RuleGen

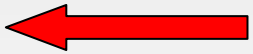
Introduction / Demo

RULE·GEN
data quality assurance

Overview

- RuleGen: what, why and how?
- Transition effect
  - Transition effect query
  - Constraint validation query
- Execution model
- Framework components
- Demo
- Use cases
- Demo
- Service procedures

What is RuleGen?

- A framework written in pl/sql that generates pl/sql to maintain data integrity constraints
  - Static constraints
    - Attribute constraints — SQL-check constraint
    - Tuple constraints — SQL-check constraint
    - Table constraints  RuleGen
    - Database constraints  RuleGen
  - Transition constraints — Future release
    - Require transaction context
- RuleGen is for professional database developers
  - Not for end-users...

Why do this *in* the DBMS?

1. To ensure data integrity! ☺
2. To be immune for the ongoing yafet(*) technology explosion outside the DBMS
   - Your data will survive many yafet technology revolutions

0. Because, like PK's and FK's, this is where this code should be

(*) Yet Another Frontend Technology
(see: TheHelsinkiDeclaration.blogspot.com)

How?

- CSF1: A database design driven development approach
  - Resulting in sound relational design
- CSF2: A seperate *database team*
  - Responsible for implementation of database design
    - Including all integrity constraints
    - Proper tooling ➔ RuleGen

Transition Effect

- Main concept of RuleGen

- Transactions change our database (state)
- They constitute a *transition*
  - Old database-state → new database-state
  - New database-state needs to be validated

- Transition effect describes the rows that were transacted

Transition Effect

- We can transact (DML) in three different ways
  - Insert statement
  - Update statement
  - Delete statement
- Correspondingly we have three transition effects
  - Inserted-rows (Insert Transition Effect: ITE)
  - Updated-rows (UTE)
  - Deleted-rows (DTE)

Transition Effect: Inserted-rows

empno	ename	job	sal	deptno
101	'Chris'	'MANAGER'	7900	10
102	'Kathy'	'TRAINER'	6000	12
103	'Thomas'	'CLERK'	2100	10
104	'David'	'TRAINER'	5600	10
105	'Renu'	'CLERK'	3000	12
106	'Bob'	'MANAGER'	8500	10
107	'Sue'	'CLERK'	2700	12

- **`Insert into emp(empno,ename,job,sal,deptno) values(108,'Toon','CEO','10000',10);`**

empno	ename	job	sal	deptno
108	'Toon'	'CEO'	10000	10

ITE

Transition Effect: Inserted-rows

empno	ename	job	sal	deptno
101	'Chris'	'MANAGER'	7900	10
102	'Kathy'	'TRAINER'	6000	12
103	'Thomas'	'CLERK'	2100	10
104	'David'	'TRAINER'	5600	10
105	'Renu'	'CLERK'	3000	12
106	'Bob'	'MANAGER'	8500	10
107	'Sue'	'CLERK'	2700	12

- **Insert into emp(empno,ename,job,sal,deptno)**
  (select 107+rownum,ename,job,sal,deptno
  from EMP
  where job='TRAINER');

empno	ename	job	sal	deptno
108	'Kathy'	'TRAINER'	6000	12
109	'David'	'TRAINER'	5600	10

ITE

Transition Effect: Updated-rows

empno	ename	job	sal	deptno
101	'Chris'	'MANAGER'	7900	10
102	'Kathy'	'TRAINER'	6000	12
103	'Thomas'	'CLERK'	2100	10
104	'David'	'TRAINER'	5600	10
105	'Renu'	'CLERK'	3000	12
106	'Bob'	'MANAGER'	8500	10
107	'Sue'	'CLERK'	2700	12

- **`Update EMP set sal=sal+100 where job='CLERK';`**

UTE

old Empno	new Empno	old Ename	new Ename	old Job	new Job	old Sal	new Sal	old Deptno	new Deptno
103	103	'Thomas'	'Thomas'	'CLERK'	'CLERK'	2100	2200	10	10
105	105	'Renu'	'Renu'	'CLERK'	'CLERK'	3000	3100	12	12
107	107	'Sue'	'Sue'	'CLERK'	'CLERK'	2700	2800	12	12

Transition Effect: Deleted-rows

empno	ename	job	sal	deptno
101	'Chris'	'MANAGER'	7900	10
102	'Kathy'	'TRAINER'	6000	12
103	'Thomas'	'CLERK'	2100	10
104	'David'	'TRAINER'	5600	10
105	'Renu'	'CLERK'	3000	12
106	'Bob'	'MANAGER'	8500	10
107	'Sue'	'CLERK'	2700	12

