Search-Based Testing of Relational Schema Integrity Constraints Across Multiple Database Management Systems

Gregory M. Kapfhammer[1] & Phil McMinn[2] & Chris J. Wright[2]

[1]Allegheny College, USA

[2]University of Sheffield, UK

Sixth IEEE International Conference on Software Testing, Verification and Validation (ICST 2013)

Tuesday, March 19, 2013

ALLEGHENY COLLEGE

The University Of Sheffield.

Databases Are Everywhere!

Relational Database Management Systems

Databases Are Everywhere!

PostgreSQL

Apache Derby

HyperSQL

Relational Database Management Systems

IBM DB2

SQLite

MySQL®

Databases Are Everywhere!

Deployment Locations for Databases

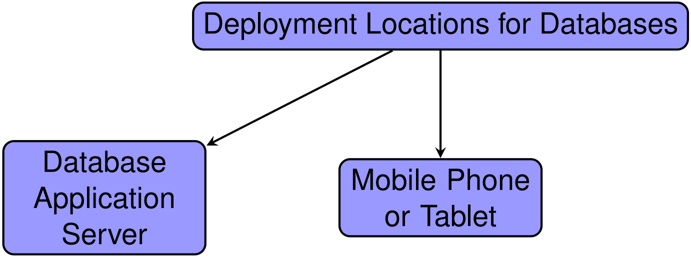
Databases Are Everywhere!

Deployment Locations for Databases

Database Application Server

PostgreSQL

Databases Are Everywhere!



Databases Are Everywhere!

Deployment Locations for Databases

- Database Application Server
- Mobile Phone or Tablet
- Office and Productivity Software

Databases Are Everywhere!

Deployment Locations for Databases

- Database Application Server
- Mobile Phone or Tablet
- Office and Productivity Software

Government

Databases Are Everywhere!

Deployment Locations for Databases

- Database Application Server
- Mobile Phone or Tablet
- Office and Productivity Software

Government

Astrophysics

Relational Database Schema

PostgreSQL

Relational Database Management System

Relational Database Schema

E-commerce

PostgreSQL

Relational Database Management System

Relational Database Schema

PostgreSQL

E-commerce

Relational Database Management System

Schema

Relational Database Schema

PostgreSQL

E-commerce

Relational Database Management System

Schema

State

Relational Database Schema

PostgreSQL

E-commerce

Relational Database Management System

Schema

State

Integrity Constraints

Relational Database Schema

PostgreSQL

E-commerce

Relational Database Management System

Schema

State

Integrity Constraints

`PRIMARY KEY`

Relational Database Schema

PostgreSQL

E-commerce

Relational Database Management System

Schema

State

Integrity Constraints

`PRIMARY KEY`

`FOREIGN KEY`

Relational Database Schema

E-commerce

PostgreSQL

Relational Database Management System

Schema

State

Integrity Constraints

`PRIMARY KEY`

`FOREIGN KEY`

Arbitrary `CHECK`

Relational Database Schema

PostgreSQL

E-commerce

Relational Database Management System

Schema

State

Relational Components

Relational Database Schema

PostgreSQL

E-commerce

Relational Database Management System

Schema

State

Relational Components

Tables

Relational Database Schema

PostgreSQL

E-commerce

Relational Database Management System

Schema

State

Relational Components

Tables

Rows

Relational Database Schema

PostgreSQL

E-commerce

Relational Database Management System

Schema | State

Relational Components

Tables | Rows | Columns

Relational Database Schema

PostgreSQL

E-commerce

Relational Database Management System

Schema

State

The Relational Schema is Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

INSERT →

Relational Database Management System

Schema

State

The Relational Schema is Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

INSERT ✓ →

Relational Database Management System

Schema

State

The Relational Schema is Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

`INSERT` ✓ →

Relational Database Management System

Schema

State

The Relational Schema is Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

INSERT →

Relational Database Management System

Schema | State

The Relational Schema is Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

INSERT ✗

Relational Database Management System

Schema

State

The Relational Schema is Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

`INSERT` ✗

Relational Database Management System

Schema

State

The Relational Schema is Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

Relational Database Management System

Schema

State

The Relational Schema is *Not* Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

✓INSERT →

Relational Database Management System

Schema

State

The Relational Schema is *Not* Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

✓INSERT ✗

Relational Database Management System

Schema

State

The Relational Schema is *Not* Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

✓INSERT ✗

Relational Database Management System

Schema

State

The Relational Schema is *Not* Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

✗INSERT

Relational Database Management System

Schema

State

The Relational Schema is *Not* Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

✗ INSERT ✓

Relational Database Management System

Schema

State

The Relational Schema is *Not* Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

✗INSERT ✓

Relational Database Management System

Schema

State

The Relational Schema is *Not* Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

SELECT ✓ →

Relational Database Management System

Schema

State

The Relational Schema is *Not* Working Correctly

Relational Database Schema

PostgreSQL

E-commerce

SELECT ✓

RESULT ✗

Relational Database Management System

Schema

State

The Relational Schema is *Not* Working Correctly

Relational Database Schema

Not working correctly!

E-commerce

SELECT ✓

RESULT ✗

PostgreSQL

Relational Database Management System

Schema

State

The Relational Schema is *Not* Working Correctly

Need for Relational Schema Testing

The Data Warehouse Institute reports that North American organizations experience a $611 billion annual loss due to poor data quality

Need for Relational Schema Testing

The Data Warehouse Institute reports that North American organizations experience a $611 billion annual loss due to poor data quality

Scott W. Ambler argues that the “virtual absence” of database testing — the validation of the contents, schema, and functionality of the database — is the primary cause of this loss

Need for Relational Schema Testing

The Data Warehouse Institute reports that North American organizations experience a $611 billion annual loss due to poor data quality

Scott W. Ambler argues that the “virtual absence” of database testing — the validation of the contents, schema, and functionality of the database — is the primary cause of this loss

This paper presents *SchemaAnalyst*, a search-based system for testing the complex integrity constraints in relational schemas

Defects in Relational Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT   CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Defects in Relational Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

The highlighted integrity constraints determine what data is valid

Defects in Relational Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT   CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

The highlighted integrity constraints determine what data is valid

Defects in Relational Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT   CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

The highlighted integrity constraints determine what data is valid

Defects in Relational Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID         CHAR(6) NOT NULL,
 SEGMENT_NUMBER    INT     NOT NULL,
 ORIGINAL_AIRPORT  CHAR(3),
 DEPART_TIME       TIME,
 DEST_AIRPORT      CHAR(3),
 ARRIVE_TIME       TIME,
 MEAL              CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

The highlighted integrity constraints determine what data is valid

Defects in Relational Schemas

```
CREATE TABLE FlightAvailable (
 FLIGHT_ID                 CHAR(6) NOT NULL,
 SEGMENT_NUMBER            INT     NOT NULL,
 FLIGHT_DATE               DATE    NOT NULL,
 ECONOMY_SEATS_TAKEN       INT,
 BUSINESS_SEATS_TAKEN      INT,
 FIRSTCLASS_SEATS_TAKEN    INT,
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 FOREIGN KEY(FLIGHT_ID, SEGMENT_NUMBER)
  REFERENCES Flights(FLIGHT_ID, SEGMENT_NUMBER)
);
```

