The un-structured student.

Colin Myers
Paul Douglas

School of Informatics

The Un-Structured Student

Colin Myers
Software Engineering
University of Westminster
London W1W 6UW
colin@wmin.ac.uk

Paul Douglas
Software Engineering
University of Westminster
London W1W 6UW
p.douglas@wmin.ac.uk

Abstract

SQL (Structured Query Language) is intended to liberate users from the complex syntax, complex semantics and complex memory management that would be required of a procedural approach to relational database manipulation, and so allow users to concentrate on problem-solving. However, students still have problems with language features, program concepts and the legacy of their prior learning. This paper aims to assist lecturers teaching SQL. We present typical student mistakes, attempt to explain why these mistakes arise, and propose possible remedies.

1 Introduction

1.1 Learning SQL

SQL is claimed to user-friendly. Our experience is that students find it more user-friendly than procedural languages but still experience many problems, with SQL syntax, semantics and pragmatics. Many of these problems can be resolved, and most students eventually come to terms with SQL, but generally only employ a limited subset of the language and as a result only address a limited subset of possible queries.

These short-comings have many possible explanations, ranging from the language design, teaching and learning approaches, and the inherent difficulty of recognizing and formulating complex tasks.

Those problems that are a result of the design and purpose of SQL include: inconsistencies within the SQL language (e.g. [1]), conflict between DDL, DML and DCL sub-languages that is not apparent in deductive languages (e.g. [4]) and inadequate modelling power compared to Semantic and Object-Oriented solutions (e.g. [3]). However, it is not our purpose to propose that SQL be re-written or that students drop SQL, rather we aim to improve student learning, and to extend their effective use of SQL.

1.2 Background

The writers of this paper have had a considerable experience of teaching and assessing database systems and SQL in particular to a variety of BSc and MSc students across a number of Universities, both in the UK and also in the USA, Hong Kong, and Singapore. Informally, we perceive no strong differences in initial learning difficulties from any of the different student cohorts.

1.3 Organization

Section 2 of this paper presents a presents a relational schema used for assessment purposes. Section 3 works through a number of natural (ie English) language questions and shows typical student mistakes, followed possible reasons for these mistakes and some suggestions how to deal with these mistakes. The Conclusion collates our recommendations with suggestions for future work.

2 The Database

2.1 Sample Schema

```
The following is a schema for the Secret
Service of an unnamed country:

 CREATE TABLE Spies (
   SpyCode          INT
   SpyName          VARCHAR(30)
   Speciality       VARCHAR(30)
   Experience       INT       -- in years
   PRIMARY KEY      (SpyCode)
 );
 CREATE TABLE Missions (
   MissionName      VARCHAR(30)
   Location         CITY      -- User
                              defined domain
   Danger           INT
   PRIMARY KEY      (MissionName)
```

24th British National Conference on Databases (BNCOD'07)
0-7695-2912-7/07 $20.00 © 2007 IEEE

```
);
CREATE TABLE MissionSpy (
  MissionName      VARCHAR(30)
  SpyCode          INT
  Expendability    INT     -- 0..9, high
                indicates can be sacrificed
  PRIMARY KEY      (MissionName, SpyCode)
);
```

2.2 Different Schemas

Over the years, we have used a number of structurally equivalent databases; for example: Concerts and Musicians; Magicians and Spells; Employers and Departments; that is, they are all some variant on Date's Supplier and Parts schema [1]. We have noticed little difference, if any, in student performance using one schema or another – except that student feedback indicates that they may prefer schemas such as Concerts and Musicians rather than Employers and Departments. This is perhaps because they have had no real exposure to the latter or because the latter is intrinsically dull and that the former provides a 'cognitive hook'.

For pedagogic purposes, we have kept the schemas small (two or three relations) and have intended them to be self-explanatory (and indeed, we have seldom had any questions regarding their meaning). Experience has shown that larger, more complex and less obvious schemas will increase student error, whatever the underlying query set. This does not mean that we have restricted our assessment to simplistic queries, although we have tried to avoid any obligation on the student to master the schema topic.

3 The Questions

3.1 Student Performance

This section presents a series of questions that require SQL DML expressions. The questions get progressively harder, and in timed assessments this is normally indicated by showing the mark awarded for each task. Typical Student responses are commented upon and some recommendations for more effective teaching and learning are made as appropriate.

The first few questions are relatively simple but even so we find that some students, whatever the cohort's learning and experiential background, still make errors. As expected, the later questions show correspondingly less student success.

3.2 DML Query Diagnosis

a) List full details of spies with more than 5 years' experience.