- **`Delete from EMP`**
  `where deptno=12;`

empno	ename	job	sal	deptno
102	'Kathy'	'TRAINER'	6000	12
105	'Renu'	'CLERK'	3000	12
107	'Sue'	'CLERK'	2700	12

DTE

Transition Effect (TE)

- RuleGen maintains the transition effect for you
- Transition effect is very useful to determine whether a given constraint needs to be validated at all
- By querying TE and looking for *constraint specific property* you can tell whether constraint can potentially be violated

Example

- "At most one president allowed in EMP"
- Only if inserted-rows show that a president has just been inserted, should we check this constraint
- In all other insert cases, constraint validation is not needed

Example

- “At most one president allowed in EMP”
- Only if updated-rows show that ...
- In all other update cases, constraint validation is not needed

Example

- “At most one president allowed in EMP”
- Only if deleted-rows show that ...

Transition Effect Queries

- For every constraint you supply RuleGen with three queries telling RuleGen when to check a constraint
  - TE-query on inserted-rows
  - TE-query on updated-rows
  - TE-query on deleted-rows
- If these queries return rows ➔ constraint must be validated
- If these queries return no rows ➔ constraint validation not necessary

Back to the Example

- "At most one president allowed in EMP"

- ```
  Select distinct 'x'
  from inserted_rows
  where job='PRESIDENT';
  ```
  ```

Back to the Example

- "At most one president allowed in EMP"

- ```
  Select distinct 'x'
  from updated_rows
  where old_job<>'PRESIDENT'
    and new_job='PRESIDENT';
  ```
  ```

Back to the Example

- "At most one president allowed in EMP"

- ```
  Select 'x'
  from deleted_rows
  where 0=1;
  ```
  ```

Execution Model

- RuleGen will validate the constraint as many times as rows are returned by the TE-query (done by after-statement trigger)
- **For r in (TE-query)**
  Loop
  <validate the constraint>
  End loop;

The DISTINCT in the TE-query
prevents the constraint being validated too many times

Execution Model

- You supply RuleGen with another query to **`<validate the constraint>`**
- This query is on your tables, and validates the constraint, by looking for a violation
  - Constraint validation (CV) query

- If CV-query returns no rows ➔ Constraint is OK
- If CV-query returns 1 row ➔ Constraint is violated
  - CV-query should return the error message to be raised

Example CV-query

- "At most one president allowed in EMP"