The highlighted integrity constraints determine what data is valid

Defects in Relational Schemas

```
CREATE TABLE FlightAvailable (
  FLIGHT_ID                  CHAR(6) NOT NULL,
  SEGMENT_NUMBER             INT     NOT NULL,
  FLIGHT_DATE                DATE    NOT NULL,
  ECONOMY_SEATS_TAKEN        INT,
  BUSINESS_SEATS_TAKEN       INT,
  FIRSTCLASS_SEATS_TAKEN     INT,
  PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
  FOREIGN KEY(FLIGHT_ID, SEGMENT_NUMBER)
    REFERENCES Flights(FLIGHT_ID, SEGMENT_NUMBER)
);
```

The highlighted integrity constraints determine what data is valid

Defects in Relational Schemas

```
CREATE TABLE FlightAvailable (
 FLIGHT_ID                   CHAR(6) NOT NULL,
 SEGMENT_NUMBER              INT     NOT NULL,
 FLIGHT_DATE                 DATE    NOT NULL,
 ECONOMY_SEATS_TAKEN         INT,
 BUSINESS_SEATS_TAKEN        INT,
 FIRSTCLASS_SEATS_TAKEN      INT,
 PRIMARY KEY (FLIGHT_ID, SEGMENT_NUMBER),
 FOREIGN KEY (FLIGHT_ID, SEGMENT_NUMBER)
  REFERENCES Flights (FLIGHT_ID, SEGMENT_NUMBER)
);
```

The highlighted integrity constraints determine what data is valid

Defects in Relational Schemas

```
CREATE TABLE FlightAvailable (
  FLIGHT_ID                  CHAR(6) NOT NULL,
  SEGMENT_NUMBER             INT     NOT NULL,
  FLIGHT_DATE                DATE    NOT NULL,
  ECONOMY_SEATS_TAKEN        INT,
  BUSINESS_SEATS_TAKEN       INT,
  FIRSTCLASS_SEATS_TAKEN     INT,
  PRIMARY KEY (FLIGHT_ID, SEGMENT_NUMBER),
  FOREIGN KEY (FLIGHT_ID, SEGMENT_NUMBER)
   REFERENCES Flights (FLIGHT_ID, SEGMENT_NUMBER)
);
```

The highlighted integrity constraints determine what data is valid

Defects in Relational Schemas

Defect: The schema does not contain the correct primary key!

Defects in Relational Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT   CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Defect: The schema does not contain the correct primary key!

Defects in Relational Schemas

```
CREATE TABLE FlightAvailable (
 FLIGHT_ID                   CHAR(6) NOT NULL,
 SEGMENT_NUMBER              INT     NOT NULL,
 FLIGHT_DATE                 DATE    NOT NULL,
 ECONOMY_SEATS_TAKEN         INT,
 BUSINESS_SEATS_TAKEN        INT,
 FIRSTCLASS_SEATS_TAKEN      INT,
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 FOREIGN KEY(FLIGHT_ID, SEGMENT_NUMBER)
  REFERENCES Flights(FLIGHT_ID, SEGMENT_NUMBER)
);
```

Defect: The schema does not contain the correct primary key!

Defects in Relational Schemas

```
CREATE TABLE FlightAvailable (
  FLIGHT_ID                  CHAR(6) NOT NULL,
  SEGMENT_NUMBER             INT     NOT NULL,
  FLIGHT_DATE                DATE    NOT NULL,
  ECONOMY_SEATS_TAKEN        INT,
  BUSINESS_SEATS_TAKEN       INT,
  FIRSTCLASS_SEATS_TAKEN     INT,
  PRIMARY KEY (FLIGHT_ID, SEGMENT_NUMBER),
  FOREIGN KEY (FLIGHT_ID, SEGMENT_NUMBER)
    REFERENCES Flights (FLIGHT_ID, SEGMENT_NUMBER)
);
```

Question: What kind of INSERT(s) will reveal this defect?

Defects in Relational Schemas

```
INSERT INTO Flights
            VALUES('UA20', 1, ... ) ✓
```

Question: What kind of `INSERT`(s) will reveal this defect?

Defects in Relational Schemas

```
INSERT INTO Flights
            VALUES('UA20', 1, ...) ✓
INSERT INTO Flights
            VALUES('UA20', 2, ...) ✗
```

Question: What kind of `INSERT`(s) will reveal this defect?

Defects in Relational Schemas

```
INSERT INTO Flights
            VALUES('UA20', 1, ...) ✓
INSERT INTO Flights
            VALUES('UA20', 2, ...) ✗
```

Explanation: A flight with two different segments is no longer allowed!

Question: What kind of `INSERT`(s) will reveal this defect?

Defects in Relational Schemas

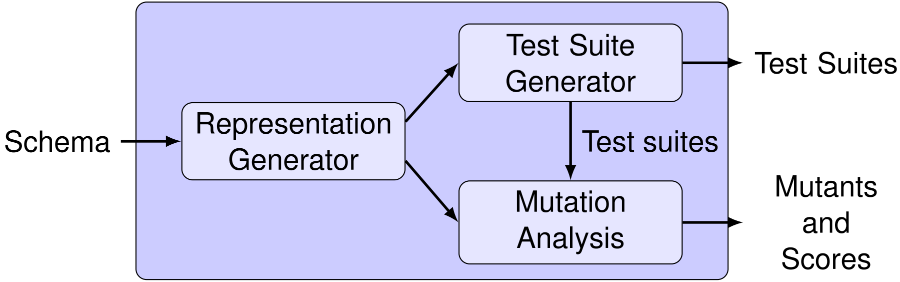
SchemaAnalyst automatically generates these `INSERT`s and this data!

```
INSERT INTO Flights
            VALUES('UA20', 1, ... ) ✓
INSERT INTO Flights
            VALUES('UA20', 2, ... ) ✗
```

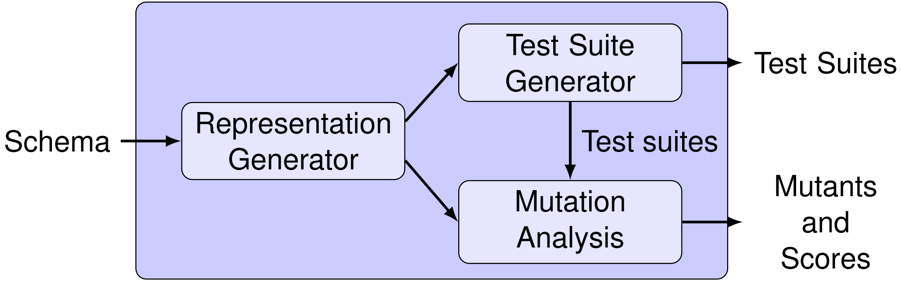
Explanation: A flight with two different segments is no longer allowed!

Question: What kind of `INSERT`(s) will reveal this defect?

