Towards Bridging the Expressiveness Gap Between Relational and Deductive Databases

Fernando Sáenz Pérez
Grupo de Programación Declarativa (GPD)
Universidad Complutense de Madrid

Outline

1. Introduction
2. DES
3. Enabling Interoperability
4. Extending DBMS Expressiveness
5. Performance
6. Conclusions

1. Introduction – Expressiveness Gap

- Expressiveness:
  - Expressive Power (Theoretical Expressiveness)
    - What things can I say?
  - Conciseness and Readiness (Practical Expressiveness)
    - How much will it cost to me?

1. Introduction – Languages

- Relational languages:
  - Relational Algebra
  - (Tuple, Domain) Relational Calculus
  - SQL
- Deductive Language:
  - Datalog

1. Introduction – Languages

- Can current SQL express all Datalog can do?
  - No:
    - Non-linear recursive queries
    - Mutual recursive views
    - Recursion limitations
- Can usual Datalog express all SQL can do?
  - No:
    - Duplicates
    - Nulls

1. Introduction

- Overcoming Datalog Theoretical Expressiveness Limitations:
  - Well, simply let’s add those absent features
  - LDL++ includes a limited form of duplicates
  - DES includes both unrestricted duplicates and nulls

1. Introduction

- That’s Ok, but I’m an SQL programmer, don’t bother me with such a logic language
    - But, What if you can use SQL with the power of Datalog?
    - DES does allow it!
    - Even with already available relational databases