```
• Select 'At most one president
          allowed. Found'||
          to_char(num_pres)||'.' as MSG
  from (select count(*) as num_pres
        from EMP
        where job='PRESIDENT')
  where num_pres > 1;
```

Execution Model

- **For r in (TE-query)**
  Loop
    Execute CV-query;
    If %FOUND
    Then raise_application_error(MSG);
    End if;
  End loop;

In summary:
Per constraint you tell RuleGen WHEN (TE-queries) and HOW (CV-query) the constraint should be validated

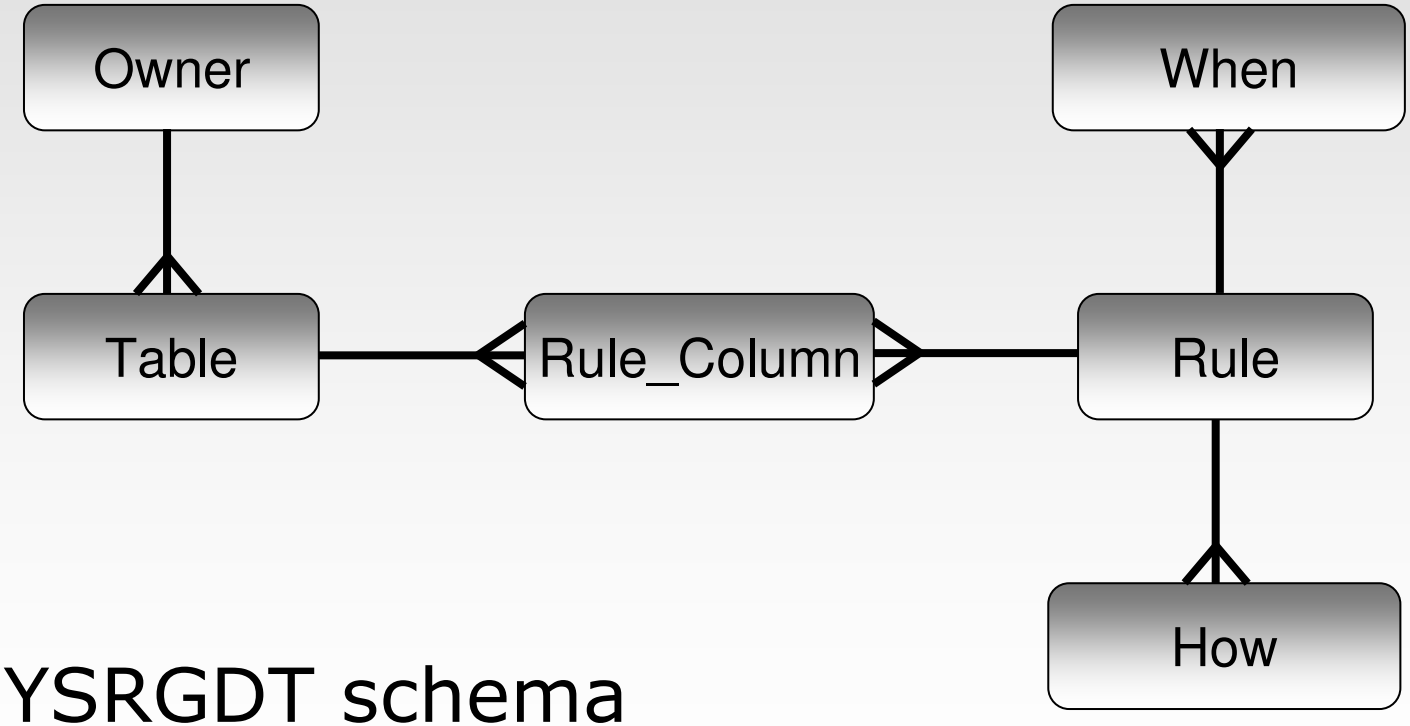
