Semantic Errors in SQL Queries

Christian Goldberg & Stefan Brass

Computer Science Institute
Martin-Luther-University Halle
Germany
1. Introduction
2. Unnecessary Complications
3. Violation of Standard Patterns
4. Possible Runtime Errors
5. Conclusions
Classification of errors

Errors in SQL queries

- Syntactic errors
  - Task must be known

- Semantic errors
  - Task-independent
Example 1:

```
SELECT ENAME
FROM EMP
WHERE JOB = 'CLERK' AND JOB = 'MANAGER'
```

- Empty result in all database states (certainly not intended)
- In one exam exercise 10 of 70 students wrote such an inconsistent condition
- In general not decidable
1. Introduction

Importance of detecting semantic errors

- **Education:**
  - Better comprehension and easier learning of SQL (applicable in learning software)

- **Software development:**
  - Faster „debugging“ of SQL code
  - Greater confidence that no bugs remain
  - Improved style (better readability)
  - Detection of unnecessarily complicated / inefficient formulations
1. Introduction

Errors in last 2 exams (percent):

- Syntax: 39%
- Syntax & Semantic: 33%
- Semantic: 16%
- Wrong Task: 12%

Legend:
- Syntax
- Syntax & Semantic
- Semantic
- Wrong Task
1. Introduction

- DB schema
- SQL query

sqllint

semantic error warnings
1. Introduction
2. Unnecessary Complications
3. Violation of Standard Patterns
4. Possible Runtime Errors
5. Conclusions
2. Unnecessary Complications

- Unnecessarily complicated query
  ➔ „probably not intended“

**Situation:**

1. User wrote query A.
2. B exists equivalent to A.
3. B is significantly simpler than A:
   B results from A by deleting parts of the query

- Why do users write such queries?
Possible reasons why the user wrote A:

- B recognized as not correct for the task.
  - A is also not correct: semantic error.
- B not recognized as equivalent to A.
  - User does not yet master SQL: education
- B recognized as equivalent to A:
  - A is faster? But SQL is declarative!
  - A is better readable? But SQL queries are short!
2. Unnecessary Complications

What means „equivalent“?

- Only for states with non-empty relations

Example 2.1:

```sql
SELECT DISTINCT DNAME
FROM DEPT, EMP
```

```sql
SELECT DISTINCT DNAME
FROM DEPT
```
2. Unnecessary Complications

- Minimize the query result without loss of information

Example 2.2:

```sql
SELECT EMPNO, ENAME, JOB
FROM EMP
WHERE JOB = 'MANAGER'
```

```sql
SELECT EMPNO, ENAME
FROM EMP
WHERE JOB = 'MANAGER'
```
2. Unnecessary Complications

- Complications possible in all query parts:
  - **SELECT**: Unnecessary DISTINCT, Constant / duplicate output columns
  - **FROM**: Unused tuple vars, Unnecessary joins
  - **WHERE**: Implied, tautological or inconsistent subconditions, Unnecessary general comparison operator
  - **GROUP BY**: singleton groups, only one group
  - ...  
  - **Entire query unnecessary** (Inconsistent cond.)
Example 2.3 (singleton groups):

```
SELECT EMPNO, MAX(SAL)
FROM EMP
WHERE JOB = 'MANAGER'
GROUP BY EMPNO
```

Example 2.4 (comparison operator):

```
SELECT ENAME, SAL
FROM EMP
WHERE SAL >= (SELECT MAX(SAL) FROM EMP)
```
2. Unnecessary Complications

Relation to Query Optimization:

- General connection to semantic query optimization
- Some problems solved in theory, but only few implemented in current DBMS
- Optimization often during query execution → not enough time for complex algorithms
- Optimizer needs strict equivalence, we can apply heuristic assumptions
- Correcting programming errors is not the task of the optimizer: “garbage in, garbage out”
1. Introduction
2. Unnecessary Complications
3. Violation of Standard Patterns
4. Possible Runtime Errors
5. Conclusions
3. Violations of Standard Patterns

- Missing join conditions
- Uncorrelated EXISTS-subqueries
- SELECT clause of subquery uses no tuple variable from the subquery
- Conditions in subquery that can be moved up
- Comparison between different domains
- HAVING without GROUP BY
- DISTINCT in SUM and AVG
- Wildcards without LIKE
3. Violations of Standard Patterns

Duplicates

- Often an indication for another error, e.g. missing join conditions
- Handling difficult as duplicates are sometimes desired → declaration of “soft keys” would be useful

Example 3:

```
SELECT JOB  
FROM EMP   
WHERE DEPTNO = 20

SELECT ENAME  
FROM EMP   
WHERE DEPTNO = 20
```
1. Introduction
2. Unnecessary Complications
3. Violation of Standard Patterns
4. Possible Runtime Errors
5. Conclusions
4. Possible Runtime Errors

Example 4:

\[
\text{SELECT ENAME, JOB} \\
\text{FROM EMP E} \\
\text{WHERE E.DEPTNO = ( SELECT DEPTNO} \\
\text{FROM DEPT D} \\
\text{WHERE D.LOC = 'NEW YORK')} \\
\]

- Runtime error if more than one department in New York
- Dependent on the database state
- Not necessarily found during testing
Example 4 (check for this runtime error):

```
SELECT * 
FROM DEPT D1, DEPT D2 
WHERE D1.LOC = 'NEW YORK' 
AND D2.LOC = 'NEW YORK' 
AND D1.DEPTNO != D2.DEPTNO
```

- Constructed subquery is consistent if and only if the runtime error can occur
- Many error checks can be reduced in this way to a consistency check
4. Possible Runtime Errors

Types of runtime errors:

- Subquery terms that might return more than one tuple
- SELECT INTO that might return more than one tuple
- No indicator variable for arguments that might be NULL
- Difficult type conversions (string → number)
- Runtime errors in datatype functions
1. Introduction
2. Unnecessary Complications
3. Violation of Standard Patterns
4. Possible Runtime Errors
5. Conclusions
5. Conclusions

Most frequent semantic errors

<table>
<thead>
<tr>
<th>Percentage</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>15%</td>
<td>Many Duplicates</td>
</tr>
<tr>
<td>13%</td>
<td>Missing Join Conditions</td>
</tr>
<tr>
<td>13%</td>
<td>Inconsistent Conditions</td>
</tr>
<tr>
<td>11%</td>
<td>Unnecessary Joins</td>
</tr>
<tr>
<td>8%</td>
<td>Unused Tuple Variables</td>
</tr>
<tr>
<td>6%</td>
<td>GROUP BY with Singleton Groups</td>
</tr>
<tr>
<td>5%</td>
<td>Implied, Tautological or Inconsistent Subconditions</td>
</tr>
</tbody>
</table>

- 2 Exams, each with 3 SQL-exercises
- 153 and 148 participants
5. Conclusions

- Current database systems print no warnings, only error messages if query is not executable.
- SQL is different from programming languages, but not that different.
- Regarding warnings, database systems are more than 10 years behind current compilers.
- The paper describes 37 classes of semantic errors.
- Given precise error descriptions, it is usually not difficult to develop testing algorithms for these errors.
- However, because most questions are in general undecidable, only an SQL subset can be handled.