A COMPUTER-BASED ENVIRONMENT FOR THE STUDY OF RELATIONAL QUERY LANGUAGES

Meurig Beynon, Abhir Bhalerao, Chris Roe, Ashley Ward

Department of Computer Science
University of Warwick
Gibbet Hill Rd, Coventry, CV4 7AL
{wmb,abhir,croe,ashley}@dcs.warwick.ac.uk

ABSTRACT

In this paper, we describe an environment developed to support a rich learning experience in which practice and theory in relational databases are better integrated, enabling students from various backgrounds to appreciate the significance of relational theory and the logical flaws in SQL. Our lightweight open source software includes the aspects of a commercial database system that are most relevant to teaching relational databases and can be run on several platforms.

Keywords

Relational Databases, SQL, Relational Algebra, Computer-Assisted Learning, Empirical Modelling.

1. INTRODUCTION

The core database module for second year students on computing-related courses at the University of Warwick combines a practical introduction to SQL with lectures on relational algebra. For many students - especially those on joint degrees involving business and engineering, who have less prior exposure to basic set-theory, algebra and logic - relating practical use of SQL to the underlying algebraic theory of relations is difficult. This problem is exacerbated by the fact that standard SQL [1] deviates from the ideal model of a relational query language as it was originally conceived by Codd [7]. In particular, SQL has what the relational database proponents Date and Darwen identify as "logical flaws" that subvert the natural mathematical semantics that relational theory provides [12]. For instance, standard SQL allows duplicate rows in a table, applies loose type checking in creating unions, and only supports an indirect and clumsy representation of natural join.

At Warwick, the educational problems identified above have been addressed by developing an environment in which students can explore the relationship between practical use of standard SQL and the underlying relational algebra. For this purpose, we have designed and implemented a pure relational algebra language ("EDDI") and developed translators both for a variant of SQL whose semantics is consistent with relational theory ("SQLZERO"), and for a subset of standard SQL. In the SQL-EDDI environment, students can study the evaluation of relational algebra expressions, and relate these to the translation and interpretation of SQL and SQLZERO queries.

The paper describes the SQL-EDDI environment and discusses its implications from an educational and a computer science perspective.

2. EDDI: RELATIONAL ALGEBRA

The EDDI interpreter[1] is a front end to the EDEN[2] interpreter, the principal tool that has been developed in connection with the Empirical Modelling project at the University of Warwick [2]. EDDI is one of several *definitive notations* in EDEN that can be used to formulate scripts that express spreadsheet-like dependencies between variables.

EDDI is loosely modelled on the Information Systems Base Language (ISBL) developed in 1976 [13]. The six basic operations of ISBL are union, difference, intersection, projection, selection and natural join. These are defined in EDDI (as in ISBL) as `R+S` (union), `R-S` (difference), `R.S` (intersection), `R%A,B,...,Z` (projection), `R:F` (selection) and `R*S` (natural join), where `R` and `S` are relations, `A`, `B` and `Z` are attributes and `F` is a Boolean condition. ISBL has two types of statement. If <exp> is a relational algebra expression then `LIST <exp>` prints the current value of `exp`, whilst `R = <exp>` assigns this value to the relational variable `R`.

[1] The Eden Definitive Database Interface

[2] The Evaluator of DEfinitive Notations

Teaching, Learning and Assessment in Databases, Coventry 2003

```
1. %eddi
2. allfruits (name CHAR, begin INT, end INT);
3. allfruits << ["granny",8,10],["lemon",5,12],["kiwi",5,6],["passion",5,7];
4. allfruits << ["orange",4,11],["grape",3,6],["lime",4,7],["pear",4,8];
5. allfruits << ["cox",1,12],["red",4,8];
6. apple (name CHAR, price REAL, qnt INT);
7. apple << ["cox",0.20,8],["red",0.35,4],["granny",0.25,10];
8. citrus (name CHAR, price REAL, qnt INT);
9. citrus <<["lime",0.30,3],["orange",0.55,8],["kiwi",0.75,5],["lemon",0.50,2];
10. soldfruit (name CHAR, unitsold INT);
11. soldfruit << ["cox",100],["granny",153],["red",70];
12. soldfruit << ["kiwi",23],["lime",15],["lemon",55],["orange",78];
13. fruits is allfruits % name;
14. popcitrus is (fruits.citrus % name).(soldfruit : unitsold > 50 % name);
15. nonapplesncox = allfruits-
      (allfruits*apple%name,begin,end)+allfruits:name==”cox”;
```