Suggested Answer:

```
SELECT * FROM Spies
 WHERE Experience > 5
```

Comment a1:

We thought this was the easy question. However, students still find ways of going wrong.

Some students replace the keyword SELECT with PROJECT. This is probably a confusion with the PROJECT function in relational algebra. We note, that many standard database textbooks introduce relational algebra (and indeed relation calculus) before SQL. (We do not claim an exhaustive survey but our observation is true for those textbooks shown in Table 1, whilst [1] interleaves his treatment with the SQL equivalent appearing second.) This approach is intended to help understand how the relational model can be manipulated, presumably because set operations are familiar from preliminary discrete mathematics courses.

However, SQL is closer to an implementation of a relational calculus than a relational algebra and thus students find the mapping of SQL onto the latter unnecessarily complicated.

P.Atzeni et al: Database Systems, McGraw Hill, 1999
T.Connolly & C.Begg: Database Systems, Addison-Wesley, 2004
S.Dietrich: Understanding Relational Database Query Languages, Prentice Hall,2001
R.Elmasri & S.Navathe: Fundamentals of Database Systems, Addison-Wesley, 2006
H.Garcia et al: Database Systems, Prentice-Hall, 2002
R.Ramakrishnan & J.Gehrke: Database Management Systems, McGraw Hill, 2003
A.Silberschatz et al: Database System Concepts, McGraw Hill, 2005

Table 1: Selected Database Textbooks

Recommendation a1:

Relational algebra operators of SQL are seldom used for query handling (thus: AND generally replacing INTERSECTION, OR generally replacing UNION, and implicit JOINs replacing CARTESIAN PRODUCTS). Perhaps it would be better to introduce relational algebra where it is used, for example, for implementation and query optimization.

Comment a2:

Many students will enumerate each attribute, and some will even anchor each attribute with the relation name, giving an answer such as

```
SELECT S.SpyCode, S.SpyName,
  S.Speciality, S.Experience
FROM Spies S
WHERE Experience > 5
```

Perhaps, this is a matter of lack of trust. Students observe that using the relation name always succeeds but that omitting the name sometimes fails. They do not observe that it only fails where there is an attribute name clash.

Some mistrusting students also believe that every result needs naming. For example:

```
SELECT * FROM Spies AS ExperiencedSpies
 WHERE Experience > 5
```

Once again this is probably done because it succeeds and sometimes not giving names fails. Students have not grasped the fact that names are only needed if the system cannot provide one, for example, when the same relation appears more than once in a query.

Recommendation a2:

For such cases, show the students both versions and appeal to their laziness in typing. Alternatively, show that the simpler version conforms to the golden rule of programming: 'less code = less mistakes' (sometimes!).

Comment a3:

Many students fail to appreciate the distinction between data and meta-data and thus 'decorate' numeric values, particularly where the attribute represents dates, money etc. For example:

```
SELECT * FROM Spies
 WHERE Experience > 5 years
```

Recommendation a3:

To avoid such transliteration, students need to be provided with a sound understanding of the way that data is stored, and place less trust in the English-like claims for SQL.

b) List, in reverse order, all locations in the Secret Service.

Suggested Answer:

```
SELECT DISTINCT Location FROM Missions
 ORDER BY Location DESCENDING
```

Comment b1:

Almost all students omit the DISTINCT keyword. The query executes but with potential replicates. It is possible to create database snapshots where there will not be any replicates so they don't see any problem.

In the 'real' world this translates to slow development times, especially since errors are often only revealed during testing rather than at query construction (the queries are syntactically sound).

Recommendation b1:

Show snapshots where there are replicates. Emphasize that the DISTINCT can usually be omitted only where the KEY is part of the projection list.

Comment b2:

Some students spell ORDER BY as GROUP BY and spell DESCENDING as DESC (ok in some dialects), as REVERSE or even as ASCENDING.

Recommendation b2:

Students often neglect to practise these simple tasks, for the very fact that they are simple to understand – and then fail on assessment. Since it is fairly common to present results in some sequence in the real world, students should be obliged to write queries involving ordering and other simple formatting requirements.

c) List the names of spies (SpyName) and their missions (MissionNames), for spies who specialize in either Snooping or Forgery.

Suggested Answer:

```
SELECT SpyName, MissionName
FROM Spies S, MissionSpy MS
WHERE S.SpyCode = MS.SpyCode
  AND S.Speciality IN ('Snooping',
                       'Forgery')
  -- also ok: (S.Speciality =
  'Snooping' OR S.Speciality = 'Forgery')
```

Note that we have used the range variables S and MS to save typing; some queries get very cluttered if the base relations have long, meaningful names.