Execution Model

- **For c in (involved constraints)**
  Loop
    For r in (TE-query-of-c)
    Loop
      Execute CV-query-of-c;
      If %FOUND
      Then raise_app_error(MSG);
      End if;
    End loop;
  End loop;

A Few Comments

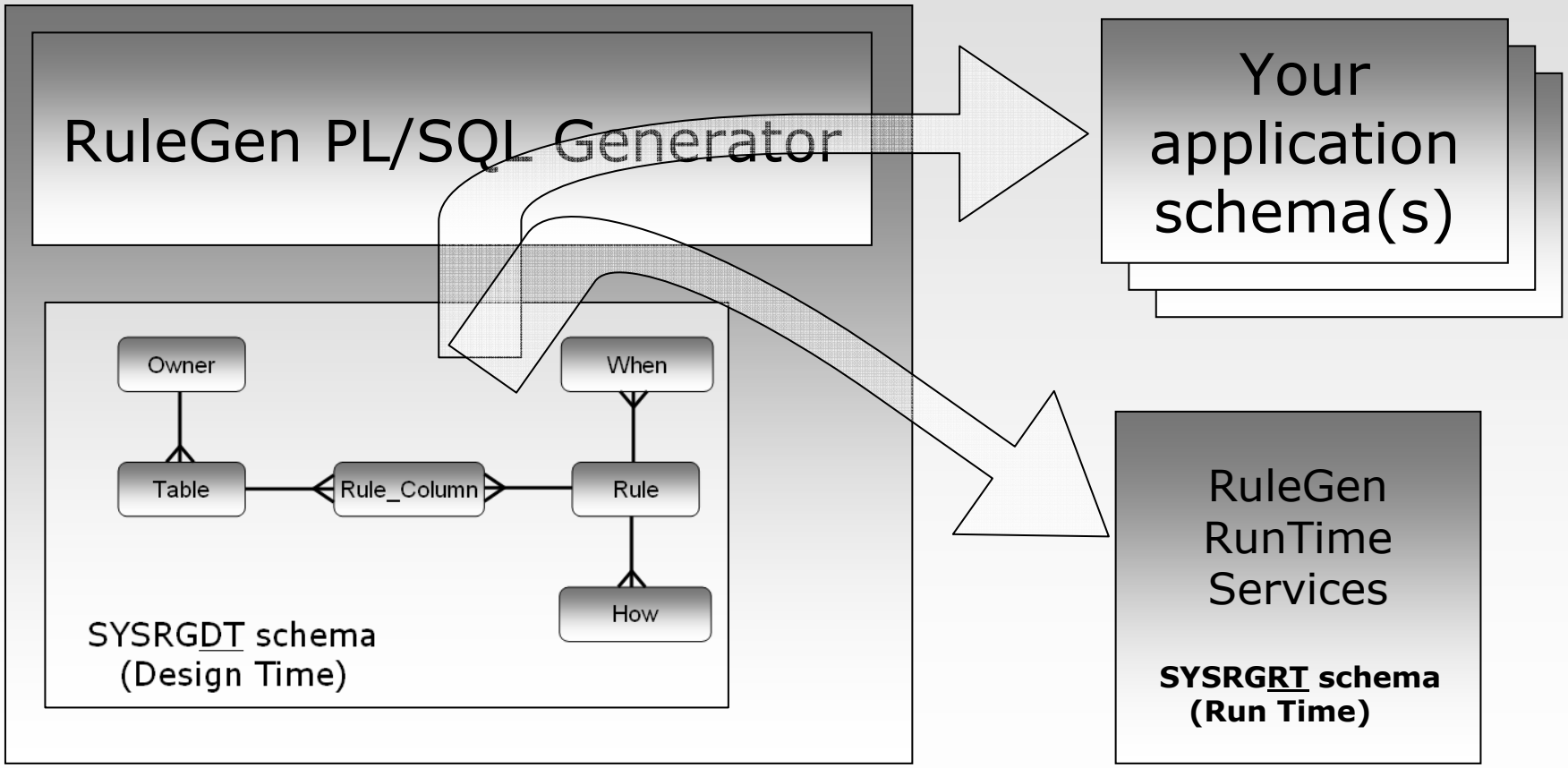
1. You must tell RuleGen per constraint what the involved columns are
   - For efficient TE maintenance
   - For determining 'involved constraints' when updating
2. You can pass parameter values from TE-query to CV-query
