Generating Test Data for Killing SQL Mutants: A Constraint-based Approach

Shetal Shah, S. Sudarshan, Suhas Kajbaje, Sandeep Patidhar, Bhanu Pratap Gupta, Devang Vira

CSE Department, IIT Bombay

Presented By: Sunny Raj Rathod
Outline

● Motivation
● Mutation Testing
● Related Work
● Contributions
● Extensions
● Implementation[XDa-TA]
● Experiments
● Future Work
Testing SQL Queries: A Challenge

- Complex SQL queries hard to get right
- Question: How to check if an SQL query is correct?
  - Formal verification is not applicable since we do not have a separate specification and an implementation
  - State of the art solution: manually generate test databases and check if the query gives the intended result
    - Often misses errors
Generating Test Data: Prior Work

- Automated Test Data generation
  - Based on database constraints, and SQL query
    - Agenda [Chays et al., STVR04]
  - Reverse Query Processing [Binning et al., ICDE07] takes desired query output and generates relation instances
    - Handle a subset of Select/Project/Join/GroupBy queries
  - Extensions of RQP for performance testing
    - guarantees cardinality requirements on relations and intermediate query results

- None of the above guarantee anything about detecting errors in SQL queries

- Question: How do you model SQL errors?
  - Answer: Query Mutation
Mutation Testing

- Mutant: Variation of the given query
  - Mutations model common programming errors, like
    - Join used instead of outerjoin (or vice versa)
    - Join/selection condition errors
      - $<$ vs. $\leq$, missing or extra condition
    - Wrong aggregate (min vs. max)
  - Mutant may be the intended query
Mutation Testing of SQL Queries

- Traditional use of mutation testing has been to check coverage of dataset
  - Generate mutants of the original program by modifying the program in a controlled manner
  - A dataset kills a mutant if query and the mutant give different results on the dataset
  - A dataset is considered complete if it can kill all non-equivalent mutants of the given query

- Our goal: generating dataset for testing query
  - Test dataset and query result on the dataset are shown to human, who verifies that the query result is what is expected given this dataset
  - Note that we do not need to actually generate and execute mutants
Related Work

- **Prior work:**
  - Tuya and Suarez-Cabal [IST07], Chan et al. [QSIC05] defined a class of SQL query mutations
  - Shortcoming: do not address test data generation
  - More recently (and independent of our work) de la Riva et al [AST10] address data generation using constraints, with the Alloy solver
    - Do not consider alternative join orders, No completeness results, Limitations on constraints
Our Contributions

- Principled approach to test data generation for given query
- Define class of mutations:
  - Join/outerjoin
  - Selection condition
  - Aggregate function

- Algorithm for test data generation that kills all non-equivalent mutants in above class for a (fairly large) subset of SQL.
  - Under some simplifying assumptions
  - With the guarantee that generated datasets are small and realistic, to aid in human verification of results
Killing Join Mutants: Example 1

- **Example 1: Without** foreign key constraints
  - Schema: \( r(A), s(B) \)
  
  ![Diagram](image)

- To kill this mutant: ensure that for some \( r \) tuple there is no matching \( s \) tuple
- Generated test case: \( r(A) = \{(1)\}; s(B) = {} \)
- **Basic idea, version 1 [ICDE 2010]**
  - run query on given database,
  - from result extract matching tuples for \( r \) and \( s \)
  - delete \( s \) tuple to ensure no matching tuple for \( r \)
  
  **Limitation: foreign keys, repeated relations**
Example 2: Extra join above mutated node

Schema: $r(A,B)$, $s(C,D)$, $t(E)$

To kill this mutant we must ensure that for an $r$ tuple there is no matching $s$ tuple, but there is a matching $t$ tuple.

