Dynamic Invariant Detection for Relational Databases

Jake Cobb\textsuperscript{1}, Gregory M. Kapfhammer\textsuperscript{2},
James A. Jones\textsuperscript{3}, Mary Jean Harrold\textsuperscript{4}

\textsuperscript{1}Georgia Institute of Technology
\textsuperscript{2}Allegheny College
\textsuperscript{3}University of California, Irvine

WODA 2011 – July 18, 2011
Outline

Background
  Dynamic Invariants
  Relational Databases

Database Invariants
  Mapping
  Implementation

Results
  Subjects
  Invariant Quality
  Schema Modification
Dynamic Invariants

Definition
A dynamic invariant is a property that is observed to hold during a series of executions.

- Not guaranteed for all possible executions.
- May reflect property of:
  - Program
  - Inputs
Daikon [Ernst et al. 2001] is a dynamic invariant detection engine.

- Collect data traces for variables at *program points*.
- Compare to pool of potential invariants.
- Output remaining invariants that meet confidence threshold.
Daikon [Ernst et al. 2001] is a dynamic invariant detection engine.

- Collect data traces for variables at *program points*.
- Compare to pool of potential invariants.
- Output remaining invariants that meet confidence threshold.
**Daikon** [Ernst et al. 2001] is a dynamic invariant detection engine.

- Collect data traces for variables at *program points*.
- Compare to pool of potential invariants.
- Output remaining invariants that meet confidence threshold.

![Diagram of Daikon process]

Program \(\xrightarrow{\text{Instrument}}\) Program' \(\xrightarrow{\text{Execute}}\) Trace File
Daikon \cite{Ernst et al. 2001} is a dynamic invariant detection engine.

- Collect data traces for variables at \textit{program points}.
- Compare to pool of potential invariants.
- Output remaining invariants that meet confidence threshold.
Many applications of dynamic invariants in software engineering:

- Programmer understanding
- Run-time checking
- Integration testing
- Interface discovery
- Test-input generation
- ...
Relational Databases

Relational Model

<table>
<thead>
<tr>
<th>TableA</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ColumnA</td>
<td>ColumnB</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>'Data'</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>'Values'</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TableB</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ColumnC</td>
<td>ColumnD</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
SQL (Structured Query Language) is a standard and query language for relational database management systems (RDBMS).
SQL (Structured Query Language) is a standard and query language for relational database management systems (RDBMS).

Data Definition
A schema is a collection of table definitions.

```
CREATE TABLE person (  
id    INT,  
name   VARCHAR(100) NOT NULL,  
age    INT(3),  
PRIMARY KEY (id)
)
```
SQL (Structured Query Language) is a standard and query language for relational database management systems (RDBMS).

Data Definition
A schema is a collection of table definitions.

```
CREATE TABLE person (
    id INT,
    name VARCHAR(100) NOT NULL,
    age INT(3),
    PRIMARY KEY (id)
)
```

Create, Read, Update and Delete (CRUD) Operations

```
INSERT INTO person (id, name, age) VALUES (1, 'John', 38)
SELECT name FROM person WHERE age >= 30 AND age <= 40
UPDATE person SET name = 'Jan' WHERE id = 2
DELETE FROM person WHERE id = 2
```
Outline

Background
  Dynamic Invariants
  Relational Databases

Database Invariants
  Mapping
  Implementation

Results
  Subjects
  Invariant Quality
  Schema Modification
## Structural Mapping

<table>
<thead>
<tr>
<th>Program Element</th>
<th>DB Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Point</td>
<td>Table</td>
</tr>
<tr>
<td>Variable</td>
<td>Column</td>
</tr>
<tr>
<td>Occurrence</td>
<td>Row</td>
</tr>
</tbody>
</table>

Detect invariants for:
- Individual columns.
- Between columns in a given row.

Example:
```
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>age</th>
<th>employed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'John Smith'</td>
<td>38</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>'Jan Downing'</td>
<td>22</td>
<td>2</td>
</tr>
</tbody>
</table>
```
Detect invariants for:

- Individual columns.
- Between columns in a given row.
Structural Mapping

<table>
<thead>
<tr>
<th>Program Element</th>
<th>DB Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Point</td>
<td>Table</td>
</tr>
<tr>
<td>Variable</td>
<td>Column</td>
</tr>
<tr>
<td>Occurrence</td>
<td>Row</td>
</tr>
</tbody>
</table>

Detect invariants for:

- Individual columns.
- Between columns in a given row.

