Virtual Mutation Analysis of Relational Database Schemas

Phil McMinn  University of Sheffield
Gregory M. Kapfhammer  Allegheny College
Chris J. Wright  University of Sheffield
Relational Databases – Why Should We (Still) Care?

A vital component of many software systems

Despite the wave of interest in “NoSQL” technologies, Relational Databases are still popular (and faster)

For developers: schemas provide self-documentation
Relational Databases – Why Should We (Still) Care?

A vital component of many software systems

Despite the wave of interest in “NoSQL” technologies, Relational Databases are still popular (and faster)

For developers: schemas provide self-documentation

Relational Databases are still important, popular and relevant
CREATE TABLE Station (  
    ID INTEGER PRIMARY KEY,  
    CITY CHAR(20),  
    STATE CHAR(2),  
    LAT_N INTEGER NOT NULL  
        CHECK (LAT_N BETWEEN 0 and 90),  
    LONG_W INTEGER NOT NULL  
        CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);  

CREATE TABLE Stats (  
    ID INTEGER REFERENCES STATION(ID),  
    MONTH INTEGER NOT NULL  
        CHECK (MONTH BETWEEN 1 AND 12),  
    TEMP_F INTEGER NOT NULL  
        CHECK (TEMP_F BETWEEN 80 AND 150),  
    RAIN_I INTEGER NOT NULL  
        CHECK (RAIN_I BETWEEN 0 AND 100),  
    PRIMARY KEY (ID, MONTH)  
);
A Relational Database Schema

CREATE TABLE Station (  
    ID INTEGER PRIMARY KEY,  
    CITY CHAR(20),  
    STATE CHAR(2),  
    LAT_N INTEGER NOT NULL  
      CHECK (LAT_N BETWEEN 0 and 90),  
    LONG_W INTEGER NOT NULL  
      CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);  

CREATE TABLE Stats (  
    ID INTEGER REFERENCES STATION(ID),  
    MONTH INTEGER NOT NULL  
      CHECK (MONTH BETWEEN 1 AND 12),  
    TEMP_F INTEGER NOT NULL  
      CHECK (TEMP_F BETWEEN 80 AND 150),  
    RAIN_I INTEGER NOT NULL  
      CHECK (RAIN_I BETWEEN 0 AND 100),  
    PRIMARY KEY (ID, MONTH)  
);
A Relational Database Schema

CREATE TABLE Station (  
    ID INTEGER PRIMARY KEY,  
    CITY CHAR(20),  
    STATE CHAR(2),  
    LAT_N INTEGER NOT NULL  
        CHECK (LAT_N BETWEEN 0 and 90),  
    LONG_W INTEGER NOT NULL  
        CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);  

CREATE TABLE Stats (  
    ID INTEGER REFERENCES Station(ID),  
    MONTH INTEGER NOT NULL  
        CHECK (MONTH BETWEEN 1 AND 12),  
    TEMP_F INTEGER NOT NULL  
        CHECK (TEMP_F BETWEEN 80 AND 150),  
    RAIN_I INTEGER NOT NULL  
        CHECK (RAIN_I BETWEEN 0 AND 100),  
    PRIMARY KEY (ID, MONTH)  
);
Integrity Constraints

CREATE TABLE Station (  
    ID INTEGER PRIMARY KEY,  
    CITY CHAR(20),  
    STATE CHAR(2),  
    LAT_N INTEGER NOT NULL  
    CHECK (LAT_N BETWEEN 0 and 90),  
    LONG_W INTEGER NOT NULL  
    CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);  

CREATE TABLE Stats (  
    ID INTEGER REFERENCES STATION(ID),  
    MONTH INTEGER NOT NULL  
    CHECK (MONTH BETWEEN 1 AND 12),  
    TEMP_F INTEGER NOT NULL  
    CHECK (TEMP_F BETWEEN 80 AND 150),  
    RAIN_I INTEGER NOT NULL  
    CHECK (RAIN_I BETWEEN 0 AND 100),  
    PRIMARY KEY (ID, MONTH)  
);  

