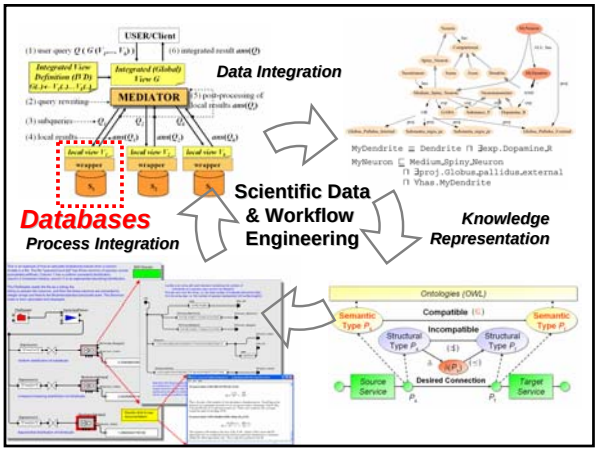
Course Overview

1. Data Integration and …
   - **structured (relational) databases**
   - knowledge-based extensions, ontologies
   - semi-structured (XML) databases

2. Scientific Workflows
   - Dataflow process networks
   - Web service workflows
   - The Kepler system

3. Student projects on (1) and (2)



Perfect Recall: Database Systems (➔ 165A)

- A Database System (DBS) consists of a Database (DB) and a Database Management System (DBMS)
- A Database is a (typically very large) integrated collection of interrelated data which are stored in files.
- Data can come from commercial or scientific applications and (usually) represent some abstraction/piece of the modeled real world.
- E.g, a scientific database might contain information about known biological, chemical, astronomical entities, lab experiments, etc
- A Database Management System is a collection of software packages designed to store, access, and manage databases. It provides users and applications with an environment that is convenient and efficient to use.

Relational Database Model

- Think of a relational DB as a number of *tables*, each have a particular *schema*:
  - Course(Instructor, Name, Quarter, Department)
- The *table/relation name* “Course”, identifies which table we are talking about.
- The *attribute/column name* (e.g., “Instructor”) corresponds to the “column header”
- Elements aka *instances* or *tuples* of a table/relation can be written, e.g., as follows:

  Course(“Gertz”, “ECS165A”, “W-2005”, “CS”).

  Course(“Ludaescher”, “ECS289F”, “W-2005”, “CS”).

  …

Example

Course

Instructor	Name	Quarter	Department
Gertz	ECS165A	W-2005	CS
Ludaescher	ECS289F	W-2005	CS
…	…	…	…

- The same in ***Datalog notation*** – as a set of ***facts***:

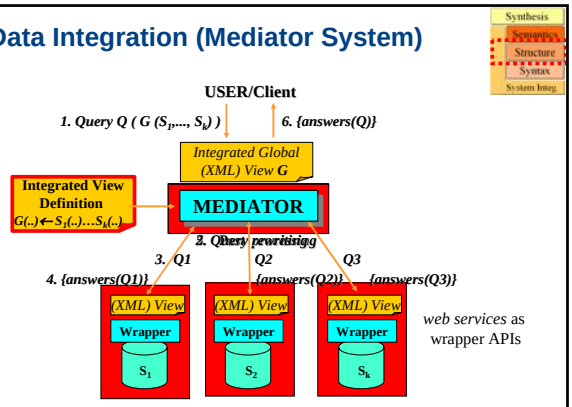
  course(‘Ludaescher’, ‘ECS289F’, ‘W-2005’, ‘CS’).

  course(… , … , … , …).

Hmm.. looks like a Spreadsheet …

- … but there are differences.
- What are they?

Data Integration (Mediator System)



Query Languages

- Databases can be ***queried***!
- We state a question, usually in terms of the given database schema, about the stored data.
- ***Query languages*** such as Datalog and SQL (Structured Query Language) are *declarative* (just say what you're interested in) – you do not need to give the details *how* to retrieve the data, but can focus on the *what* (to retrieve).