Figure 1: An EDDI extract illustrating the definition of the FRUITS database

ISBL also supports a form of view definition. This exploits a call-by-name mechanism invoked by preceding variable names in formulae by “`N!`”. The current values of such variables are then used on evaluation. ISBL allows attribute renaming so that `R%A,B→C` denotes the relation `R(A,B)` with the second attribute renamed as `C`. This feature allows the specification of a Cartesian product, ensuring that ISBL satisfies Codd’s criterion for completeness of a relational query language [7].

EDDI realises all the functions of ISBL but uses different syntactic conventions. Figure 1 shows the EDDI code to define a small example database. Lines 2-12 are the EDDI DDL to define the basic tables. Lines 13 and 14 define two views derived from these tables, denoted by the `is` syntax for a definition. Line 15 defines the explicit contents of a table, using the `=` syntax for assignment. A subsequent change to the contents of the `allfruits` table will affect the values of `fruits` and `popcitrus` but not that of `nonapplesncox`.

3. SQLZERO: A TRULY RELATIONAL VARIANT OF SQL

A direct translation of standard SQL into EDDI is problematic (indeed impossible!) because of the flaws in its design mentioned in §1, which led Darwen to call SQL the “askew wall” [10]. The flaws are illustrated by the queries on our example database shown in Figure 2.

Query 1a) returns duplicate rows for the `cox`, `red` and `granny` tuples. Queries 2a) and 2b) return tables with the same contents but different attribute names (respectively `unitsold` and `qnt`) in the second column. Query 3a) returns a table with six columns (`allfruits.name, begin, end, apple.name, price, qnt`), two of which are identical. To specify the natural join of `allfruits` and `apple` requires the more complex query 3b).

Duplicate rows:

```
1a) SQL:  (SELECT name FROM apple) UNION (SELECT name FROM allfruits)
1b) EDDI: ?apple % name + allfruits % name;
```

Loose type checking in creating unions:

```
2a) SQL:  (SELECT * FROM soldfruit) UNION (SELECT name, qnt FROM citrus)
2b) SQL:  (SELECT name, qnt FROM citrus) UNION (SELECT * FROM soldfruit)
2c) EDDI: ?soldfruit + citrus % name, qnt;
```

Indirect and clumsy representation of natural join:

```
3a) SQL:  SELECT * FROM allfruits, apple
3b) SQL:  SELECT allfruits.name, begin, end, price, qnt FROM allfruits, apple
            WHERE allfruits.name=apple.name
3c) EDDI: ?allfruits * apple;
```

Figure 2: Some example SQL queries and their EDDI equivalents

Each of the above three queries illustrates a different way in which standard SQL is problematic. In standard SQL, query 1a) does not return a legitimate mathematical object, query 2a) - although it is strictly meaningless - returns a result, and query 3a) arguably returns an inappropriate result.

