Semantic Errors in Database Queries

Stefan Brass

TU Clausthal, Germany
From April: University of Halle, Germany

Classification of Errors

- Syntax Error: Input not element of query language.
  $\Rightarrow$ DBMS will find error, good theory.
- Semantic error: Syntactically correct, but does not (always) compute intended answer.
  - ◇ Meaningful, but wrong for given task.
  - ◇ Obviously not right, no matter for which task.
    $\Rightarrow$ Topic of this talk.

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

Relevant to This Seminar?

- Important for any query/specification language.
- Most of us teach databases or formal methods.
  - ◇ E.g. in one exam exercise: 10 out of 70 students had an inconsistent condition in an SQL query.
  - ◇ Even professionals sometimes make mistakes.
- When features are combined in one application, there might be problems/conflicts, some of which could be detected with these methods.
- Work in progress: I need feedback.

Contents

1. Inconsistent Conditions
2. Integrity Constraints
3. Duplicates
4. General Remarks, Unnecessary Complications
5. Conclusion

Undecidability

- The consistency of logical formulas is undecidable (as well known from the literature).

  E.g. PKP encoded as SQL query [Abiteboul/Hull/Vianu 1994].

- This does not mean that one cannot do anything:
  - ◇ For a relatively large SQL subset the question is decidable.
  - ◇ Otherwise heuristic warnings (might be wrong).

    If warning is wrong: Think about alternative, easier formulations (might help also programmer) or comment ("This is ok").

- Similar to `lint` semantic checker for C.

Inconsistency: Simple Case (1)

- Conditions of the Form $A\,\theta\,B$ or $A\,\theta\,c$, connected with `AND`, `OR`, `NOT` (no subqueries, no aggregations, no datatype operations).

  A, B attributes, c constant, θ comparison op. (=, <>, <, <=, >, >=).

- See [Guo, Sun, Weiss, 1996].
- Compute DNF, test conjunctions with graph:
  - ◇ Nodes are equivalence classes of attributes and constants (wrt "$=$").
  - ◇ Edges are labelled "$<$" or "$\leq$".

Inconsistency: Simple Case (2)

- Example: Find indexes over the columns `FIRST_NAME` and `LAST_NAME` in the Oracle data dictionary.
- Solution in an exam (wrong: inconsistent):

```
SELECT U.INDEX_NAME
FROM   COLS C1, COLS C2, USER_IND_COLUMNS U
WHERE  U.COLUMN_NAME = C1.COLUMN_NAME
AND    U.COLUMN_NAME = C2.COLUMN_NAME
AND    C1.COLUMN_NAME = 'FIRST_NAME'
AND    C2.COLUMN_NAME = 'LAST_NAME'
```

- Only one node: `U.COLUMN_NAME`, `C1.COLUMN_NAME`, `C2.COLUMN_NAME`, `'FIRST_NAME'`, `'LAST_NAME'`.

Inconsistency: Subqueries (1)

- One can handle also `NOT EXISTS` subqueries that do not contain further subqueries:

```
SELECT ...
FROM   R_1 X_1, ..., R_n X_n
WHERE  φ
AND    NOT EXISTS (SELECT ...
                   FROM   S_1 Y_1, ..., S_m Y_m
                   WHERE  ψ)
```

- It suffices to search for a model (DB state) that contains only one tuple for each X_i.

  If there is any model of the query condition, one can remove all tuples except those used in one assignment for X_i that makes the condition true. `NOT EXISTS` is anti-monotic: Removing tuples never makes it false.

Inconsistency: Subqueries (2)

- Consider the mappings (substitutions)

$$f\colon \{Y_1, \ldots, Y_m\} \to \{X_1, \ldots, X_n\}$$

such that $f(Y_i) = X_j$ only if $S_i = R_j$.

These are the variable assignments for the tuple variables of the inner query if there are only the tuples named by the tuple variables in the outer query.

- Let $f_1 \ldots, f_k$ be all such f. The above query is consistent iff the following formula is consistent:

$$\varphi \wedge \neg f_1(\psi) \wedge \cdots \wedge \neg f_k(\psi)$$

- Fits with theorem of Bernays/Schönfinkel: $\forall \ldots \exists \ldots$

Inconsistency: Subqueries (3)

- Task: Find non-indexed columns.
- Solution in an exam (wrong: inconsistent):

```
SELECT X.TABLE_NAME, X.COLUMN_NAME
FROM   COLS X, USER_IND_COLUMNS Y
WHERE  X.TABLE_NAME = Y.TABLE_NAME
AND    X.COLUMN_NAME = Y.COLUMN_NAME
AND    NOT EXISTS
       (SELECT *
        FROM   USER_IND_COLUMNS Y
        WHERE  X.TABLE_NAME = Y.TABLE_NAME
        AND    X.COLUMN_NAME = Y.COLUMN_NAME)
```

Inconsistency: Subqueries (4)

- Often, also nested subqueries can be handled.
- Skolem-function s_X of sort R for each $\exists$-quantified (inside even number of `NOT`) tuple variable X over R.
- Parameters of s_X: Correspond to $\forall$-quantified tuple variables Y in the scope of which X is declared,
  - $\diamond$ and which appear in the subquery in which X is declared (even stricter arity reduction possible).
- Consistency is decidable if only finitely many terms can be built from these Skolem functions.

