Automating the Detection of Snapshot Isolation Anomalies

- **Sudhir Jorwekar** (IIT Bombay)
- **Alan Fekete** (Univ. Sydney)
- **Krithi Ramamritham** (IIT Bombay)
- **S. Sudarshan** (IIT Bombay)

Motivation

- Non-serializable executions are possible in Snapshot Isolation.
- Many industry applications run on systems that use Snapshot Isolation as the isolation level
  - E.g. Oracle, PostgreSQL, SQL Server etc.

Motivation

- Non-serializable executions are possible in Snapshot Isolation.
- Many industry applications run on systems that use Snapshot Isolation as the isolation level
  - E.g. Oracle, PostgreSQL, SQL Server etc.

Theory for identifying such anomalies already exists.
(Needs manual analysis)

Motivation

- Non-serializable executions are possible in Snapshot Isolation.
- Many industry applications run on systems that use Snapshot Isolation as the isolation level
  - E.g. Oracle, PostgreSQL, SQL Server etc.

Theory for identifying such anomalies already exists.
(Needs manual analysis)

Challenges

To have a tool to examine the application and see whether or not anomalies are possible when it executes on SI platform.

Automating the fixing the anomalies.

Agenda

1. Introduction to Snapshot Isolation Protocol
    1. Examples of SI-Anomalies
    2. Existing Theory for Detecting SI-Anomalies

4. Analyzing the transaction programs

3. Reducing the false positive

4. Results

What is Snapshot Isolation?

What is Snapshot Isolation?

Snapshot Isolation [Berenson et.al. SIGMOD'95]

What is Snapshot Isolation?

Snapshot Isolation [Berenson et.al. SIGMOD'95]

A transaction T executing with Snapshot Isolation

What is Snapshot Isolation?

Snapshot Isolation [Berenson et.al. SIGMOD'95]

A transaction T executing with Snapshot Isolation

- takes snapshot of committed data at start

What is Snapshot Isolation?

Snapshot Isolation [Berenson et.al. SIGMOD'95]

A transaction T executing with Snapshot Isolation

- takes snapshot of committed data at start
- always reads/modifies data in its own snapshot

What is Snapshot Isolation?

Snapshot Isolation [Berenson et.al. SIGMOD'95]

A transaction T executing with Snapshot Isolation

- takes snapshot of committed data at start
- always reads/modifies data in its own snapshot
- updates of concurrent transactions are not visible to T

What is Snapshot Isolation?

Snapshot Isolation [Berenson et.al. SIGMOD'95]

A transaction T executing with Snapshot Isolation

- takes snapshot of committed data at start
- always reads/modifies data in its own snapshot
- updates of concurrent transactions are not visible to T
- writes of T complete when it commits

What is Snapshot Isolation?

Snapshot Isolation [Berenson et.al. SIGMOD'95]

A transaction T executing with Snapshot Isolation

- takes snapshot of committed data at start
- always reads/modifies data in its own snapshot
- updates of concurrent transactions are not visible to T
- writes of T complete when it commits
- T commits only if no other concurrent transaction has already written the data that T intends to write.

First Committer Wins

First Committer Wins

T_1 : deposits 40 in X	T_2: deposits 70 in X
R(X, 100)	
	R(X, 100)
	W(X, 170)
W(X, 140)	
Commit	
	Commit request : Serialization problem is detected by SI. ABORT! Avoids lost update anomaly.

