (ssn) REFERENCES Employees,
 FOREIGN KEY (did) REFERENCES Departments)
```

Modifying earlier mapping?

```
CREATE TABLE Employees
(ssn CHAR(11),
 name CHAR(20),
 lot INTEGER,
 PRIMARY KEY (ssn))

CREATE TABLE Departments
(did INTEGER,
 dname CHAR(20),
 budget REAL,
 PRIMARY KEY (did))

CREATE TABLE Works_In
(ssn CHAR(11),
 did INTEGER,
 since DATE,
 PRIMARY KEY (ssn, did),
 FOREIGN KEY (ssn) REFERENCES Employees,
 FOREIGN KEY (did) REFERENCES Departments)
```
Participation Constraints Without Key Constraints?

Now Departments has **total participation** in `Works_In` (without a key constraint):

Unfortunately, we **cannot** model such total participation case in SQL (yet). For now, when you map such diagrams to SQL in your homework and projects, just **mention you cannot** capture such participation constraints (yet) in SQL.
CREATE TABLE Employees
(ssn CHAR(11),
nname CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))
CREATE TABLE Dep_Policy (
    pname CHAR(20),
    age INTEGER,
    cost REAL,
    ssn CHAR(11),
    PRIMARY KEY (pname, ssn),
    FOREIGN KEY (ssn) REFERENCES Employees
    ON DELETE CASCADE
)

CREATE TABLE Employees
(ssn CHAR(11),
 name CHAR(20),
 lot INTEGER,
 PRIMARY KEY (ssn))

Weak entity set and identifying relationship set are translated into a single table.

When identifying entity is deleted, all corresponding weak entities must also be deleted (i.e., ON DELETE CASCADE)
## Mapping ISA Hierarchies

```sql
CREATE TABLE Employees
(ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))
```

<table>
<thead>
<tr>
<th>Employees</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ssn: string</td>
<td>name: string</td>
<td>lot: integer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hourly_Emps</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>hourly_wages: real</td>
<td></td>
<td></td>
</tr>
<tr>
<td>hours_worked: integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Contract_Emps</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>contract_id: integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Mapping ISA Hierarchies

General approach uses 3 relations: Employees, Hourly_Emps, and Contract_Emps

- **Employees** “records” every employee with the attributes of Employees entity set
- **Hourly_Emps** “records” the ssn of every hourly employee, which is a foreign key into Employees, plus attributes hourly_wages and hours_worked; must delete an Hourly_Emps tuple if the referenced Employees tuple is deleted (i.e., **ON DELETE CASCADE**)
- **Contract_Emps** is defined analogously as Hourly_Emps

- **Queries** involving all employees are efficient, but those involving just Hourly_Emps or Contract_Emps require a join to get employee attributes beyond ssn, which can be costly
Mapping ISA Hierarchies

CREATE TABLE Employees(  
  ssn CHAR(11),  
  name CHAR(20),  
  lot INTEGER,  
  PRIMARY KEY(ssn))

CREATE TABLE Hourly_Emps(  
  ssn CHAR(11),  
  hourly_wages REAL,  
  hours_worked INTEGER,  
  PRIMARY KEY (ssn),  
  FOREIGN KEY (ssn) REFERENCES Employees  
  ON DELETE CASCADE)

CREATE TABLE Contract_Emps(  
  ssn CHAR(11),  
  contract_id INTEGER,  
  PRIMARY KEY (ssn),  
  FOREIGN KEY (ssn) REFERENCES Employees  
  ON DELETE CASCADE)

Use this 3-table mapping if:  
- Hourly_Emps and Contract_Emps can overlap, or  
- Hourly_Emps and Contract_Emps cannot overlap but they do not cover Employees, or  
- Employees participates in one or more relationship sets or additional ISA hierarchies
Mapping ISA Hierarchies: A Special Case

Now we have:

1. Hourly_Emps and Contract_Emps cannot overlap
2. Hourly_Emps and Contract_Emps cover Employees

If, additionally, (3) Employees does not participate in any relationship sets or other ISA hierarchies, then we can use just 2 tables (rather than 3); all 3 conditions (1), (2), and (3) must hold for this mapping to work.
Mapping ISA Hierarchies: A Special Case

CREATE TABLE Hourly_Emps(
    ssn CHAR(11),
    name CHAR(20),
    lot INTEGER,
    hourly_wages REAL,
    hours_worked INTEGER,
    PRIMARY KEY (ssn))

CREATE TABLE Contract_Emps(
    ssn CHAR(11),
    name CHAR(20),
    lot INTEGER,
    contract_id INTEGER,
    PRIMARY KEY (ssn))

Note this works only if **all 3 conditions** in previous slide hold; we don’t create a dedicated Employees table in this special case
Mapping ISA Hierarchies

What if Hourly_Emps and Contract_Emps can overlap?
Follow the 3-table design in the earlier slide (otherwise name and lot would be redundantly stored twice for employees in the intersection, which is bad for data integrity)

What if Employees participates in a relationship set or another ISA hierarchy?
Again, follow the 3-table design in the earlier slide (otherwise you couldn’t refer to “generic” employees with a single foreign key from other tables, which you need to do in this case)
Mapping Aggregation

CREATE TABLE Employees
(ssn CHAR(11),
nname CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))

CREATE TABLE Departments
(did INTEGER,
dname CHAR(20),
budget REAL,
PRIMARY KEY (did))

CREATE TABLE Locations
(address CHAR(50),
capacity INTEGER,
PRIMARY KEY (address))

CREATE TABLE Is_At
(did INTEGER,
address CHAR(50),
FOREIGN KEY (did)
REFERENCES Departments,
FOREIGN KEY (address)
REFERENCES Locations

NOTE: Cannot yet model participation constraint of Departments into Is_At
Mapping Aggregation

CREATE TABLE Employees
(ssn CHAR(11),
name CHAR(20),
lot INTEGER,
PRIMARY KEY (ssn))

CREATE TABLE Departments
(did INTEGER,
dname CHAR(20),
budget REAL,
PRIMARY KEY (did))

CREATE TABLE Locations
(address CHAR(50),
capacity INTEGER,
PRIMARY KEY (address))

CREATE TABLE Is_At
(did INTEGER,
address CHAR(50),
PRIMARY KEY (did, address),
FOREIGN KEY (did)
REFERENCES Departments,
FOREIGN KEY (address)
REFERENCES Locations)

NOTE: Cannot yet model participation constraint of Departments into Is_At

CREATE TABLE Works_In
(ssn CHAR(11),
did INTEGER,
address CHAR(50),
PRIMARY KEY (ssn, did, address),
FOREIGN KEY (ssn)
REFERENCES Employees,
FOREIGN KEY (did, address)
REFERENCES Is_At (did, address))

NOTE: Completely analogous to mapping of relationship set not involving aggregation
Relational Model: Summary

• Relational model provides tabular representation of data
  • Simple and intuitive, most widely used

• Integrity constraints can be specified based on application semantics

• DBMS never allows violations of integrity constraints