   - Depends upon the constraint (demo)
3. You can control the order of constraint evaluation

Components: Repository

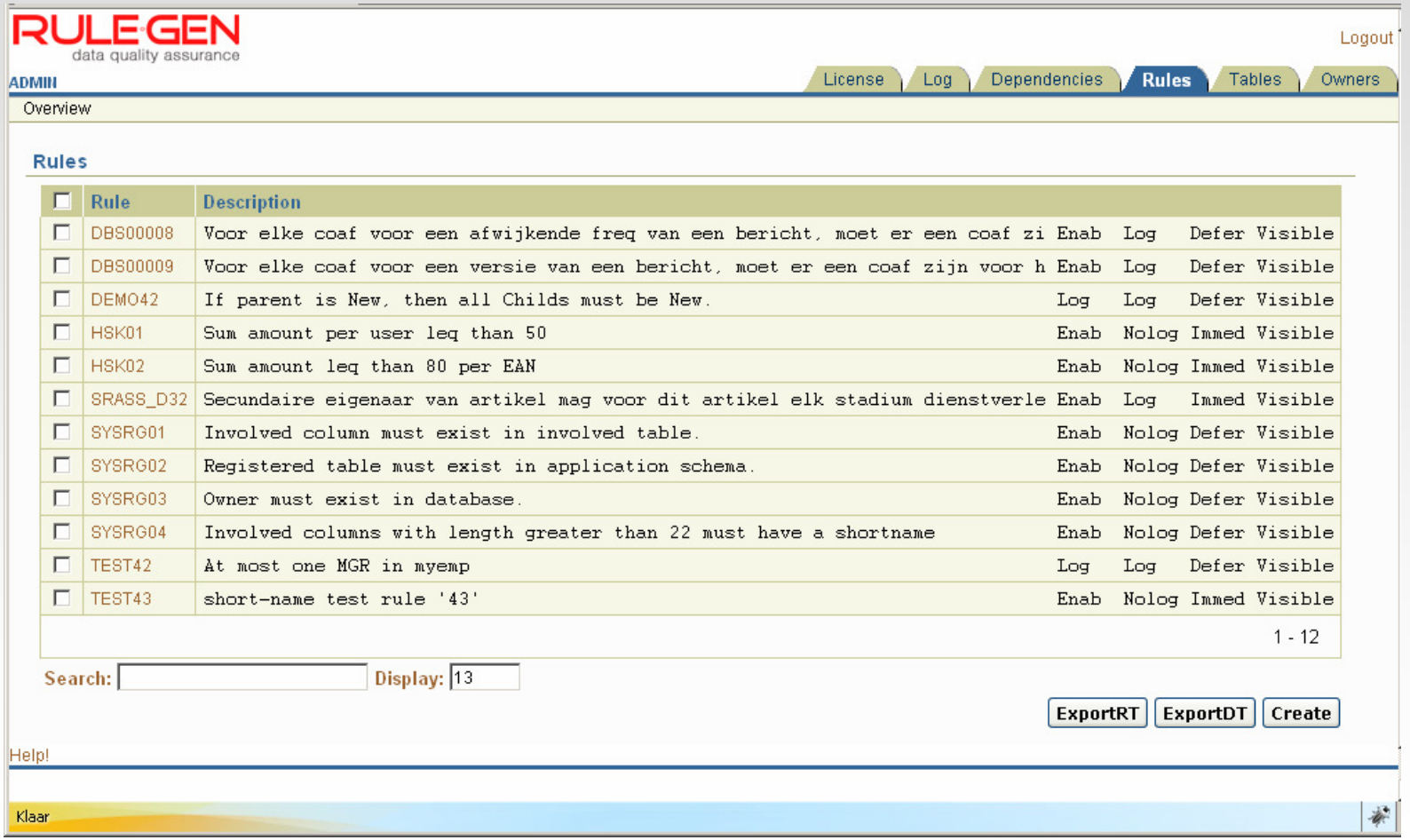


SYSRGDT schema
(Design Time)

Components: Generator and Runtime



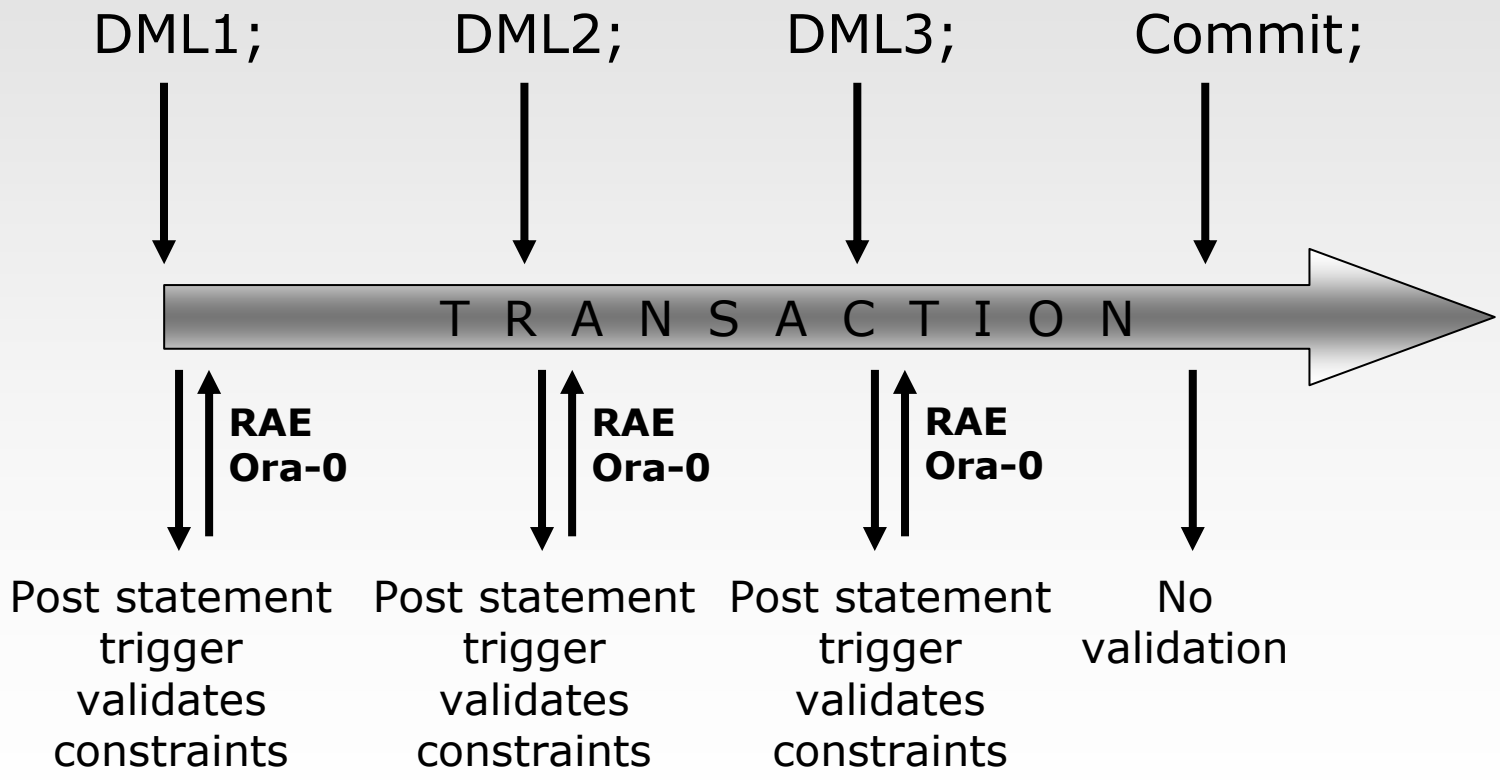
Components: Apex Frontend



Demo

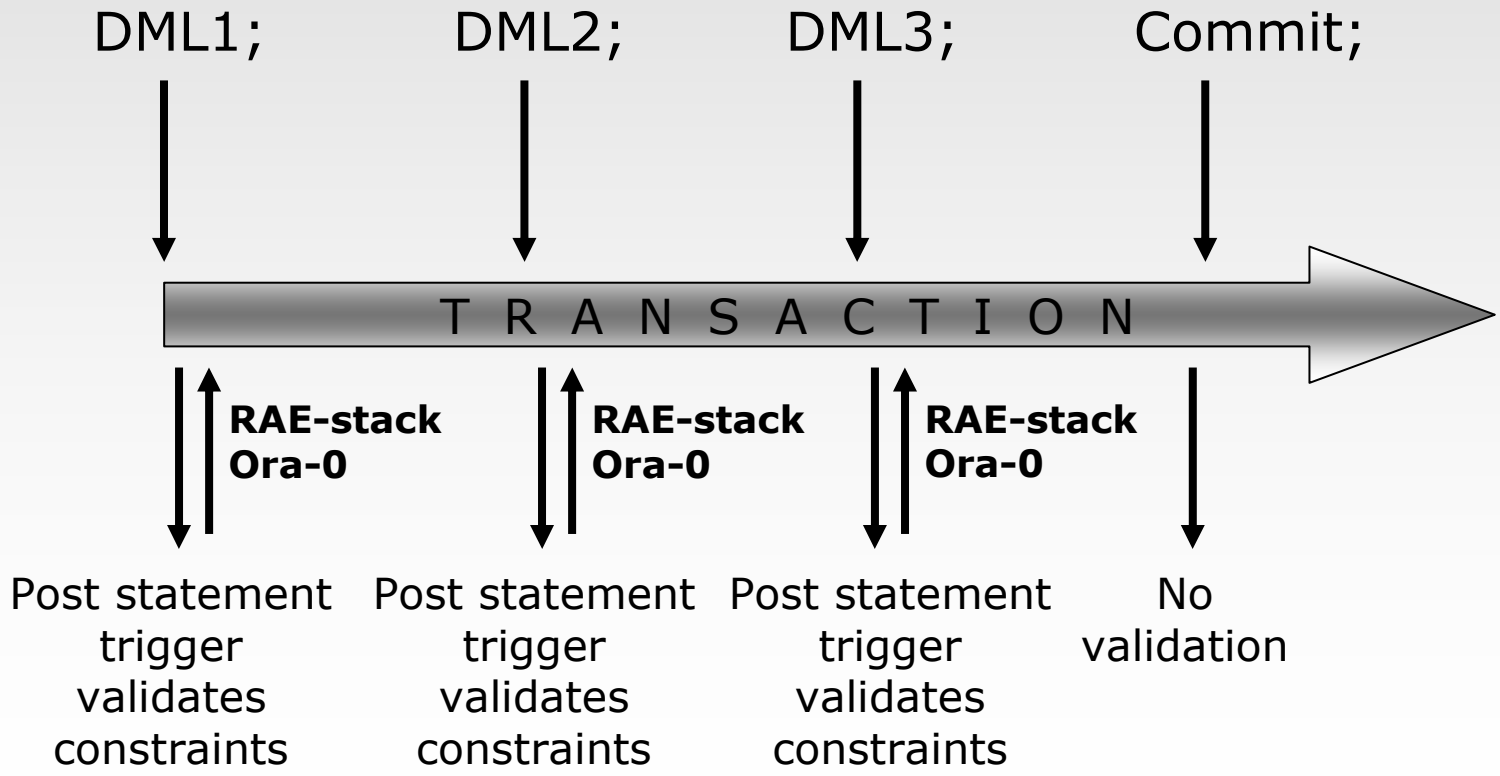
- Show “At most one president in EMP”
- Show “At least two employees per DEPT”

Use Case: Immediate Checking

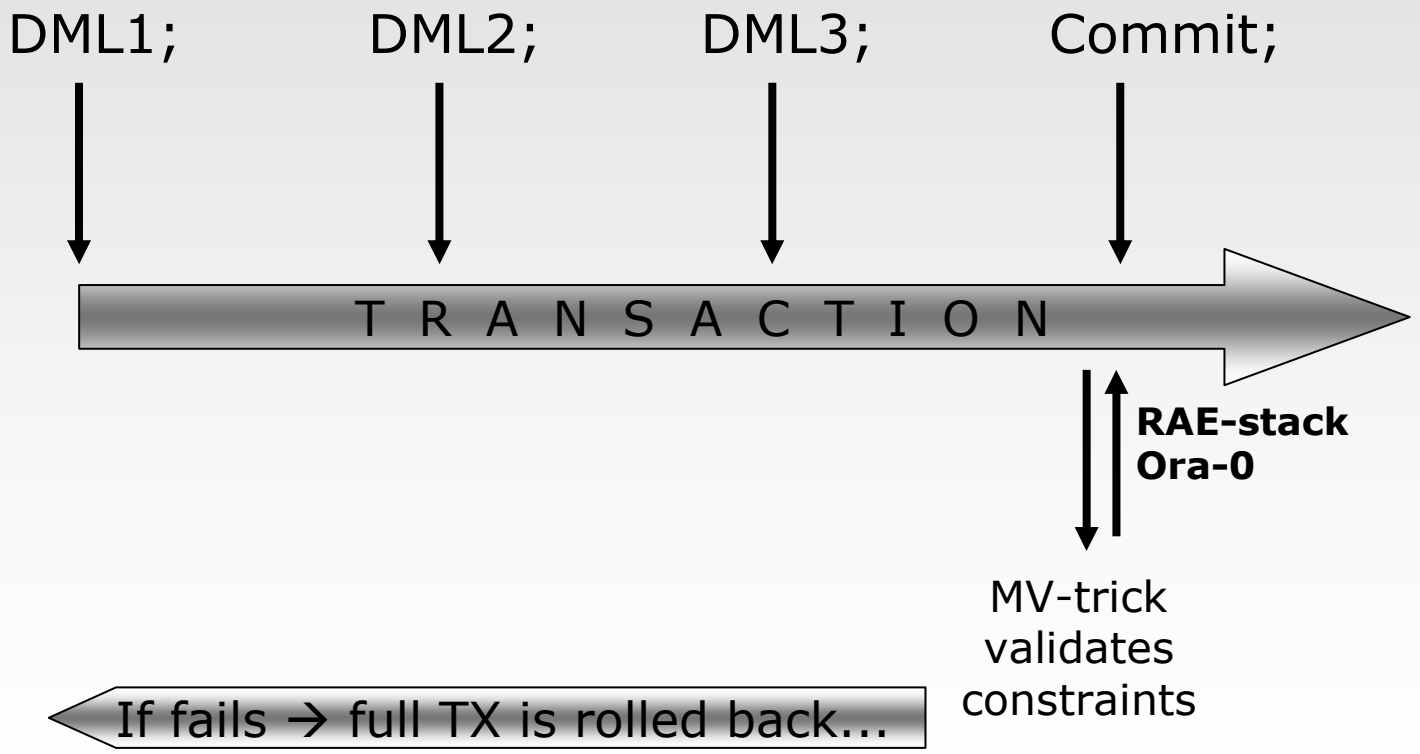


Use Case: Immediate Checking