Generated test case: $r(A,B)=\{(1,2)\}$; $s(C,D)=\{}$; $t(E)=\{(2)\}$
**Example 3**: Equivalent mutation due to join

- **Schema**: $r(A,B)$, $s(C,D)$, $t(E)$

- **Note**: right outer join this time
- Any result with a $r.B$ being null will be removed by join with $t$
- Similarly equivalence can result due to selections
teaches $\bowtie$ instructor is equivalent to teaches $\bowtie$ instructor if there is a foreign key from teaches.ID to instructor.ID

BUT: teaches $\bowtie$ $\sigma_{\text{dept}=\text{CS}}$(instructor) is not equivalent to teaches $\bowtie$ $\sigma_{\text{dept}=\text{CS}}$(instructor)

Key idea: have a teaches tuple with an instructor not from CS

Selections and joins can be used to kill mutations
Killing Join Mutants: Equivalent Trees

- **Space of join-type mutants**: includes mutations of join operator of a single node for all trees equivalent to given query tree.
- Datasets should kill mutants across all such trees.
Equivalent Trees and Equivalence Classes of Attributes

- Whether query conditions written as
  - A.x = B.x AND B.x = C.x or as
  - A.x = B.x AND A.x = C.x

should not affect set of mutants generated

- Solution: Equivalence classes of attributes

---

![Diagram]

- a. Given Query
- b. Equivalent Query
- c. Join Reordering on (b)
- d. Intended Query
Assumptions

- A1, A2: Only primary and foreign key constraints; foreign key columns not nullable
- A3: Single block SQL queries; no nested subqueries
- A4: Expr/functions: Only arithmetic exprs
- A5: Join/selection predicates: conjunctions of \{expr relop expr\}
- A6: Queries do not explicitly check for null values using IS NULL
- A7: In the presence of full outer join, at least one attribute from each of its inputs present in the select clause (and A8 for natural join: see paper)
Data Generation in 2 Steps

- **Step 1: Generation of constraints**
  - Constraints due to the schema
  - Constraints due to the query
  - Constraints to kill a specific mutant

- **Step 2: Generation of data from constraints**
  - Using solver, currently CVC3
Running Example: University Schema (Book)

```
SELECT *
FROM crse, dept, teaches
WHERE crse.dept_name = dept.dept_name
AND crse.course_id = teaches.course_id
```

Relations:

- `crse(course_id, dept_name, credits)`
- `dept(dept_name, building, budget)`
- `teaches(instructor_id, course_id, semester, acadyear)`
Data Generation Algorithm - Overview

procedure generateDataSet(query q)
  ➡ preprocess query tree
  ➡ generateDataSetForOriginalQuery()
  ➡ killEquivalenceClasses()
  ➡ killOtherPredicates()
  ➡ killComparisonOperators()
  ➡ killAggregates()
Preprocess Query Tree

• Build Equivalence Classes from join conditions
  – A.x = B.y and B.y = C.z then
    Equivalence class: A.x, B.y and C.z

• Foreign Key Closure
  – A.x -> B.y and B.y -> C.z then A.x -> C.z

• Retain all join/selection predicates other than equijoin predicates
Dataset for Original Query

- Generate datatype declarations for CVC3

  DATATYPE COURSE_ID = BIO101 | BIO301 | BIO399 | CS101 | CS190 | CS315 | CS319 | CS347 | CS630 | CS631 | CS632 | EE181 | FIN201 | HIS351 | MU199 | PHY101 END;

  CREDITS : TYPE = SUBTYPE (LAMBDA (x: INT) : x > 1 AND x < 5);

- Array of tuples of constraint variables, per relation

  CRSE_TupleType: TYPE = [COURSE_ID, DEPT_NAME, CREDITS];
  O_CRSE: ARRAY INT OF CRSE_TupleType;
  TEACHES_TupleType: TYPE = [INSTRUCTOR_ID, COURSE_ID, SEMESTER, ACADYEAR];
  O_TEACHES: ARRAY INT OF TEACHES_TupleType

  O_CRSE[1].0 is a constraint variable corresponding to COURSE_ID of the first tuple
Dataset for Original Query

- One or more constraint tuples from array, for each occurrence of a relation

  O_CRSE_INDEX_INT : TYPE = SUBTYPE (LAMBDA (x: INT) : x > 0 AND x < 2);
  O_DEPT_INDEX_INT : TYPE = SUBTYPE (LAMBDA (x: INT) : x > 0 AND x < 2);
  O_TEACHES_INDEX_INT : TYPE = SUBTYPE (LAMBDA (x: INT) : x > 0 AND x < 2);

- More than 1 tuple required for aggregation, repeated occurrences or to ensure f.k. Constraints

- Equality conditions between variables based on equijoins

  ASSERT (O_CRSE[1].1 = O_DEPT[1].0) ;
  ASSERT O_CRSE[1].0 = O_TEACHES[1].