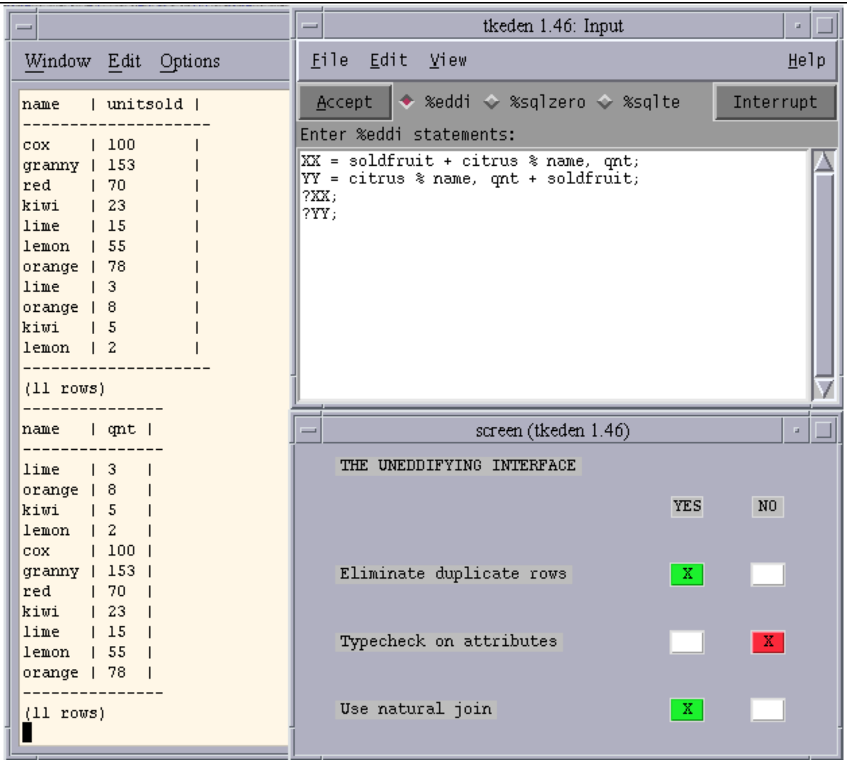
We have developed a front-end to EDDI that uses SQL-style syntax, named SQLZERO. SQLZERO is faithful to the relational theory developed by Codd [7] as the foundation for relational databases. It differs from standard SQL in that

- `SELECT` is treated as a synonym for `SELECT DISTINCT`,
- type checking on constructing union, intersection and difference of relations takes account of both domain types and attribute names,
- `SELECT * FROM X,Y` is interpreted as a natural join of relations.

These evaluation characteristics reflect the sound mathematical foundation on which the EDDI interpreter operates. They give SQLZERO characteristics as a query language that are very different from those of commercial implementations of SQL. By way of illustration, Figure 1 shows the EDDI translations of the SQL queries 1a), 2a) and 3a). Following the default evaluation conventions for SQLZERO, query 1b) returns a set of *distinct* fruit names. Query 2c) causes a semantic error when type checked. Query 3c) returns the natural join of the two tables.

Figure 3 shows the interpretation of queries 2a) and 2b) in the SQL-EDDI environment [3]. To assist the user to appreciate the intimate connection between the relational query language SQLZERO and relational algebra, the environment includes the SQLTE interface that displays the results of SQLZERO to EDDI translation without carrying out the evaluation. As has been pointed out by Date and Darwen, the relationship between standard SQL and relational algebra is much less satisfactory [8,9,11,12].

The SQL-EDDI environment exposes the relationship between variants of SQL and relational algebra. It also serves to distinguish between design flaws in SQL and essential features of relational query languages, such as aggregate operators, that cannot be directly interpreted within relational algebra. The characteristics of SQLZERO are determined by two features: the EDDI evaluation strategy, and the method of translation. More insight into how SQL subverts the sound mathematical foundations offered by EDDI can be gained through trying to derive an EDDI implementation for standard SQL by modifying the evaluation and translation strategies in SQLZERO. For this purpose, it proves helpful to introduce an interface for controlling the evaluation of EDDI queries. This interface ("The Uneddifying Interface") allows the evaluation conventions in EDDI to be changed interactively (see Figure 3). The interface has two functions: it



```
name   | unitsold |
--------------------
cox    | 100      |
granny | 153      |
red    | 70       |
kiwi   | 23       |
lime   | 15       |
lemon  | 55       |
orange | 78       |
lime   | 3        |
orange | 8        |
kiwi   | 5        |
lemon  | 2        |
--------------------
(11 rows)
---------------
name   | qnt |
---------------
lime   | 3   |
orange | 8   |
kiwi   | 5   |
lemon  | 2   |
cox    | 100 |
granny | 153 |
red    | 70  |
kiwi   | 23  |
lime   | 15  |
lemon  | 55  |
orange | 78  |
---------------
(11 rows)
```

Enter %eddi statements:

```
XX = soldfruit + citrus % name, qnt;
YY = citrus % name, qnt + soldfruit;
?XX;
?YY;
```

THE UNEDDIFYING INTERFACE

	YES	NO
Eliminate duplicate rows	X	
Typecheck on attributes		X
Use natural join	X	

Figure 3 - The SQL-EDDI environment in use

exposes what deviations from relational theory are required to implement standard SQL, and it also shows that such deviations have unpleasant conceptual and practical consequences.

