Proving the Safety of SQL Queries

Christian Goldberg

Joint work with: Stefan Brass

Institute of Computer Science
Martin-Luther-University
Halle, Germany

QSIC 2005, Melbourne, Australia
1. Introduction
2. Possible Runtime Errors in SQL
3. The Consistency Check
4. Restrictions and Possible Solutions
5. Conclusions
1. Introduction

• Many applications retrieve data via SQL.
• Although SQL is declarative, runtime errors can occur which depend on the data.
• Today’s DBMS do not print any warnings.
• It is undecidable whether a query is safe, i.e. runtime errors cannot occur in any database state.
1. Introduction

• Classification of errors in SQL queries:

  Errors in SQL queries
    ▼
  Syntactic errors    Semantic errors
    ▼
  Task must be known    Task independent
1. Introduction

• Classification of 1225 SQL queries in exam solutions

<table>
<thead>
<tr>
<th>Error Description</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct (no errors)</td>
<td>27 %</td>
</tr>
<tr>
<td>Only Syntactic Errors</td>
<td>24 %</td>
</tr>
<tr>
<td><strong>Only Semantic Errors</strong></td>
<td><strong>18 %</strong></td>
</tr>
<tr>
<td>Both (Semantic and Syntactic Errors)</td>
<td>11 %</td>
</tr>
<tr>
<td>Wrong Task</td>
<td>10 %</td>
</tr>
<tr>
<td>Not counted</td>
<td>10 %</td>
</tr>
</tbody>
</table>

• In 18% of the cases, sensible error messages could have been produced with our techniques

A quite complete list of semantic errors in SQL queries can be found in [S. Brass and C. Goldberg, Journal of Systems & Software – Elsevier 2005 / QSIC 2004]
1. Introduction
2. Possible Runtime Errors in SQL
3. The Consistency Check
4. Restrictions and Possible Solutions
5. Conclusions
2. Possible Runtime Errors

- Types of runtime errors in SQL queries:
  - Subquery term that might return more than one tuple
  - `SELECT INTO` that might return more than one tuple
  - No indicator variable for argument that might be null
  - Runtime error in datatype function
  - Difficult type conversion (string → number)

- One can check for possible runtime errors by reducing the problem to a consistency check.
2. Possible Runtime Errors

• Example 1:

```sql
SELECT S.EMAIL
INTO :E
FROM STUDENTS S
WHERE S.FNAME = :FN
AND S.LNAME = :LN
```

STUDENTS (SID, FNAME, LNAME, EMAIL)
EXERCISES (ENO, TOPIC, MAXPOINTS)
GRADES (SID->STUDENTS, ENO->EXERCISES, POINTS)

• Runtime error if query returns more than one tuple (at least two students with same FNAME and LNAME)
2. Possible Runtime Errors

• Example 1 (reduced to a consistency check):

```sql
SELECT * 
FROM STUDENTS S1, STUDENTS S2
WHERE S1.FNAME = :FN
AND S1.LNAME = :LN
AND S2.FNAME = :FN
AND S2.LNAME = :LN
AND S1.SID <> S2.SID
```

• Original query is safe iff the above query has always an empty result.

(\texttt{WHERE}-condition is inconsistent)
2. Possible Runtime Errors

- Example 2:

```sql
SELECT S.FNAME, S.LNAME
FROM STUDENTS S
WHERE 10 = (SELECT G.POINTS
            FROM GRADES G
            WHERE G.SID = S.SID
            AND G.ENO = 1)
```

- Runtime error if such a subquery returns more than one tuple.
2. Possible Runtime Errors

- Example 3:

```sql
SELECT G.SID, E.ENO, 
100 * G.POINTS/E.MAXPOINTS 
FROM GRADES G, EXERCISES E 
WHERE G.ENO = E.ENO 
ORDER BY G.SID, E.ENO
```

- Runtime error if there exists an exercise with `MAXPOINTS=0` (division by zero).
1. Introduction
2. Possible Runtime Errors in SQL
3. The Consistency Check
4. Restrictions and Possible Solutions
5. Conclusions
3. The Consistency Check

• Our algorithm decides the consistency of the following subset of SQL:

  – No datatype operations (+, -, …)
  – No aggregation functions (SUM, AVG, …)
  – One level of subqueries
    Type: Bernays-Schönfinkel: $\exists \ldots \forall \ldots$
    (but often more)
  – Only $\text{exists}$ subqueries (no real restriction)
  – No LIKE
3. The Consistency Check

- New combination of known techniques:
  - Adding integrity constraints to the query (\textit{NOT EXISTS} violation of integrity constraint)
  - Skolemization
  - Conversion to DNF, pushing down comparison operators by “turning them around”
  - Algorithm of Guo, Sun, Weiss (1996) as a consistency check for conjunctions of conditions with =, <, >, <=, >=, <>

- Besides this: handling of null values (three-valued logic)
3. The Consistency Check