Anomaly: Write Skew (with updates)

Constraint: $X+Y>=0$

Initially, $X = 100$ and $Y = 0$

Anomaly: Write Skew (with updates)

Constraint: X+Y>=0
Initially, X = 100 and Y = 0

T_1 : Withdraw 70 from X	**T_2: Withdraw 90 from Y**
R(X, 100)	
R(Y, 0)	
	R(X, 100)
	R(Y, 0)
	W(Y, -90)
W(X, 30)	
	Commit
Commit	

Anomaly: Write Skew (with updates)

Constraint: X+Y>=0
Initially, X = 100 and Y = 0

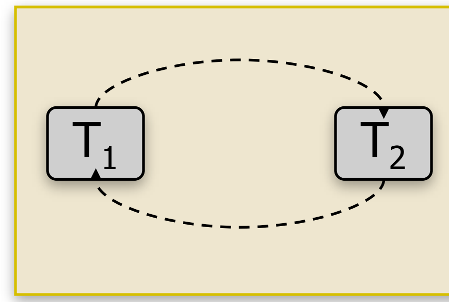
T_1 : Withdraw 70 from X	T_2: Withdraw 90 from Y
R(X, 100)	
R(Y, 0)	
	R(X, 100)
	R(Y, 0)
	W(Y, -90)
W(X, 30)	
	Commit
Commit	

X+Y= – 60

Anomaly: Write Skew (with updates)

Constraint: X+Y>=0
Initially, X = 100 and Y = 0

T_1 : Withdraw 70 from X	T_2: Withdraw 90 from Y
R(X, 100)	
R(Y, 0)	
	R(X, 100)
	R(Y, 0)
	W(Y, -90)
W(X, 30)	
	Commit
Commit	

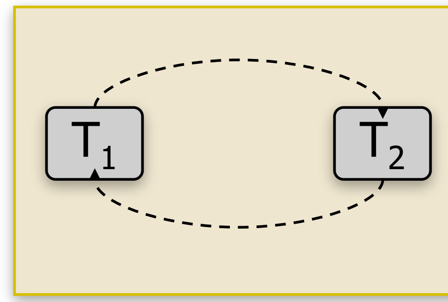


X+Y= – 60

Anomaly: Write Skew (with updates)

Constraint: X+Y>=0
Initially, X = 100 and Y = 0

T_1 : Withdraw 70 from X	T_2: Withdraw 90 from Y
R(X, 100)	
R(Y, 0)	
	R(X, 100)
	R(Y, 0)
	W(Y, -90)
	Commit



X+Y= – 60

Dependency is called vulnerable under SI if it does not prevent transactions from executing concurrently.
E.g., the rw dependency without ww dependency is vulnerable.

Anomaly: Write Skew (with Inserts)

1. A voucher with unique voucher# is to be created for every bill
2. Programmer codes :

    m = select max(vno) ;
    insert new tuple (billno, voucher#=m+1)

3. Let max(vno)=10 and new vouchers for billnumbers X and Y are to be created

Anomaly: Write Skew (with Inserts)

1. A voucher with unique voucher# is to be created for every bill
2. Programmer codes :

    m = select max(vno) ;
    insert new tuple (billno, voucher#=m+1)

3. Let max(vno)=10 and new vouchers for billnumbers X and Y are to be created

T_1	T_2
R(max(vno), 10)	
	R(max(vno), 10)
Insert (X,11)	
	Insert (Y, 11)
	commit
commit	

Anomaly: Write Skew (with Inserts)

1. A voucher with unique voucher# is to be created for every bill
2. Programmer codes :

   m = select max(vno) ;

   insert new tuple (billno, voucher#=m+1)

3. Let max(vno)=10 and new vouchers for billnumbers X and Y are to be created

T_1	T_2
R(max(vno), 10)	
	R(max(vno), 10)
Insert (X,11)	
	Insert (Y, 11)
	commit
commit	

Duplicate voucher# created!

Detecting Anomalies: Static Analysis

Goal is to ensure that **every** possible execution in given application is serializable (not just a particular execution).

1) Application consists of transaction programs
    - from which different transactions are generated depending on
        - the control structures
        - the parameter values
2) Transactions might interleave in different ways.
3) Hence, it is infeasible to enumerate every possible execution.

