PRISM++
Update Rewriting and Integrity Constraint Maintenance

Carlo Curino
Hyun J. Moon, Alin Deutsch, Carlo Zaniolo
Motivation

- Information Systems Evolution... it’s hard!
• Information Systems Evolution... it’s hard!
• Information Systems Evolution... it’s hard!
Motivation

- Information Systems Evolution... it’s hard!
Motivation

• Information Systems Evolution... it’s hard!
Motivation

• Information Systems Evolution... it’s hard!
Motivation: Schema Evolution

- change schema
- migrate data
- fix queries/updates
- check/modify app code
### Motivation: Schema Evolution

The total number of query and update templates is typically rather small (less than a thousand for Wikipedia), therefore, the cache sub-model can be stored in RAM directly. To this purpose, we adapted the template name of the DB they were targeting (Wikipedia has many DB sharing an identical schema). To this purpose, we adapted the template name of the DB they were targeting (Wikipedia has many DB sharing an identical schema).

The experiments have been conducted on a system with the HW/SW environment not required (e.g., prior to the schema version used in the experiments). In addition, pruning removes all the schema versions from a schema indirectly reachable via foreign keys from the initial footprint and the other mappings that are not necessary for the rewriting (i.e., predicates and (iv-a) substitute the parameters with the original input values).

The last question that remains to be answered is how to translate the C&B rewriting procedure by reusing results produced for similar structural SMOs (ICMO rewriting is not based on the complement, i.e., what to keep). Update shares the same issues involving joins with other tables), while propagating the tables to be rewritten.

- **ATutor**: Educational CMS, # of schema versions 216, lifetime (years) 5.7
- **CERN DQ2**: Scientific DB, # of schema versions 51, lifetime (years) 1.3
- **Dekiwiki**: CRM, ERP, # of schema versions 11, lifetime (years) 1.11
- **E107**: CMS, # of schema versions 16, lifetime (years) 5.4
- **Ensembl**: Scientific DB, # of schema versions 412, lifetime (years) 9.8
- **KT-DMS**: CMS, # of schema versions 105, lifetime (years) 4
- **Nucleus CMS**: CMS, # of schema versions 51, lifetime (years) 6.7
- **PHPWiki**: Wiki, # of schema versions 18, lifetime (years) 4.11
- **SlashCode (slashdot.org)**: News Website, # of schema versions 256, lifetime (years) 8.10
- **Tikiwiki**: Wiki, # of schema versions 99, lifetime (years) 0.9
- **Mediawiki (Wikipedia.org)**: Wiki, # of schema versions 242, lifetime (years) 6.2
- **Zabbix**: Monitoring solution, # of schema versions 196, lifetime (years) 8.3

### Average of 31 schema version per year
### Motivation: Schema Evolution

#### Table 5: Evolution histories of popular IS in our dataset

<table>
<thead>
<tr>
<th>System Name</th>
<th>System type</th>
<th># of schema versions</th>
<th>lifetime (years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATutor</td>
<td>Educational CMS</td>
<td>216</td>
<td>5.7</td>
</tr>
<tr>
<td>CERN DQ2</td>
<td>Scientific DB</td>
<td>51</td>
<td>1.3</td>
</tr>
<tr>
<td>Dekiwiki</td>
<td>CRM, ERP</td>
<td>11</td>
<td>1.11</td>
</tr>
<tr>
<td>E107</td>
<td>CMS</td>
<td>16</td>
<td>5.4</td>
</tr>
<tr>
<td>Ensembl</td>
<td>Scientific DB</td>
<td>412</td>
<td>9.8</td>
</tr>
<tr>
<td>KT-DMS</td>
<td>CMS</td>
<td>105</td>
<td>4</td>
</tr>
<tr>
<td>Nucleus CMS</td>
<td>CMS</td>
<td>51</td>
<td>6.7</td>
</tr>
<tr>
<td>PHPWiki</td>
<td>Wiki</td>
<td>18</td>
<td>4.11</td>
</tr>
<tr>
<td>SlashCode (slashdot.org)</td>
<td>News Website</td>
<td>256</td>
<td>8.10</td>
</tr>
<tr>
<td>Tikiwiki</td>
<td>Wiki</td>
<td>99</td>
<td>0.9</td>
</tr>
<tr>
<td>Mediawiki (Wikipedia.org)</td>
<td>Wiki</td>
<td>242</td>
<td>6.2</td>
</tr>
<tr>
<td>Zabbix</td>
<td>Monitoring solution</td>
<td>196</td>
<td>8.3</td>
</tr>
</tbody>
</table>