Comment c1:

Some students omit the JOIN clause (S.SpyCode = Ms.SpyCode). In many similar circumstances the query produces the correct result but at the expensive cost of executing a CARTESIAN PRODUCT.

Recommendation c1:

We suggest some hand-evaluation or simulation of how such answers produce their results. We note that some students will complain that they have been told elsewhere to ignore efficiency issues. This is not always true and students need to appreciate that with large data volumes their queries will take too long to execute.

Comment c2:

Some students introduce an extra join clause, e.g.

```
AND Missions.MissionName = Ms.MissionName
```

Once again, this is probably a matter of lack of trust: it works or seems to work, so do it. Actually, in this case, it is only guaranteed to work if MissionName in MissionSpy is referenced by MissionName in Missions – which would be a strange thing. But many students do not understand how

0-7695-2912-7/07 $20.00 © 2007 IEEE

foreign keys, especially what is the referencer and what is the referenced.

Recommendation c2:

As above, we suggest showing an execution trace of this type of excessive query. One additional problem is that students, quite correctly, are told not to bother about efficiency, and that the optimizer will automatically optimize. Perhaps, naively, they believe that it will always remove redundancy.

Comment c3:

Some students will choose attributes not asked for in the question. Sometimes this yields simplistic solutions; for example selecting SpyCode rather than SpyName in some queries could eliminate the need to join on the Spies table. On the other hand this may provide too many attributes: in the above case perhaps selecting Spies.* or SpyCode as well as SpyName.

Recommendation c3:

Students need to recognize that clients need only pay for what they request. In assessment, this means penalizing students who are over-generous.

Comment c4:

Students have trouble with conditional expressions. It is common for them to write:

```
S.Speciality = 'Snooping' OR 'Forgery'
```

Again, we blame student trust in SQL. The problem is that SQL claims to be English-like. Not unnaturally, students are willing to believe it.

d) List all spies with 'Lee' as part of their name, who have a mission in Paris, as long as the expendability of the mission has been decided.

Suggested Answer:

```
SELECT SpyName
FROM Spies S, Missions M, MissionSpy MS
 WHERE S.SpyCode = MS.SpyCode
   AND MS.MissionName = M.MissionName
   AND M.Location = 'Paris'
   AND S.SpyName LIKE '%Lee%'
   AND MS.Expendability IS NOT NULL
```

Comment d1:

As in the previous example, students give a variable number of JOINS. We have similar recommendations. We note that the greater the complexity of the query, the less likely that students actually give the correct number of explicit joins.

Comment d2:

Students are not comfortable with SQL search conditions. Many students will silently ignore the pattern matching requirement and just write S.SpyName = 'Lee'. In a similar manner, students fail with DATE extraction, confuse COUNT with SUM etc.

Recommendation d2:

Perhaps we are to blame. The power of SQL's DML is its declarative manipulation of relations: teaching its aggregate functions, search conditions and other non-declarative features is consequently neglected. On the other hand, these are features are generally not difficult to use, and so assessment could be done in non-timed assignments.

Comment d3:

Students have problems with NULL values. Apart from obvious syntax errors using equality tests rather than testing IS or IS NOT, students are just as likely to check against 0 or the empty string. Students also have problems OUTER JOINs possibly because these results incorporate NULL values.

e) Give counts of how many missions exist in each location.

Suggested Answer:

```
SELECT Location, COUNT(MissionName)
FROM Missions
GROUP BY Location
```

Comment e1:

Students get their syntax confused. The COUNT or wrong attribute appears in the GROUP BY clause or the wrong attribute appears in the COUNT function.

Comment e2:

As noted previously, some students select additional attributes (presumably to be 'helpful' by providing additional information to the user) which causes the query to fail because the attributes chosen are not single-valued for the group. Similarly, some students will always use (*) as the COUNT parameter; this only works if the desired parameter is the only candidate key.

Recommendation e2:

Students need to understand how GROUP BY partitions relations. We find that 'hand working' snapshot examples can really help with GROUP BY, especially as this can be done meaningfully with small data sets using single base tables.

f) If a mission has more than five spies then list its name, together with a count of how many spies undertake in it.

