# DES: A Deductive Database System

des.sourceforge.net

Fernando Sáenz Pérez
Grupo de Programación Declarativa (GPD)
Dept. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid
Facultad de Informática

# Contents

1. Introduction
2. Features
3. Query Languages
4. Outer Joins
5. Aggregates
6. DES as a Test-Bed for Research
7. Impact Factor
8. Conclusions

# 1. Introduction

- Databases:
  From relational to deductive
- (Declarative) Query Languages:
  From SQL to Datalog

# 1. Introduction: Datalog

- A database query language stemming from Prolog

| Prolog | Datalog |
|---|---|
| Predicate | Relation |
| Goal | Query |

- Meaning of a predicate
  (Multi)set of derivable facts
  - Intensionally (Rules or Clauses)
  - Extensionally (Facts)

# 1. Introduction: Datalog

- What a typical database user would expect from a query language?
  - Finite data, finite computations (terminating queries)
    - No terms or bound depth
    - Be aware of built-in infinite relations!
  - All answer tuples at once
    - Prolog returns serveral answers upon backtracking

# 1. Introduction: Systems

- Deductive database systems:
  LDL++, DLV, Coral, XSB, SDS, Declare, ConceptBase, ...
- Yet another system, Why?
- We needed an interactive system targeted at teaching Datalog in classrooms
- So, what a whole set of features we would ask for such a system?

# 2. Features

## 2.1. Required Features

- A system oriented at teaching
- User-friendly:
  - Installation
  - Usability
- Multiplatform (Windows, Linux, Mac, ...)
- Interactive
- Database updates

# 2.2. DES Concrete Features (1/2)

- Free, Open-source, Multiplatform, Portable
- Query languages sharing EDB/IDB:
  - Datalog
  - (Recursive) SQL
- Database updates:
  - SQL DML
  - Commands
- Temporary Datalog views
- Duplicates (v2.0)
- Declarative debugging of Datalog programs
- Test case generation for SQL views
- Datalog and SQL tracers (v2.0)

# 2.2. DES Concrete Features (2/2)

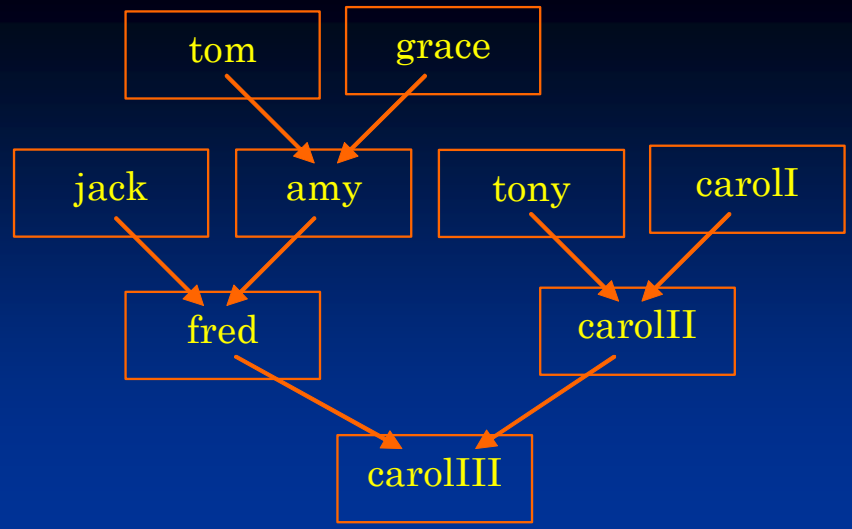
- Null value support *à la* SQL
- Outer joins for both SQL and Datalog
- Aggregates
- Negation
- Integrity constraints:
  - Domain
  - Referential integrity
- Full-fledged arithmetics
- Type system for SQL tables and views
- Souce-to-source program transformations:
  - Safety
  - Performance (simplifications)
- Tabling-based implementation

# 3. Query Languages

## 3.1. Datalog (1/2)

- Program: Set of rules.
- Rule:
  - `head :- body.`
  - `head.`
- Head: Positive atom.
- Body: Conjunctions (,) and disjunctions (;) of literals
- Literal: Atom, negated atom or a built-in.
- Query:
  - Literal with variables or constants in arguments
  - Body (Conjunctive queries, ...)
  - Temporary views

# 3.1. Datalog (2/2) Example




```
father(tom,amy).
father(jack,fred).
father(tony,carolII).
father(fred,carolIII).

mother(graceI,amy).
mother(amy,fred).
mother(carolI,carolII).
mother(carolII,carolIII).

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

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

```
DES-Datalog> ancestor(tom,X)
{
  ancestor(tom,amy),
  ancestor(tom,carolIII),
  ancestor(tom,fred)
}

DES-Datalog> father(X,Y),mother(Y,Z)
answer(X,Y,Z) :-
    father(X,Y),
    mother(Y,Z).
{
  answer(tom,amy,fred),
  answer(tony,carolII,carolIII)
}
```

# 3.2. SQL (1/3)

- DQL:
  - `SELECT … FROM … WHERE`
  - `WITH RECURSIVE …`
- DML:
  - `INSERT …`
  - `UPDATE …`
  - `DELETE …`
- DDL:
  - `CREATE [OR REPLACE] TABLE …`
  - `CREATE [OR REPLACE] VIEW …`
  - `DROP …`

# 3.2. SQL (2/3) Example

```
CREATE VIEW parent(parent,child) AS
 SELECT * FROM father
  UNION
 SELECT * FROM mother;

CREATE OR REPLACE VIEW ancestor(ancestor,descendant) AS
 WITH RECURSIVE rec_ancestor(ancestor,descendant) AS
  SELECT * FROM parent
   UNION
  SELECT parent,descendant
  FROM parent,rec_ancestor
  WHERE parent.child=rec_ancestor.ancestor
 SELECT * FROM rec_ancestor;