The Uneddifying Interface allows three aspects of EDDI evaluation to be adjusted: these reflect the problematic aspects of the SQL queries 1a), 2a), 2b) and 3a). Using the radio buttons, the user can switch between different evaluation conventions, disallowing or allowing multiple rows, imposing strict or loose type checking and using natural or "unnatural" join.

Experimentation with the Uneddifying Interface shows how subverting the mathematical model in EDDI complicates the semantics of relational queries, and leads to violations of the laws of relational algebra. Further analysis shows that the poor design of standard SQL also makes it harder for the software designer to implement the query processing. For example, when the EDDI interpreter returns "unnatural" join, the interpretation of the query `?allfruits*apple%name;` becomes obscure because of the hidden and context-dependent conventions for renaming common attributes. This problem is compounded in more complicated joins. To solve this we have to make the renaming conventions more transparent and use natural join with appropriate renaming to emulate "unnatural" join. This shows that changing the evaluation strategy alone is not sufficient to transform SQLZERO into standard SQL.

4. DEVELOPMENT OF SQL-EDDI

The SQL-EDDI environment has been constructed using Empirical Modelling principles and tools. Space does not permit a detailed discussion to this approach to system development – for more information, the reader is referred to [4]. From the perspective of relational theory, the most significant feature of Empirical Modelling is its use of principles for representing state that can be seen as generalising relational tables and views. State transitions are effected by redefinitions of variables that are analogous to the modification of a table or the introduction of a view.

The application of Empirical Modelling principles allows the flexible development of open-ended resources that are specifically targeted at supporting a particular educational requirement in computer-assisted learning. This is in contrast to educational software products that take the form of monolithic environments whose functionality is carefully preconceived to admit many different applications, but are effectively not open to modification by its users. The model-building approach we have developed is especially well suited to situations in which there is a specific focus of interest, but a whole range of different agents and perspectives must be recruited on-the-fly as a situation evolves.

The development of the SQL-EDDI environment is a practical illustration of the above approach. The specific target here is clarifying the relationship between standard SQL and relational algebra. The account of the features of the SQL-EDDI environment given in the previous sections can also be construed as an account of its development, much of which was carried out on-the-fly in parallel with the teaching of the course to undergraduate students. The functionality of the current system was not envisaged at the outset, but emerged incrementally as the development proceeded. For instance, in the implementation of standard SQL, it was not clear in advance that it was inappropriate to use unnatural join, or that modification of the translator would be necessary. All the required changes to the evaluation strategy and the parser: developing, testing and debugging, are effected by modifying small groups of definitions followed by experimentation and interpretation.

As another illustration, the specific educational objectives can be specialised further to consider issues such as introducing complementary visual representations to enable students and tutors to assess the correctness of queries informally, and integrating formative and summative evaluation mechanisms into the database environment. To achieve this, it is necessary to specialise the database to a particular application domain. For example, over the last three years, we have used a specific data set relating to the scheduling of final year project orals as the basis for practical exercises and assessment. We would ideally like to show the results of database queries such as "list all times on Friday for which Room 104 is available" by displaying the corresponding time slots on a timetable grid. By such specialisation of the data set, we envisage being able to develop a rich resource of special purpose visualisation and assessment tools that can grow opportunistically in response to new teaching needs. In building these visualisation and assessment tools, we can take advantage of special characteristics of the data set as we discover them. For instance, we shall be able to identify classes of queries that admit simple direct visualisations that are well suited for giving experience to novices and for convenient marking. The openness of the Empirical Modelling approach to development is crucial for realising these aims.