Inconsistency: Null Values

- Satisfiable if the attribute B can be null:

```
SELECT X.A FROM R X
WHERE  NOT EXISTS (SELECT * FROM R Y
                   WHERE  Y.B = Y.B)
```

- Can be handled by enclosing subquery formulas into new logical operator NTF ("null to false"):

p	TRUE	UNDEFINED	FALSE
NTF(p)	TRUE	FALSE	FALSE

- Pushed down to atomic formulas (possibly converted to NTT), simple extension of model construction.

Contents

1. Inconsistent Conditions
2. Integrity Constraints
3. Duplicates
4. General Remarks, Unnecessary Complications
5. Conclusion

Integrity Constraints (1)

- Task (again): Indexes over `FIRST_NAME`, `LAST_NAME`.
- Solution in an exam (wrong: inconsistent):

```
SELECT U.INDEX_NAME
FROM   USER_IND_COLUMNS X, USER_IND_COLUMNS Y
WHERE  X.INDEX_NAME = Y.INDEX_NAME
AND    X.COLUMN_POSITION = Y.COLUMN_POSITION
AND    X.COLUMN_NAME = 'FIRST_NAME'
AND    Y.COLUMN_NAME = 'LAST_NAME'
```

- `INDEX_NAME` and `COLUMN_POSITION` are together a key for `USER_IND_COLUMNS`.
- Therefore, `X` and `Y` must be the same tuple.

Integrity Constraints (2)

- Domains: `CHECK(NULLABLE IN ('Y','N'))`.
  Then the following condition is certainly wrong:

  ```
  WHERE X.NULLABLE = 'y'
  ```

- Strong relation to semantic query optimization.

  While for optimization purposes, only fast algorithms are useful, we would be willing to invest more time for detecting/locating an error.

- In order to find possible errors, also "soft constraints" are useful (satisfied only in most cases).

  E.g. the columns "`TABLE_NAME`" and "`COLUMN_NAME`" are normally disjoint, although there can be exceptions. Therefore, the condition "`X.TABLE_NAME = Y.COLUMN_NAME`" should generate a warning.

Integrity Constraints (3)

- In order to check the consistency of a query

  SELECT ... FROM ... WHERE φ

  given a constraint

  $$\forall X_1 \colon R_1 \ \ldots \ \forall X_n \colon R_n \, (\psi)$$

  one can check this query with the above method:

```
SELECT ... FROM ...
WHERE  φ
AND    NOT EXISTS (SELECT *
                   FROM   R1 X1, ..., Rn Xn
                   WHERE  ¬ψ)
```

Contents

1. Inconsistent Conditions

2. Integrity Constraints

3. Duplicates

4. General Remarks, Unnecessary Complications

5. Conclusion

Duplicate Answers (1)

- Duplicates in the answer are often an indication for an error (e.g. a missing join condition).
- If a query generates many duplicates, it is difficult to read and understand the result:

```
SELECT JOB FROM EMP
```

- One would better write one of the following:
  - `SELECT DISTINCT JOB FROM EMP`
  - `SELECT JOB, COUNT(*) FROM EMP GROUP BY JOB`

Duplicate Answers (2)

- If duplicates are seldom, they are no problem:

```
SELECT ENAME FROM EMP WHERE DEPTNO = 10
```

- In the unlikely case that there should be two employees with the same name in the same department, one would want to see the duplicate.
- The intention behind this query is to ask for employee objects, duplicates might only occur because the name is an imperfect way to identify them.

  The name or the combination of name and department is a kind of "soft key". At a high warning level, the user should be informed that the identification is not reliable.

Duplicate Answers (3)

- Let the following query be given:

$$\begin{array}{ll} \texttt{SELECT} & t_1, \ \ldots, \ t_k \\ \texttt{FROM} & R_1 \ X_1, \ \ldots, \ R_n \ X_n \\ \texttt{WHERE} & \varphi \end{array}$$

- The result cannot contain duplicates if and only if the following query has an inconsistent condition:

$$\begin{array}{ll} \texttt{SELECT} & * \\ \texttt{FROM} & R_1 \ X_1, \ \ldots, \ R_n \ X_n, \ R_1 \ X'_1, \ \ldots, \ R_n \ X'_n \\ \texttt{WHERE} & \varphi \wedge \varphi' \wedge t_1 = t'_1 \wedge \cdots \wedge t_k = t'_k \\ \texttt{AND} & (X_1 \neq X'_1 \vee \cdots \vee X_n \neq X'_n) \end{array}$$

Duplicate Elimination (4)

Simpler sufficient condition for "No Duplicates":

- Let $\mathcal{K}$ be the set of attributes that appear as output columns under `SELECT`.