DES-SQL> SELECT * FROM ancestor WHERE ancestor='tom';
```

# 3.2. SQL (3/3)
# Simplified Syntax

```
CREATE OR REPLACE VIEW ancestor(ancestor, descendant) AS
 SELECT parent, child FROM parent
  UNION
 SELECT parent, descendant
 FROM parent, ancestor
 WHERE parent.child=ancestor.ancestor;
```

# 3.3. Datalog and SQL

- Deductive engine (DE):
  - Tabling implementation
- Datalog programs are solved by DE
- Compilation of SQL views and queries to Datalog programs
- SQL queries are also solved by DE
- Corollary: SQL and Datalog do share the deductive database!
- Datalog programs can refer to predicates defined as views in SQL

# 3.4. ODBC Connections

- New feature in version 2.0, released on August
- Access to Relational DBMS
  - MySQL
  - MS Access
  - Oracle
  - ...
- SQL statements injected to the DBMS engine
- Query results are cached by the Datalog engine
- So, interoperability is allowed!

# 4. Outer Joins (1/2)

- Null values:
  - Cte.: `null`
  - Functions: `is_null(Var)`
  - `is_not_null(Var)`
- Outer join built-ins:
  - Left: `lj(Left_Rel,Right_Rel,ON_Condition)`
  - Right: `rj(Left_Rel,Right_Rel,ON_Condition)`
  - Full: `fj(Left_Rel,Right_Rel,ON_Condition)`

# 4. Outer Joins (2/2)

```
lj(a(X), b(Y), X=Y)
SELECT * FROM a LEFT JOIN b ON x=y;

lj(a(X), b(X), true)
SELECT * FROM a LEFT JOIN b WHERE x=y;

lj(a(X), rj(b(Y), c(U,V), Y=U), X=Y)
SELECT * FROM a LEFT JOIN (b RIGHT JOIN c ON y=u) ON x=y;
```

# 5. Aggregates (1/5)

- Aggregate functions:
  - `count` – `COUNT(*)`
  - `count(Var)` – `COUNT(Column)`
  - `min(Var)`
  - `max(Var)`
  - `sum(Var)`
  - `avg(Var)`
  - `times(Var)`

# 5. Aggregates (2/5)

- Predicate `group_by/3`

```
group_by(
   Relation_A,          % FROM / WHERE
   [Var_1,…,Var_n],     % Grouping columns
   Relation_B)          % HAVING / Projection
```

# 5. Aggregates (3/5)
## Example

- Number of employees for each department:

```
DES-Datalog> group_by(employee(N,D,S),
                      [D],
                      R=count)

Info: Processing:
  answer(D,R) :-
    group_by(employee(N,D,S),[D],R = count).
{
  answer(accounting,3),
  answer(null,2),
  answer(resources,1),
  answer(sales,5)
}
Info: 4 tuples computed.
```

employee

| *Name* | *Department* | *Salary* |
|---|---|---|
| anderson | accounting | 1200 |
| andrews | accounting | 1200 |
| arlingon | accounting | 1000 |
| nolan | null | null |
| norton | null | null |
| randall | resources | 800 |
| sanders | sales | null |
| silver | sales | 1000 |
| smith | sales | 1000 |
| Steel | sales | 1020 |
| Sullivan | sales | null |

# 5. Aggregates (3/5)
## Example (contd.)

- Active employees (those with assigned salaries):

```
DES-Datalog> group_by(employee(N,D,S),
                      [D],
                      R=count(S))

Info: Processing:
  answer(D,R) :-
    group_by(employee(N,D,S),[D],R = count(S)).
{
  answer(accounting,3),
  answer(null,0),
  answer(resources,1),
  answer(sales,3)
}
Info: 4 tuples computed.
```

employee

| *Name* | *Department* | *Salary* |
|---|---|---|
| anderson | accounting | 1200 |
| andrews | accounting | 1200 |
| arlingon | accounting | 1000 |
| nolan | null | null |
| norton | null | null |
| randall | resources | 800 |
| sanders | sales | null |
| silver | sales | 1000 |
| smith | sales | 1000 |
| Steel | sales | 1020 |
| Sullivan | sales | null |

# 5. Aggregates (3/5)
## Example (contd.)

- Active employees of departments with more than one active employee:

```
DES-Datalog> group_by(employee(N,D,S),
                      [D],
                      count(S)>1)