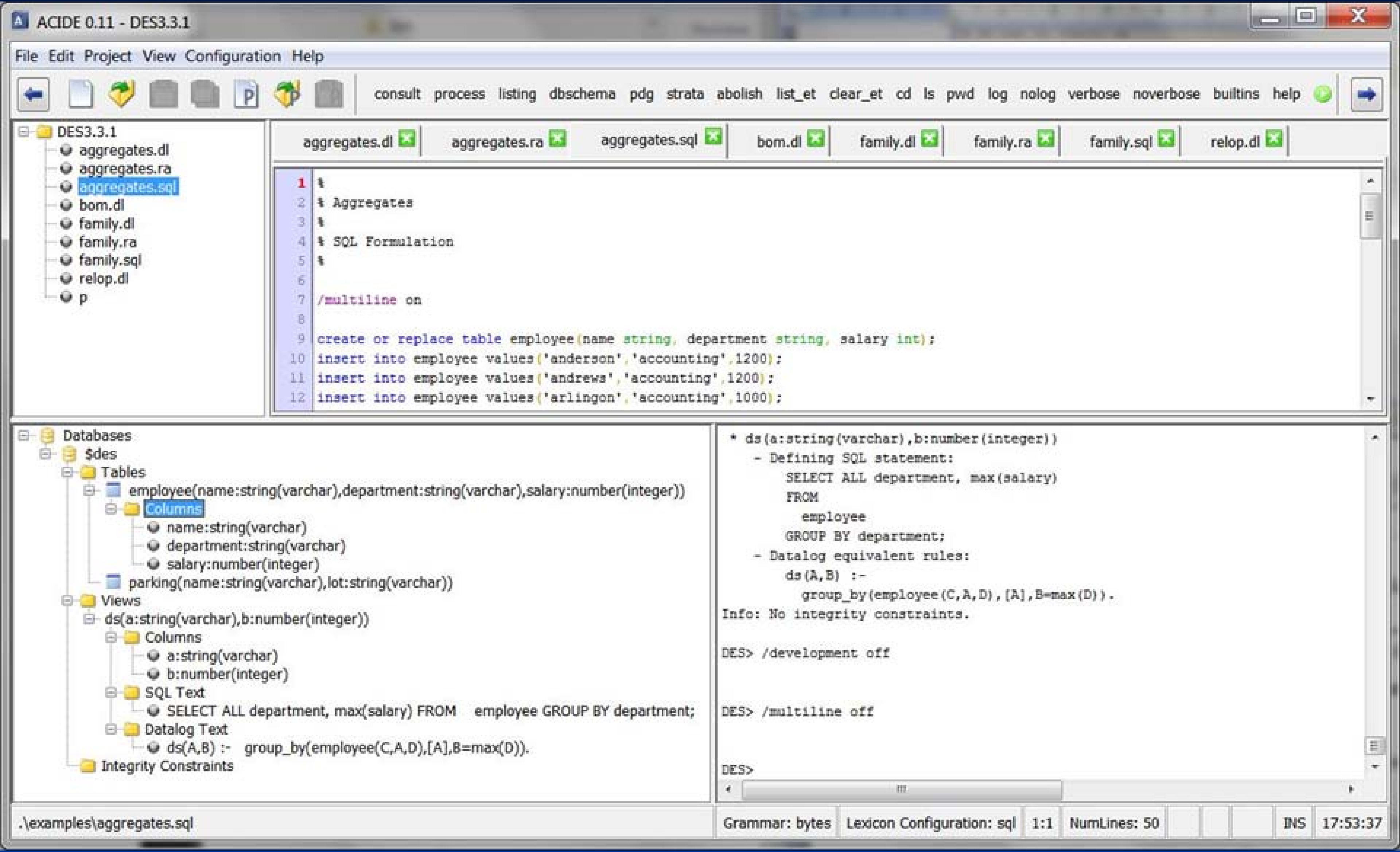
1. Introduction

- Could a major SQL vendor do the same?
- Well, this is the rationale of the proposal, just take it and improve current relational systems!
- But, in the meantime, what could I do?
- Just use database interoperability in DES

2. DES – Datalog Educational System

- Interactive system targeted at teaching databases
- User-friendly (Installation, Usability, >47K downloads)
- Free, Open-source, Multiplatform, Portable
- Query languages sharing EDB/IDB:
  - Datalog following ISO Prolog standard
  - (Recursive) SQL following ANSI/ISO standard
  - (Extended) Relational Algebra
- Null value support *à la* SQL
- Outer joins for RA, SQL and Datalog
- Duplicates
- Aggregates
- Stratified negation ... and many more
- **`des.sourceforge.net`**

DES running under ACIDE



- DES3.3.1
  - aggregates.dl
  - aggregates.ra
  - aggregates.sql
  - bom.dl
  - family.dl
  - family.ra
  - family.sql
  - relop.dl
  - p

```
%
% Aggregates
%
% SQL Formulation
%

/multiline on

create or replace table employee(name string, department string, salary int);
insert into employee values('anderson','accounting',1200);
insert into employee values('andrews','accounting',1200);
insert into employee values('arlingon','accounting',1000);
```

- Databases
  - $des
    - Tables
      - employee(name:string(varchar),department:string(varchar),salary:number(integer))
        - Columns
          - name:string(varchar)
          - department:string(varchar)
          - salary:number(integer)
      - parking(name:string(varchar),lot:string(varchar))
    - Views
      - ds(a:string(varchar),b:number(integer))
        - Columns
          - a:string(varchar)
          - b:number(integer)
        - SQL Text
          - SELECT ALL department, max(salary) FROM employee GROUP BY department;
        - Datalog Text
          - ds(A,B) :- group_by(employee(C,A,D),[A],B=max(D)).
    - Integrity Constraints

```
* ds(a:string(varchar),b:number(integer))
   - Defining SQL statement:
       SELECT ALL department, max(salary)
       FROM
         employee
       GROUP BY department;
   - Datalog equivalent rules:
       ds(A,B) :-
         group_by(employee(C,A,D),[A],B=max(D)).
Info: No integrity constraints.

DES> /development off


DES> /multiline off


DES>
```