Dependencies should be identified
- Between transaction-programs
- for every possible interleaving of transaction programs

Detecting Anomalies: Static Analysis

SDG: Static Dependency Graph [Fekete et al. TODS'05]

Nodes : Transaction Programs as nodes.
Edges : Let T_1 and T_2 be any execution instances of transaction program P_1 and P_2 respectively

- $P_1 \rightarrow P_2$ if there can exist some T_1 that conflicts with some T_2
- it is marked vulnerable if dependency does not prevent concurrent execution $P_1 \xrightarrow{VUL} P_2$

Detecting Anomalies: Static Analysis

SDG: Static Dependency Graph [Fekete et al. TODS'05]

Nodes : Transaction Programs as nodes.
Edges : Let T_1 and T_2 be any execution instances of transaction program P_1 and P_2 respectively

- $P_1 \rightarrow P_2$ if there can exist some T_1 that conflicts with some T_2
- it is marked vulnerable if dependency does not prevent concurrent execution $P_1 \xrightarrow{VUL}® \; P_2$

Conditions for Vulnerability

rw conflict from T_1 to T_2 without ww conflict.

Detecting Anomalies: Static Analysis

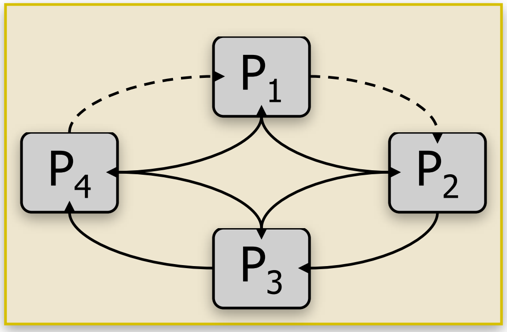
SDG: Static Dependency Graph [Fekete et al. TODS'05]

Nodes : Transaction Programs as nodes.
Edges : Let T_1 and T_2 be any execution instances of transaction program P_1 and P_2 respectively

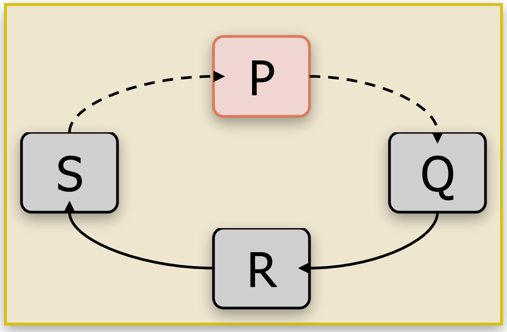
- $P_1 \rightarrow P_2$ if there can exist some T_1 that conflicts with some T_2
- it is marked vulnerable if dependency does not prevent concurrent execution $P_1 \xrightarrow{VUL} P_2$

Conditions for Vulnerability

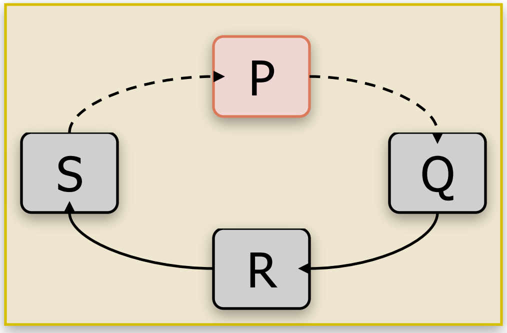
rw conflict from T_1 to T_2 without ww conflict.



Detecting Anomalies: Static Analysis



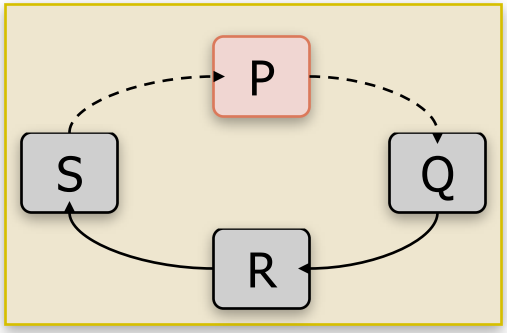
Detecting Anomalies: Static Analysis



Pivot

A transaction program P is a pivot if in static dependency graph (SDG), there is a **cycle** containing subpath with $S \xrightarrow{\text{VUL}} P \xrightarrow{\text{VUL}} Q$

Detecting Anomalies: Static Analysis



Pivot

A transaction program P is a pivot if in static dependency graph (SDG), there is a **cycle** containing subpath with $S \xrightarrow{\text{VUL}} P \xrightarrow{\text{VUL}} Q$

Theorem [Fekete TODS'05]

Absence of pivot implies serializable execution under SI.