- Other selection and join conditions become constraints
Dataset for Original Query (DB Constraints)

- **Constraints for primary and foreign keys**
  - f.k. from crse.deptname to dept.dept_name
    - \[\text{ASSERT} \ \forall i \ \exists j \ (\text{O\_CRSE}[i].1 = \text{O\_DEPT}[j].0)\];
  - p.k. on R.A
    - \[\text{ASSERT} \ \forall i \ \forall j \ (\text{O\_CRSE}[i].0 = \text{O\_CRSE}[j].0) \Rightarrow \text{“all other attrs equal”}\]
    - Why not assert primary key value is distinct (supported by CVC3)?
  - Since range is over finite domain, p.k. and f.k. constraints can be unfolded

- **Unfolded constraints:**
  - f.k.: \[\text{ASSERT} \ \text{O\_CRSE}[1].1 = \text{O\_DEPT}[1].0 \ \text{OR} \ \text{O\_CRSE}[1].1 = \text{O\_DEPT}[2].0\]
  - p.k.: \[\text{ASSERT} \ (\text{O\_DEPT}[1].0 = \text{O\_DEPT}[2].0) \Rightarrow (\text{O\_DEPT}[1].1 = \text{O\_DEPT}[2].1) \\text{AND} \ (\text{O\_DEPT}[1].2 = \text{O\_DEPT}[2].2)\]
Helper Functions

• CvcMap
  – Takes a rel and attr and returns $r[i].pos$ where
  – $r$ is base relation of rel
  – pos is the position of attribute attr
  – $i$ is an index in the tuple array

• GenerateEqConds($P$)
  – Generates equality constraints amongst all elements of an equivalence class $P$
Killing Join Mutants: Equijoin

\textbf{killEquivalenceClasses()}

- for each equivalence class \( ec \) do
  - Let allRelations := Set of all \(<\text{rel}, \text{attr}>\) pairs in \( ec \)
  - for each element \( e \) in allRelations do
    - \( \text{conds} := \text{empty set} \)
    - Let \( e := R.a \)
    - \( S := \left( \text{set of elements in } ec \text{ which are foreign keys referencing } R.a \text{ directly or indirectly} \right) \cup R:a \)
    - \( P := ec - S \)
    - if \( P:\text{isEmpty}() \) then
      - continue
    - else … main code for generating constraints (see next slide)
Killing Join Mutants: EquiJoins

- `conds.add(generateEqConds(P))`
- `conds:add(`
  
  “NOT EXISTS i: R[i].a = ” + cvcMap(P[0]))

- for all other equivalence classes oe do
  - `conds.add(generateEqConds(oe))`

- for each other predicate p do
  - `conds:add(cvcMap(p))`

- `conds.add(genDBConstraints()) /*P.K. and F.K*/`
- `callSolver(conds)`
- if solution exists then
  - `create a dataset from solver output`
Killing Other Predicates

- Create separate dataset for each attribute in predicate
- e.g. For Join condition B.x = C.x + 10
  - Dataset 1 (nullifying B:x):
    - \texttt{ASSERT NOT EXISTS (i : B\_INT) : (B[i].x = C[1].x + 10);}  
  - Dataset 2 (nullifying C:x):
    - \texttt{ASSERT NOT EXISTS (i : C\_INT) : (B[1].x = C[i].x + 10);}
Comparison Operation Mutations

• Example of comparison operation mutations:
  A < 5 vs. A <= 5 vs. A > 5 vs A >= 5 vs. A=5, vs A <> 5

• Idea: generate separate dataset for three cases (leaving rest of query unchanged):
  • A < 5
  • A = 5
  • A > 5

• This set will kill all above mutations
Aggregation Operation Mutations

- Aggregation operations
  - `count(A)` vs. `count(distinct A)`
  - `sum(A)` vs `sum(distinct A)`
  - `avg(A)` vs `avg(distinct A)`
  - `min(A)` vs `max(A)`
  - and mutations amongst all above operations