.\examples\aggregates.sql

Grammar: bytes

Lexicon Configuration: sql

1:1

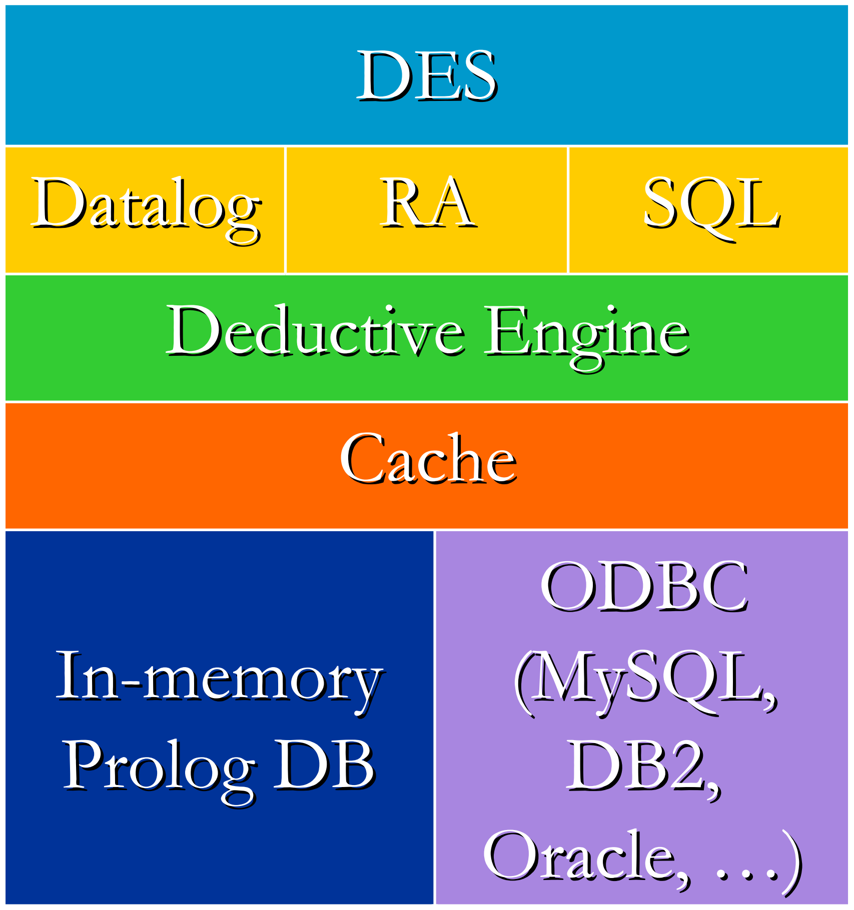
NumLines: 50

INS

17:53:37

3. Enabling Interoperability

- Architecture
- Interoperability
  - In-memory Deductive DB
  - In-memory SQL DB
  - External DB's via ODBC
  - Persistent predicates



DES

Datalog RA SQL

Deductive Engine

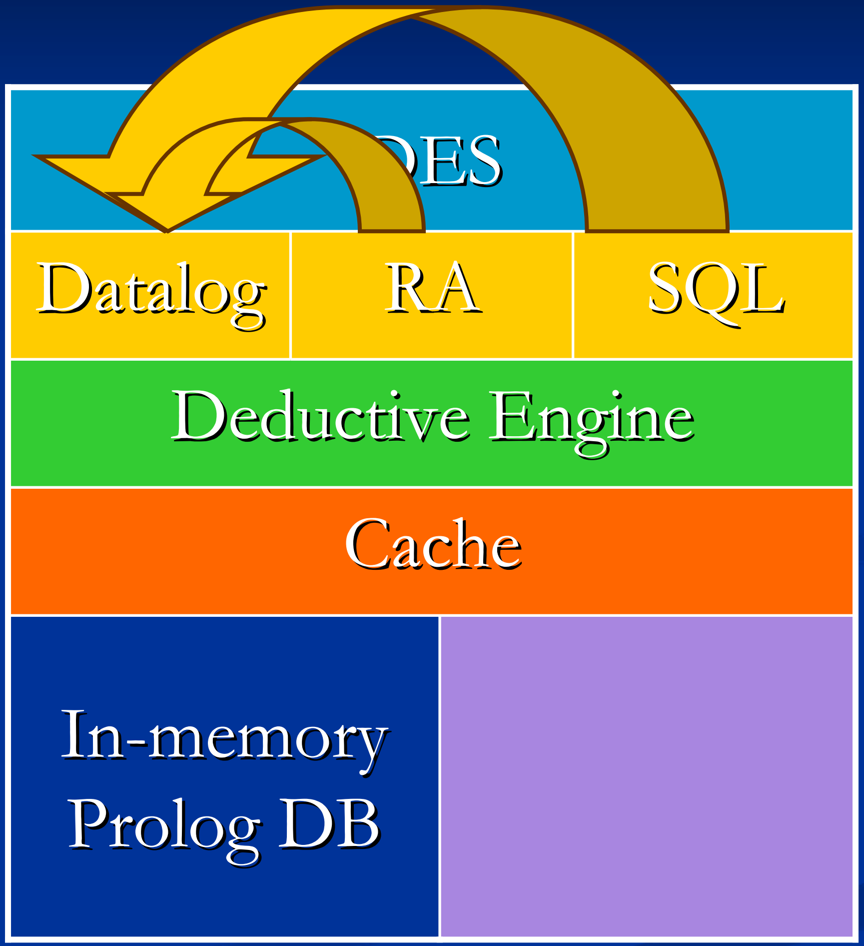
Cache

In-memory Prolog DB

ODBC (MySQL, DB2, Oracle, …)