- **Average of 31 schema version per year**
Our Previous Work

<table>
<thead>
<tr>
<th></th>
<th>Structural Evolution</th>
<th>Integrity Constraints Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data</strong></td>
<td>✓</td>
<td>❌</td>
</tr>
<tr>
<td><strong>Queries</strong></td>
<td>✓</td>
<td>❌</td>
</tr>
<tr>
<td><strong>Updates</strong></td>
<td>❌</td>
<td>❌</td>
</tr>
</tbody>
</table>

- Schema Modification Operators (SMOs)
- Query rewriting engine based on *chase* & *backchase*
What are we going to do?

- Integrity Constraints Evolution
  - Introduce integrity-constraint mod. operators (ICMOs)
  - Adapt schema modification operators (SMOs)
- Updates (and queries with negation)
  - Novel update representation (query equivalence)
  - Extended rewriting engine (support for negation and ICMOs)
Evolution Operators

• Key idea: separate structural changes (SMOs) from non-information preserving* ones (ICMOs)

*information-preserving = invertible mapping = constant information-capacity
Evolution Operators

• Key idea: separate structural changes (SMOs) from non-information preserving* ones (ICMOs)

*information-preserving = invertible mapping = constant information-capacity

Schema v1

<table>
<thead>
<tr>
<th>R</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
</table>

Wednesday, August 31, 11
Evolution Operators

- Key idea: separate structural changes (SMOs) from non-information preserving* ones (ICMOs)
Evolution Operators

- Key idea: separate structural changes (SMOs) from non-information preserving* ones (ICMOs)

DECOMPOSE R INTO S(a,b), T(a,c);

ALTER TABLE T DROP FOREIGN KEY fk2;
**Evolution Operators**

- **Key idea:** separate structural changes (SMOs) from non-information preserving* ones (ICMOs)

**Schema v1**

```
R | a | b | c
```

- no changes to schema structure
- not information preserving

**Schema v1.1**

```
S | a | b |
T | a | c |
```

- changes to schema structure
- information preserving

**DECOMPOSE R INTO S(a,b), T(a,c);**

**Schema v2**

```
S | a | b |
T | a | c |
```

- no changes to schema structure
- not information preserving

**ALTER TABLE T DROP FOREIGN KEY fk2;**
Good/Bad News

• We force every SMO to be information-preserving (data migration and query rewriting paradise!)

• ICMOs:
  • risk of data loss
  • rewriting not obvious (new alg.)
  • inverse operator (user input)
• Challenge: *migrating towards a “tighter” schema (data loss)*
Data Migration

• Challenge: migrating towards a “tighter” schema (data loss)

```
ALTER TABLE S
ADD PRIMARY KEY pk1(a)
<policy>;
```
Data Migration

- Challenge: migrating towards a “tighter” schema (data loss)

  ```sql
  ALTER TABLE S
  ADD PRIMARY KEY pk1(a)
  <policy>;
  ```

- `<policy>`:
  - **CHECK**: migrates data only if constraint already holds
  - **ENFORCE**: “canonical repair” by moving all violating tuples to special table

  ![Diagram](image)
Query Rewriting

- Challenge: evolution towards a "looser" schema (inverse is not inf-preserving)
Query Rewriting

• Challenge: evolution towards a “looser” schema (inverse is not inf-preserving)

```sql
ALTER TABLE T
DROP FOREIGN KEY fk2;
```
Query Rewriting