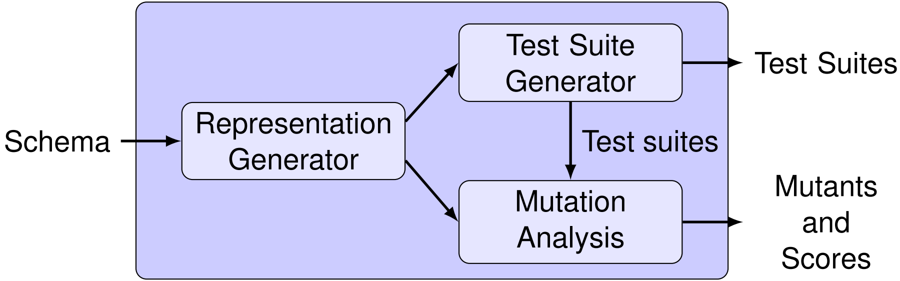
Search-Based Testing with *SchemaAnalyst*



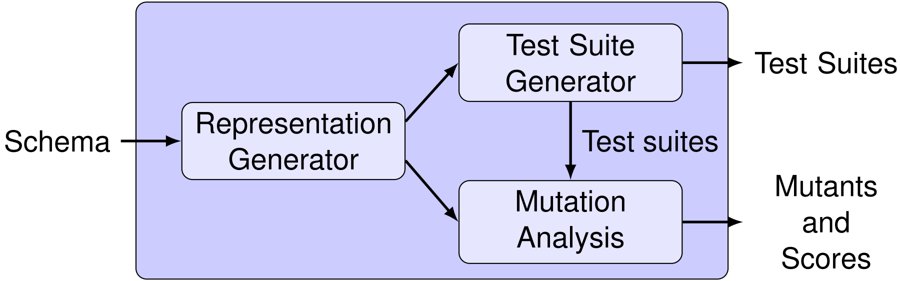
Search-Based Testing with *SchemaAnalyst*



Search-Based Testing with *SchemaAnalyst*



Search-Based Testing with *SchemaAnalyst*



Goals and Stages of Test Data Generation

Goal of test data generation?

Goals and Stages of Test Data Generation

Goal of test data generation?

INSERT INTO T_1 VALUES(1, Jan-08-99, ...) ✓

Goals and Stages of Test Data Generation

Goal of test data generation?

INSERT INTO T_1 VALUES(1, Jan-08-99, ...) ✓

INSERT INTO T_1 VALUES(1, Jan-08-99, ...) ✗

Goals and Stages of Test Data Generation

Goal of test data generation?

INSERT INTO T_1 VALUES(1, Jan-08-99, ...) ✓

INSERT INTO T_1 VALUES(1, Jan-08-99, ...) ✗

INSERT INTO T_n VALUES(true, 'L-20', ...) ✓

INSERT INTO T_n VALUES(false, 'L-1', ...) ✗

Goals and Stages of Test Data Generation

```
CREATE TABLE Flights(
 FLIGHT_ID           CHAR(6) NOT NULL,
 SEGMENT_NUMBER      INT     NOT NULL,
 ORIGINAL_AIRPORT    CHAR(3),
 DEPART_TIME         TIME,
 DEST_AIRPORT        CHAR(3),
 ARRIVE_TIME         TIME,
 MEAL                CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Goals and Stages of Test Data Generation

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Stage 1: Generate rows of data to satisfy the integrity constraints

Goals and Stages of Test Data Generation

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT   CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Stage 1: Generate rows of data to satisfy the integrity constraints

Goals and Stages of Test Data Generation

```
CREATE TABLE Flights(
  FLIGHT_ID          CHAR(6) NOT NULL,
  SEGMENT_NUMBER     INT     NOT NULL,
  ORIGINAL_AIRPORT   CHAR(3),
  DEPART_TIME        TIME,
  DEST_AIRPORT       CHAR(3),
  ARRIVE_TIME        TIME,
  MEAL               CHAR(1),
  PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
  CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Goals and Stages of Test Data Generation

```
CREATE TABLE Flights(
  FLIGHT_ID          CHAR(6) NOT NULL,
  SEGMENT_NUMBER     INT     NOT NULL,
  ORIGINAL_AIRPORT   CHAR(3),
  DEPART_TIME        TIME,
  DEST_AIRPORT       CHAR(3),
  ARRIVE_TIME        TIME,
  MEAL               CHAR(1),
  PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
  CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Stage 2: Generate rows of data to negate a constraint

Goals and Stages of Test Data Generation

```
CREATE TABLE Flights(
  FLIGHT_ID          CHAR(6) NOT NULL,
  SEGMENT_NUMBER     INT     NOT NULL,
  ORIGINAL_AIRPORT CHAR(3),
  DEPART_TIME        TIME,
  DEST_AIRPORT       CHAR(3),
  ARRIVE_TIME        TIME,
  MEAL               CHAR(1),
  PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
  CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Stage 2: Generate rows of data to negate a constraint

Goals and Stages of Test Data Generation

```
CREATE TABLE Flights(
 FLIGHT_ID         CHAR(6) NOT NULL,
 SEGMENT_NUMBER    INT     NOT NULL,
 ORIGINAL_AIRPORT CHAR(3),
 DEPART_TIME      TIME,
 DEST_AIRPORT     CHAR(3),
 ARRIVE_TIME      TIME,
 MEAL             CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

A fitness function computes a numeric value minimized by search

Goals and Stages of Test Data Generation

```
CREATE TABLE Flights(
 FLIGHT_ID         CHAR(6) NOT NULL,
 SEGMENT_NUMBER    INT     NOT NULL,
 ORIGINAL_AIRPORT  CHAR(3),
 DEPART_TIME       TIME,
 DEST_AIRPORT      CHAR(3),
 ARRIVE_TIME       TIME,
 MEAL              CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Data's fitness is closer to zero when nearer to a primary key value

Goals and Stages of Test Data Generation

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT   CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Types, primary and foreign keys, `UNIQUE`, `NOT NULL`, and `CHECK`