3.1. In-Memory Database

- Datalog:
  - Relations (Data)
  - Schema (Type assertions)
- SQL:
  - Tables, Views
  - Schema (DDL statements)
- RA:
  - Views
- Deductive database sharing
  - Datalog queries ↔ SQL / RA queries
  - Datalog typed relations ↔ SQL tables and views



DES

Datalog

RA

SQL

Deductive Engine

Cache

In-memory Prolog DB

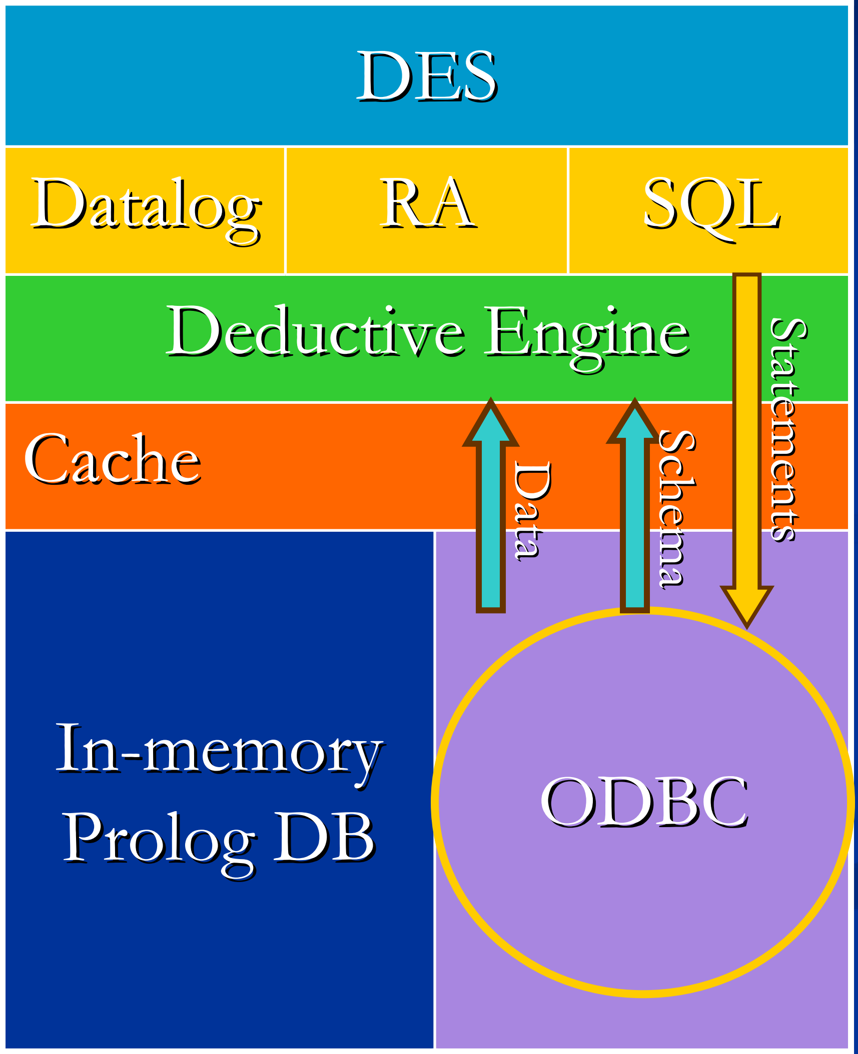
3.1. In-Memory Database

Datalog	SQL
`DES> :-type(employee(name: varchar, dept:varchar, salary:integer)).` `DES> :-pk(employee,[name]).`	`DES> create table employee(name varchar primary key, dept varchar, salary integer);`
`DES> /assert employee(’Smith’,’Sales’,15000).`	`DES> insert into employee values(’Smith’,’Sales’,15000);`