Transaction Programs in SQL Language

Identifying Set of Transaction Programs (SQL)

1. Program Analysis.
    - May not be possible for large applications.
2. SQL traces at backend.
    - May not cover all the transaction programs.

We apply our analysis to the set of transaction programs obtained.

Characteristics of Transaction Programs (in SQL)

- SQL statements

  SELECT, INSERT, DELETE etc.
- Parameterization

  WHERE col=:UserInput

Identifying Dependencies

Identifying Dependencies

rset(P) (resp. wset(P)) is the set of columns read (resp. written) by P

Identifying Dependencies

rset(P) (resp. wset(P)) is the set of columns read (resp. written) by P

Update Customer Information Transaction Program (UCI)

```
begin;
        select * from customer where id=:id;
        update customer set name=?, address=? where id=:id;
commit;
```

Identifying Dependencies

rset(P) (resp. wset(P)) is the set of columns read (resp. written) by P

Update Customer Information Transaction Program (UCI)

```
begin;
        select * from customer where id=:id;
        update customer set name=?, address=? where id=:id;
commit;
```

rset(UCI) ={customer.id, customer.name, customer.address}
wset(UCI) ={customer.name, customer.address}

Syntactic Column-based Analysis of Transaction Programs

Column-based Syntactic Dependency Graph (CSDG)

Syntactic Column-based Analysis of Transaction Programs

Column-based Syntactic Dependency Graph (CSDG)

- nodes are transaction programs.
- an edge is marked as pseudovulnerable (PVUL) whenever

  $rset(Pi) \cap wset(Pj) \neq \Theta$

  - $wset(Pi) \cap wset(Pj) \neq \Theta$ does not imply ww conflict

Syntactic Column-based Analysis of Transaction Programs

Column-based Syntactic Dependency Graph (CSDG)

- nodes are transaction programs.
- an edge is marked as pseudovulnerable (PVUL) whenever
  rset(Pi) ∩ wset(Pj) ≠ Θ
  - wset(Pi) ∩ wset(Pj) ≠ Θ does not imply ww conflict

P_B is a **syntactic pseudopivot** if some cycle of edges in CSDG contains a subpath $P_A \xrightarrow{PVUL}® P_B \xrightarrow{PVUL}® P_C$

Syntactic Column-based Analysis of Transaction Programs

Column-based Syntactic Dependency Graph (CSDG)

- nodes are transaction programs.
- an edge is marked as pseudovulnerable (PVUL) whenever
  $rset(Pi) \cap wset(Pj) \neq \Theta$
  - $wset(Pi) \cap wset(Pj) \neq \Theta$ does not imply ww conflict

P_B is a **syntactic pseudopivot** if some cycle of edges in CSDG contains a subpath $P_A \xrightarrow{PVUL} P_B \xrightarrow{PVUL} P_C$

Note: Every pivot is a syntactic pseudopivot. [but not vice-versa]

Syntactic Column-based Analysis of Transaction Programs

Column-based Syntactic Dependency Graph (CSDG)

- nodes are transaction programs.
- an edge is marked as pseudovulnerable (PVUL) whenever
  rset(Pi) ∩ wset(Pj) ≠ Θ
  - wset(Pi) ∩ wset(Pj) ≠ Θ does not imply ww conflict

P_B is a **syntactic pseudopivot** if some cycle of edges in CSDG contains a subpath $P_A \xrightarrow{\text{PVUL}} P_B \xrightarrow{\text{PVUL}} P_C$