Suggested Answer:

```
SELECT MissionName, COUNT(SpyCode)
FROM MissionSpy
GROUP BY MissionName
HAVING COUNT(SpyCode) > 5
```

Comment f1:

Some students master the semantics of GROUP BY but fail with the syntax of HAVING. In these cases, they will write:

0-7695-2912-7/07 $20.00 © 2007 IEEE

```
WHERE COUNT(SpyCode) > 5
```

Recommendation f1:

Students need to be shown clearly the relationship of GROUP BY .. HAVING to SELECT .. WHERE. This is another type of task where lots of practice helps.

g) Give a count of all spies who do not have a mission.

Suggested Answer:

```
SELECT COUNT (*) FROM Spies S
 WHERE NOT EXISTS
       (SELECT * FROM MissionSpy MS
        WHERE S.SpyCode = MS.SpyCode)
```

Comment g1:

Here it gets harder to generalize the nature of the error, and hence to characterize its genesis. However, we note that expressions with `HAVING COUNT = 0` or some phrase involving IS NULL or IS NOT NULL frequently appears in answers. Sometimes, indeed, they may accidentally give the correct answer for certain snapshots.

Comment g2:

We note the ambiguity of the word 'all'. It is used in a different context in various questions and in natural language. We do not, however, recommend that no queries are ever asked that use 'all' rather that students are encouraged to challenge any questions that may be ambiguous. A clear understanding of what is being queried is essential.

Recommendation g3:

Transliteration to/from SQL can again be useful here in demonstrating cases where the SQL does *not* mean quite the same thing as English.

Comment g3:

Students often fail to distinguish between subqueries that essentially define dynamically-derived data sets, and those that are correlated to an outer query, though the effects of the latter are quite different. This in turn tends to lead to the syntax error of omitting the join from the inner query. This problem is compounded in that no data from the nested query forms part of the result of the outer query. Hence the confusion of many students when, in answer to the question 'what exactly should I select in the inner query' they are told that it really doesn't matter. In brief, students are not confident with the semi-join concept.

Recommendation g3:

A sound understanding of how queries are executed can help. Often students will find a difficult solution where an alternative method of achieving the same result exists; for example, it might be better to use a join than a nested subquery.

h) List SpyCodes for spies who only have missions in GeorgeTown.

Suggested Answer:

```
(SELECT DISTINCT SpyCode
 FROM MissionSpy MS, Mission M
 WHERE MS.MissionName = M.MissionName
 AND M.Location = 'GeorgeTown')
EXCEPT
(SELECT DISTINCT SpyCode
 FROM MissionSpy MS, Mission M
 WHERE MS.MissionName = M.MissionName
 AND M.Location <> 'GeorgeTown')
```

Comment h1:

Another easy question to ask but hard to answer, and generally meeting with poor student response. In addition, to the comments for question-g. We note that some students will silently ignore the 'only' qualifier in the question; their answers are thus just the first SELECT clause.

This is also interesting from an assessment technique point of view. It is common that questions get progressively harder the further down the page (with correspondingly more marks allocated); yet students often fail to recognize that such problems are likely to have a more complex solution.

Comment h2:

This is another type of question for which students may attempt a procedural solution; creating a temporary relation to hold the Spies in GeorgeTown, another for Spies not in GeorgeTown and then executing the relational algebra subtraction. Indeed, our proposed answer is similar, but presented as a single expression. This perhaps indicates that not all problems have a nice declarative solution and that procedural approaches are sometimes more intuitive.

Comment h3:

We note that students also get confused answering such questions as: `Get Spies who participate in all Missions.` They either students either submit simplistic or overcomplex solutions.

Comment h4:

Sometimes students have a partial memory of a solution to a congruent or similar query, and random EXISTS, NOT EXISTS, ALL, ANY clauses are scattered across the SQL 'solution'.

Recommendations h:*

Again, there is no simple panacea to these problems. Our approach is plenty of examples and to show the results of plausible but not quite correct attempts.

0-7695-2912-7/07 $20.00 © 2007 IEEE

i) Spies with more than ten years' experience are considered Spy Masters. Get spy details for all spies who go on more than one mission with a Spy Master who only shares the same speciality as themselves.

Suggested Answer:

We leave this as an exercise for the reader.

Comment i1:

Students do not do well with such over-complex questions, especially under test conditions.

Recommendation i1:

It is self evident that questions should be unambiguous and comprehensible. The first check is whether the problem makes any sense. The second check should be if it is motivating. Students will object to being asked to do hard things merely for the sake of doing them.

We recognize, of course, the desire to stretch the better student, but timed assessments are not the appropriate vehicle to strive for such discrimination.

j) Use the sample schema, to write sample queries demonstrating the use of the SQL features that you have learned.

Comment j1:

Weak students do not do well with such an open-ended request; tending to give simple answers. This is especially true for timed tests.