`DES> employee(N,D,S), S>10000.`
`{`
`employee(’Smith’,’Sales’,15000)`
`}`
	`DES> select * from employee where salary > 10000;`
	`answer(employee.name:string(varchar),employee.dept:string(varchar),employee.salary:number(integer)) ->`
	`{`
	`answer(’Smith’,’Sales’,15000)`
	`}`

3.2. Connecting to External DBs

- SQL Statements are issued to the open ODBC connection
- Both data and schema can be retrieved from the external DB
- Existing tables and views in the external DB are visible to the deductive engine



3.2. Connecting to External DBs

```
DES> /open_db mysql
```

External DB

```
DES> create table manager(mgr varchar(10), emp
varchar(10));

DES> insert into manager values (’E1’, ’E2’),
(’E2’, ’E3’), (’E2’, ’E4’);
```

Deductive Engine

+

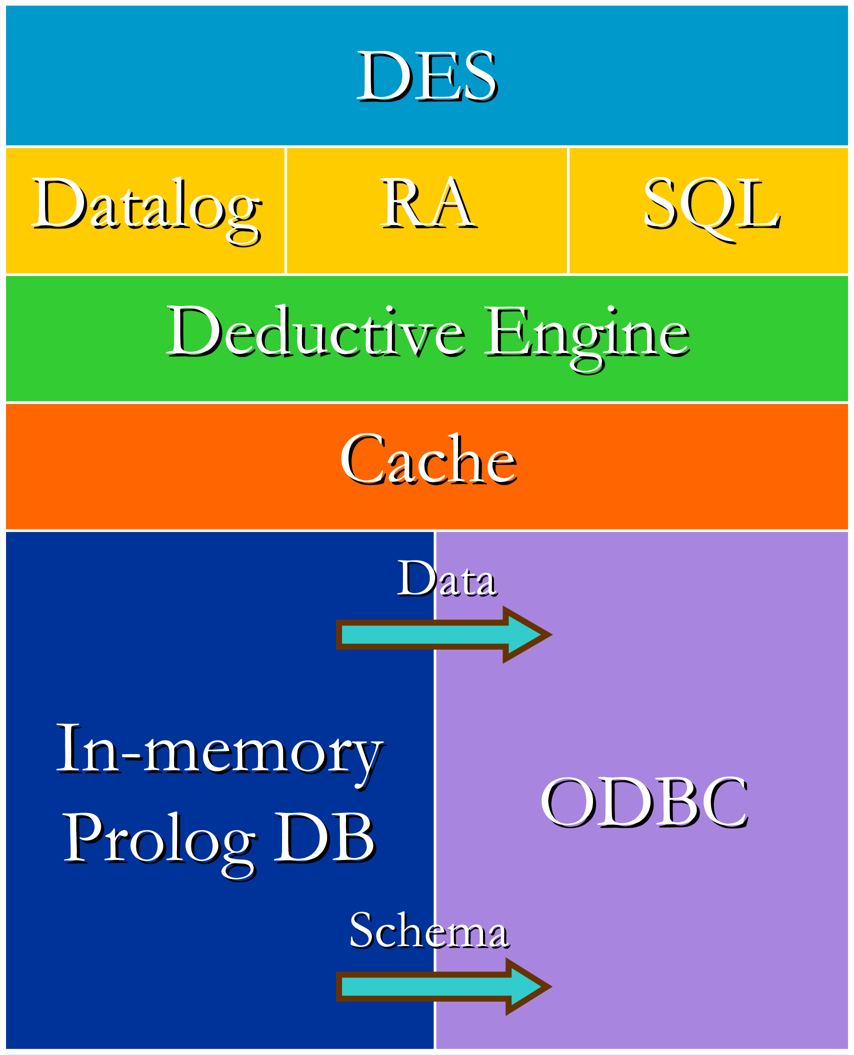
External DB

```
DES> managers(M,E) :- manager(M,E) ;
                      manager(M,M2),managers(M2,E).

{
  managers(’E1’,’E2’), managers(’E1’,’E3’),
  managers(’E1’,’E4’), managers(’E2’,’E3’),
  managers(’E2’,’E4’)
}
```

3.3. Persisting Datalog Predicates

- Durability (ACID Properties) for Deductive Databases
- Given a predicate, it can be made persistent
  - EDB (Facts)
  - IDB (Rules)
  - Schema (Type information)



