A Framework for Testing Database Applications

David Chays, Saikat Dan, Phyllis G. Frankl,
Filippos I. Vokolos, Elaine J. Weyuker

Sonia Ng
November 17, 2009

Introduction

- Why is Database (DB) testing important?
  - Central role in operations of modern organizations
  - Need to manage large amounts of data while still:
    - Protecting the integrity of the data
    - Relieving the user from low-level details

- But, relatively little attention has been given to this
- Solution proposed:
  - Develop a systematic, partially-automatable tool to test DB's

Introduction

- Aspects to the correctness of a database system:
  - Does the application program behave as specified?
  - Does the database schema correctly reflect real world data?
  - Are security and privacy protected appropriately?
  - Are the data in the database accurate?
  - Does the DBMS perform all insertions, deletions and updates?
- The paper focuses on the correctness of DB applications
  - Also, restricting attention to relational databases

Introduction

- Outline
  - Background and terminology
  - Issues arising in testing database applications and the approach
  - Overview of the tool
  - Further implementation details
  - Example illustrating the capabilities of the tool
  - Comparison of the approach to other commercial tools
  - Directions for on-going work

Background and Terminology

- Relational databases and SQL
  - Relations often thought of as tables
  - Relation schema = relation name and attributes (columns)
    - In other words, the structure of the table
  - Attributes = Has a name (A_i) and a domain/type (dom(A_i))
  - Domains = must be atomic types. Not complex types
  - Relation/relation state = a set of tuples at a specific time
    - Each set of tuples is an element of the Cartesian product dom(A_1)X… dom(A_n)
  - Relational database schema = set of relation schemas with a set of integrity constraints

Background and Terminology

- Types of constraints:
  - Domain constraints
  - Uniqueness constraints
  - Not-NULL constraints
  - Referential integrity constraints (foreign key constraints)
  - Semantic integrity constraints
- SQL
  - Language used to define and manipulate relational databases
  - Semi-declarative language
    - Expressing what should be done rather than how

Background and Terminology

- Figure 1: A database schema definition in SQL

```
create table s (sno char(5), sname char(20), status decimal(3),
       city char(15), primary key(sno));
create table p (pno char(6) primary key, pname char(20), color char(6),
       weight decimal(3), city char(15));
create table sp (sno char(5), pno char(6), qty decimal(5),
       primary key(sno,pno),
       foreign key(sno) references s, foreign key(pno) references p);
```

Issues in Testing DB Applications

- We will use these specifications in order to outline issues:
  - Input:
    - customer's ID, name of telephone feature
  - Return:
    - 0 = ID number or feature name is invalid
    - 1 = customer's location and feature compatibility approved. Feature added, billing table updated, sent out notice
    - 2 = customer lives in area where feature is not available
    - 3 = Feature is available in the area but is incompatible with subscribed features

Issues in Testing DB Applications

- Role of DB state
  - It's not just about input and output. The state of the DB must be considered.
- Approaches to deal with DB state:
  - 1. Ignore it
  - 2. Consider DB state as an aspect of the environment
  - 3. Treat it as part of the input/output spaces

Issues in Testing DB Applications

- Problems:
  - Controllability
  - Observability
- Eg: Adding a new feature gives rise to several test cases:
  - Feature already subscribed
  - Not available in the area
  - Not compatible with already subscribed features
  - Available in the area and compatible
  - Customer has no features at all

Issues in Testing DB Applications

- Populating the DB
  - Live data
  - Synthetic data
- Synthetic:
  - Issues with data population. Must produce valid and interesting data