Info: Processing:
answer(D) :-
 group_by(employee(N,D,S),
          [D],
          (A = count(S),A > 1)).
{
  answer(accounting),
  answer(sales)
}
Info: 2 tuples computed.
```

employee

| *Name* | *Department* | *Salary* |
|---|---|---|
| anderson | accounting | 1200 |
| andrews | accounting | 1200 |
| arlingon | accounting | 1000 |
| nolan | null | null |
| norton | null | null |
| randall | resources | 800 |
| sanders | sales | null |
| silver | sales | 1000 |
| smith | sales | 1000 |
| Steel | sales | 1020 |
| Sullivan | sales | null |

# 5. Aggregates (4/5)

- Aggregate Predicates:
  - `count(Rel)` – `COUNT(*)`
  - `count(Rel,Var)` – `COUNT(Column)`
  - `min(Rel,Var)`
  - `max(Rel,Var)`
  - `sum(Rel,Var)`
  - `avg(Rel,Var)`
  - `times(Rel,Var)`

# 5. Aggregates (5/5)

## Example

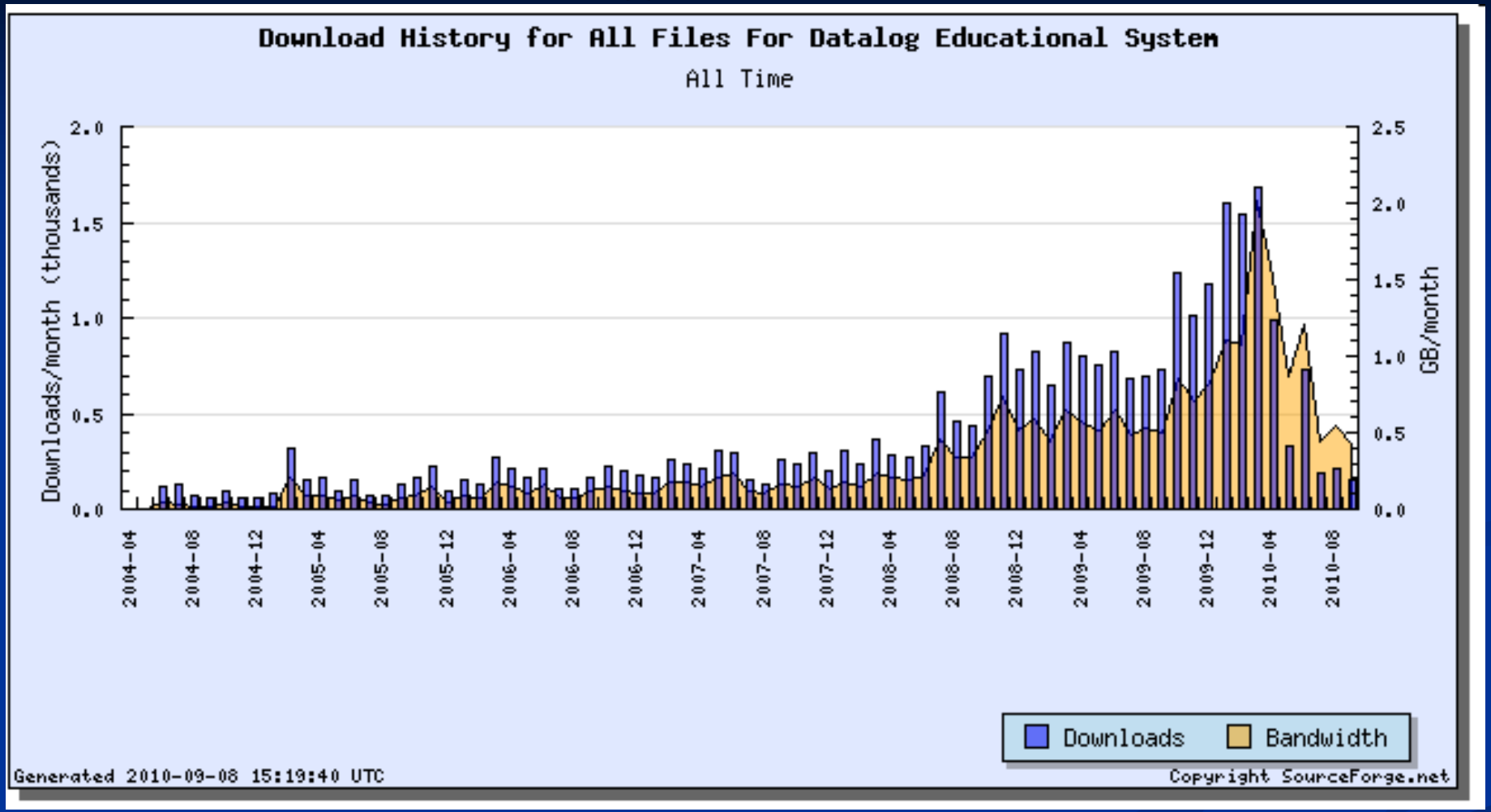
```
% SQL Program

CREATE OR REPLACE VIEW
shortest_paths(Origin,Destination,Length) AS
WITH RECURSIVE
 path(Origin,Destination,Length) AS
(SELECT edge.*,1 FROM edge)
UNION
(SELECT
  path.Origin,edge.Destination,path.Length+1
 FROM path,edge
 WHERE path.Destination=edge.Origin and
       path.Length <
      (SELECT COUNT(*) FROM Edge) )
SELECT Origin,Destination,MIN(Length)
FROM path
GROUP BY Origin,Destination;

% SQL Query
SELECT * FROM shortest_paths;
```

```
% Datalog Program

path(X,Y,1) :-
  edge(X,Y).
path(X,Y,L) :-
  path(X,Z,L0),
  edge(Z,Y),
  count(edge(A,B),Max),
  L0<Max,
  L is L0+1.