3.3. Persisting Datalog Predicates

Schema

```
DES> :-type(manager(mgr:string,
                emp:string))
```

EDB
+
IDB

```
DES> /assert manager('E1', 'E2')
DES> /assert manager('E2', 'E3')
DES> /assert manager('E2', 'E4')
DES> /assert managers(M,E) :-
                  manager(M,E) ;
                  manager(M,M2),managers(M2,E).
```

- In-Memory DB

Persist Predicate

```
DES> :-persistent(manager/2,mysql)
```

- External DB

Deductive Engine
+
External DB

```
DES> managers(M,E)
{
  managers('E1','E2'), managers('E1','E3'),
  managers('E1','E4'), managers('E2','E3'),
  managers('E2','E4')
}
```

Outline

1. Introduction
2. DES
3. Enabling Interoperability
4. Extending DBMS Expressiveness
5. Performance
6. Conclusions

4.1. Contrived Formulations

- *Practical Expressiveness*
- Aliases (or, why do we need them?)

```
select * from t, t;  -- Cartesian Product
```

```
select * from t as t1, t as t2;
```

4.1. Contrived Formulations

- Lack of operators

```
select * from s full join q on s.sno=q.sno;
```

```
select * from s left join q on s.sno=q.sno
union all
select * from s right join q on s.sno=q.sno;
```

4.1. Contrived Formulations

- Syntax Restrictions (e.g., nesting)

```
select * from s left join
(select * from q right join sp on
  q.sno=sp.sno where q.qname<sp.pname)
on s.sno=q.sno where s.name=q.qname;
```

4.2. Recursive SQL

Practical and Theoretical Expressiveness

- Linear recursion. Precludes:
  - Fibonacci and the like
  - Graph algorithms
- Acyclic graphs
- No EXCEPT
- No DISTINCT
- UNION ALL
- No aggregates
  - Minimal paths
  - Bound limits

4.2. Recursive SQL

```
with recursive path(ori,des) as
(select edge.*,1 from edge
union all
select path.ori,edge.des from path,edge
  where path.des=edge.ori)
select * from path;
```

```
path(Ori,Des) :-
    edge(Ori,Des)
    ;
    edge(Ori,Int), path(Int,Des).
```

4.2. Recursive SQL

```
create view path(ori,des) as
select edge.* from edge
union
select path.ori,edge.des from path,edge
  where path.des=edge.ori;
```

```
create view path(ori,des) as
with recursive rec_path as
(select edge.* from edge
union all
select rec_path.ori,edge.des from
  rec_path,edge where rec_path.des=edge.ori)
select * from rec_path;
```

4.2. Recursive SQL

- Unrestricted SQL in External DB:

```
:-persistent(edge(ori:int,des:int),mysql).
:-persistent(path(ori:int,des:int),mysql).

with recursive path(ori, des) as
select * from edge
union
select p1.ori,p2.des from path p1, path p2
  where p1.des=p2.ori
select * from path;
```

4.3. The Division RA Operator

```
select sno from
   select sno,pno from spj
   division
   select pno from p where weight=17;
```

```
select distinct sno from spj
except
   select sno from
      (select sno, pno from
         (select sno from spj) as t1,
      (select pno from p where weight=17) as t2
  except
     select sno, pno from spj ) as t3;
```

4.3. The Division RA Operator

- Or simply:

```
v(SNO) :- spj(SNO,PNO,_,_,_) division p(PNO,_,_,17,_)
```

4.4. Functional Dependencies

- $\alpha \longrightarrow \beta$

```
CREATE TABLE emp(name string,
                 zip string,
                 city string determined by zip);
```

- $\{\texttt{zip}\} \longrightarrow \{\texttt{city}\}$

4.5. Hypothetical Queries

- "What-if" queries for decision support systems
- Assuming tuples in EDB
  - SQL

```
assume select 3,1 in edge select * from path;
```

  - Datalog $(\forall x,y(\neg(path(x,y)\leftarrow edge(3,1)))$

```
edge(3,1) => path(X,Y).
```

4.6. Hypothetical Queries

- Assuming tuples in EDB (TC of **edge**):
  - SQL

```
assume
  select e1.ori, e2.des
  from edge e1, edge e2 where e1.des=e2.ori
in edge
  select * from edge;
```