• Example 4:

```
SELECT S.FNAME, S.LNAME, G.ENO
FROM STUDENTS S, GRADES G
WHERE S.SID = G.SID
AND NOT EXISTS
  (SELECT *
   FROM GRADES B
   WHERE B.ENO = G.ENO
   AND B.POINTS > G.POINTS)
```

STUDENTS (SID, FNAME, LNAME, EMAIL)
EXERCISES (ENO, TOPIC, MAXPOINTS)
GRADES (SID->STUDENTS, ENO->EXERCISES, POINTS)
• Adding integrity constraints:

\[ \ldots \]

\[
\text{AND NOT EXISTS} \quad \text{-- Foreign Key}
\]

\[
(\text{SELECT} \quad *
\quad \text{FROM} \quad \text{GRADES X} \quad \text{-- } \forall
\quad \text{WHERE} \quad \text{NOT EXISTS}
\quad (\text{SELECT} \quad *
\quad \text{FROM} \quad \text{STUDENTS Y} \quad \text{-- } \exists
\quad \text{WHERE} \quad \text{X.SID} = \text{Y.SID})
\]

\[ \ldots \]
3. The Consistency Check: Example (3)

• Adding integrity constraints (cont’d):

\[ \text{AND NOT EXISTS -- Primary key of STUDENTS} \]
\[ (\text{SELECT * FROM STUDENTS S1, STUDENTS S2 -- } \forall \text{ WHERE S1.SID = S2.SID AND (S1.FNAME <> S2.FNAME OR S1.LNAME <> S2.LNAME OR S1.EMAIL <> S2.EMAIL)}) \]
3. The Consistency Check: Example (4)

- Introducing Skolem constants and functions with the outer universal tuple variables as parameters for the existential tuple variables

Possible reduction of parameter list: Require that parameter variables appear in the subquery where the existential variable is declared.

- We use the relations as sorts:
  - S: STUDENTS
  - G: GRADES
  - Y(X: GRADES): STUDENTS
3. The Consistency Check: Example (5)

- Herbrand universe for the sorts:
  - **STUDENTS**: S, Y(G)
  - **GRADERS**: G

- Compute the flat form of the condition:
  - Substitute all existential tuple variables with Skolem functions (with parameters)
  - Substitute all **EXISTS**-conditions in an odd number of **NOT** with the disjunction of all assignments of universal tuple variables with the Herbrand universe elements according to the sorts
3. The Consistency Check: Example (6)

- Flat form: (without case of empty domain)

```sql
S.SID = G.SID
AND NOT (G.ENO=G.ENO AND G.POINTS>G.POINTS)
-- Foreign key:
AND NOT (NOT (G.SID=Y(G).SID))
-- Primary key for STUDENTS:
AND NOT (S.SID = S.SID AND
  (S.FNAME <> S.FNAME OR
   S.LNAME <> S.LNAME OR
   S.EMAIL <> S.EMAIL))
AND NOT (S.SID = Y(G).SID AND
  (S.FNAME <> Y(G).FNAME OR
   S.LNAME <> Y(G).LNAME OR
   S.EMAIL <> Y(G).EMAIL))
AND ...
```
• Simplified flat form:

\[
\begin{align*}
S.\text{SID} &= G.\text{SID} \\
\text{AND} \quad G.\text{SID} &= Y(G).\text{SID} \\
\text{AND} \quad S.\text{FNAME} &= Y(G).\text{FNAME} \\
\text{AND} \quad S.\text{LNAME} &= Y(G).\text{LNAME} \\
\text{AND} \quad S.\text{EMAIL} &= Y(G).\text{EMAIL}
\end{align*}
\]

• The result is a model with:
  – One tuple (or two identical tuples) \( S \) and \( Y(G) \) in \textit{STUDENTS}
  – Only one tuple \( G \) in \textit{GRADES} (all with same \( S.\text{SID} \))
1. Introduction
2. Possible Runtime Errors in SQL
3. The Consistency Check
4. Restrictions and Possible Solutions
5. Conclusions
4. Restrictions and Possible Solutions

- The Herbrand universe has to be finite.
- E.g. cyclic foreign keys would lead to an infinite number of possible tuples.
- Also multiple nested subqueries can cause an infinite Herbrand universe if they produce cyclic dependencies between relations.
- Possible solution with heuristical approach:
  - Assume, a fixed number of tuples (e.g. 2 tuples) suffice in the critical relation. If a model is found, the query is consistent.
1. Introduction
2. Possible Runtime Errors in SQL
3. The Consistency Check
4. Restrictions and Possible Solutions
5. Conclusions
5. Conclusions

• In the past (1985) C compilers checked only as much as needed for executing the code.

• Today’s DBMS are still at this state: No warnings, only syntax errors.

• We develop a semantic checker for SQL called sqllint (like lint for C):

   http://dbs.informatik.uni-halle.de/sqllint/
5. Conclusions

• The check for the safety of SQL queries can be reduced to a consistency check.

• The Consistency Check can handle a surprisingly large subset of SQL.

• It can also be used for a variety of other semantic errors (not only runtime errors).

• Our future work focuses on enlarging this subset (e.g. `LIKE`, aggregations, …)