Prevent invalid data being entered into the database

Encode domain logic
Integrity Constraints

```
CREATE TABLE Station (  
    ID INTEGER PRIMARY KEY,  
    CITY CHAR(20),  
    STATE CHAR(2),  
    LAT_N INTEGER NOT NULL  
        CHECK (LAT_N BETWEEN 0 and 90),  
    LONG_W INTEGER NOT NULL  
        CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);

CREATE TABLE Stats (  
    ID INTEGER REFERENCES STATION(ID),  
    MONTH INTEGER NOT NULL  
        CHECK (MONTH BETWEEN 1 AND 12),  
    TEMP_F INTEGER NOT NULL  
        CHECK (TEMP_F BETWEEN 80 AND 150),  
    RAIN_I INTEGER NOT NULL  
        CHECK (RAIN_I BETWEEN 0 AND 100)  
    PRIMARY KEY (ID, MONTH)  
);
```

Prevent invalid data being entered into the database

Encode domain logic
Testing the Schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W) VALUES(1, 'Austin', 'TX', 30, 98);
Correctly accepted by the schema

INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W) VALUES(1, 'Austin', 'TX', NULL, 98);
Correctly rejected by the schema

INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W) VALUES(1, 'Austin', 'TX', 91, 98);
Correctly rejected by the schema
```
Testing the Schema

```sql
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W) VALUES(1, 'Austin', 'TX', 30, 98);
Correctly accepted by the schema

INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W) VALUES(1, 'Austin', 'TX', NULL, 98);
Correctly rejected by the schema

INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W) VALUES(1, 'Austin', 'TX', 91, 98);
Correctly rejected by the schema
```
Testing the Schema

```sql
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W) VALUES(1, 'Austin', 'TX', 30, 98);
```

Correctly accepted by the schema

```sql
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W) VALUES(1, 'Austin', 'TX', NULL, 98);
```

Correctly rejected by the schema

```sql
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W) VALUES(1, 'Austin', 'TX', 91, 98);
```

Correctly rejected by the schema
Testing the Schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES(1, 'Austin', 'TX', 30, 98);
```
Correctly **accepted** by the schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES(1, 'Austin', 'TX', NULL, 98);
```
Correctly **rejected** by the schema

```
INSERT INTO Station(ID, CITY, STATE, LAT_N, LONG_W)
VALUES(1, 'Austin', 'TX', 91, 98);
```
Correctly **rejected** by the schema
Why Do We Need to Do This?
Why Do We Need to Do This?

To trap common errors when designing a schema

For example: lack of uniqueness property on usernames, out of range values
Why Do We Need to Do This?

To trap common errors when designing a schema
For example: lack of uniqueness property on usernames, out of range values

To test development behaviour vs deployment
DBMSs have subtly different behaviors
Why Do We Need to Do This?

To trap common errors when designing a schema
For example: lack of uniqueness property on usernames, out of range values

To test development behaviour vs deployment
DBMSs have subtly different behaviors

Nobody throws away a database of data
To test the success of database migrations
Why Do We Need to Do This?

To trap common errors when designing a schema
For example: lack of uniqueness property on usernames, out of range values

To test development behaviour vs deployment
DBMSs have subtly different behaviors

Nobody throws away a database of data
To test the success of database migrations

Industry advice
Destroying database consistency can have huge cost implications
Mutation Analysis

Once a test suite has been created, its fault finding capability can be estimated with mutation analysis.

For relational database schema testing, mutants are created by making small changes to the schema.

```sql
CREATE TABLE Station (  
    ID INTEGER PRIMARY KEY,  
    CITY CHAR(20),  
    STATE CHAR(2),  
    LAT_N INTEGER NOT NULL  
    CHECK (LAT_N BETWEEN 0 and 90),  
    LONG_W INTEGER NOT NULL  
    CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
```
Mutation Analysis

Once a test suite has been created, its fault finding capability can be estimated with mutation analysis.

For relational database schema testing, mutants are created by making small changes to the schema.
Mutation Analysis

Once a test suite has been created, its fault finding capability can be estimated with **mutation analysis**.

For relational database schema testing, mutants are created by making small changes to the schema.
Mutation Analysis

Once a test suite has been created, its fault finding capability can be estimated with *mutation analysis*.

For relational database schema testing, mutants are created by making small changes to the schema:

```sql
CREATE TABLE Station (
    ID INTEGER PRIMARY KEY,
    CITY CHAR(20) UNIQUE,
    STATE CHAR(2),
    LAT_N INTEGER NOT NULL
        CHECK (LAT_N BETWEEN 0 and 90),
    LONG_W INTEGER NOT NULL
        CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)
);
```
Mutation Analysis is Costly

schema → mutants
Mutation Analysis is Costly

- schema → mutants
- mutant + test suite → database
- Mutant killed / alive

FAIL PASS
Mutation Analysis is Costly

SchemaAnalyst

- schema -> mutants
- mutant + test suite -> database
- Mutant killed / alive

FAIL
PASS
Mutation Analysis is Costly
Mutation Analysis is Costly

DO FEWER
Mutation Analysis is Costly

DO FEWER

DO SMARTER
Mutation Analysis is Costly

DO FEWER

DO SMARTER

DO FASTER
Mutation Analysis is Costly

SchemaAnalyst

- schema ➔ mutants

- mutant
  + test suite

- Mutant killed / alive

- database
Mutation Analysis is Costly

SchemaAnalyst

- schema
- mutants
- mutant
- test suite
- database

Mutant killed / alive

+
Mutation Analysis is Costly

- Schema 
- Mutants

High cost of communicating with the DBMS and executing SQL queries on it

- SchemaAnalyst
- Mutant killed / alive

- Database
  - FAIL
  - PASS
Reducing the Cost

SchemaAnalyst

- schema → mutants
- mutant + test suite → database
- Mutant killed / alive

+
Reducing the Cost

SchemaAnalyst

Local, no communication overhead

mutant

Mutant killed / alive

database
Reducing the Cost

SchemaAnalyst

- Schema -> Mutants
- Mutant + Test Suite
  - Mutant killed / alive
  - Model of Database
Reducing the Cost

SchemaAnalyst

Virtual Mutation Analysis

Model of database

Mutant killed / alive

Test suite

Mutant

Schema

 mutants
Reducing the Cost

Virtual Mutation Analysis

SchemaAnalyst

Lower execution overhead

Mutant killed / alive
CREATE TABLE Station (  
    ID INTEGER PRIMARY KEY,  
    CITY CHAR(20),  
    STATE CHAR(2),  
    LAT_N INTEGER NOT NULL  
      CHECK (LAT_N BETWEEN 0 and 90),  
    LONG_W INTEGER NOT NULL  
      CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);
CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,  
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL  
    CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL  
    CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)
);

Integrity constraint predicate $icp_1$  
$(nr(ID) \neq \text{NULL}) \land (\forall er \in \text{Station} : nr(ID) \neq er(ID))$
CREATE TABLE Station (  ID INTEGER PRIMARY KEY,  CITY CHAR(20),  STATE CHAR(2),  LAT_N INTEGER NOT NULL CHECK (LAT_N BETWEEN 0 and 90),  LONG_W INTEGER NOT NULL CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180) );

Integrity constraint predicate \textit{icp1}

\[(nr(ID) \neq \text{NULL}) \land (\forall r \in \text{Station}: nr(ID) \neq er(ID))\]

\textit{icp2}

\[(nr(LAT\_N) \neq \text{NULL})\]

\textit{icp3}

\[(nr(LAT\_N) \geq 0 \land nr(LAT\_N) \leq 90)\]
The Model

CREATE TABLE Station (  
    ID INTEGER PRIMARY KEY,  
    CITY CHAR(20),  
    STATE CHAR(2),  
    LAT_N INTEGER NOT NULL  
    CHECK (LAT_N BETWEEN 0 and 90),  
    LONG_W INTEGER NOT NULL  
    CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);

Integrity constraint predicate icp1
\[(nr(ID) \neq \text{NULL}) \land (\forall er \in \text{Station}: nr(ID) \neq er(ID))\]

Integrity constraint predicate icp2
\[(nr(LAT_N) \neq \text{NULL})\]

Integrity constraint predicate icp3
\[(nr(LAT_N) \geq 0 \land nr(LAT_N) \leq 90)\]

Integrity constraint predicate icp4

Integrity constraint predicate icp5
The Model

CREATE TABLE Station (  
  ID INTEGER PRIMARY KEY,  
  CITY CHAR(20),  
  STATE CHAR(2),  
  LAT_N INTEGER NOT NULL CHECK (LAT_N BETWEEN 0 and 90),  
  LONG_W INTEGER NOT NULL CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);

Form an acceptance predicate for the table:

\[ ap = icp1 \land icp2 \land icp3 \land icp4 \land icp5 \]

Integrity constraint predicate \( icp1 \)

\[ (nr(ID) \neq NULL) \land (\forall er \in Station : nr(ID) \neq er(ID)) \]

Integrity constraint predicate \( icp2 \)

\[ (nr(LAT_N) \neq NULL) \]

Integrity constraint predicate \( icp3 \)

\[ (nr(LAT_N) \geq 0 \land nr(LAT_N) \leq 90) \]