Question

- What's the difference between *keyword-based search* and *querying a database?*

- But watch out
  - … some recent work in the database community on "keyword search in databases"…

DATALOG

- A **relational database** is given as a set of **facts**:

```
employee(john, 40000, toys). ...
employee(mary, 65000, cs). ...
...
dept(cs, mary). ...
```

- A **Datalog program** defines **views** by means of **rules** of the form **Head** ← **Body**:
  `boss(Emp,Mgr) ← employee(Emp, Salary, DeptNo), dept(DeptNo,Mgr)`
  `highpaid(Emp) ← employee(Emp, Salary, _), Salary > 60000`
- **EDB**: extensionally defined relations (facts): `employee/3, dept/2`
- **IDB**: intensionally (i.e., rule-) defined relations (views): `boss/2`
- A **query** is a view with a distinguished **answer**/n relation:
  `answer(Emp,Mgr) ← employee(Emp, Salary, DeptNo), dept(DeptNo,Mgr)`

Notation:

- **lowercase**: **relation names** (`employee/3, highpaid/1, ...`) and **constants** (aka data values: `john, toys, 50000, ...`)
- **UPPERCASE/Capitalized**: **variables** (`Emp, X, ...`) ("_" means: don't care)

DATALOG: Examples of Relational Operations

Relational operations have concise representations! Examples:

```
sel(X,Y) :- p(X,Y), X=a, not X=Y.   % SELECT some tuples from p(X,Y)

proj(X) :- p(X,Y).                  % PROJECT on the first argument

join(X,Y,Z) :- p(X,Y), q(Y,Z).      % JOIN p(A,B), q(C,D) s.t. B=C

prod(X,Y) :- p(X), q(Y).            % PRODUCT of p(X) and q(Y)

intersect(X) :- p(X), q(X).         % INTERSECTION of p(X), q(X)

diff(X) :- p(X), not q(X).          % SET-DIFFERENCE: p(X) \ q(X)

union(X) :- p(X).                   % UNION of p(X), ...
union(X) :- q(X).                   % ... and q(X)
```

Rules have a "logical reading" (i.e., rules are formulas):

$$\forall X\ (\ \mathtt{diff}(X) \leftarrow \mathtt{p}(X) \wedge \neg\, \mathtt{q}(X)\ ).$$
$$\forall X\ (\ \mathtt{union}(X) \leftarrow \mathtt{p}(X) \vee \mathtt{q}(X)\ ).$$

What is a Query?

- A **query expression** e.g. in SQL or in Datalog denotes a query (but we still don't know what a query is…)
- A **query** is a (generic*) **mapping** *f* from instances of an input schema (EDB) to instances of an output schema (IDB):
  f : inst(EDB) → inst(IDB)
- Note: Different query expressions can denote the same query (mapping). Example…?

What is a Query?

- A query is a **generic mapping** *f* from instances of an input schema (EDB) to instances of an output schema (IDB):
  f : inst(EDB) → inst(IDB)

*__generic__: invariant under *renamings r, i.e.,*
f (r (I)) = r(f(I)) for all database instances I of the schema EDB

- Examples: Consider EBD = {p(X), emp(N,S)}. Which of the following are generic?
  - f_even: "T" if | {x | p(x) is in DB I} | is even
  - f_jeff: { (N,S) | emp(N,S) in DB I, N = "Jeff" }

Problem

- How can one *evaluate* DATALOG queries? That is, given a database *instance* (= a set of *facts*), how can one obtain the *answer* to a given query (=*rule* or *set of rules*) ?