With regard to the visualisation of the project oral timetabling we have previously developed a model that allows semi-automated timetabling of student projects [5]. The interface to the model enables the users to visualise the current state of the timetable.

We have also explored the potential for integrating assessment into the SQL-EDDI environment. In our database courses, we have assessed students' knowledge of SQL and relational algebra through both practical coursework and pencil and paper based tests. Tests, individualised to avoid plagiarism, were generated using the underlying EDEN environment, based on tables relating to the project timetabling problem. This approach could be developed further so that students could interact with an SQL-EDDI environment during a summative examination of their practical skills. Further possibilities include providing a formative assessment environment where students could gauge their own understanding through question and answer sessions with individualised feedback focused on their areas of weakness. Such an environment could take advantage of peer assessment strategies such as are to be found in OASYS [6].

The challenge is to integrate both the visualisation and testing environments with the SQL-EDDI environment. The integration task is simplified by using definitions to establish dependencies between components in the environment.

5. FUTURE WORK AND CONCLUSIONS

There is much scope for future work. Possible extensions to the SQL-EDDI environment include:

- supporting a larger subset of SQL features, such as more sophisticated data definition, integrity constraints and support for nulls [9].
- implementation of other relational query languages such as QUEL.
- an interface for studying the optimisation of relational database queries.

SQL-EDDI has not yet been formally evaluated, but Darwen's lectures on the "Askew Wall" [10] in 2003 seemed to be more fully appreciated by students who had previously gained practical experience with the issues through SQL-EDDI than their predecessors. Another significant feature has been the way in which final year project students have integrated relational data models in EDDI with dependency structures in EDEN to obtain richer data models than relations alone afford. In conclusion, our research on SQL-EDDI has demonstrated interesting connections between Empirical Modelling and relational data modelling through which each can be of benefit to the other.

6. REFERENCES

[1] International Organisation for Standardisation (ISO): Database Language SQL, Document ISO/IEC 9075, 1999.

[2] The Empirical Modelling Research Group Website. http://www.dcs.warwick.ac.uk/modelling.

[3] The SQL-EDDI environment. Online at http://empublic.dcs.warwick.ac.uk/projects/sqleddiWard2003.

[4] W.M.Beynon, R.Cartwright, P-H.Sun, A.Ward. Interactive Situation Models for Information Systems Development. Proceedings of SCI'99 and ISAS'99, Vol. 2, pp 9-16, Orlando, USA, July 1999.

[5] W.M.Beynon, A.Ward, S.Maad, A.Wong, S.Rasmequan, S.Russ. The Temposcope: A Computer Instrument for the Idealist Timetabler. Proceedings of the 3rd International Conference on the Practice and Theory of Automated Timetabling, Constance, Germany, August 16-18, 2000.

[6] A.Bhalerao, A.Ward. Towards Electronically Assisted Peer Assessment: A case study. In Association of Learning Technology Journal (ALT-J), Vol. 9, No. 1, April 2001.

[7] E.F.Codd. A Relational Model of Data for Large Shared Data Banks. CACM 13, No.6, 1970.

[8] H.Darwen. The Duplicity of Duplicate Rows. In C.J.Date (with H.Darwen), Relational Database Writings 1991-1994. Reading, Mass., Addison-Wesley, 1995.

[9] H.Darwen. The Nullologist in Relationland. In C.J.Date (with H.Darwen), Relational Database Writings 1991-1994. Reading, Mass., Addison-Wesley, 1995.

[10] H.Darwen. The Third Manifesto Lecture (Formerly entitled The ASKEW Wall). Online at http://www.hughdarwen.freeola.com/TheThirdManifesto.web/TTM-TheAskewWall-printable.pdf

[11] C.J.Date. What's wrong with SQL? In C.J.Date, Relational Database Writings 1985-1989. Reading, Mass., Addison-Wesley, 1990.

[12] C.J.Date, H.Darwen. Foundations for Future Database Systems: The Third Manifesto (2nd Edition). Addison-Wesley, 2000.

[13] S.J.P.Todd. The Peterlee Relational Test Vehicle--A System Overview, IBM Systems Journal 15(4), pp. 285-308, 1976.