% Datalog Query:
shortest_paths(X,Y,L) :-
  min(path(X,Y,Z),Z,L).
```

# 6. DES as a Test-Bed for Research

- Test case generation for SQL views
- Datalog declarative (algorithmic) debugging
- Datalog and SQL tracers
- Novel proposal for outer joins in Datalog
- Theses, Papers, Academia... See DES Facts at its web page

# 7. “Impact Factor”




Up to more than 1,500 downloads a month
More than 30,000 downloads since 2004
More than 10,000 entries in Google

# 8. Conclusions

- Successful implementation guided by need
- Widely used, both for teaching and research
- Not really novel for each feature but as a whole
  - Datalog and SQL integration
  - Interactive, user-friendly, multiplatform system
  - Just download it and play!
  - Nevertheless the aforementioned novel features
  - Still, many things to do...

# Limitations (Future Work)

- Data are constants, no terms
- Datalog database updates
- SQL coverage still incomplete
- Precise syntax error reports
- Single-line inputs
- Constraints (*à la* CLP)
- Performance
- ... only to name a few!

# DES Facts

- Efficient Integrity Checking for Databases with Recursive Views
  Davide Martinenghi and Henning Christiansen
  In Advances in Databases and Information Systems: 9th East European Conference, ADBIS 2005, Tallinn, Estonia, September 12-15, 2005 : Proceedings
  Autor Johann Eder, Hele-Mai Haav, Ahto Kalja, Jaan Penjam
  ISBN 3540285857, 9783540285854
- PhD
  Computer Science and Engineering Department
  University of Nebraska - Lincoln, USA
- PhD
  University of Texas at San Antonio, USA

**Industry:**

- XLOG Technologies GmbH, Zürich
- CaseLab : Applied Operations Research
- Ideacube

**Links to DES:**

- ACM SIGMOD Online Publicly Available Database Software from Nonprofit Organizations
- The ALP Newsletter. vol. 21 n. 1
- Datalog Wikipedia German
- Datalog Wikipedia English
- Wapedia
- SWI-Prolog. Related Web Resources
- SICStus Prolog. Third Party Software. Other Research Systems
- SOFTPEDIA. Datalog Educational System 1.7.0
- Famouswhy
- DBpedia
- BDD-Based Deductive DataBase (bddbddb)
  Other implementations of Datalog/Prolog
- Reach Information
- Ask a Word
- Acronym finder
- Acronym Geek
- .

- CS240A: Databases and Knowledge Bases
- The University of Arizona
  CsC372
- The State University of New York
  University at Buffalo
  CSE 636: Data Integration
- The University of British Columbia
  CS304: Introduction to Relational Databases
  Datalog Tutorial
- Master's of Information Technology in Arkansas Tech University,
  Russellville
- The University of Texas at Austin
  CS2

*Australia:*

- INFO2820: Database Systems 1 (Advanced) (2010 - Semester 1)
  Engineering and Information Technologies
  The University of Sydney
  Tab "Resources"
- INFO2120/2820: Database Systems 1 (2009 - Semester 1)
  School of Information Technologies
  The University of Sydney
  Tutorial 3
- Allan Hancock College > > INFO > > 2120 Fall, 2009
  Description: School of Information Technologies
  INFO2120/2820: Database Systems I 1.Sem./2009
  Tutorial 3: SQL and Relational Algebra 23.03.2009

*Africa:*

- Faculty of Sciences and Technologies of Mohammedia (FSTM) - Morocco

# Temporary Views

relop.dl

```
1  % Relations
2  a(a1).
3  a(a2).
4  a(a3).
5
6  b(b1).
7  b(b2).
8  b(a1).
9
10 c(a1,b2).
11 c(a1,a1).
12 c(a2,b2).
13
14 % Relational Operations
15
16 % pi(X)(c(X,Y))
17 projection(X) :- c(X,Y).
18
19 %sigma(X=a2)(a)
20 selection(X) :- a(X), X=a2.
21
22 % a X b
23 cartesian(X,Y) :- a(X), b(Y).
24
25 % a |x| b
26 join(X) :- a(X), b(X).
27
28 % a U b
29 union(X) :- a(X).
30 union(X) :- b(X).
31
32 % a - b
33 difference(X) :- a(X), not(b(X)).
```

```
DES> d(X) :- a(X), not(b(X))

Info: Computing predicate dependency graph...
Info: Computing strata...
Info: Computing by stratum of [b(_62518)].
{
  d(a2),
  d(a3)
}

DES> a(X) :- b(X)

Info: Computing predicate dependency graph...
Info: Computing strata...
{
  a(a1),
  a(a2),
  a(a3),
  a(b1),
  a(b2)
}

DES>
```

C:\Archivos de program | Grammar: | Lexicon Conf | 6:7 | CA | NL | 5:38

# Datalog Declarative Debugging

- Motivation:
  - Abstract the solving-oriented debugging procedure
- Roots:
  - [Shaphiro83], Algorithmic Program Debugging
- Semantics-oriented

# Declarative Debugger

```
between(X,Z):- br(X),br(Y),br(Z),X<Y,Y<Z .
```

Pairs of non-consecutive elements in the sequence

```
next(X,Y) :-  br(X), br(Y), X<Y, not(between(X,Y)).
next(nil,X) :-  br(X), not(has_preceding(X)).
```

Consecutive elements in a sequence (starting at nil)

```
has_preceding(X) :-  br(X), br(Y), X > Y.
```

Elements having preceding values in the sequence

```
even(nil).
even(X) :- odd(Z), next(Z,X).
```

Elements in an even position+nil

```
odd(Y) :-  even(Z), next(Z,Y).
```

Elements in an odd position

```
br_is_even :- even(X), not(next(X,Y)).
```

Succeeds if the cardinality is even

```
br(a).
br(b).
```

Base relation (sequence of elements)

# Declarative Debugger

```
between(X,Z):- br(X),br(Y),br(Z),X<Y,Y<Z .
```

Pairs of non-consecutive elements in the sequence

```
next(X,Y) :-  br(X), br(Y), X<Y, not(between(X,Y)).
next(nil,X) :-  br(X), not(has_preceding(X)).
```

Consecutive elements in a sequence (starting at nil)

```
has_preceding(X) :-  br(X), br(Y), X < Y.
```

Elements having preceding values in the sequence

```
even(nil).
even(X) :- odd(Z), next(Z,X).
```

Elements in an even position+nil

