Transforming SQL Queries for Database Testing

Calvin Li and Ben Mishkanian

6/6/15

Motivation – DBMS Testing

- **Random query generation** alone can be used to test for crashing bugs, but not functional correctness.
- **Differential testing** can be used to find functional bugs, but it can not be applied directly, since SQL is not fully standardized across implementations.
- We need the coverage of random testing and the functional validation of differential testing, without having to translate across SQL dialects.
- We focus our tests on SQLite, as it is has not been tested as extensively as other DBMSs.

Hybrid Testing Approach

1. Generate a set Q of random SQL queries
2. Fill a database with randomly generated data.
3. Scan the database to find a set P of invariant predicates (ones that are always true or always false)
4. For each query in Q, generate a semantically equivalent query by appending a valid invariant in P to the WHERE clause.
5. Run both the original query and the transformed query on the DB, checking if the output is identical.

Example

- Database `db1` has table `employees`, containing columns `Name`, `Income`, and `Age`

1. Scanning employees, we find out that the largest income is 100000 and the largest age is 50. We may generate the following invariants:
   - `employees.Income <= 100000`
   - `employees.Age <= 50`
2. Given the randomly generated query

   ```
   SELECT Name FROM employees WHERE id=3;
   ```

   We generate the transformed query

   ```
   SELECT Name FROM employees WHERE id=3 AND employees.Age <= 50;
   ```

Generating Random Queries and Data

- The **Random Query Generator** tool generates queries based on a grammar file given as input:

```
select_rule:
    SELECT distinct_all result_column RC_cont from_rule where_rule group_by_rule ;

distinct_all:
    | DISTINCT | ALL ;

RC_cont:
    | , result_column RC_cont ;
```

snippet of grammar defining SQLite syntax

- It also generates randomized database tables based on a configuration file given as input:

```
$fields = {
        types => [ 'int', 'varchar' ],
        indexes => ['nokey', 'key' ],
        null => [undef ],
        signed => ['signed', 'unsigned']
};

$data = {
        numbers => [ 'digit', 'null', 'mediumint' ],
        strings => [ 'letter', 'english' , 'string(32)']
};
```

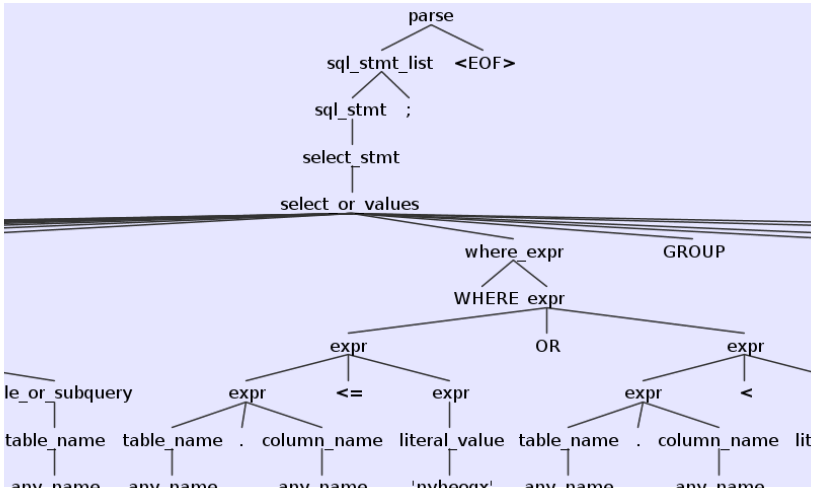
snippet of schema configuration

Finding Database Invariants

- There are many different ways to find predicates that are always true/false
- Possible ideas:
  - Comparison to the min/max of a column
  - Checking if a cell in column X is also in a subquery that returns column X
  - Comparison to the sum of a column

Query Transformation

- We use **ANTLR**, a tool that generates for our SQLite grammar file.
- Antlr generates a parse tree, which we can then traverse to both retrieve information about the query and insert additional tokens.



- We go to the node that covers the table used in the query, using that to generate an appropriate invariant.
- Then we go to the node for `where_expr` and insert our invariant there.

Evaluation

- We ran the program for 2000 queries so far
- Record:
  - original query
  - transformed query
  - Comparison
  - error messages
- Transformations are simple for now: 3 true invariants **`AND`**ed to the **`where`** clause.
- Over 400 (~20%) queries produced different outputs.
- Further investigation will be needed to determine which queries are the result of bugs in SQLite.

Future Work

- Process queries with different outputs
  - Determine if there is a bug within our scripts
  - Reduce query to get minimal test case that reveals bug.
  - See if bug is new
  - Automate this process as much as possible
- Implement more sophisticated ways to transform queries
  - Mix **`AND`** and **`OR`** operands
  - Compare to result of an aggregate subquery
  - Use a membership condition (also involves subqueries)
  - Transform **`JOIN`** clause in addition to **`WHERE`**

The End

Questions?