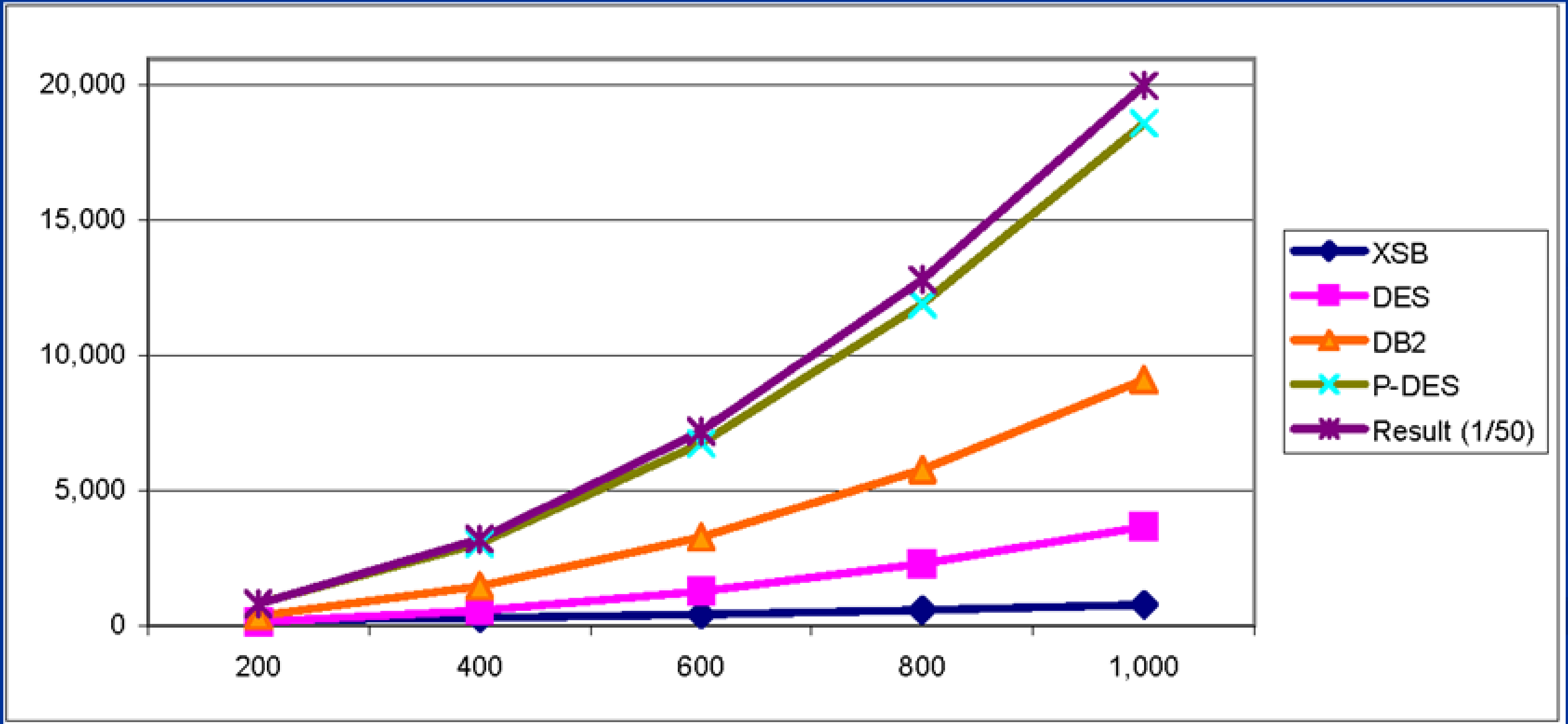
  - Datalog

∀x,y,o,d(¬(edge(x,y)←(edge(o,d)ˇ¬edge(o,i)ˇ¬edge(i,d)))

```
(edge(Ori,Des):-edge(Ori,I),edge(I,Des)) =>
  edge(X,Y)
```

5. Performance

- Cartesian Product (up to 1,000,000 tuples in the result set)



Conclusions

- Datalog vs. SQL. Which is better?
  - Who minds, they both express FOPL (without functions)
  - Give me both, let the user decide
- Is this all new?
  - LDL++ includes duplicates (but not for recursive rules)
  - Some DBMS's include a restricted form of recursion
  - Novel hypothetical queries
- Applications
  - I'd really liked all those nice features when developing for Repsol-YPF, Enagás, ...
- Current DBMS vendors: Please relax SQL limitations!