```
odd(Y) :-  even(Z), next(Z,Y).
```

Elements in an odd position

```
br_is_even :- even(X), not(next(X,Y)).
```

Succeeds if the cardinality is even

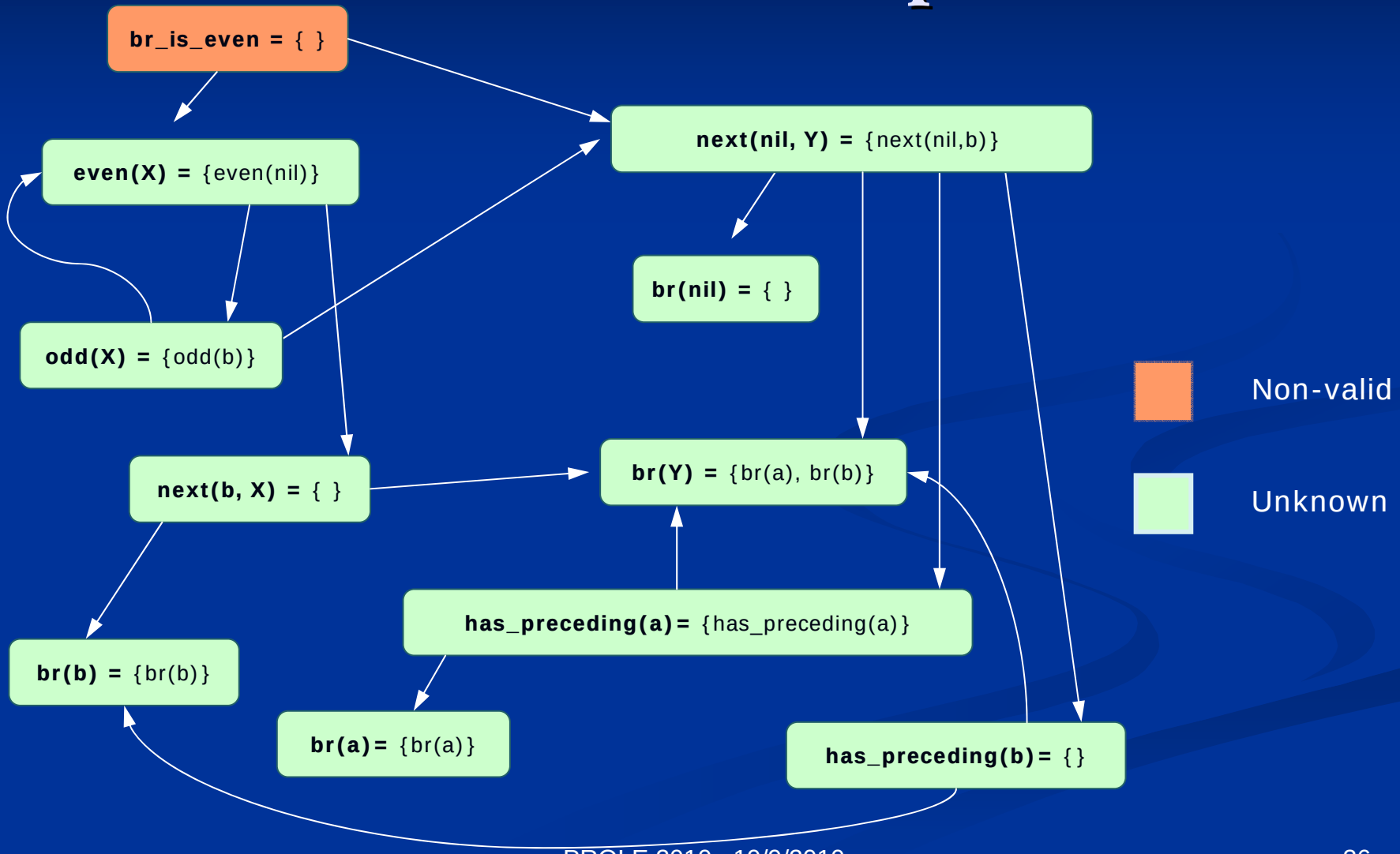
```
br(a).
br(b).
```

Base relation (sequence of elements)

# Declarative debugging: a practical session