- Idea: given relation `r(G, O, A)` and query
  ```sql
  select aggop(A) from r group by G
  ```
  Tuples `(g1, o1, a1), (g1, o2, a1), (g1, o3, a2)`, with `a1 <> 0` will kill above pairs of mutations
  - Additional constraints to ensure killing mutations across pairs
Aggregation Operation Mutants

- Issues:
  - Database/query constraints forcing A to be unique for a given G
  - Database/query constraints forcing A to be a key
  - Database/query constraints forcing G to be a key

- Carefully crafted set of constraints, which are relaxed to handle such cases
Completeness Results

**Theorem:** For the class of queries, with the space of join-type and selection mutations defined in the paper, the suite of datasets generated by our algorithm is complete. That is, the datasets kill all non-equivalent mutations of a given query.

- Completeness results for restricted classes of aggregation mutations
  - Aggregation as top operation of tree, under some restrictions on joins in input
Complexity

- Number of datasets generated is linear in query size

- Although solving constraints is in general NP-hard, and even undecidable with arbitrary constraints, it is tractable in special cases.
Extensions

• Unintended Joins
• Nested subqueries
• Handling NULLs
• String Constraints
• Distinct
• Others – Set ops, Parameterized Queries, Date-Time, Insert, Update, Delete, Disjunctions

Sources:

Extending XData to kill SQL query mutants in the wild

XDa-TA : Automating Grading of SQL Query Assignments
Unintended Join Conditions

- Unintended join conditions can be explicitly added by the user in the where clause of the query or by using **natural joins** instead of theta joins.

- Example:
  - Schema:
    - student (id, name, dept name)
    - course (course id, name, dept name)
    - takes (id, course id, sec id, semester, year)
  - Query to find the list of all courses taken by a student with id = 1234 is:
    
    ```
    SELECT course id, course name FROM student
    INNER JOIN takes on(id)
    INNER JOIN course ON(course id) WHERE student.id = 1234
    ```
    
  - Dataset Generated:
    - Student (1234, Alice, EE)
    - course (CS-317, Database Systems, CS)
    - takes (1234, CS-317, 1, Fall, 2014)
Constrained Aggregation Operation

- Aggregation Constraints: Example: \( \text{SUM} (r.a) > 20 \)

- CVC3 requires us to specify how many tuples \( r \) has.

- Hence, before generating CVC3 constraints we must
  (a) estimate the number of tuples \( n \), required to satisfy an aggregation constraint
  (b) translate this number \( n \) to appropriate number of tuples for each base relation so that the input of the aggregation contains exactly \( n \) tuples.
Changed Group By Attributes

• Schema: takes (id, course id, sec id, semester, year, section)

• Example: find the number of students taking each course every time it is offered.

  SELECT count(id), course id, semester, year FROM takes
  GROUP BY course id, semester, year

• Erroneous query misses out students who have taken the same course in different sections.

  SELECT count(id), course id, semester, year FROM takes
  GROUP BY course id, semester, year, section

• Example tuples for dataset:
  t1 (1234, CS-317, 1, Fall, 2014, section 1)
  t2 (1234, CS-317, 1, Fall, 2014, section 2)
Handling NULLs

- For text attributes, enumerate a few more values in the enumerated type and designate them NULLs.
  Example: for an attribute course_id, we enumerate values NULL_course_id_1, NULL_course_id_2, etc.

- For numeric values, we model NULLs as any integer in a range of negative values that we define to be not part of the allowable domain of that numeric value.

- Add constraints forcing those attribute values to take on one of the above mentioned special values representing NULL.