• Challenge: evolution towards a “looser” schema (inverse is not inf-preserving)

```
ALTER TABLE T
DROP FOREIGN KEY fk2;

ALTER TABLE T
ADD FOREIGN KEY fk2(a)
REFERENCES S(a) <policy>;
```
Query Rewriting

- Challenge: evolution towards a “looser” schema (inverse is not inf-preserving)

```
ALTER TABLE T
DROP FOREIGN KEY fk2;

ALTER TABLE T
ADD FOREIGN KEY fk2(a)
REFERENCES S(a) <policy>;
```

- `<policy>`:
  - CHECK: checks constraint before running query
  - ENFORCE: limits query scope to non-violating tuples
  - IGNORE: runs query as-is
Query Rewriting Example

- ENFORCE: limits query scope to non-violating tuples
**Query Rewriting Example**

- **ENFORCE:** limits query scope to non-violating tuples

```
ALTER TABLE S
ADD PRIMARY KEY pk1(a)
ENFORCE;
```

```
ALTER TABLE S
DROP PRIMARY KEY;
```
**Query Rewriting Example**

- **ENFORCE**: limits query scope to non-violating tuples

```
SELECT a, b FROM S
WHERE a=1;
```

```
ALTER TABLE S
DROP PRIMARY KEY;
```

```
ALTER TABLE S
ADD PRIMARY KEY pk1(a)
ENFORCE;
```
Query Rewriting Example

Q

SELECT a,b FROM S
WHERE a=1;

S a b

Q’

SELECT a,b FROM S
WHERE a=1 and NOT EXISTS
(SELECT * FROM S as s2
WHERE S.a=s2.a AND
S.b!=s2.b);

ALTER TABLE S
DROP PRIMARY KEY;

S a b

ALTER TABLE S
ADD PRIMARY KEY pk1(a)
ENFORCE;

ENFORCE: limits query scope to non-violating tuples
Negation...

- Intuition: *rewrite independently negative and positive part of the query*
**Intuition:** rewrite independently negative and positive part of the query

```
SELECT a, b FROM S
WHERE a = 1 AND NOT EXISTS (SELECT * FROM S as s2
WHERE S.a = s2.a AND S.b != s2.b);
```

**Q’**

```
RENAME COLUMN b
IN S TO x;
```

<table>
<thead>
<tr>
<th>S</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>S</th>
<th>a</th>
<th>x</th>
</tr>
</thead>
</table>
Negation...

- Intuition: *rewrite independently negative and positive part of the query*

\[ \text{SELECT } a, b \text{ FROM } S \]
\[ \text{WHERE } a=1 \text{ and NOT EXISTS (SELECT * FROM } S \text{ as } s2 \]
\[ \text{WHERE } S.a=s2.a \text{ AND } S.b!=s2.b) \];

\[ \text{RENAME COLUMN } b \]
\[ \text{IN } S \text{ TO } x; \]

\[ \text{SELECT } a, x \text{ FROM } S \]
\[ \text{WHERE } a=1 \text{ and NOT EXISTS (SELECT * FROM } S \text{ as } s2 \]
\[ \text{WHERE } S.a=s2.a \text{ AND } S.x!=s2.x) \];
• Intuition: rewrite independently negative and positive part of the query

Q':

```
SELECT a,b FROM S
WHERE a=1 and NOT EXISTS
(SELECT * FROM S as s2
WHERE S.a=s2.a AND
S.b!=s2.b);
```

Q'':

```
SELECT a,x FROM S
WHERE a=1 and NOT EXISTS
(SELECT * FROM S as s2
WHERE S.a=s2.a AND
S.x!=s2.x);
```

This application of Chase & Back-Chase is sound but not complete
So far...