```
DES> /debug br_is_even

Debugger started ...
Is br(b) = {br(b)} valid(v)/non-valid(n) [v]? v
Is has_preceding(b) = {} valid(v)/non-valid(n) [v]? n
Is br(X) = {br(b),br(a)} valid(v)/non-valid(n) [v]? v
! Error in relation: has_preceding/1
! Witness query: has_preceding(b) = { }
```

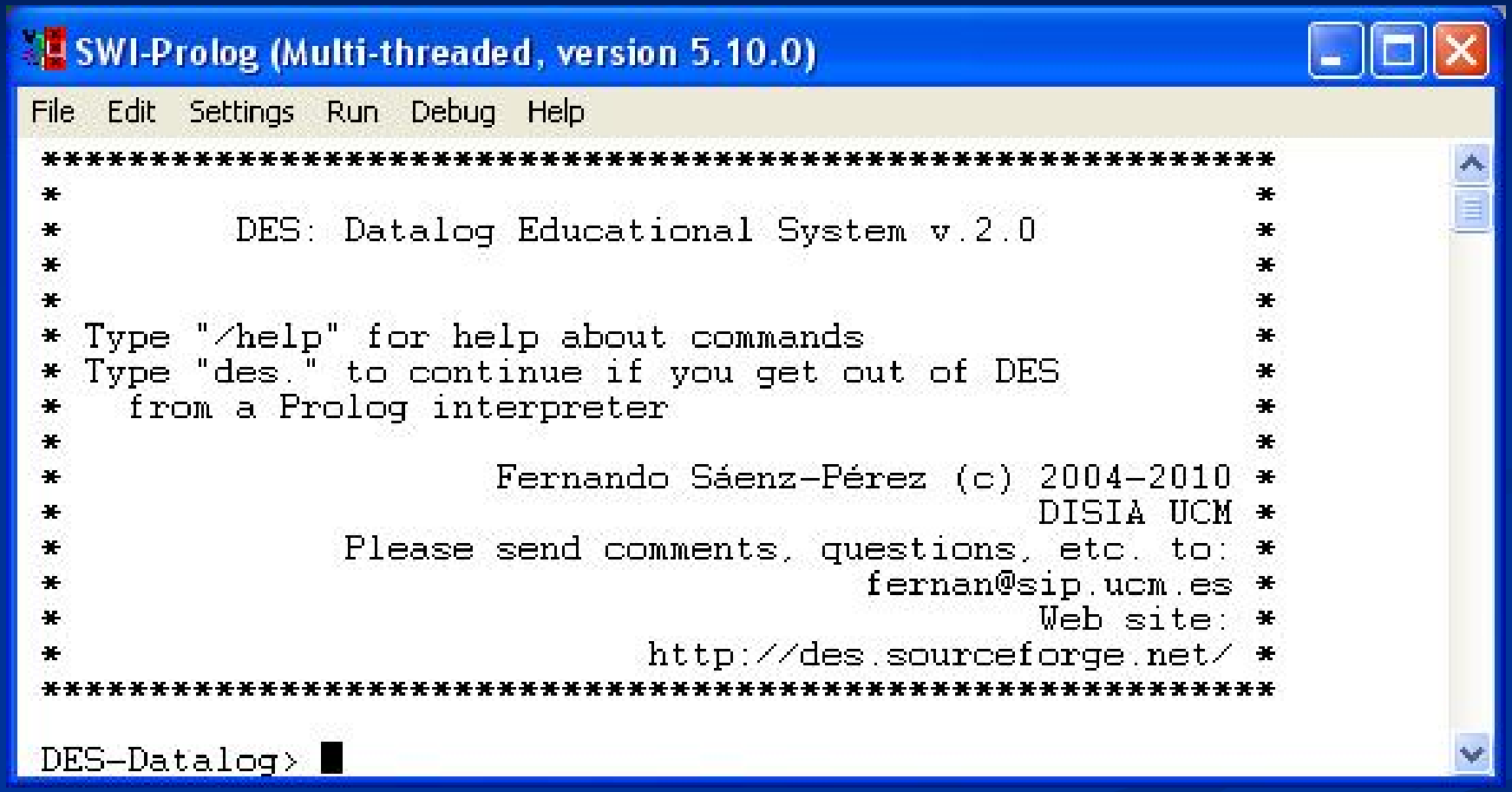
# Declarative Debugging: Semantic Graph



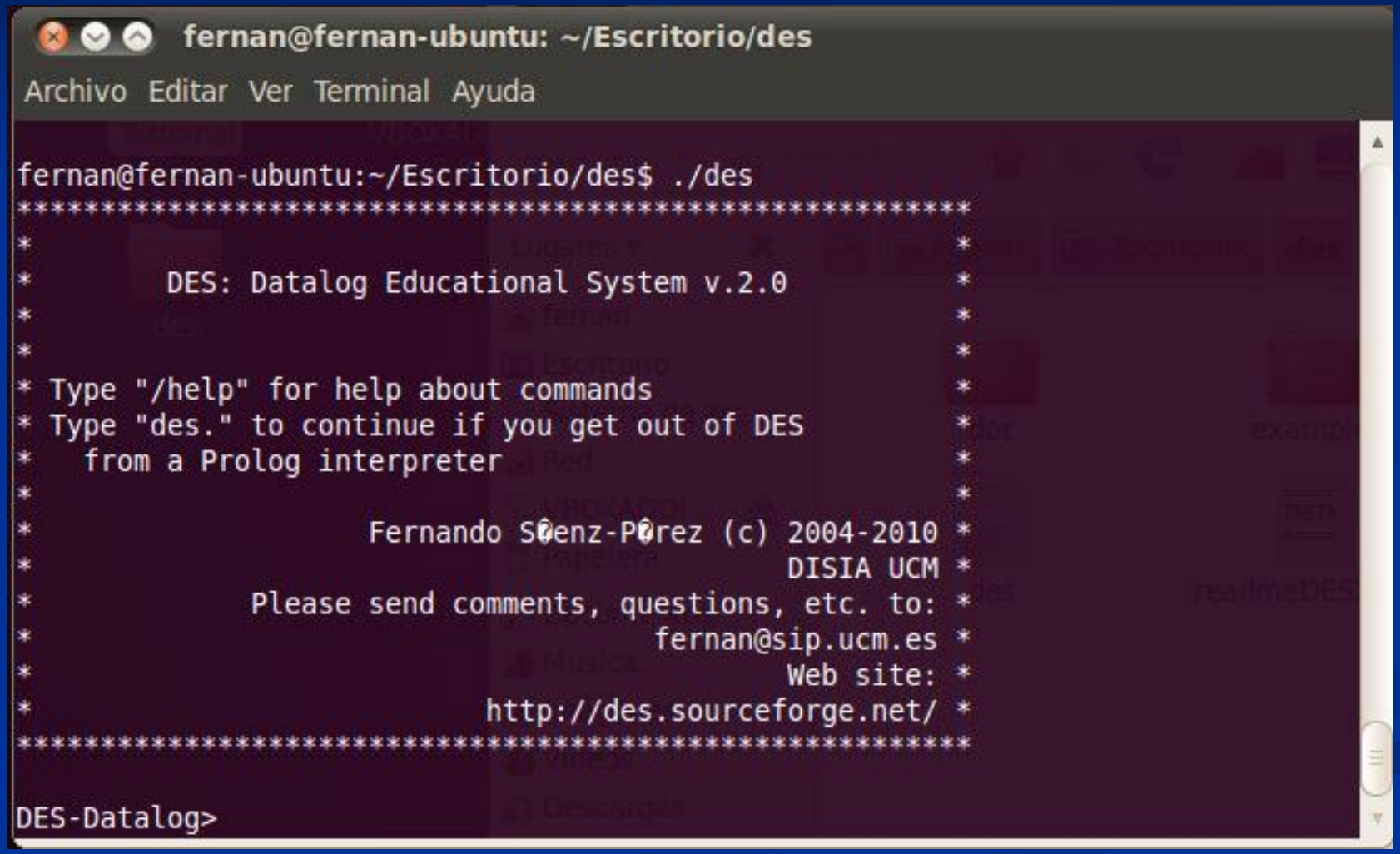
# Installing DES

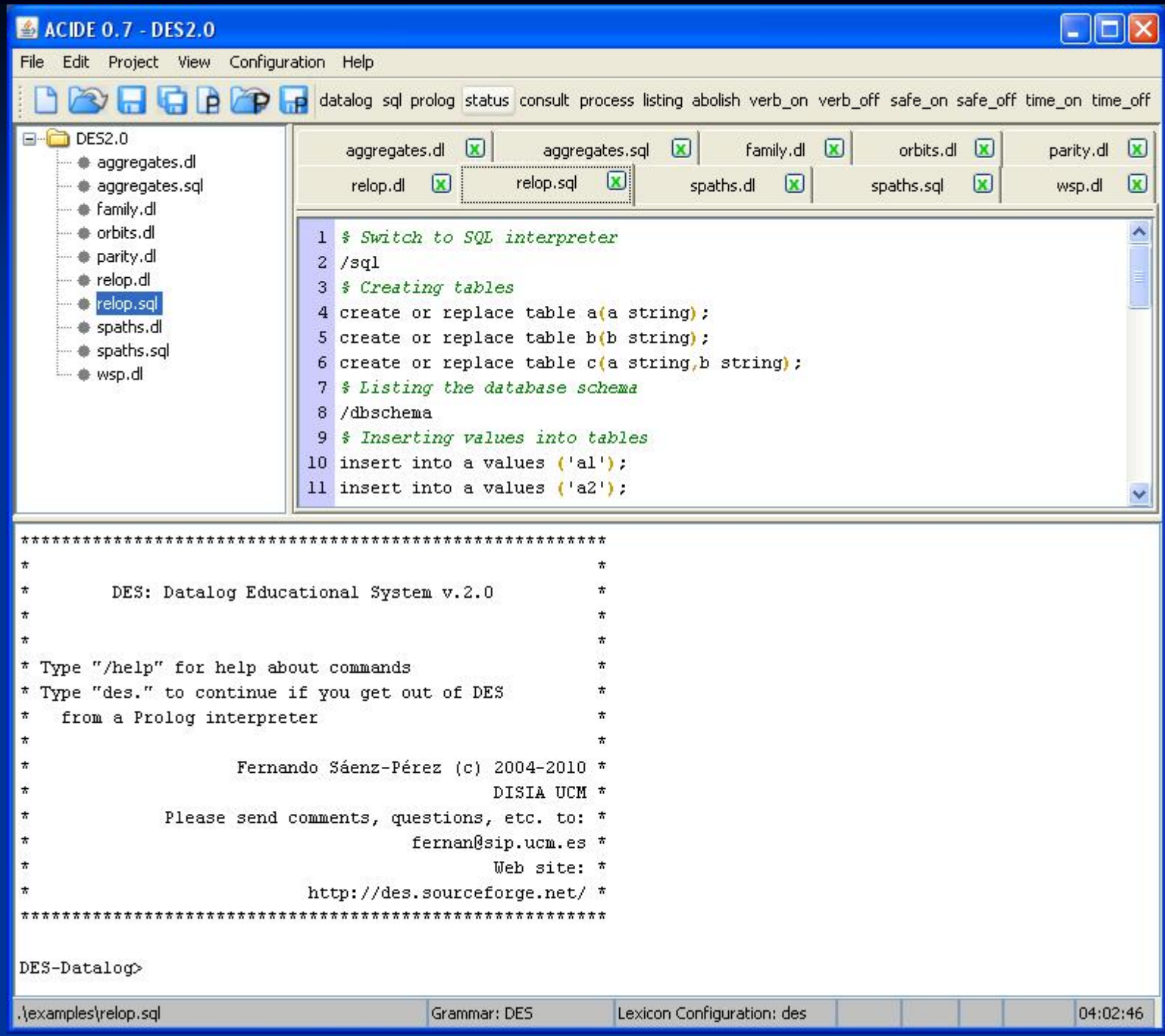
- Distro under GPL in Sourceforge:
  - Sources
  - Portable Executables (Windows, Linux)
  - Portable Bundle including Java IDE (Windows)
- Starting the system. Either:
  - From a Prolog interpreter (Ciao, GNU, Sicstus, SWI)
  - Simply execute the binary
  - Start the Java application

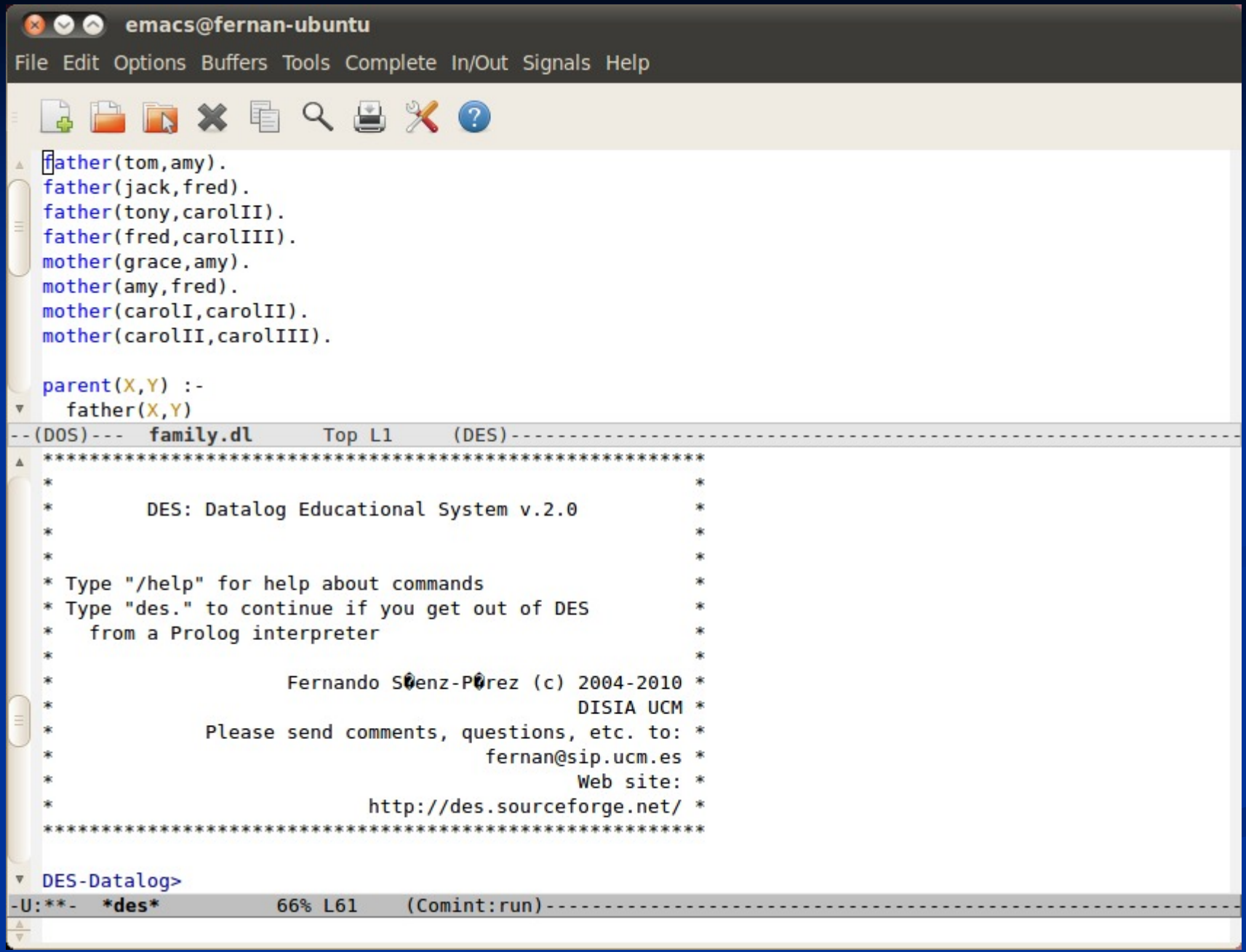
# DES running as a Windows application



# DES running in a Linux terminal




ACIDE 0.7 - DES2.0
File Edit Project View Configuration Help
datalog sql prolog status consult process listing abolish verb_on verb_off safe_on safe_off time_on time_off
DES2.0