- Add constraints to force all other values to be non null.
String Constraints

- $S_1 \text{ likeop } \text{pattern}$
- $S_1 \text{ relop } \text{constant}$
- $\text{strlen}(S) \text{ relop } \text{constant}$
- $S_1 \text{ relop } S_2$

where $S_1$ and $S_2$ are string variables, likeop is one of LIKE, ILIKE (case insensitive like), NOT LIKE and NOT ILIKE
relop operators are $=, <, \leq, >, \geq, <>$, and case-insensitive equality denoted by $\sim=$.
String Constraints

• String solver

• String constraint mutation: \{=, \textless{}, \textgreater{}, <, >, \leq, \geq\}
  (1) S1 = S2 (2) S1 > S2 (3) S1 < S2

• LIKE predicate mutation: \{LIKE, ILIKE, NOT LIKE, NOT ILIKE\}
  • Dataset 1 satisfying the condition S1 LIKE pattern.
  • Dataset 2 satisfying condition S1 ILIKE pattern, but not S1 LIKE pattern
  • Dataset 3 failing both the LIKE and ILIKE conditions
XDa-TA

• For each query in an assignment, a correct SQL query is given to the tool, which generates datasets for killing mutants of that query.

• Modes:
  i) admin mode
  ii) student mode.

• Assignment can be marked as:
  1. learning assignment
  2. graded assignment.

Source:
XDa-TA : Automating Grading of SQL Query Assignments
Performance Results

- University database schema from Database System Concepts 6th Ed
- Queries with joins, with varying number of foreign keys imposed
## Results for inner join queries

<table>
<thead>
<tr>
<th>Query</th>
<th>#Joins (#Relations)</th>
<th>#FK</th>
<th>#Datasets Generated</th>
<th>#Mutants Killed</th>
<th>Total Time(s) without Unfolding</th>
<th>Total Time(s) with Unfolding</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 (2)</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>0.430</td>
<td>0.040</td>
</tr>
<tr>
<td>1</td>
<td>1 (2)</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0.370</td>
<td>0.030</td>
</tr>
<tr>
<td>2</td>
<td>2 (3)</td>
<td>0</td>
<td>4</td>
<td>6</td>
<td>1.680</td>
<td>0.140</td>
</tr>
<tr>
<td>2</td>
<td>2 (3)</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>1.000</td>
<td>0.100</td>
</tr>
<tr>
<td>2</td>
<td>2 (3)</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>0.990</td>
<td>0.060</td>
</tr>
<tr>
<td>3</td>
<td>3 (4)</td>
<td>0</td>
<td>6</td>
<td>18</td>
<td>3.990</td>
<td>0.229</td>
</tr>
<tr>
<td>3</td>
<td>3 (4)</td>
<td>1</td>
<td>5</td>
<td>13</td>
<td>1.729</td>
<td>0.190</td>
</tr>
<tr>
<td>3</td>
<td>3 (4)</td>
<td>4</td>
<td>3</td>
<td>6</td>
<td>1.230</td>
<td>0.179</td>
</tr>
<tr>
<td>4</td>
<td>4 (5)</td>
<td>0</td>
<td>7</td>
<td>122</td>
<td>7.190</td>
<td>0.279</td>
</tr>
<tr>
<td>4</td>
<td>4 (5)</td>
<td>4</td>
<td>4</td>
<td>62</td>
<td>2.310</td>
<td>0.190</td>
</tr>
<tr>
<td>5</td>
<td>5 (6)</td>
<td>0</td>
<td>9</td>
<td>450</td>
<td>26.800</td>
<td>0.570</td>
</tr>
<tr>
<td>5</td>
<td>5 (6)</td>
<td>4</td>
<td>6</td>
<td>245</td>
<td>2.960</td>
<td>0.380</td>
</tr>
<tr>
<td>6</td>
<td>6 (7)</td>
<td>0</td>
<td>11</td>
<td>1499</td>
<td>68.450</td>
<td>0.790</td>
</tr>
<tr>
<td>6</td>
<td>6 (7)</td>
<td>6</td>
<td>6</td>
<td>507</td>
<td>3.809</td>
<td>0.520</td>
</tr>
</tbody>
</table>

**TABLE I**

**Results for Inner Join Queries**
Results for queries with selections, aggregations

<table>
<thead>
<tr>
<th>Query</th>
<th>#Joins</th>
<th>#Selections</th>
<th>#Aggregations</th>
<th>#Data sets Gen.</th>
<th>#Mutants killed</th>
<th>Total Time(s) without Unfolding</th>
<th>Total Time(s) with Unfolding</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>5</td>
<td>0.12</td>
<td>0.12</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>7</td>
<td>0.08</td>
<td>0.08</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>9</td>
<td>41.40</td>
<td>0.65</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>6</td>
<td>9</td>
<td>5.69</td>
<td>1.23</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>9</td>
<td>18</td>
<td>6.54</td>
<td>1.67</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>5</td>
<td>14</td>
<td>53.95</td>
<td>1.05</td>
</tr>
</tbody>
</table>