  The elements of $\mathcal{K}$ are of the form "Tuplevariable.Attribute". $\mathcal{K}$ is the set of attributes that have a unique value for a given output row.

- Add to $\mathcal{K}$ attributes A such that $A = c$ with a constant c appears in the `WHERE`-condition.

  This test assumes that the condition is a conjunction. Of course, a condition $c = A$ is treated in the same way. Conditions in subqueries do not count (subqueries are simply removed before the test is done).

Duplicate Elimination (5)

Sufficient condition for "No Duplicates", continued:

- As long as something changes, do the following:
  - ◇ Add to $\mathcal{K}$ attributes A such that $A = B$ appears in the `WHERE`-condition and $B \in \mathcal{K}$.
  - ◇ If $\mathcal{K}$ contains a key of a tuple variable, add all other attributes of this tuple variable.
- If $\mathcal{K}$ contains a key of every tuple variable listed under `FROM`, the query cannot generate duplicates.

  If the query contains `GROUP BY`, one checks instead whether all `GROUP BY` columns are contained in $\mathcal{K}$.

Contents

1. Inconsistent Conditions

2. Integrity Constraints

3. Duplicates

4. General Remarks, Unnecessary Complications

5. Conclusion

General Framework (1)

- If the task is not known, it is in general difficult to declare a query as "semantically wrong".
- But if the condition is inconsistent, the query result is certainly smaller than intended.
- When is the query result probably too large?
  - ◇ Many duplicates.
  - ◇ Missing join conditions.
  - ◇ Constant result columns.
  - ◇ Two result columns are always identical.

  $\Rightarrow$ Smaller output contains the same information.

General Framework (2)

- Unnecessary complications in the query indicate a semantic error or bad use of the query language.
- One is tempted to declare a query Q as bad if there is an equivalent query Q' that is shorter than Q.
- Two possibilities why the user did not choose Q':
  - ◇ The user knew that Q' would not correctly formalize the task (thus, also Q is wrong).
  - ◇ The user did not know that the query Q could also be written in this way.
- Inconsistent condition: Entire query unnecessary.

General Framework (3)

- "Shorter" must be measured carefully: Good names or explicit tuple variables in attribute references should not increase the size.
- Unnecessarily imprecise/weak comparison:

```
SELECT EMPNO, ENAME, SAL FROM EMP
WHERE  SAL >= (SELECT MAX(SAL)
               FROM   EMP)
```

- Better: "=". Several students used `IN`, but the subquery is guaranteed to return only one value.

  Fits into this framework (with right definition of query length: `>=` and `IN` have higher weight than `=`).

General Framework (4)

- It might be too strict to exclude equivalent queries Q' that have a very different structure than Q.
- However, if Q' can be constructed from Q by local modifications that reduce the size ("deletions"), then Q is indeed not good.
- E.g. if a subcondition is replaced by `TRUE` or `FALSE` (and the natural simplification done), the result should not be equivalent to the original query.
- Formalization as a relation $Q \longrightarrow Q'$ between queries.

General Framework (5)

- More examples for unnecessary complications:
  - Unnecessary joins
  - Two tuple variables are always equal
  - Unnecessary `DISTINCT`, also `MAX(DISTINCT ...)`
  - `EXISTS(SELECT A, B, C FROM ...)`
  - `GROUP BY`, but no aggregation (→ adv. duplicate elim.)
  - `GROUP BY`: key, groups consist of one element each
  - `GROUP BY`: constant column, only one group
  - `HAVING`-condition could be written under `WHERE`
  - Subquery under `FROM`: merge with main query

General Framework (6)

- The equivalence condition must be slightly relaxed:

```
SELECT D.DNAME
FROM   DEPT D, EMP E
WHERE  D.DEPTNO = 10
```

- It should be possible to delete the unused tuple variable `E`, but the equivalence needs assumptions:
  - ◇ Relations are normally not empty.
  - ◇ The duplicates generated by the presence of `E` are irrelevant.
- Also constant and double output columns etc. can be considered irrelevant.

Other Things to Check

- More indicators for errors:
  - Non-correlated `EXISTS`-subqueries.
  - `SUM(DISTINCT ...)`, `AVG(DISTINCT ...)`.
  - `HAVING` without `GROUP BY`
- Warnings for possible runtime errors:
  - Conditions of the form `A = (SELECT ...)`, but the subquery might return more than one value.
  - Same problem: `SELECT ... INTO ...`
  - No indicator variable, but null value possible.
- Style checks, e.g. no "shadowed" tuple variables.

Contents

1. Inconsistent Conditions

2. Integrity Constraints

3. Duplicates

4. General Remarks, Unnecessary Complications

5. Conclusion

Summary

- Current DBMS give only an error message if the query cannot be executed.
- There are no warnings in case of strange constructs that are likely to be semantic errors.

  Compared to the state of the art in compiler construction, systems like Oracle are about 10 years behind.
- The problems are in general undecidable, but there are algorithms for a surprisingly large SQL subset.
- Important for database education, software development, feature integration, query optimization.