Goals and Stages of Test Data Generation

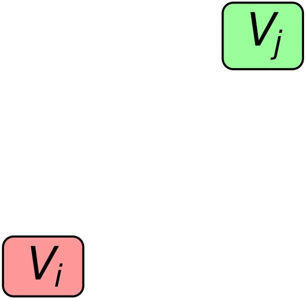
```
CREATE TABLE Flights(
  FLIGHT_ID          CHAR(6) NOT NULL,
  SEGMENT_NUMBER     INT     NOT NULL,
  ORIGINAL_AIRPORT   CHAR(3),
  DEPART_TIME        TIME,
  DEST_AIRPORT       CHAR(3),
  ARRIVE_TIME        TIME,
  MEAL               CHAR(1),
  PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
  CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

See the paper for more details about the computation of fitness

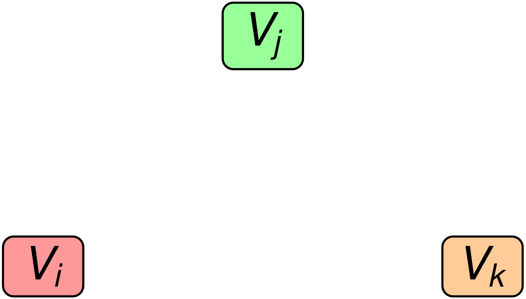
Alternating Variable Method

$\boxed{V_i}$

Alternating Variable Method



Alternating Variable Method



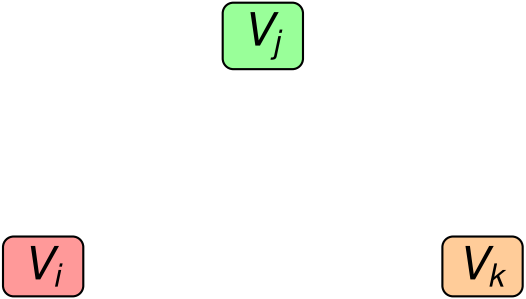
Alternating Variable Method

V_j

V_i V_k

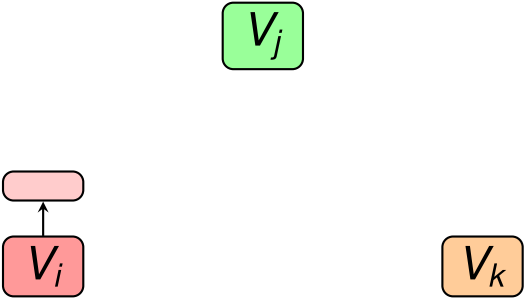
Use the defaults to form the initial values of the `INSERT` variables

Alternating Variable Method



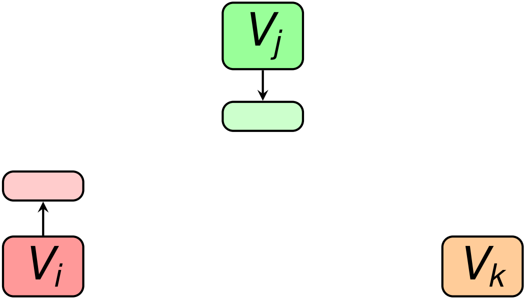
Use exploratory moves to determine the correct direction for search

Alternating Variable Method



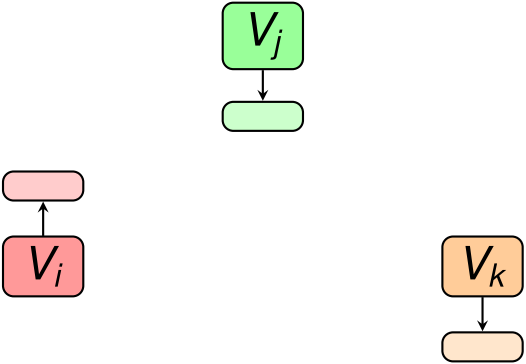
Use exploratory moves to determine the correct direction for search

Alternating Variable Method



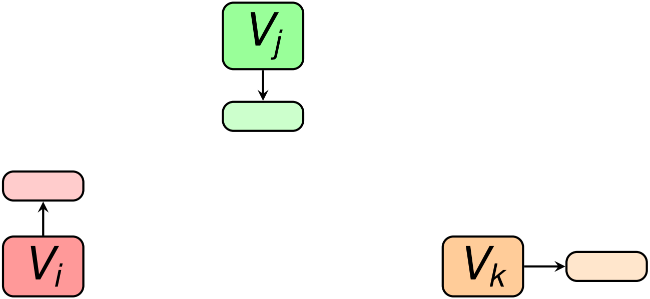
Use exploratory moves to determine the correct direction for search

Alternating Variable Method



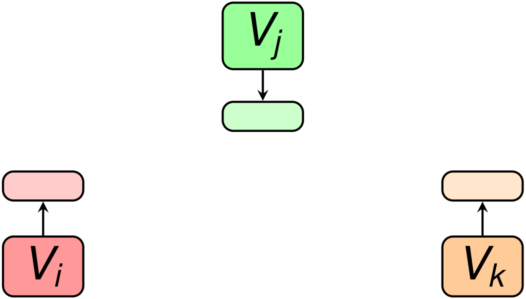
Use exploratory moves to determine the correct direction for search

Alternating Variable Method



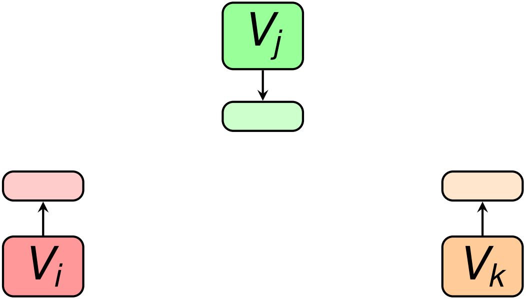
Use exploratory moves to determine the correct direction for search

Alternating Variable Method



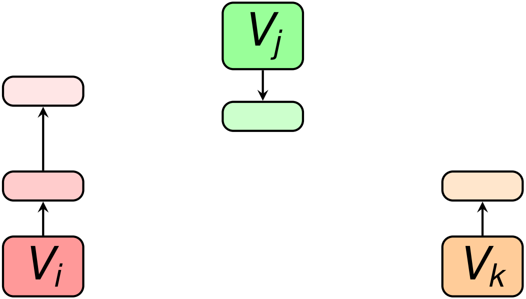
Use exploratory moves to determine the correct direction for search

Alternating Variable Method



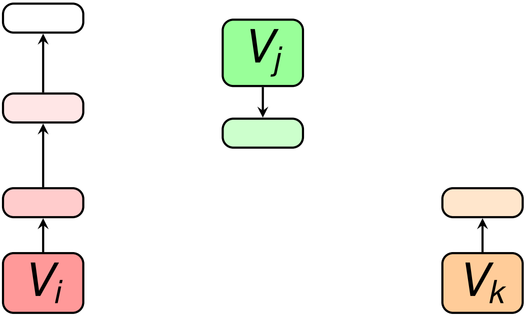
Use pattern moves to accelerate the improvements in fitness

Alternating Variable Method



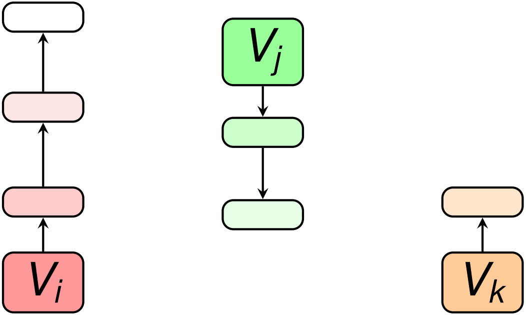
Use pattern moves to accelerate the improvements in fitness

Alternating Variable Method



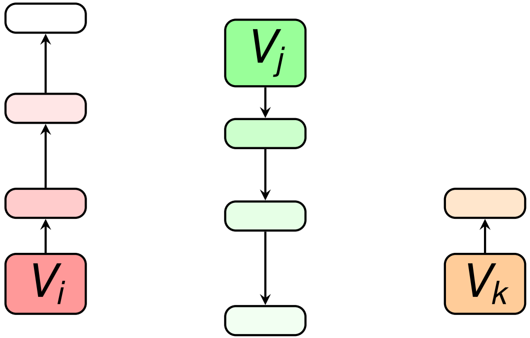
Use pattern moves to accelerate the improvements in fitness

Alternating Variable Method



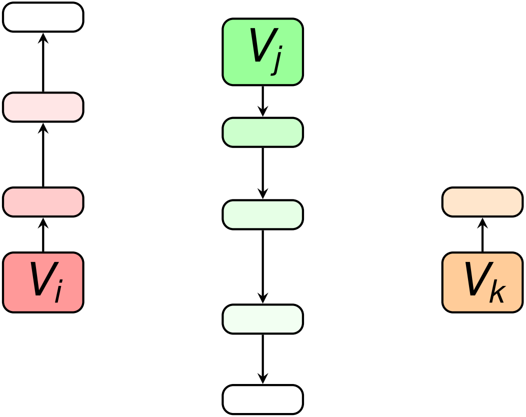
Use pattern moves to accelerate the improvements in fitness

Alternating Variable Method



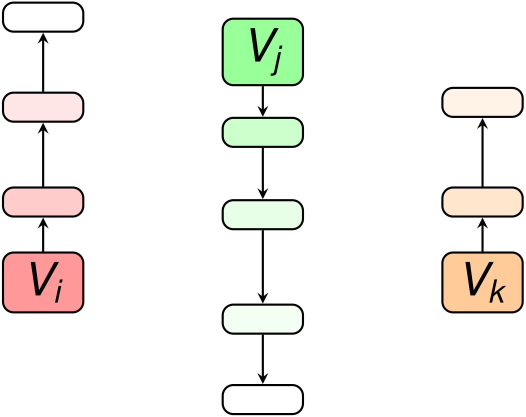
Use pattern moves to accelerate the improvements in fitness

Alternating Variable Method



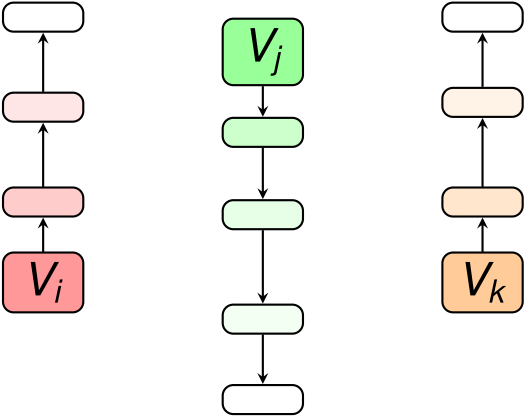
Use pattern moves to accelerate the improvements in fitness

Alternating Variable Method



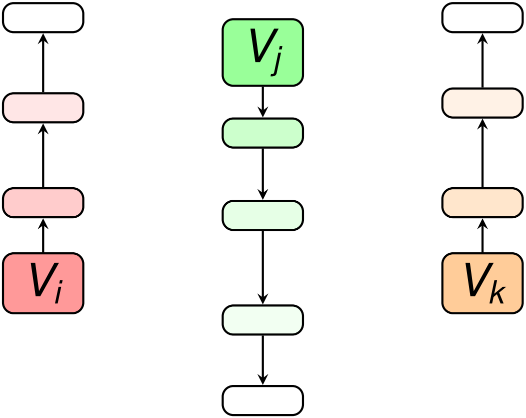
Use pattern moves to accelerate the improvements in fitness

Alternating Variable Method



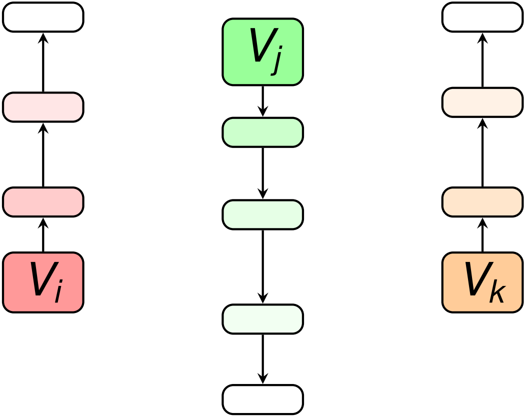
Use pattern moves to accelerate the improvements in fitness

Alternating Variable Method



AVM terminates when the fitness is zero or an exploration cycle fails

Alternating Variable Method



Restart AVM with random column values when an exploration cycle fails

Mutation Operators for Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT   CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Mutation Operators for Schemas