**Table II**

Results for Queries with Selection/Aggregation
<table>
<thead>
<tr>
<th>QId</th>
<th>DS</th>
<th>Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q0</td>
<td>5</td>
<td>CREATE VIEW rich_instructors AS SELECT id, name, dept_name, salary FROM instructor WHERE salary &gt; 50000</td>
</tr>
<tr>
<td>Q1</td>
<td>2</td>
<td>SELECT course_id, title FROM course</td>
</tr>
<tr>
<td>Q2</td>
<td>5</td>
<td>SELECT course_id, title FROM course WHERE dept_name = 'Comp. Sci.'</td>
</tr>
<tr>
<td>Q3</td>
<td>9</td>
<td>SELECT DISTINCT course.course_id, course.title, ID FROM course NATURAL JOIN teaches WHERE teaches.semester = 'Spring' AND teaches.year = '2010'</td>
</tr>
<tr>
<td>Q4</td>
<td>6</td>
<td>SELECT DISTINCT student.id, student.name FROM takes NATURAL JOIN student WHERE course_id = 'CS-101'</td>
</tr>
<tr>
<td>Q5</td>
<td>8</td>
<td>SELECT DISTINCT course.dept_name FROM course NATURAL JOIN section WHERE section.semester = 'Spring' AND section.year = '2010'</td>
</tr>
<tr>
<td>Q6</td>
<td>5</td>
<td>SELECT course_id, title FROM course WHERE credits &gt; 3</td>
</tr>
<tr>
<td>Q7</td>
<td>8</td>
<td>SELECT course_id, COUNT(DISTINCT id) FROM course NATURAL LEFT OUTER JOIN takes GROUP BY course_id</td>
</tr>
<tr>
<td>Q8</td>
<td>11</td>
<td>SELECT DISTINCT course_id, title FROM course NATURAL JOIN section WHERE semester = 'Spring' and year = 2010 and course_id NOT IN (SELECT course_id FROM prereq)</td>
</tr>
<tr>
<td>Q9a</td>
<td>25</td>
<td>WITH s as (SELECT id, time_slot_id, year, semester FROM takes NATURAL JOIN section GROUP BY id, time_slot_id, year, semester HAVING count(time_slot_id) &gt; 1) SELECT DISTINCT id, name FROM s NATURAL JOIN student</td>
</tr>
<tr>
<td>Q</td>
<td>#</td>
<td>SQL Statement</td>
</tr>
<tr>
<td>-----</td>
<td>---</td>
<td>------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Q9b</td>
<td>22</td>
<td><code>SELECT distinct A.id, A.name FROM (SELECT * from student NATURAL JOIN takes NATURAL JOIN section) A, (SELECT * from student NATURAL JOIN takes NATURAL JOIN section) B WHERE A.name = B.name and A.time_slot_id = B.time_slot_id and A.course_id &lt;&gt; B.course_id and A.semester = B.semester and A.year = B.year</code></td>
</tr>
<tr>
<td>Q10</td>
<td>7</td>
<td><code>SELECT DISTINCT dept_name FROM course WHERE credits = (SELECT max(credits) FROM course)</code></td>
</tr>
<tr>
<td>Q11</td>
<td>4</td>
<td><code>SELECT DISTINCT instructor.ID,name,course_id FROM instructor LEFT OUTER JOIN TEACHES ON instructor.ID = teaches.ID</code></td>
</tr>
<tr>
<td>Q12</td>
<td>5</td>
<td><code>SELECT student.id, student.name FROM student WHERE lower(student.name) like '%sr%'</code></td>
</tr>
<tr>
<td>Q13</td>
<td>10</td>
<td><code>SELECT id, name FROM student NATURAL LEFT OUTER JOIN (SELECT id, name, course_id FROM student NATURAL LEFT OUTER JOIN takes WHERE year = 2010 and semester = 'Spring') S WHERE course_id IS NULL</code></td>
</tr>
<tr>
<td>Q14</td>
<td>19</td>
<td><code>SELECT DISTINCT * FROM takes T WHERE (NOT EXISTS (SELECT id,course_id FROM takes S WHERE grade ! = 'F' AND T.id=S.id AND T.course_id=S.course_id) and T.grade IS NOT NULL) or (T.grade ! = 'F' AND T.grade IS NOT NULL)</code></td>
</tr>
</tbody>
</table>
## Query grading results