aggregates.dl
aggregates.sql
family.dl
orbits.dl
parity.dl
relop.dl
relop.sql
spaths.dl
spaths.sql
wsp.dl
aggregates.dl aggregates.sql family.dl orbits.dl parity.dl
relop.dl relop.sql spaths.dl spaths.sql wsp.dl
1 % Switch to SQL interpreter
2 /sql
3 % Creating tables
4 create or replace table a(a string);
5 create or replace table b(b string);
6 create or replace table c(a string,b string);
7 % Listing the database schema
8 /dbschema
9 % Inserting values into tables
10 insert into a values ('a1');
11 insert into a values ('a2');
DES: Datalog Educational System v.2.0
Type "/help" for help about commands
Type "des." to continue if you get out of DES
from a Prolog interpreter
Fernando Sáenz-Pérez (c) 2004-2010
DISIA UCM
Please send comments, questions, etc. to:
fernan@sip.ucm.es
Web site:
http://des.sourceforge.net/
DES-Datalog>
.\examples\relop.sql
Grammar: DES
Lexicon Configuration: des
04:02:46


emacs@fernan-ubuntu
File Edit Options Buffers Tools Complete In/Out Signals Help
father(tom,amy).
father(jack,fred).
father(tony,carolII).
father(fred,carolIII).