```
CREATE TABLE Flights(
  FLIGHT_ID          CHAR(6) NOT NULL,
  SEGMENT_NUMBER     INT     NOT NULL,
  ORIGINAL_AIRPORT   CHAR(3),
  DEPART_TIME        TIME,
  DEST_AIRPORT       CHAR(3),
  ARRIVE_TIME        TIME,
  MEAL               CHAR(1),
  PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
  CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Use mutation analysis to assess the adequacy of `INSERT`s and values

Mutation Operators for Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT   CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Primary Keys: Remove, replace, and add column operators

Mutation Operators for Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID         CHAR(6) NOT NULL,
 SEGMENT_NUMBER    INT     NOT NULL,
 ORIGINAL_AIRPORT  CHAR(3),
 DEPART_TIME       TIME,
 DEST_AIRPORT      CHAR(3),
 ARRIVE_TIME       TIME,
 MEAL              CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Primary Keys: Remove, replace, and add column operators

Mutation Operators for Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT   CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Primary Keys: Remove, replace, and add column operators

Mutation Operators for Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID         CHAR(6) NOT NULL,
 SEGMENT_NUMBER    INT     NOT NULL,
 ORIGINAL_AIRPORT  CHAR(3),
 DEPART_TIME       TIME,
 DEST_AIRPORT      CHAR(3),
 ARRIVE_TIME       TIME,
 MEAL              CHAR(1),
 PRIMARY KEY(ORIGINAL_AIRPORT, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Primary Keys: Remove, replace, and add column operators

Mutation Operators for Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT   CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Primary Keys: Remove, replace, and add column operators

Mutation Operators for Schemas

```
CREATE TABLE Flights(
  FLIGHT_ID           CHAR(6) NOT NULL,
  SEGMENT_NUMBER      INT     NOT NULL,
  ORIGINAL_AIRPORT    CHAR(3),
  DEPART_TIME         TIME,
  DEST_AIRPORT        CHAR(3),
  ARRIVE_TIME         TIME,
  MEAL                CHAR(1),
PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER, DEST_AIRPORT),
  CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

Primary Keys: Remove, replace, and add column operators

Mutation Operators for Schemas

```
CREATE TABLE Flights(
  FLIGHT_ID           CHAR(6) NOT NULL,
  SEGMENT_NUMBER      INT     NOT NULL,
  ORIGINAL_AIRPORT    CHAR(3),
  DEPART_TIME         TIME,
  DEST_AIRPORT        CHAR(3),
  ARRIVE_TIME         TIME,
  MEAL                CHAR(1),
  PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
  CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

`UNIQUE`: Handle in a fashion similar to the primary key operator

Mutation Operators for Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT   CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

`NOT NULL`: Reverse the status for all non-primary key columns

Mutation Operators for Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID         CHAR(6) NOT NULL,
 SEGMENT_NUMBER    INT     NOT NULL,
 ORIGINAL_AIRPORT  CHAR(3),
 DEPART_TIME       TIME,
 DEST_AIRPORT      CHAR(3),
 ARRIVE_TIME       TIME,
 MEAL              CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