<table>
<thead>
<tr>
<th>QId</th>
<th>Queries</th>
<th>XDa-TA</th>
<th>USm</th>
<th>ULg</th>
<th>TA</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>√</td>
<td>×</td>
<td>√</td>
<td>×</td>
<td>√</td>
</tr>
<tr>
<td>Q0</td>
<td>72</td>
<td>72</td>
<td>0</td>
<td>72</td>
<td>0</td>
<td>72</td>
</tr>
<tr>
<td>Q1</td>
<td>55</td>
<td>53</td>
<td>2</td>
<td>53</td>
<td>2</td>
<td>53</td>
</tr>
<tr>
<td>Q2</td>
<td>57</td>
<td>56</td>
<td>1</td>
<td>56</td>
<td>1</td>
<td>56</td>
</tr>
<tr>
<td>Q3</td>
<td>71</td>
<td>58</td>
<td>13</td>
<td>59</td>
<td>12</td>
<td>59</td>
</tr>
<tr>
<td>Q4</td>
<td>78</td>
<td>52</td>
<td>26</td>
<td>52</td>
<td>26</td>
<td>75</td>
</tr>
<tr>
<td>Q5</td>
<td>72</td>
<td>49</td>
<td>23</td>
<td>61</td>
<td>11</td>
<td>56</td>
</tr>
<tr>
<td>Q6</td>
<td>61</td>
<td>55</td>
<td>6</td>
<td>55</td>
<td>6</td>
<td>59</td>
</tr>
<tr>
<td>Q7</td>
<td>77</td>
<td>52</td>
<td>25</td>
<td>54</td>
<td>23</td>
<td>75</td>
</tr>
<tr>
<td>Q8</td>
<td>79</td>
<td>46</td>
<td>33</td>
<td>67</td>
<td>12</td>
<td>65</td>
</tr>
<tr>
<td>Q9</td>
<td>80</td>
<td>12</td>
<td>68</td>
<td>56</td>
<td>24</td>
<td>10</td>
</tr>
<tr>
<td>Q9a</td>
<td>80</td>
<td>9</td>
<td>71</td>
<td>56</td>
<td>24</td>
<td>10</td>
</tr>
<tr>
<td>Q9b</td>
<td>80</td>
<td>8</td>
<td>72</td>
<td>56</td>
<td>24</td>
<td>10</td>
</tr>
<tr>
<td>Q9</td>
<td>80</td>
<td>8</td>
<td>72</td>
<td>56</td>
<td>24</td>
<td>10</td>
</tr>
<tr>
<td>Q10</td>
<td>74</td>
<td>73</td>
<td>1</td>
<td>73</td>
<td>1</td>
<td>74</td>
</tr>
<tr>
<td>Q11</td>
<td>69</td>
<td>53</td>
<td>16</td>
<td>53</td>
<td>16</td>
<td>53</td>
</tr>
<tr>
<td>Q12</td>
<td>70</td>
<td>62</td>
<td>8</td>
<td>67</td>
<td>3</td>
<td>63</td>
</tr>
<tr>
<td>Q13</td>
<td>72</td>
<td>64</td>
<td>8</td>
<td>63</td>
<td>9</td>
<td>63</td>
</tr>
<tr>
<td>Q14</td>
<td>67</td>
<td>39</td>
<td>28</td>
<td>53</td>
<td>14</td>
<td>57</td>
</tr>
</tbody>
</table>

**Table 2: Query grading results**
Future Work

• Ongoing work
  • Integration with course management systems such as Moodle or Blackboard using the Learning Tools Interoperability (LTI) standard

• Future work:
  • Handling SQL features not supported currently
  • Multiple queries
  • Form parameters
Questions
Thank You