Design of the Tool

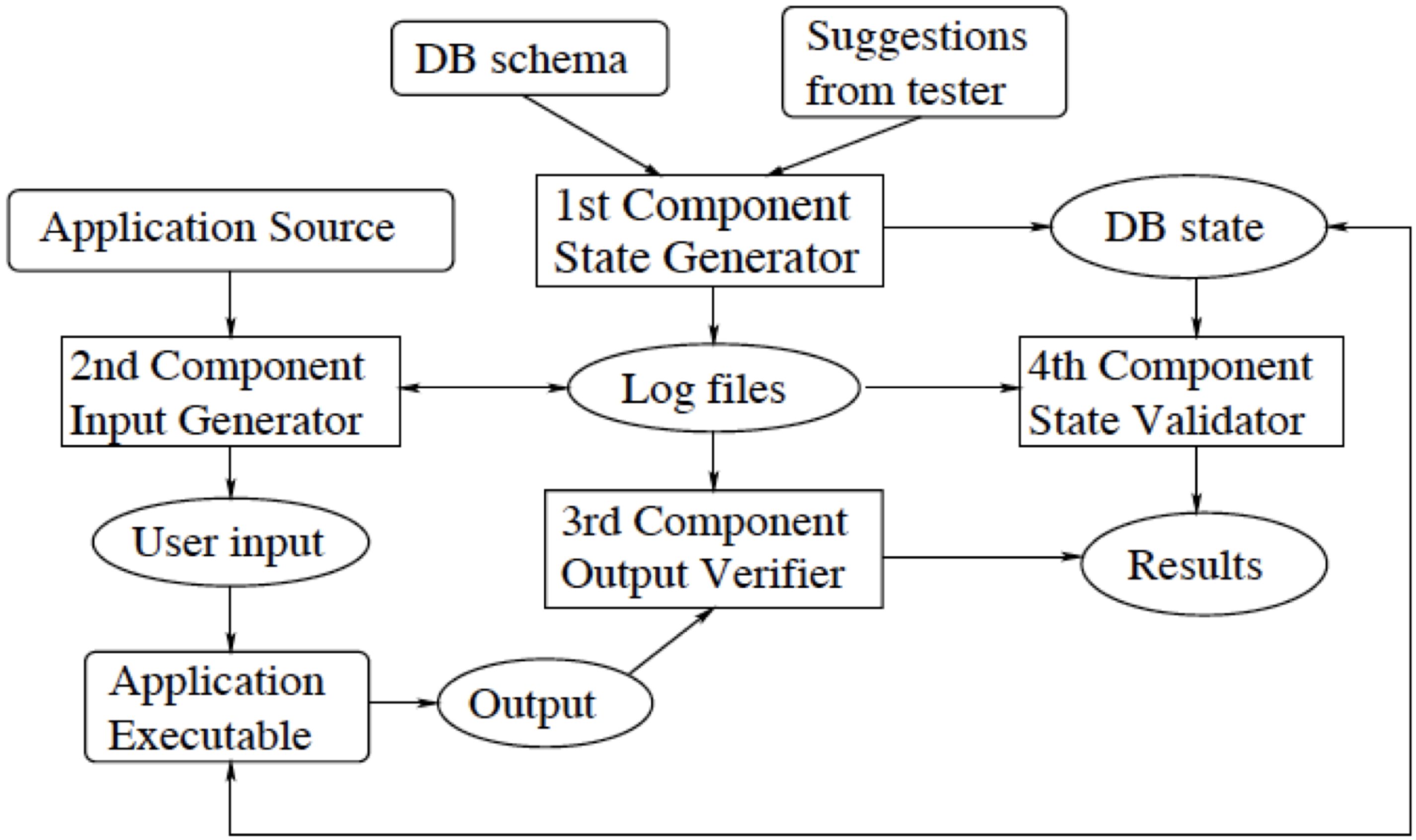


Figure 2: A framework for the tool set architecture

Design of the Tool

```
−−choice_name: low
10
20
30
− − −−
−−choice_name: medium
300
400
− − −−
−−choice_name: high
5000
6000
```