NOT NULL: Reverse the status for all non-primary key columns

Mutation Operators for Schemas

```
CREATE TABLE Flights(
  FLIGHT_ID          CHAR(6) NOT NULL,
  SEGMENT_NUMBER     INT     NOT NULL,
  ORIGINAL_AIRPORT   CHAR(3) NOT NULL,
  DEPART_TIME        TIME,
  DEST_AIRPORT       CHAR(3),
  ARRIVE_TIME        TIME,
  MEAL               CHAR(1),
  PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
  CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

NOT NULL: Reverse the status for all non-primary key columns

Mutation Operators for Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT   CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

`CHECK`: Remove the constraint for each of the checked columns

Mutation Operators for Schemas

```
CREATE TABLE Flights(
 FLIGHT_ID          CHAR(6) NOT NULL,
 SEGMENT_NUMBER     INT     NOT NULL,
 ORIGINAL_AIRPORT   CHAR(3),
 DEPART_TIME        TIME,
 DEST_AIRPORT       CHAR(3),
 ARRIVE_TIME        TIME,
 MEAL               CHAR(1),
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 CHECK(MEAL IN ('B', 'L', 'D', 'S'))
);
```

CHECK: Remove the constraint for each of the checked columns

Mutation Operators for Schemas

```
CREATE TABLE FlightAvailable (
 FLIGHT_ID                  CHAR(6) NOT NULL,
 SEGMENT_NUMBER             INT     NOT NULL,
 FLIGHT_DATE                DATE    NOT NULL,
 ECONOMY_SEATS_TAKEN        INT,
 BUSINESS_SEATS_TAKEN       INT,
 FIRSTCLASS_SEATS_TAKEN     INT,
 PRIMARY KEY(FLIGHT_ID, SEGMENT_NUMBER),
 FOREIGN KEY(FLIGHT_ID, SEGMENT_NUMBER)
  REFERENCES Flights(FLIGHT_ID, SEGMENT_NUMBER)
);
```

Foreign Keys: Remove each column from the key

Mutation Operators for Schemas

```
CREATE TABLE FlightAvailable (
 FLIGHT_ID                  CHAR(6) NOT NULL,
 SEGMENT_NUMBER             INT     NOT NULL,
 FLIGHT_DATE                DATE    NOT NULL,
 ECONOMY_SEATS_TAKEN        INT,
 BUSINESS_SEATS_TAKEN       INT,
 FIRSTCLASS_SEATS_TAKEN     INT,
 PRIMARY KEY (FLIGHT_ID, SEGMENT_NUMBER),
 FOREIGN KEY (FLIGHT_ID, SEGMENT_NUMBER)
  REFERENCES Flights (FLIGHT_ID, SEGMENT_NUMBER)
);
```

Foreign Keys: Remove each column from the key

Mutation Operators for Schemas