Recommendation j1:

This sort of open-ended question may be excellent for un-timed assignments, giving the stronger student a chance to explore the less common language features and formulate harder queries – whilst still allowing the slower student to achieve a good mark (and more importantly a good understanding) through extra effort.

3.3 DML Updates, DDL and DCL Diagnosis

For reasons of space, we have restricted our investigation of student responses to tasks involving DML queries. We note, however, that DML updates (inserts, deletes and modification), and DDL and DCL activities often give rise to similar observations to those enumerated above, compounded by the fact that students often confuse the syntax of the different activities. Another major source of problems is the handling of integrity issues, especially foreign and candidate keys.

3.4 Reading SQL

3.4.1 Translating SQL

We sometimes give SQL queries and ask students to provide their English language equivalent. This approach has mixed success. We believe students should be able to 'read' SQL – after all a large part of programming is debugging and amending other people's code (or indeed your own code that was written so long ago that you have had time to forget what it does). However, under timed assessments, many students are sadly inarticulate. We illustrate this with a typical example of student attempts to 'translate' SQL queries:

```
Express the following SQL query in
English:

  SELECT SpyName, MissionName
  FROM Spies S, MissionSpy MS
  WHERE S.SpyCode = MS.SpyCode
  AND S.Speciality IN ('Snooping',
                       'Forgery')
```

Typical answers just echo the SQL:

> Get SpyName and MissionName from Spies and MissionSPy where SpyName SpyCode is joined with MissionSPy SpyCode and SpyName Speciality is in Snooping or Forgery.

Such answers are not wrong and can show some understanding of SQL (e.g. JOINs and SETs) however they would not be much use to somebody who did not understand the relational model.

More complex queries, or those involving GROUP BY or EXISTS generally result in answers that are even nearer to SQL transliterations and further from English. Perhaps, SQL is a victim of its own publicity that it provides an English-like query language.

Recommendation:

However, we still encourage such questions, but not within timed assessments. Rather, they provide useful interactive tutorial exercises and definitely help tutors gauge students understanding.

3.4.2 Debugging SQL

Another approach is to provide a schema, a snapshot database, and pairs of English language questions and incorrect, though well-formed SQL answers. Students are then asked to show why the SQL is wrong and what it actually does: a useful and often-required debugging skill.

Students perform better here than when asked to translate SQL into English (see above). Although the task is more open-ended ('why is the SQL wrong?'), in some ways it is more mechanical and requires less communications skill. We recommend adding such tasks to tutorials and also in timed assessments. Within tutorials, we also recommend asking students to correct ill-formed SQL statements.

0-7695-2912-7/07 $20.00 © 2007 IEEE

4 Conclusions

4.1 Overall Recommendations

Our experience suggests a simple four-point plan to improving student learning of SQL.

Firstly, avoid 'cognitive overload' and teach SQL before teaching the relational algebra and calculus.

Secondly, avoid the 'miracle' approach to teaching. We note that most database textbooks teach SQL by means of a series of English language query and SQL solution pairs. We have found that student performance improves by also showing wrong answers. By anticipating common errors, we have found that their frequency decreases. This approach has proven successful in teaching other programming languages, in particular functional languages [2].

Thirdly, more examples need to be placed on the common 'complex' query types, in particular those involving devil words such as 'all' and 'only'. This is in contradistinction to most databases textbooks which, at best, give an equal weight to all of their query types [see Table 1]. However, it is necessary to be selective in handling complexity and so ensure that students understand the nature of the question, and can anticipate by sample snapshots what correct and incorrect answers will produce.

Fourthly, widen student experience by giving more emphasis on reading SQL, both by translating SQL expressions into natural language and also by judging how well possibly incorrect SQL expressions answer given problems.

We welcome other examples of common and illuminating student mistakes to throw into our rattlebag, to help explain and hopefully reduce such errors.

4.2 Future Work

We intend to expend our analysis to those aspects of SQL (ie DML updates, DDL and DCL) not covered in this paper, and to gather statistical evidence of where students experience difficulties to help focus teaching and learning activities.

References

[1] C.Date. *Introduction to Database Systems (8th ed)*. Addison-Wesley, 2004.

[2] C. . C.Myers. The dysfunctional student. *FPLE LCNS 1022 (ed P.Hartel & R.Plasmeijer)*, 1996.

[3] A. . D. (eds). *Research Foundations in Object-Oriented and Semantic Database Systems*. 1990.

[4] N. et al. *Database Programming Languages*. Prentice Hall, 1996.

0-7695-2912-7/07 $20.00 © 2007 IEEE