Integrity constraint predicate \( icp4 \)

Integrity constraint predicate \( icp5 \)

\[ (nr(LONG_W) \neq NULL) \]
The Model

CREATE TABLE Station (  
    ID INTEGER PRIMARY KEY,  
    CITY CHAR(20),  
    STATE CHAR(2),  
    LAT_N INTEGER NOT NULL  
       CHECK (LAT_N BETWEEN 0 and 90),  
    LONG_W INTEGER NOT NULL  
       CHECK (LONG_W BETWEEN SYMMETRIC 180 AND -180)  
);  

Integrity constraint predicate $icp1$  
$(nr(ID) \neq \text{NULL}) \land (\forall r \in \text{Station} : nr(ID) \neq er(ID))$

$icp2$  
$(nr(LAT_N) \neq \text{NULL})$

$icp3$  
$(nr(LAT_N) \geq 0 \land nr(LAT_N) \leq 90)$

$icp4$  
$icp5$

Form an acceptance predicate for the table:

$ap = icp1 \land icp2 \land icp3 \land icp4 \land icp5$

True when DBMS would accept the data  
False otherwise
Virtual DBMS Models

PostgreSQL

SQLite

HyperSQL

HSQLDB - 100% Java Database
Empirical Study

RQ1. What is the relative efficiency of the virtual approach?

RQ2. What are the time savings?

RQ3. How do mutation scores compare when the standard approach is run for as long as the virtual one?
Subject Schemas

<table>
<thead>
<tr>
<th>Schema</th>
<th>Tables</th>
<th>Columns</th>
<th>Checks</th>
<th>Foreign Keys</th>
<th>Not Nulls</th>
<th>Primary Keys</th>
<th>Uniques</th>
<th>Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoffeeOrders</td>
<td>5</td>
<td>20</td>
<td>0</td>
<td>4</td>
<td>10</td>
<td>5</td>
<td>0</td>
<td>19</td>
</tr>
<tr>
<td>Employee</td>
<td>1</td>
<td>7</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>Inventory</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Iso3166</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>JWhoisServer</td>
<td>6</td>
<td>49</td>
<td>0</td>
<td>0</td>
<td>44</td>
<td>6</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>MozillaPermissions</td>
<td>1</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>NistWeather</td>
<td>2</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>Person</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>Products</td>
<td>3</td>
<td>9</td>
<td>4</td>
<td>2</td>
<td>5</td>
<td>3</td>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>Total</td>
<td>21</td>
<td>114</td>
<td>13</td>
<td>7</td>
<td>71</td>
<td>21</td>
<td>1</td>
<td>113</td>
</tr>
</tbody>
</table>
RQ1: Efficiency

Database Schema

Mutation Analysis Time (Log Transformed)
RQ1: Efficiency

Virtual Mutation Analysis is significantly more efficient for Postgres and HyperSQL, but not SQLite.
RQ2: Time Savings

The chart shows the percentage of mean time saved across different databases and the number of mutants. The x-axis represents the number of mutants, while the y-axis represents the percentage of mean time saved. The databases compared are HyperSQL, PostgreSQL, and SQLite. The data points indicate that HyperSQL generally has a higher percentage of mean time saved compared to PostgreSQL and SQLite, especially at lower numbers of mutants. PostgreSQL and SQLite show a more variable performance, with PostgreSQL generally saving more time at higher numbers of mutants.
RQ2: Time Savings

Virtual Mutation Analysis yields large time savings for Postgres and HyperSQL but not always with SQLite, leading to an average time saving of 51% overall.
RQ3: Comparison

![Bar charts comparing Total Number of Mutants for different databases and schemata (Selective vs. Virtual).]
RQ3: Comparison

Virtual Mutation Analysis evaluates more mutants
RQ3: Comparison

Database Schema

Selectivity vs. Mutation Score for different database management systems (HyperSQL, PostgreSQL, SQLite) and database schemas (CoffeeOrders, Employee, Inventory, Iso3166, JWhoisServer, MozillaPermissions, NistWeather, Person, Products).
RQ3: Comparison

Virtual Mutation Analysis is the best option when highly accurate scores are needed under a time constraint.
Conclusions

Virtual Mutation Analysis Technique:

Removes the need to use a real DBMS for relational database schema mutation testing

More cost-effective while still being accurate:

• More efficient for 22 of 27 configurations studied
• Yields time savings of 13 to 99%