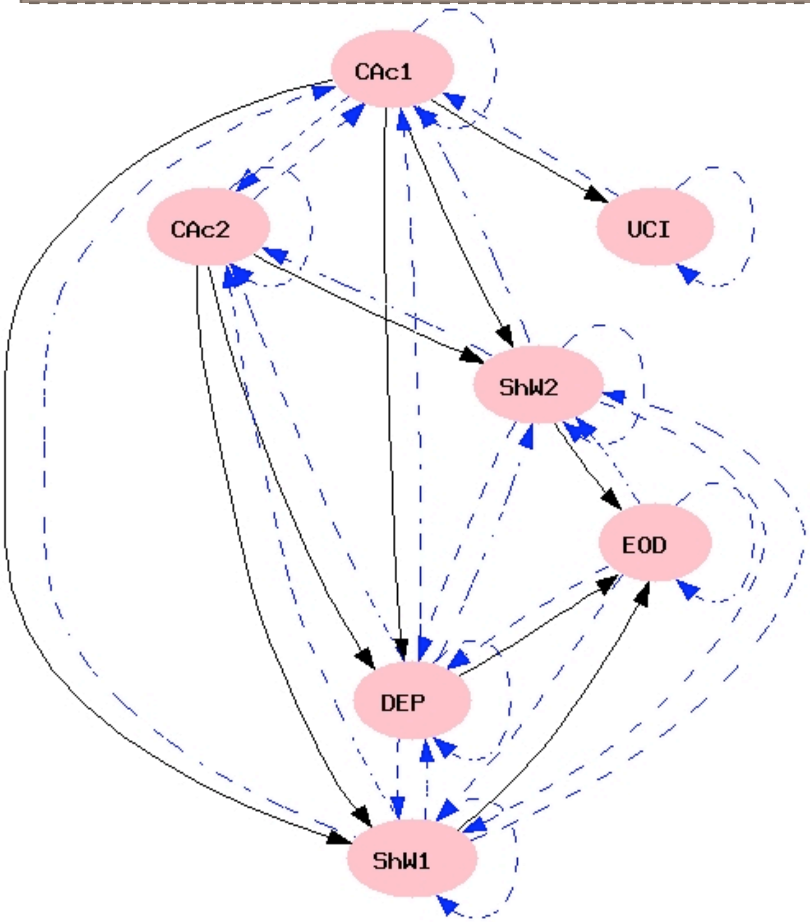
Note: Every pivot is a syntactic pseudopivot. [but not vice-versa]

Theorem

If a set of transaction programs contain no syntactic pseudopivots, then every execution under SI will in fact be serializable.

