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

<table>
<thead>
<tr>
<th>Prolog</th>
<th>Datalog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicate</td>
<td>Relation</td>
</tr>
<tr>
<td>Goal</td>
<td>Query</td>
</tr>
</tbody>
</table>

- 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 several 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 \textit{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
- Source-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 ...
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';
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)

\[ lj(a(X), b(Y), X=Y) \]
\[ \text{SELECT * FROM a LEFT JOIN b ON x=y;} \]

\[ lj(a(X), b(X), \text{true}) \]
\[ \text{SELECT * FROM a LEFT JOIN b WHERE x=y;} \]

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

```prolog
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.
```
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.
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Department</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>anderson</td>
<td>accounting</td>
<td>1200</td>
</tr>
<tr>
<td>andrews</td>
<td>accounting</td>
<td>1200</td>
</tr>
<tr>
<td>arlingon</td>
<td>accounting</td>
<td>1000</td>
</tr>
<tr>
<td>nolan</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>norton</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>randall</td>
<td>resources</td>
<td>800</td>
</tr>
<tr>
<td>sanders</td>
<td>sales</td>
<td>null</td>
</tr>
<tr>
<td>silver</td>
<td>sales</td>
<td>1000</td>
</tr>
<tr>
<td>smith</td>
<td>sales</td>
<td>1000</td>
</tr>
<tr>
<td>Steel</td>
<td>sales</td>
<td>1020</td>
</tr>
<tr>
<td>Sullivan</td>
<td>sales</td>
<td>null</td>
</tr>
</tbody>
</table>
5. Aggregates (4/5)

- **Aggregate Predicates:**
  - count(Rel) –– \texttt{COUNT}(*)
  - count(Rel,Var) –– \texttt{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!
Temporary Views

```prolog
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)
}
```

$Relations$
1. a(a1).
2. a(a2).
3. a(a3).
4. b(b1).
5. b(b2).
6. b(a1).
7. c(a1, b2).
8. c(a1, a1).
9. c(a2, b2).

$Relational\ Operations$
10. pi(X)(c(X,Y)).
11. projection(X) :- c(X,Y).
12. sigma(X=a2)(a).
13. selection(X) :- a(X), X=a2.
14. X b.
15. cartesian(X,Y) :- a(X), b(Y).
16. a |x| b.
17. join(X) :- a(X), b(X).
18. a U b.
19. union(X) :- a(X).
20. union(X) :- b(X).
21. a - b.
22. difference(X) :- a(X), not(b(X)).
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)).

Consecutive elements in a sequence (starting at nil)

next(nil,X) :- br(X), not(has_preceding(X)).

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).

Base relation (sequence of elements)

br(b).
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)).

Consecutive elements in a sequence (starting at nil)

next(nil,X) :- br(X), not(has_preceding(X)).

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).

Base relation (sequence of elements)

br(b).
Declarative debugging: a practical session

DES> /debug br_is_even

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

- `br_is_even = {}`
- `even(X) = {even(nil)}`
- `odd(X) = {odd(b)}`
- `next(b, X) = {}`
- `next(nil, Y) = {next(nil, b)}`
- `br(nil) = {}`
- `br(Y) = {br(a), br(b)}`
- `br(a) = {br(a)}`
- `br(b) = {br(b)}`
- `has_preceding(a) = {has_preceding(a)}`
- `has_preceding(b) = {}`
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

**SWI-Prolog (Multi-threaded, version 5.10.0)**

---

**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>**
DES running in a Linux terminal

```
fern@fern-ubuntu:~/Escritorio/des

fern@fern-ubuntu:~/Escritorio/des$ ./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 Sánchez-Pérez (c) 2004-2010
DISIA UCM
Please send comments, questions, etc. to:
fern@sip.ucm.es
Web site:
http://des.sourceforge.net/

DES-Datalog>
```
$ Switch to SQL interpreter
/sql
$ 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);
$ Listing the database schema
/dbschema
$ 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>
father(tom, amy).
father(jack, fred).
father(tony, carolII).
father(fred, carolIII).
mother(grace, amy).
mother(amy, fred).
mother(carolII, carolII).
mother(carolII, carolIII).

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

PROLE 2010   10/9/2010

42