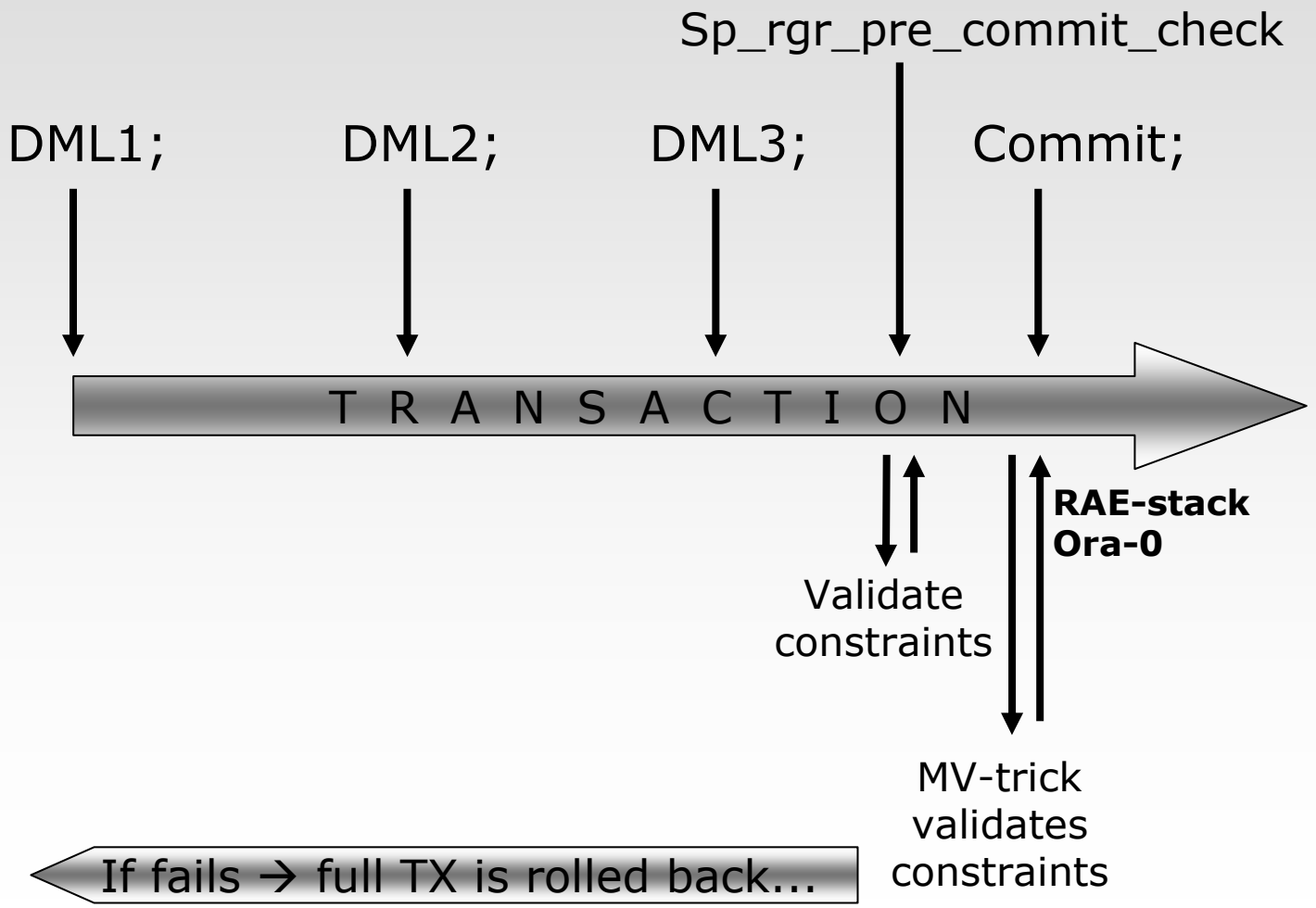
Optional "Continue-on-error" execution model



Use Case: Deferred Checking



Use Case: Deferred Checking



Execution Model

- Immediate / deferred: runtime property of a constraint
  - You can have deferred and immediate constraints at the same time
- You can modify this within your session per constraint

Demo

- Re-demo
  - "At least two employees per DEPT"
  - "At most one president in EMP"

Services

- Sp_rgr_set_exec_model: Stop_on_error / Cont_on_error
- Sp_rgr_set_check_moment: Immediate / Deferred
- Sp_rgr_pre_commit_check
- Sp_rgr_set_check_status: Enabled / Disabled
- Sp_rgr_disable/enable_rulegen
- Sp_rgr_set_log_status: On / Off
- Sp_rgr_filter_sqlerr_stack
- Sp_rgr_set_rule_scope
- Sp_rgr_clear_rule_scope
- Sp_rgr_validate_rule

Installation

1. SYSRGDT schema
   - Holds repository + design-time code
2. SYSRGRT schema
   - Holds runtime services
3. SYSRGFE schema
   - Holds code for Apex frontend
4. Apex frontend application
5. AM4DP schema (optional)
   - Demo environment: database design+ rules of AM4DP book

Availability

- RuleGen release 3.0 production release available now
- Requires 10Gr2 or higher
- Requires Apex 3.2 or higher
- Free ‘developers laptop’ license

Contact

- Toon.Koppelaars@RuleGen.com

History

- 2004: Start development version 1
  - ~ 90% code generation
- 2005: Version 1 in use at one location
- 2007: Development of version 2
  - 100% code generation
- 2008: Founded RuleGen BV
- 2010: Development of version 3
  - Rewrite into open architecture