DATALOG: Fixpoint Semantics (Bottom-Up)

Relations can be defined (directly or indirectly) in terms of themselves (⇒ **recursive** relations/rules):

```
e(a,b). e(b,c).  ...            % FACTS (EDB-relation: e/2)

tc(X,Y) :- e(X,Y).              % RULES (IDB-relation: tc/2)
tc(X,Y) :- e(X,Z), tc(Z,Y).
```

- **Bottom-Up** Evaluation (**Fixpoint Semantics**): Apply rules **iteratively** (in so-called T_P-**rounds**) until a **fixpoint** is reached $(P := Facts \cup Rules)$:

$$\boxed{\begin{array}{l} I_0 := \emptyset \\ I_{n+1} := I_n \cup T_P(I_n) \end{array}}$$

The sequence $I_0 \subseteq I_1 \subseteq I_2 \ldots$ converges[1] to the **least fixpoint** $lfp(T_P)$ of T_P

T_P (**Immediate Consequences**) operator:

$$T_P(I) := \{\sigma(Head) \mid (Head \leftarrow Body) \in P,\ I \models \sigma(Body)\}$$

[1] proviso: P positive

Example: Transitive Closure

```
e(a,b). e(b,c). e(c,d).          % FACTS (EDB-relation: e/2)

tc(X,Y) :- e(X,Y).               % RULES (IDB-relation: tc/2)
tc(X,Y) :- e(X,Z), tc(Z,Y).
```

n	I_n
0	$\emptyset$
1	$\{\mathbf{e(a,b)}, \mathbf{e(b,c)}, \mathbf{e(c,d)}\}$
2	$\{e(a,b), e(b,c), e(c,d)\} \cup \{\mathbf{tc(a,b)}, \mathbf{tc(b,c)}, \mathbf{tc(c,d)}\}$
3	$\{e(a,b), e(b,c), e(c,d)\} \cup \{tc(a,b), tc(b,c), tc(c,d), \mathbf{tc(a,c)}, \mathbf{tc(b,d)}\}$
4	$\{e(a,b), e(b,c), e(c,d)\} \cup \{tc(a,b), tc(b,c), tc(c,d), tc(a,c), tc(b,d), \mathbf{tc(a,d)}\}$

$\Rightarrow$ if the longest path in e/2 has length n, then $O(n)$ rounds are needed!

(Exercise: how about the following rule? `tc(X,Y)` $\leftarrow$ `tc(X,Z), tc(Z,Y).`)

DATALOG: Minimal Model Semantics

- Rules can be seen as first-order formulas:
  `p(X) :- q(X,Y)` $:\Leftrightarrow \forall X(p(X) \leftarrow \exists Y q(X,Y))$.
- A **model** of a program P ($= Fact \cup Rules$) is an interpretation of the relations which satisfies all rules.
  (Here, we're dealing with Herbrand interpretations and models, i.e., which interpret constants "syntactically". Then, an interpretation can be written as a set of *true* tuples; all other tuples are regarded as *false*.)
- From the possible different models find the "intuitive", "intended" models, i.e., which make true only what is "strictly necessary", given the rules of $P \Rightarrow$ minimal models:
- A model M of P is **minimal**, if there is no other model $M' \subset M$.

Query Languages for Relational Databases

- SQL — SELECT ... FROM ... WHERE ...
- Relational Algebra (RA) — $\sigma, \pi, \bowtie, \delta, \cup, \setminus$
- Relational Calculus (RC) — $\forall x F,\ \exists x F,\ F \wedge G,\ F \vee G, \neg F$
- **Datalog** — $\approx$ RC + Recursion

EXAMPLE: Given relations `employee(Emp, Salary, DeptNo)` and `dept(DeptNo, Mgr)`, find all (employee, manager) pairs:

- SQL:
```
SELECT Emp, Mgr
FROM employee, dept
WHERE employee.DeptNo = dept.DeptNo
```
- RA: $\pi_{\texttt{Emp,Mgr}}(\texttt{employee} \bowtie \texttt{dept})$
- RC: $F(\texttt{Emp,Mgr}) =$ $\exists \texttt{Salary, DeptNo} : (\texttt{employee(Emp, Salary, DeptNo)} \wedge \texttt{dept(DeptNo,Mgr)})$
- **Datalog**: `boss(Emp,Mgr)` $\leftarrow$ `employee(Emp, Salary, DeptNo), dept(DeptNo,Mgr)`