Announcements

- Web site and Piazza are "live":
  - sites.google.com/site/ecs165bwinter2014
  - piazza.com/ucdavis/winter2014/ecs165b
  - **Sign Up!** All announcements and online discussions there! (39 already signed up :-)
- Starting next week:
  - Discussion sections
  - Office hours ➔ class site:
- **Individual** Assignment #1 out!
- **Group** Project #1 coming soon..
- Stanford DB class: db-class.org
  - Sign-up! (optional of course …)

ECS-165B Big Picture

- **Database Foundations** ("Theory")
  - Recursive queries (Datalog, SQL-with-recursive)
  - Design Theory for Relational Databases (Normalization)
- **Advanced & Hands-on Topics** ("Practice")
  - XML data management
  - Online Analytical Processing (OLAP), Data Cubes
  - Map-Reduce Parallelism Framework
  - Big Data
  - Other trends (NoSQL)

Relational Query Languages

- 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:

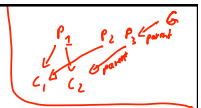
Attributes (SQL) ≠ Variables (Datalog)

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

boss(E,M) ← employee(E, S, D), dept(D1, M), D = D1 ⋈

Datalog

EDB (given) parent: p1 c1, p1 c2, p2 c1




```
grandparent(X, Y) :-
    parent(X, Z), parent(Z, Y).

father(X, Y) :-
    parent(X, Y), male(X).
mother(X, Y) :-
    parent(X, Y), female(X).

brother(X, Y) :-
    parent(P, X), parent(P, Y), male(X), X != Y.

sister(X, Y) :-
    parent(P, X), parent(P, Y), female(X), X != Y.

ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).
```

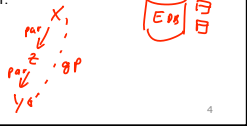
IDB (derived) views: gp, father, mother, …, ancestor

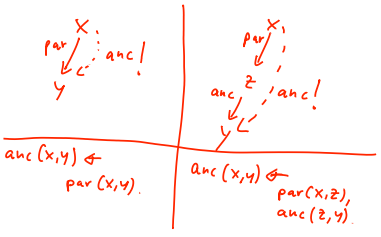
pair: child

Relation R occurs in head (lhs) ⇒ R is IDB

otherwise ⇒ R is EDB

X <> Y, X ≠ Y





Datalog

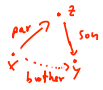
- Logical query language
  - **Data**bases + **Log**ic
  - aka "deductive databases"
- Inspired by logic programming / Prolog
  - Algorithm = Logic + Control
- "Declarative" programming
  - focus on "what" instead of "how"
  - (compare with SQL, Relational Algebra)
- From academia to industry
  - In the 1990's mostly a research vehicle
  - Some ideas made it into products (SQL recursion)
  - Renewed interest, commercial applications
    - LogicBlox (Atlanta), Lixto (Vienna, Austria), DLV (UCAL, Italy), …

SQL Problem: Recursive Queries

- Certain queries cannot be expressed in SQL (or relational algebra):
  - Bill-of-materials (BOM)
    - Set of parts, made up from subparts
  - Ancestor relations, transitive closure , …
  - Graph queries: social network analysis, biological networks, …
- Need recursion and/or iteration!
  - SQL + WITH-RECURSIVE (… next up…)
  - Datalog!

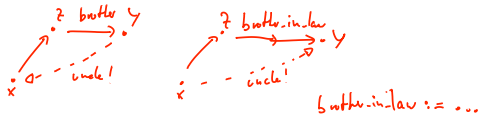
Living in the family

```
parent(X,Y) <- father(X,Y).
parent(X,Y) <- mother(X,Y).

son(X,Y)      <- parent(Y,X), male(Y).
daughter(X,Y) <- parent(Y,X), female(Y).

brother(X,Y) <- parent(X,Z), son(Z,Y), X \= Y.
sister(X,Y)  <- parent(X,Z), daughter(Z,Y), X \= Y.
```



Living in the family

```
brother_in_law(X,Y) <- sister(X,Z), spouse(Z,Y).
brother_in_law(X,Y) <- spouse(X,Z), brother(Z,Y).
sister_in_law(X,Y)  <- brother(X,Z), spouse(Z,Y).
sister_in_law(X,Y)  <- spouse(X,Z), sister(Z,Y).

uncle(X,Y) <- parent(X,Z), brother(Z,Y).
uncle(X,Y) <- parent(X,Z), brother_in_law(Z,Y).
aunt(X,Y)  <- parent(X,Z), sister(Z,Y).
aunt(X,Y)  <- parent(X,Z), sister_in_law(Z,Y).
```



uncle := parent.brother
| parent.brother_in_law

brother_in_law := ...

Datalog Syntax

- 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: Some Details

- Rule: head vs body
- Literals: positive vs negative atoms
- Relational atom vs built-in atom (arithmetic)
- Safety: every variable occurs positive in the body! (why?)
- EDB vs IDB relations (aka "predicates")

Datalog vs Relational Algebra

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):

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