```
CREATE TABLE FlightAvailable (
  FLIGHT_ID                   CHAR(6) NOT NULL,
  SEGMENT_NUMBER              INT     NOT NULL,
  FLIGHT_DATE                 DATE    NOT NULL,
  ECONOMY_SEATS_TAKEN         INT,
  BUSINESS_SEATS_TAKEN        INT,
  FIRSTCLASS_SEATS_TAKEN      INT,
  PRIMARY KEY (FLIGHT_ID, SEGMENT_NUMBER),
  FOREIGN KEY (FLIGHT_ID, SEGMENT_NUMBER)
    REFERENCES Flights (FLIGHT_ID, SEGMENT_NUMBER)
);
```

Foreign Keys: Remove each column from the key

Calculating the Mutation Score

$$M_D = \frac{|K \cup Q|}{|K \cup N|}$$

Calculating the Mutation Score

$$M_D = \frac{|K \cup Q|}{|K \cup N|}$$

Calculating the Mutation Score

$$M_D = \frac{|K \cup Q|}{|K \cup N|}$$

Calculating the Mutation Score

$$M_D = \frac{|K \cup Q|}{|K \cup N|}$$

Calculating the Mutation Score

$$M_D = \frac{|K \cup Q|}{|K \cup N|}$$

Calculating the Mutation Score

$$M_D = \frac{|K \cup \color{blue}{Q}|}{|K \cup N|}$$

HyperSQL SQLite PostgreSQL

Calculating the Mutation Score

$$M_D = \frac{|K \cup Q|}{|K \cup N|}$$

HyperSQL SQLite PostgreSQL

✓

Calculating the Mutation Score

$$M_D = \frac{|K \cup Q|}{|K \cup N|}$$

HyperSQL

SQLite ✓

PostgreSQL ✗

Calculating the Mutation Score

$$M_D = \frac{|K \cup Q|}{|K \cup N|}$$

HyperSQL	SQLite	PostgreSQL
✗	✓	✗

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
BankAccount	2	9	0	1	5	2	0	8
BookTown	23	69	1	0	17	11	0	29
Cloc	2	10	0	0	0	0	0	0
CoffeeOrders	5	20	0	4	9	5	0	18
CustomerOrder	7	32	1	7	27	7	0	42

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
BankAccount	2	9	0	1	5	2	0	8
BookTown	23	69	1	0	17	11	0	29
Cloc	2	10	0	0	0	0	0	0
CoffeeOrders	5	20	0	4	9	5	0	18
CustomerOrder	7	32	1	7	27	7	0	42

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
BankAccount	2	9	0	1	5	2	0	8
BookTown	23	69	1	0	17	11	0	29
Cloc	2	10	0	0	0	0	0	0
CoffeeOrders	5	20	0	4	9	5	0	18
CustomerOrder	7	32	1	7	27	7	0	42

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
BankAccount	2	9	0	1	5	2	0	8
BookTown	23	69	1	0	17	11	0	29
Cloc	2	10	0	0	0	0	0	0
CoffeeOrders	5	20	0	4	9	5	0	18
CustomerOrder	7	32	1	7	27	7	0	42

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
BankAccount	2	9	0	1	5	2	0	8
BookTown	23	69	1	0	17	11	0	29
Cloc	2	10	0	0	0	0	0	0
CoffeeOrders	5	20	0	4	9	5	0	18
CustomerOrder	7	32	1	7	27	7	0	42

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
BankAccount	2	9	0	1	5	2	0	8
BookTown	23	69	1	0	17	11	0	29
Cloc	2	10	0	0	0	0	0	0
CoffeeOrders	5	20	0	4	9	5	0	18
CustomerOrder	7	32	1	7	27	7	0	42

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
BankAccount	2	9	0	1	5	2	0	8
BookTown	23	69	1	0	17	11	0	29
Cloc	2	10	0	0	0	0	0	0
CoffeeOrders	5	20	0	4	9	5	0	18
CustomerOrder	7	32	1	7	27	7	0	42

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
DellStore	8	52	0	0	36	0	0	36
Employee	1	7	3	0	0	1	0	4
Examination	2	21	6	1	0	2	0	9
Flights	2	13	1	1	6	2	0	10
FrenchTowns	3	14	0	2	13	0	8	23

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
DellStore	8	52	0	0	36	0	0	36
Employee	1	7	3	0	0	1	0	4
Examination	2	21	6	1	0	2	0	9
Flights	2	13	1	1	6	2	0	10
FrenchTowns	3	14	0	2	13	0	8	23

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
Inventory	1	4	0	0	0	1	1	2
Iso3166	1	3	0	0	2	1	0	3
JWhoisServer	6	49	0	0	44	6	0	50
NistDML181	2	7	0	1	0	1	0	2
NistDML182	2	32	0	1	0	1	0	2

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
Inventory	1	4	0	0	0	1	1	2
Iso3166	1	3	0	0	2	1	0	3
JWhoisServer	6	49	0	0	44	6	0	50
NistDML181	2	7	0	1	0	1	0	2
NistDML182	2	32	0	1	0	1	0	2

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
Inventory	1	4	0	0	0	1	1	2
Iso3166	1	3	0	0	2	1	0	3
JWhoisServer	6	49	0	0	44	6	0	50
NistDML181	2	7	0	1	0	1	0	2
NistDML182	2	32	0	1	0	1	0	2

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
NistDML183	2	6	0	1	0	0	1	2
NistWeather	2	9	5	0	2	2	0	9
NistXTS748	1	3	1	0	1	0	1	3
NistXTS749	2	7	1	1	3	2	0	7
Person	1	5	1	0	5	1	0	7

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
Products	3	9	4	2	5	3	0	14
Residence	2	6	3	1	2	2	0	8
RiskIt	13	56	0	10	15	11	0	36
UnixUsage	8	32	0	7	9	7	0	23
Usda	10	67	0	0	30	0	0	30

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
Products	3	9	4	2	5	3	0	14
Residence	2	6	3	1	2	2	0	8
RiskIt	13	56	0	10	15	11	0	36
UnixUsage	8	32	0	7	9	7	0	23
Usda	10	67	0	0	30	0	0	30

Case Study Schemas

Schema	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
Products	3	9	4	2	5	3	0	14
Residence	2	6	3	1	2	2	0	8
RiskIt	13	56	0	10	15	11	0	36
UnixUsage	8	32	0	7	9	7	0	23
Usda	10	67	0	0	30	0	0	30

Case Study Schemas

	Tables	Columns	Checks	Foreign keys	Not Nulls	Primary keys	Uniques	Total Constraints
Totals	111	542	27	40	231	68	11	377

Data Generation Techniques

DBMonster

Data Generation Techniques

DBMonster

SchemaAnalyst

Data Generation Techniques

DBMonster

SchemaAnalyst

HSQLDB ✓

SQLite ✓

Postgres ✓

Data Generation Techniques

DBMonster

HSQLDB ✗

SchemaAnalyst

HSQLDB ✓

SQLite ✓

Postgres ✓

Data Generation Techniques

DBMonster

HSQLDB ✗

SQLite ✗

SchemaAnalyst

HSQLDB ✓

SQLite ✓

Postgres ✓

Data Generation Techniques

DBMonster

HSQLDB ✗

SQLite ✗

Postgres ✓

SchemaAnalyst

HSQLDB ✓

SQLite ✓

Postgres ✓

Constraint Coverage Results

Schema	AVM (%)	*DBMonster* (%)
Flights	100.0	70.0
FrenchTowns	100.0	70.0
Inventory	100.0	75.0
Iso3166	100.0	50.0
JWhoisServer	100.0	50.0

Constraint Coverage Results

Schema	AVM (%)	*DBMonster* (%)
Flights	100.0	70.0