<table>
<thead>
<tr>
<th></th>
<th>Structural Evolution</th>
<th>Integrity Constraints Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data</strong></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Queries</strong></td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td><strong>Updates</strong></td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

- Introduce ICMOs, Adapted SMOs
- Extended Query Rewriting Engine (ICMOS + neg.)
• Intuition: reuse query rewriting engine to tackle update rewriting
Intuition: *reuse query rewriting engine to tackle update rewriting*
Update Rewriting (through SMOs)

- Intuition: reuse query rewriting engine to tackle update rewriting
• Intuition: *reuse query rewriting engine to tackle update rewriting*
Update Rewriting (through SMOs)

- Intuition: reused query rewriting engine to tackle update rewriting
Intuition: *represent updates as (equivalence between) queries, exploit query rewriting*
• Intuition: represent updates as (equivalence between) queries, exploit query rewriting

UPDATE S
SET b=7
WHERE a=1;

SELECT a,7 FROM S WHERE a=1;
UNION
SELECT a,b FROM S WHERE a!=1;

SELECT a,b FROM S;

Q_{before} == Q_{after}
Intuition: *the policies specify popular special-cases of view-update problem*
Update Rewriting (through ICMOs)

- Intuition: the policies specify popular special-cases of view-update problem

- `<policy>`:
  - CHECK: checks constraint before and after running update
  - ENFORCE: limits update scope to non-violating tuples, checks violation-set is not changed
  - IGNORE: runs update as-is (allows side effects)
Optimization

- **Challenge:** rewriting complexity depends on mapping size (foreign keys and ICMOs make things harder)
- **Solution:** extract templates, cache rewritings

![Graph](image)

- **Ensembl genetic DB**
- **Synthetic Dataset**

**Wikipedia hit/miss ratio:** up to 88M
Conclusion

- Prism++ is a high-performance practical system supporting DB schema-evolution:

<table>
<thead>
<tr>
<th></th>
<th>Structural Evolution</th>
<th>Integrity Constraints Evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Queries</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Updates</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

For More info contact me: krl@yahoo-inc.com
• Solution: effectiveness of template caching

<table>
<thead>
<tr>
<th>Statement type</th>
<th>number of templates</th>
<th>avg hit/miss ratio</th>
<th>max hit/miss ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>update</td>
<td>142</td>
<td>5,661.21</td>
<td>80,870</td>
</tr>
<tr>
<td>select</td>
<td>1294</td>
<td>248,005.41</td>
<td>88,740,689</td>
</tr>
<tr>
<td>select*</td>
<td>610</td>
<td>526,096.72</td>
<td>88,740,689</td>
</tr>
</tbody>
</table>

*with improved template extraction factorizing DB names.

<table>
<thead>
<tr>
<th>Statements</th>
<th>execution time</th>
<th>rewriting time</th>
<th>overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>77.37 ms</td>
<td>1 ms</td>
<td>1.29%</td>
</tr>
<tr>
<td>S2</td>
<td>21.674 ms</td>
<td>1 ms</td>
<td>4.6%</td>
</tr>
<tr>
<td>S3</td>
<td>48.2 ms</td>
<td>1 ms</td>
<td>2.07%</td>
</tr>
</tbody>
</table>
Chase & BackChase

• Intuition: behind the scene Disjunctive Embedded Dependencies and chase-based re rewritings
Update Rewriting through SMO: Example

**UPDATE exon SET end="342" WHERE id=1**

**UPDATE eregion r, etype t SET r.end = "342" WHERE r.id=t.id**

**SELECT r.id, type, region, start, end WHERE r.id=1 AND r.id=t.id**

**SELECT id, type, region, start, end FROM exon WHERE id!=1**

**SELECT id, type, region, start, end FROM exon WHERE id!=1 FROM exon**

**UPDATE exon SET end="342" WHERE id=1**

**UPDATE eregion r, etype t SET r.end = "342" WHERE r.id=t.id**
Update Rewriting ICMOs: Example

```
ALTER TABLE exon
DROP PRIMARY KEY pk1

ALTER TABLE exon
ADD PRIMARY KEY pk1(id)
CHECK

@pre = SELECT * FROM exon e,exon e2
    WHERE e.id=e2.id AND e.rank=e2.rank AND
        (e.type!=e2.type OR e.start!=e2.start OR e.end!=e2.end);
@post = SELECT * FROM exon e WHERE e.id=1;

IF(isempty(@pre)&& isempty(@post)) INSERT INTO exon VALUES(1,2,3,4,5)
ELSE RETURN ERROR;
```