Example

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>age</th>
<th>employed</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>'John Smith'</td>
<td>38</td>
<td>5</td>
<td>...</td>
</tr>
<tr>
<td>2</td>
<td>'Jan Downing'</td>
<td>22</td>
<td>2</td>
<td>...</td>
</tr>
</tbody>
</table>
Data Mapping

Daikon Concepts

- Representation type
  - int
  - double
  - String
  - int[]

- Comparability
<table>
<thead>
<tr>
<th>Group</th>
<th>Name</th>
<th>SQL Types</th>
<th>Java Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Text</td>
<td>CHAR, VARCHAR, TEXT</td>
<td>String</td>
</tr>
<tr>
<td>2</td>
<td>Integer</td>
<td>INTEGER, NUMERIC, BIT</td>
<td>int</td>
</tr>
<tr>
<td>3</td>
<td>Decimal</td>
<td>FLOAT, DOUBLE, REAL, DECIMAL</td>
<td>double</td>
</tr>
<tr>
<td>4</td>
<td>Binary</td>
<td>BLOB, BIT</td>
<td>byte[]</td>
</tr>
<tr>
<td>5</td>
<td>Text Set</td>
<td>SET</td>
<td>String[]</td>
</tr>
<tr>
<td>6</td>
<td>Datetime</td>
<td>DATETIME, TIMESTAMP</td>
<td>String</td>
</tr>
<tr>
<td>7</td>
<td>Date</td>
<td>DATE</td>
<td>String</td>
</tr>
<tr>
<td>8</td>
<td>Time</td>
<td>TIME</td>
<td>String</td>
</tr>
<tr>
<td>9</td>
<td>Interval</td>
<td>INTERVAL</td>
<td>int</td>
</tr>
<tr>
<td>10</td>
<td>Primary Key</td>
<td>INTEGER</td>
<td>reference</td>
</tr>
</tbody>
</table>
NULL Values

- NULL is a possible value for any SQL type.
- Daikon does not accept null for primitive representation types, e.g. int.
NULL Values

- NULL is a possible value for any SQL type.
- Daikon does not accept null for primitive representation types, e.g. int.
- Introduce synthetic variable for each NULL-able column.
  - Representation type is hashcode (reference).
  - Value is either null or a constant.
Process Overview

- **Scan DB State**
- **Read Schema**
- **Infer Invariants**
- **Instrumentation Wrapper**
- **Collect Trace**
- **Infer Invariants**

**Application-independent, Fixed Data**
- Schema, Type Metadata
- All Rows, Columns

**Application-specific, Dynamic Data**
- Schema, Type Metadata
- Inserted, Updated Rows

**DBMS**

**Data Trace**

- **Schema**
Implementation

Trace Collector

- Python\(^1\) program:
  - Input: DB connection information.
  - Output: Daikon declarations and data trace files.
- Process:
  1. Read schema metadata to determine tables, columns and data mapping.
  2. Write declarations file and serialize mapping info for reuse.
  3. SELECT table contents, transform data by mapping, write to GZip’d trace file.
- Supports various RDBMS via SQLAlchemy.

\(^1\)... plus a tiny bit of Cython
Implementation

Instrumentation Wrapper

- Modified P6Spy JDBC driver wrapper.
- On connection, capture information and initiate initial metadata read and trace.
- On statement execution, append trace if data could be modified.
  - INSERT statement.
  - UPDATE statement.
  - Unknown (e.g. a stored procedure call.)
  - Ignore others, including DELETE and TRUNCATE.
Outline

Background
  Dynamic Invariants
  Relational Databases

Database Invariants
  Mapping
  Implementation

Results
  Subjects
  Invariant Quality
  Schema Modification
# Subjects

## Fixed Data Sets

<table>
<thead>
<tr>
<th>Subject</th>
<th>Tables</th>
<th>Columns</th>
<th>Rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>world</td>
<td>3</td>
<td>24</td>
<td>5302</td>
</tr>
<tr>
<td>sakila</td>
<td>23</td>
<td>131</td>
<td>50,086</td>
</tr>
<tr>
<td>menagerie</td>
<td>2</td>
<td>10</td>
<td>19</td>
</tr>
<tr>
<td>employees</td>
<td>6</td>
<td>24</td>
<td>3,919,015</td>
</tr>
</tbody>
</table>

- MySQL sample databases for training, certification and testing.
- Trace entire dataset.
## Database Applications

<table>
<thead>
<tr>
<th>Program</th>
<th>iTrust</th>
<th>JWhoisServer</th>
<th>JTrac</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables</td>
<td>30</td>
<td>7</td>
<td>13</td>
</tr>
<tr>
<td>Columns</td>
<td>177</td>
<td>57</td>
<td>126</td>
</tr>
<tr>
<td>KLOC</td>
<td>25.5 (Java), 8.6 (JSP)</td>
<td>6.7</td>
<td>12</td>
</tr>
<tr>
<td>Test Cases</td>
<td>787</td>
<td>67</td>
<td>41</td>
</tr>
</tbody>
</table>

- Java applications driven by a database.
- Wrap real DB driver in a modified P6Spy driver.
- Execute the test suite.
Invariant Quality

Meaningful Invariants
Invariants that capture a semantic relationship.
Invariant Quality

Meaningful Invariants
Invariants that capture a semantic relationship.