FrenchTowns	100.0	70.0
Inventory	100.0	75.0
Iso3166	100.0	50.0
JWhoisServer	100.0	50.0

Constraint Coverage Results

Schema	AVM (%)	*DBMonster* (%)
NistDML181	100.0	75.0
NistDML182	100.0	50.0
NistDML183	100.0	100.0
NistXTS748	100.0	72.2
NistXTS749	100.0	21.4

Constraint Coverage Results

Schema	AVM (%)	*DBMonster* (%)
NistDML181	100.0	75.0
NistDML182	100.0	50.0
NistDML183	100.0	100.0
NistXTS748	100.0	72.2
NistXTS749	100.0	21.4

Constraint Coverage Results

Schema	AVM (%)	*DBMonster* (%)
NistDML181	100.0	75.0
NistDML182	100.0	50.0
NistDML183	100.0	100.0
NistXTS748	100.0	72.2
NistXTS749	100.0	21.4

Constraint Coverage Results

Schema	AVM (%)	*DBMonster* (%)
Residence	100.0	62.5
RiskIt	100.0	4.1
Products	96.4	59.3
UnixUsage	97.8	59.3
Usda	100.0	50.0

Constraint Coverage Results

Schema	AVM (%)	*DBMonster* (%)
Residence	100.0	62.5
RiskIt	100.0	4.1
Products	96.4	59.3
UnixUsage	97.8	59.3
Usda	100.0	50.0

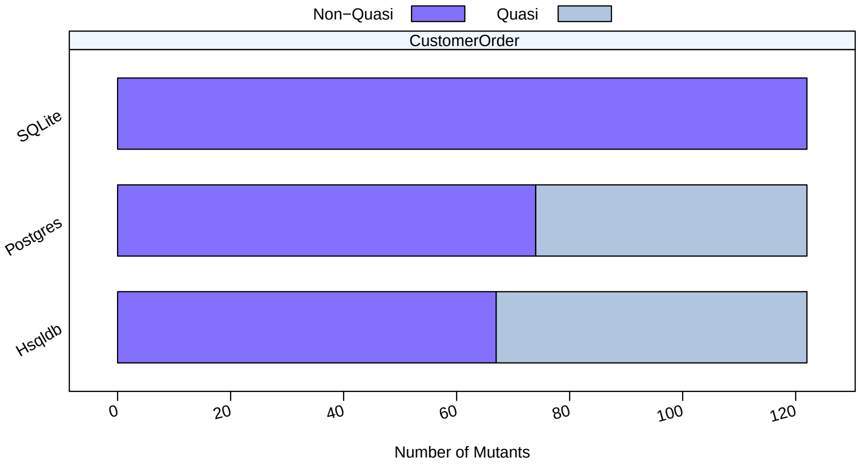
Constraint Coverage Results

Schema	AVM (%)	*DBMonster* (%)
Residence	100.0	62.5
RiskIt	100.0	4.1
Products	96.4	59.3
UnixUsage	97.8	59.3
Usda	100.0	50.0

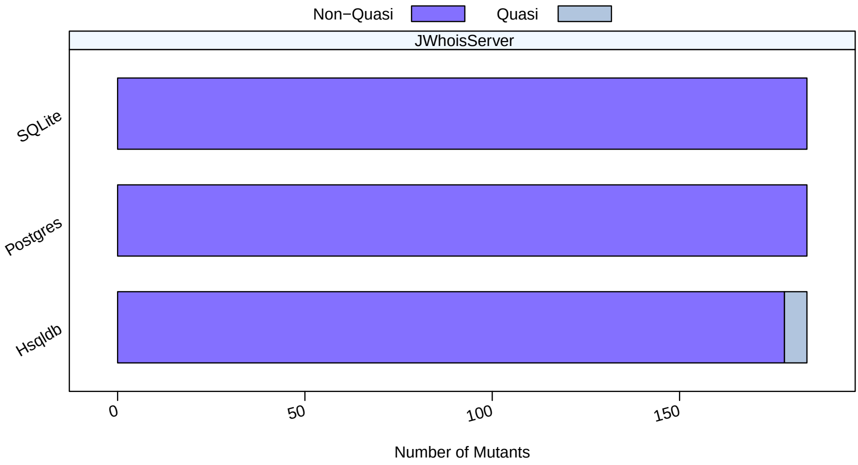
Constraint Coverage Results

Schema	AVM (%)	*DBMonster* (%)
Residence	100.0	62.5
RiskIt	100.0	4.1
Products	96.4	59.3
UnixUsage	97.8	59.3
Usda	100.0	50.0

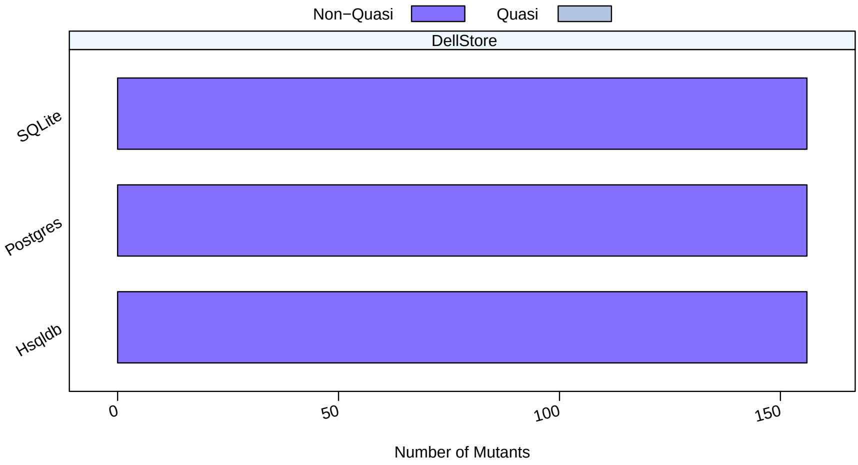
Quasi-Mutant Results



Quasi-Mutant Results



Quasi-Mutant Results



Summary: Quasi-Mutant Results

Summary: Quasi-Mutant Results

HyperSQL	SQLite	PostgreSQL
Some	None	Some

Summary: Quasi-Mutant Results

HyperSQL	SQLite	PostgreSQL
Some	None	Some

Few quasi-mutants means that the mutation scores are good effectiveness indicators

Mutation Score Results

DBMonster

SchemaAnalyst

Mutation Score Results

	DBMonster	*SchemaAnalyst*
JWhoisServer	DBI=300, $M_D = 0.2$	DBI=62, $M_D = 0.7$

Mutation Score Results

	DBMonster	*SchemaAnalyst*
JWhoisServer	DBI=300, $M_D = 0.2$	DBI=62, $M_D = 0.7$
NistDML181	DBI=13,650, $M_D = 0.5$	DBI=7, $M_D = 0.6$

Mutation Score Results

DBMonster

SchemaAnalyst

(0.0, 0.11, 0.41, 0.52, 0.68)

(0.29, 0.59, 0.65, 0.70, 0.89)

Mutation Score Results

DBMonster crashes for six schemas!

CustomerOrder
Flights
NistDML182
NistXTS748
Person
RiskIt

DBMonster

(0.0, 0.11, 0.41, 0.52, 0.68)

SchemaAnalyst

(0.29, 0.59, 0.65, 0.70, 0.89)

Mutation Score Results

DBMonster crashes for six schemas!

CustomerOrder
Flights
NistDML182
NistXTS748
Person
RiskIt

DBMonster

(0.0, 0.11, 0.41, 0.52, 0.68)

SchemaAnalyst

(0.29, 0.59, 0.65, 0.70, 0.89)

Mutation Score Results

DBMonster crashes for six schemas!

CustomerOrder
Flights
NistDML182
NistXTS748
Person
RiskIt

DBMonster

(0.0, 0.11, 0.41, 0.52, 0.68)

SchemaAnalyst

(0.29, 0.59, 0.65, 0.70, 0.89)

Mutation Score Results

DBMonster

(0.0, 0.11, 0.41, 0.52, 0.68)

SchemaAnalyst

(0.29, 0.59, 0.65, 0.70, 0.89)

Mutation Score Results

DBMonster

(0.0, 0.11, 0.41, 0.52, 0.68)

SchemaAnalyst

(0.29, 0.59, 0.65, 0.70, 0.89)

Mutation Score Results

SchemaAnalyst's mutation score is higher than *DB-Monster*'s for 96% of the schemas

DBMonster

SchemaAnalyst

(0.0, 0.11, 0.41, 0.52, 0.68)

(0.29, 0.59, 0.65, 0.70, 0.89)

Efficiency Results

DBMonster

SchemaAnalyst

Efficiency Results

DBMonster

SchemaAnalyst

(1.50, 3.01, 5.21, 16.79, 639.93)

(0.41, 1.09, 1.90, 5.07, 36.52)

Efficiency Results

DBMonster

(1.50, 3.01, 5.21, 16.79, 639.93)

SchemaAnalyst

(0.41, 1.09, 1.90, 5.07, 36.52)

Efficiency Results

SchemaAnalyst exhibits competitive data generation times that are less variable

DBMonster

(**1.50**, 3.01, 5.21, 16.79, **639.93**)

SchemaAnalyst

(**0.41**, 1.09, 1.90, 5.07, **36.52**)

Important Contributions

This paper presents *SchemaAnalyst*, a search-based system for testing the complex integrity constraints in relational schemas

Important Contributions

This paper presents *SchemaAnalyst*, a search-based system for testing the complex integrity constraints in relational schemas

The empirical study demonstrates that *SchemaAnalyst*'s efficiency is competitive with *DBMonster*'s

Important Contributions

This paper presents *SchemaAnalyst*, a search-based system for testing the complex integrity constraints in relational schemas

The empirical study demonstrates that *SchemaAnalyst*'s efficiency is competitive with *DBMonster*'s

SchemaAnalyst almost always covers 100% of the constraints in the 25 chosen relational schemas

Important Contributions

This paper presents *SchemaAnalyst*, a search-based system for testing the complex integrity constraints in relational schemas

The empirical study demonstrates that *Schema-Analyst*'s efficiency is competitive with *DBMonster*'s

SchemaAnalyst almost always covers 100% of the constraints in the 25 chosen relational schemas

SchemaAnalyst's mutation score is higher than *DBMonster*'s for 96% of the schemas

Important Contributions

This paper presents *SchemaAnalyst*, a search-based system for testing the complex integrity constraints in relational schemas

The empirical study demonstrates that *SchemaAnalyst*'s efficiency is competitive with *DBMonster*'s

SchemaAnalyst almost always covers 100% of the constraints in the 25 chosen relational schemas

SchemaAnalyst's mutation score is higher than *DBMonster*'s for 96% of the schemas

http://www.schemaanalyst.org