mother(grace,amy).
mother(amy,fred).
mother(carolI,carolII).
mother(carolII,carolIII).
parent(X,Y) :-
father(X,Y)
--(DOS)--- family.dl Top L1 (DES)
DES: Datalog Educational System v.2.0
Type "/help" for help about commands
Type "des." to continue if you get out of DES
from a Prolog interpreter
Fernando S?enz-P?rez (c) 2004-2010
DISIA UCM
Please send comments, questions, etc. to:
fernan@sip.ucm.es
Web site:
http://des.sourceforge.net/
DES-Datalog>
-U:**- *des* 66% L61 (Comint:run)

# Implementation

- DES command-line interpreter: Prolog
  - Tabling (Bottom-up Top-down driven)
  - Computation by strata saturations (negation and aggregates)
- Datalog Debugger: Prolog + Java
  - [CGS07] R. Caballero, Y. García-Ruiz, and F. Sáenz-Pérez, A new proposal for debugging datalog programs. WFLP'07
- Test Case Generator: Prolog + FD constraints
  - [CGS10a] R. Caballero, Y. García-Ruiz, and F. Sáenz-Pérez, Applying Constraint Logic Programming to SQL Test Case Generation, FLOPS 2010
- ACIDE: Java
  - A Configurable IDE (LaTeX, SQL, Prolog, Datalog, ...)