False Positives

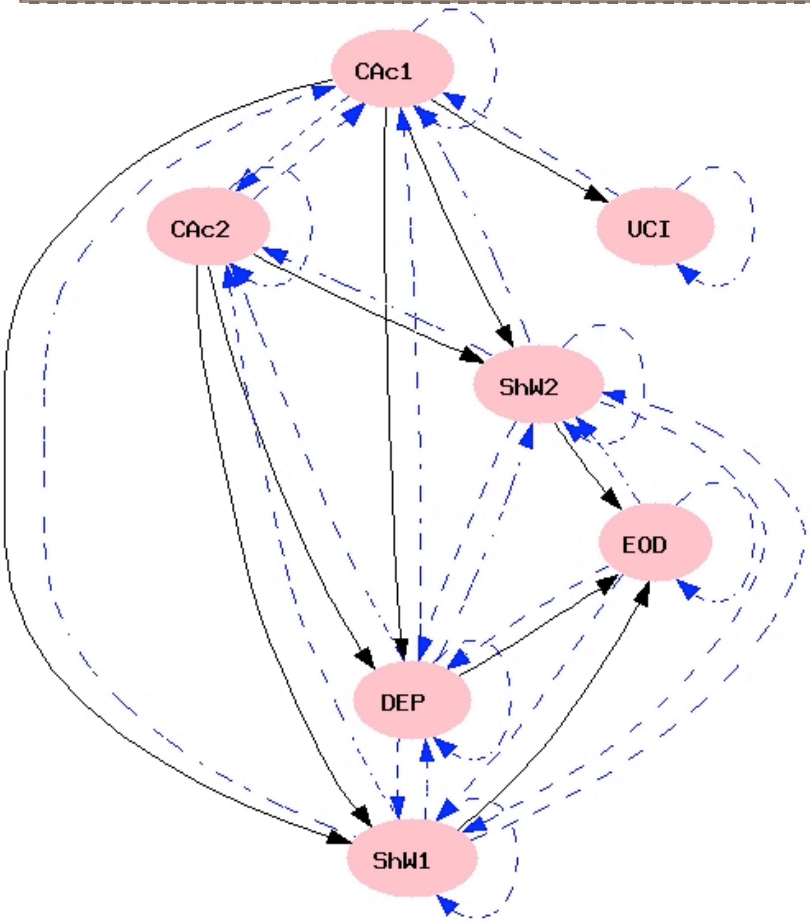
CSDG for Banking Application



Pink nodes: syntactic pseudopivots

False Positives

CSDG for Banking Application



Pink nodes: syntactic pseudopivots

False positive

Many transactions which can never cause any anomaly are detected as syntactic pseudopivots.

Eliminating False Positives 1: Modification Protected Readset

Eliminating False Positives 1: Modification Protected Readset

Update Customer Information Transaction Program (UCI)

```
begin;
        select * from customer where id=:id;
        update customer set name=?, address=? where id=:id;
commit;
```

Eliminating False Positives 1: Modification Protected Readset

Update Customer Information Transaction Program (UCI)

```
begin;
        select * from customer where id=:id;
        update customer set name=?, address=? where id=:id;
commit;
```

rset(UCI) ={customer.id, customer.name, customer.address}
wset(UCI) ={customer.name, customer.address}

Eliminating False Positives 1: Modification Protected Readset

Update Customer Information Transaction Program (UCI)

```
begin;
        select * from customer where id=:id;
        update customer set name=?, address=? where id=:id;
commit;
```

rset(UCI) ={customer.id, customer.name, customer.address}
wset(UCI) ={customer.name, customer.address}

- UCI has a pseudovulnerable **self** edge
  - due to syntactic conflict between select and update

  seems to imply two copies of UCI could create an anomaly
- But selected row is updated subsequently so first committer wins, the other aborts

Eliminating False Positives 1: Modification Protected Readset

Update Customer Information Transaction Program (UCI)

```
begin;
        select * from customer where id=:id;
        update customer set name=?, address=? where id=:id;
commit;
```

rset(UCI) ={customer.id, customer.name, customer.address}
wset(UCI) ={customer.name, customer.address}

- UCI has a pseudovulnerable **self** edge
  - due to syntactic conflict between select and update

  seems to imply two copies of UCI could create an anomaly
- But selected row is updated subsequently so first committer wins, the other aborts

Modification Protected Readset (MPR)

Eliminating False Positives 2: New Identifier Generation Test

```
begin;
    select max(accno)+1 as m from account;
    insert into account(accno, balance, type) values (:m, 0, :type);
Commit;
```

Eliminating False Positives 2: New Identifier Generation Test

```
begin;
    select max(accno)+1 as m from account;
    insert into account(accno, balance, type) values (:m, 0, :type);
Commit;
```

Select max() ... Insert

- for assigning new primary key (numeric)
- if two transactions read same max value and create same identifier, SI will not prevent concurrent execution
  - but primary key constraint will!
    - Checked outside snapshot

Eliminating False Positives 3: Existence Check Before Insert

Eliminating False Positives 3: Existence Check Before Insert

Select with given PK ... if not found (Insert values with same PK)

- Select using primary key can not conflict with Insert of other transaction having same pattern.