Figure 3: Input file for qty attribute of table sp

Design of the Tool

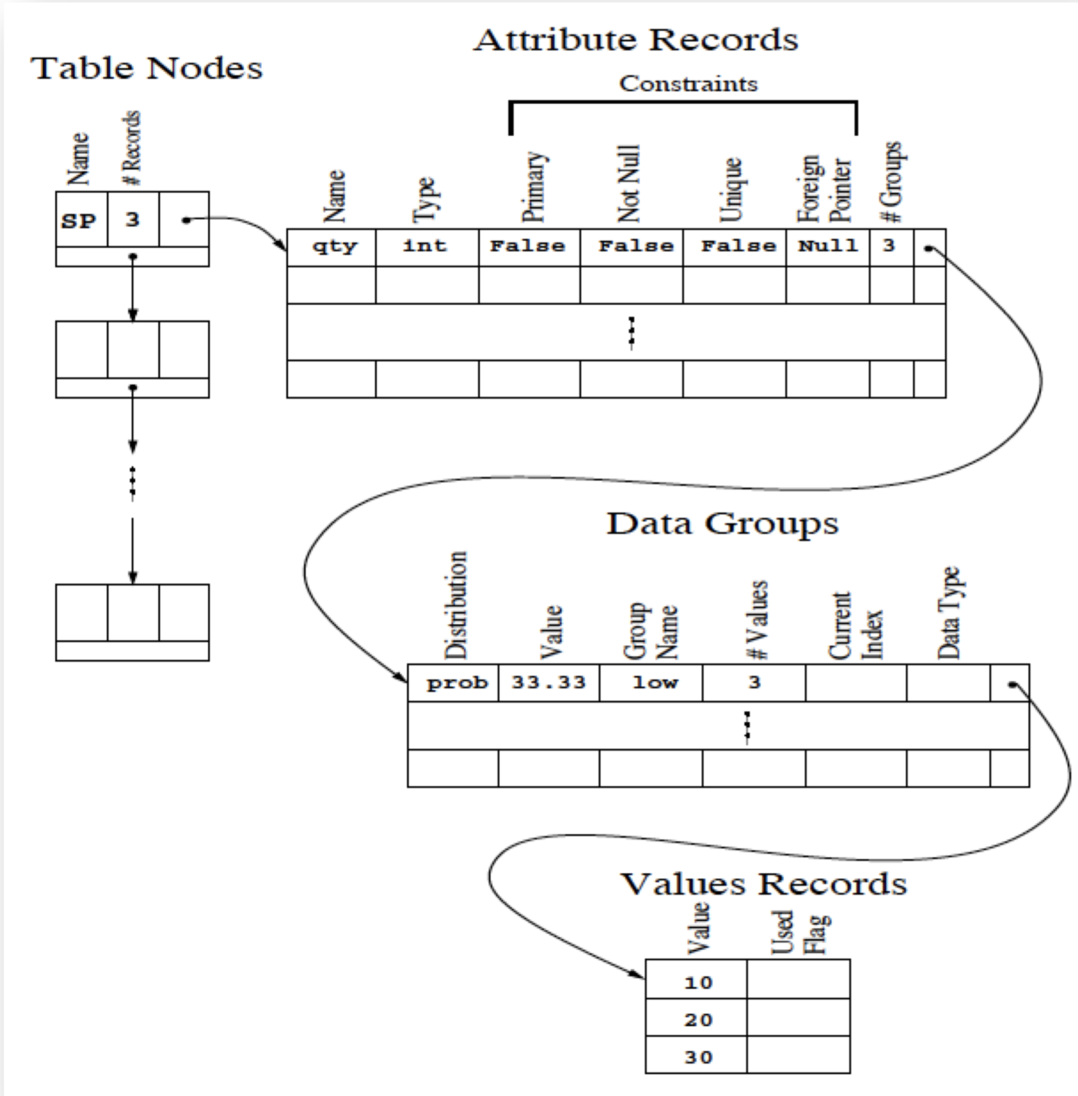
```
insert into s values ('S1',NULL,0,'Brooklyn');
insert into s values ('S2','Smith',1,'Florham-Park');
insert into s values ('S3','Jones',NULL,'Athens');
insert into s values ('S4','Blake',NULL,'Middletown');
insert into p values ('P1',NULL,'blue',100,'Brooklyn');
insert into p values ('P2','seats','green',300,'Florham-Park');
insert into p values ('P3','airbags','yellow',500,'Middletown');
insert into sp values ('S1','P1',5000);
insert into sp values ('S1','P2',300);
insert into sp values ('S1','P3',10);
insert into sp values ('S2','P1',6000);
insert into sp values ('S2','P2',400);
insert into sp values ('S2','P3',5000);
insert into sp values ('S3','P1',20);
insert into sp values ('S3','P2',300);
insert into sp values ('S3','P3',30);
insert into sp values ('S4','P1',6000);
```

Figure 4: Sample output produced by the tool

Implementation of the Tool

- Base the tool on PostgreSQL
- PostgreSQL parser can create a parse tree with all relevant information
  - Might be inconvenient/inefficient during test generation
  - Location in tree depends on exact syntax of schema definition
- So, designed a data structure that brings all the associated information into one place
  - Modified parser so that it builds the data structure as it parses the schema definition

Implementation of the Tool



Implementation of the Tool

- After parsing schema, user is queried for input files
- Annotations in input files:
  - choice_name
  - choice_prob
  - choice_freq
  - null_prob
  - null_freq
- For each attribute, an array "data groups" is dynamically created to show annotations
  - This contains a pointer to array "values" that stores actual data values

Implementation of the Tool

- For tables with constraints made up of multiple attributes:
  - Look at the combination rather than individual value
  - Array called "composite attribute records" is used
- Assessing size limits. Factors:
  - Number of attributes
  - Attribute sizes
  - Number of composite constraints
  - Amount of memory

Example

```
Enter filename for pno or ENTER if same as the column name:
Enter filename for pname or ENTER if same as the column name:
auto
Enter filename for color or ENTER if same as the column name:
Enter filename for weight or ENTER if same as the column name:
Enter filename for city or ENTER if same as the column name:
How many records for table p? 3
```

Figure 7: Sample excerpts from a session

Example

table s

sno	sname	status	city
S1	NULL	0	Brooklyn
S2	Smith	1	Florham-Park
S3	Jones	NULL	Athens
S4	Blake	NULL	Middletown

table sp

sno	pno	qty
S1	P1	5000
S1	P2	300
S1	P3	10
S2	P1	6000
S2	P2	400
S2	P3	5000
S3	P1	20
S3	P2	300
S3	P3	30
S4	P1	6000

table p

pno	pname	color	weight	city
P1	NULL	blue	100	Brooklyn
P2	seats	green	300	Florham-Park
P3	airbags	yellow	500	Middletown

Figure 8: A database state produced by the tool

Related Work

- With the exception of a paper by Davies, Beynon, and Jones, there has not been an approach specifically targeted towards DB testing
- This technique = related to specification-based test generation
  - Using category-partition technique

Conclusions and Future Work

- Focused on: "populating a database with meaning data that satisfies constraints"
- Identified issues that make testing DB applications different from other software systems
- Described the tool/approach with examples
- Determined size limitations
- Extend work by:
  - Handle domain constraints and semantic constraints
  - Handle constraints that are *not* part of the schema
  - Including "boundary values" or other "special values" that are more fault-prone