- \( \text{dept_emp}.\text{from}\_\text{date} \leq \text{dept_emp}.\text{to}\_\text{date} \)
- \( \text{employees}.\text{gender} \) one of \{ "F", "M" \}
- \( \text{employees}.\text{birth}\_\text{date} < \text{employees}.\text{hire}\_\text{date} \)
- \( \text{country}.\text{Population} \geq 0 \)
- \( \text{icdcodes}.\text{Chronic} \) one of \{ "no", "yes" \}
Spurious Invariants

▶ Vacuous invariants reflect a meaningless relationship.

▶ Lack-of-data invariants result from limited data samples.
Spurious Invariants

Vacuous invariants reflect a meaningless relationship.
- patients.phone1 <= patients.BloodType
- patients.lastName >= patients.address1
- cptcodes.Description != cptcodes.Attribute

Lack-of-data invariants result from limited data samples.
Spurious Invariants

- **Vacuous** invariants reflect a meaningless relationship.
  - patients.phone1 <= patients.BloodType
  - patients.lastName >= patients.address1
  - cptcodes.Description != cptcodes.Attribute

- **Lack-of-data** invariants result from limited data samples.
  - mntnr.login == "mntnt"
  - inetnum.changed == "2006-10-14 16:21:09"
  - person.name one of { "no name company", "persona non grata"}
Invariant Quality

Results

Number of Invariants

Type of Invariant

Vacuous
Lack−of−data
Meaningful

Number of Invariants
Schema Modification

- Some invariants can be enforced by the schema definition.
- Schema enforcement provides a stronger assurance of data integrity than application enforcement.
- Analyze enforceable invariants:
  - Already enforced by the schema.
  - Suggest modification to enforce the invariant.
### Schema Enforced

<table>
<thead>
<tr>
<th><strong>Invariant</strong></th>
<th><strong>Schema Definition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>employees.gender one of { &quot;F&quot;, &quot;M&quot; }</td>
<td>ENUM(’F’, ’M’)</td>
</tr>
<tr>
<td>countrylanguage.IsOfficial one of { &quot;F&quot;, &quot;T&quot; }</td>
<td>ENUM(’F’, ’T’)</td>
</tr>
<tr>
<td>customer.active one of { 0, 1 }</td>
<td>TINYINT(1)</td>
</tr>
<tr>
<td>inventory.film_id &gt;= 1</td>
<td>SMALLINT(5) UNSIGNED</td>
</tr>
<tr>
<td>spaces.guest_allowed one of { 0, 1 }</td>
<td>BIT(1)</td>
</tr>
</tbody>
</table>
## Schema Enforceable

<table>
<thead>
<tr>
<th>Invariant</th>
<th>Schema</th>
<th>Modification</th>
</tr>
</thead>
<tbody>
<tr>
<td>isnull(message.message) != null</td>
<td>TEXT</td>
<td>NOT NULL</td>
</tr>
<tr>
<td>isnull(film_text.description) != null</td>
<td>TEXT</td>
<td>NOT NULL</td>
</tr>
<tr>
<td>isnull(history.time_stamp) != null</td>
<td>DATETIME</td>
<td>NOT NULL</td>
</tr>
<tr>
<td>user_space_roles.user_id &gt;= 1</td>
<td>BIGINT(20)</td>
<td>UNSIGNED</td>
</tr>
<tr>
<td>pet.sex one of { &quot;f&quot;, &quot;m&quot; }</td>
<td>CHAR(1)</td>
<td>ENUM(’m’,’f’)</td>
</tr>
<tr>
<td>country.Population &gt;= 0</td>
<td>INT(11)</td>
<td>UNSIGNED</td>
</tr>
<tr>
<td>isnull(titles.to_date) != null</td>
<td>DATE</td>
<td>NOT NULL</td>
</tr>
</tbody>
</table>
Schema Modification

Results

Type of Meaningful Invariant

- Enforceable with Standards-Compliant Database
- Enforceable with Current Database
- Already Enforced

Percentage of Meaningful Invariants

- sakila
- world
- iTrust
- employees
- JTrac
- menagerie
- JWhoisServer

Percentage of Meaningful Invariants
Conclusions and Future Work

Conclusions

- Meaningful invariants may be mined from databases and database applications.
- Invariant quality depends on diverse data.
- Data integrity may be enhanced by using invariants for schema modification.

Future Work

- Invariants between multiple tables.
- Invariants for individual queries.
- Explore additional client applications.
Conclusions and Future Work

Conclusions

▷ Meaningful invariants may be mined from databases and database applications.
▷ Invariant quality depends on diverse data.
▷ Data integrity may be enhanced by using invariants for schema modification.

Future Work

▷ Invariants between multiple tables.
▷ Invariants for individual queries.
▷ Explore additional client applications.
Dynamic Invariant Detection for Relational Databases

Thank you for your time and attention.

Questions?