Eliminating False Positives 3: Existence Check Before Insert

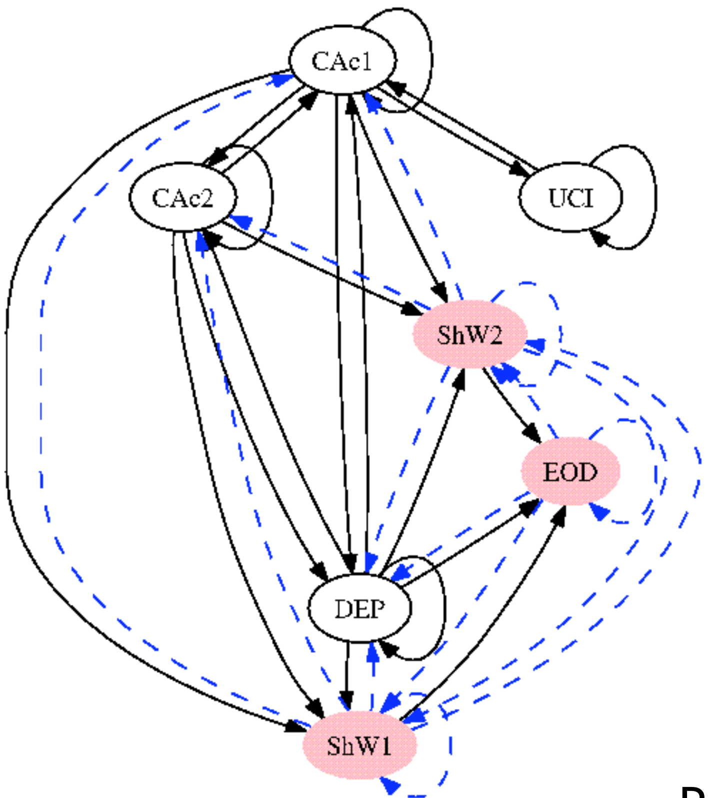
Select with given PK ... if not found (Insert values with same PK)

- Select using primary key can not conflict with Insert of other transaction having same pattern.

```
begin;
    select accno as found from account where accno=:m;
    if(found==null)
        insert into account values (:m, 0, :type);
    else
            print ‘Error: Requested account number is already in use’;
    endif
commit;
```

After Eliminating False Positives

CSDG for Banking Application



Eliminated False Positives

1. UCI: MPR
2. DEP: MPR
3. CAc1 & CAc2: EFP1

Remaining Syntactic Psuedopivots

1. ShW1 & ShW2 (Write Skew with Updates)
2. EOD (Write Skew with Insert)

Pink nodes: remaining syntactic pseudopivots

Analyzing an Application

1. Find the set of transaction programs.
2. Create CSDG using Syntactic Analysis and detect syntactic pseudopivots.
3. Reduce false positives.
4. Select appropriate techniques to avoid anomalies (manual)

After using the techniques to avoid anomalies we can rerun the analysis to check whether they worked.

Results

Acad and Finance: Real life applications in use at IITB

	TPC-C	Bank	Acad.	Finance
Distinct transactions	7	7	26	34
Syntactic Pseudopivots detected	4	7	25	34
EFP1: MPR detected	3	2	11	4
EFP2: New Identifier Generation Protection detected	0	2	3	3
EFP3: Existence Check before Insert Protection	0	0	2	0

*: there may be more pivots, we don't have application code

Conclusion

Contributions

1. Theory of Syntactic Analysis to obtain a superset of transactions that may cause anomalies.

2. Studied some general patterns of false positives and proposed sufficient conditions for identifying such transactions.

3. Developed a tool that can automate the testing of database applications for safety against SI anomalies
   - identified some genuine problems in production code.

Conclusion

Future work

1. Automating the fixing of the anomalies :
   - Developing a generic technique to decide what conflicts to materialize.
   - Efficient approximation algorithms to minimize promotions added to remove anomalies (NP hardness shown in paper).

2. Identifying more false positives :
   1. Developing a theory for including workflow constraints .
   2. Detecting FPs due to integrity constraints.
   3. Identifying some more transaction patterns.

Thank You!