A Generic business rules validation system for ORACLE Applications

Olivier Francis MARTIN

System analyst

European Laboratory for Particle Physics - CERN / AS-DB

Geneva - SWITZERLAND

Jean Francois PERRIN

Consultant

AUSY

Lyon - FRANCE

Summary

Picture this : You have just spent the remainder of your IT budget on a new software package for Human Resources. Despite its excellent functionality, it does not perform all of the complex validation that your old in-house-developed system did. How can you improve the standard software package given the following constraints :

You cannot afford to pay the vendor for modifications to the package

Modifying the package yourself is out of the question.

We describe a tool designed to implement the validation of complex business rules for any ORACLE database application - without incurring any modification to the user interface. It enhances your productís standard capabilities and improves data quality as soon as data has been entered or modified.

.

Our initial implementations was for the ORACLE Human Resources package. Our tool consists of four independent components:

- A description of the data and its location,
- A set of rules (written in a simple pseudo-code),
- A generic on-line change detection system,
- A core engine that checks data consistency by applying the rules to relevant changes.

The Validation system makes extensive use of state of the art techniques with database triggers and

dynamic PL/SQL.

Table of contents

Introduction

This chapter introduces the need for a validation mechanism, the possible uses of our tool - copyright CERN, 1996,1997 - and the objectives of this project.

Why a validation mechanism?

Quick Detection of Errors

The cost of correcting an error increases proportionally to the time taken to detect the error. Since business rules are often complex, errors frequently remain undetected for a long time (e.g. until data is 'sliced and diced' in a warehousing application or consolidated for the accounts). By that time the correction of an error requires a long investigation to locate the source of the error and correct all of its consequences and side-effects. Furthermore the corrective actions to be taken may be very complex. The objective of this tool is to allow the users to detect business rule violations or special events while the data is 'fresh'. In short, to enhance data quality in complex environments.

Implementation of Complex Business Rules

It is unrealistic to expect a commercial product to support non-trivial company specific rules (it is all the more true if the rule is complex and involves different pieces of information). Cross-check validations (i.e. rules that ensure correspondences between data located at different places in the system) are, in general, not possible to implement using a standard product.

Companies may invest large quantities of resources to customize these products and to port their changes across new releases. If they do not do so, they risk having to spend even more on resources to correct their data.

This tool has been built to overcome this deficiency without having to modify the standard interfaces nor the productís APIs. It can be installed on any ORACLE database with a minimum amount of effort. It has been designed to be independent from the existing application: it requires no modification to your commercial product. There is no limitation to the complexity nor the number of rules that can be implemented. Since the information we manage is moreoften valid for a period of time, the tool fully supports temporal data.

Checking at the Right time

The best solution would be to inform the user at the moment the error is generated. This solution generally requires modifications to the standard package.

Furthermore, this may not be what we want to do in the first place. In the case of the cross-check validation rule, a discrepancy must be temporarily allowed (the user must modify the data located at different places in sequence), and blocking the process as soon as an incoherence appears would cause a deadlock. For this category of rule, no on-line solution can be implemented.

Our tool implements a generic off-line validation mechanism for end-users and controllers.

Table of contents...

Potential uses

Auto correction by End-Users

As already stated, validation should be performed ideally as soon as possible after the data has been entered, preferably by the user who has made the changes.

To achieve this, an end user may use the tool to check the data he/she has just entered against all relevant rules (and nothing else). In order to allow this "incremental" checking, the tool must record who has done what and when in the database (on-line event recording), and for which period of time these changes are effective (e.g. changing a salary for 1997 should not trigger the validation of this employee’s previous salaries).

End-users can request a validation at any moment.

The tool is also able to automatically notify errors to end users. This is done by an overnight

batch job.

Error reports may be either sent by electronic mail and /or printed on the user's preferred printer.

Checking by controllers

In this section we list a few possible uses where verification is not made by the person who has entered the data :

- Quality Control: A supervisor could use the tool to measure the quality of the data entered by his/her services.
- Task-specific validation: Certain people may be in charge of specific data for the whole company (e.g. payroll officers, medical services...). These people are generally only interested in checking a subset of the data against a subset of business rules, regardless of who entered the data.
- Data Audit: Although the tool has been designed to incrementally validate new data, it can be used to validate the existing data. Exceptional data checking may also be performed for audit purposes, prior to archival in a data warehouse or prior to migration to a new system. In particular checking may be required in order to adapt the existing data to the level of accuracy - or to the constraints - of the new software.

Detection of exceptional cases

Validation rules may be implemented for analysis and reporting of exceptional cases: for instance to detect persons overpaid or underpaid according to complex criteria. This is the OffLAP (Off Line Analytical Processing) flavor of our tool.

Delayed checking

In a workflow, some events should trigger actions within a period of time. An example could be: when a new employee arrives, he/she has to follow the safety courses within two weeks.

It is possible to execute these validations *a posteriori* (on events that occurred some time ago). A reminder process can be put in place.

In this way, it is possible to check that a process (workflow) is completed within a certain limit of time.

Table of contents...

Constraints and goals

Application independence

For obvious reasons of cost, maintainability and support we do not want to modify the standard products, therefore the validation mechanism must be application independent.

Because we wish to use this tool for different applications, we cannot make any assumption about the data to validate : we need a dynamic description of the database.

Maintainability

We would like to simplify as much as possible the programming of the rules (which should be pseudo algorithmic). This will lead to a higher complexity of the validation engine.

Flexibility

There should be no limits to the type of rules to implement or type of data to validate. The selection of what to validate must be flexible enough to allow all different uses mentioned before.

Speed

Because we may have to validate large amounts of data against complex business rules, performance is clearly a key issue.

The tool must be accurate in validating only what it is required. It must be "event-oriented".

The designer must have a good knowledge of the application repository in order to avoid sub-optimal queries and their consequences on database performance.

Table of contents...

Concepts

This chapter introduces the underlying concepts of our tool.

Domain of a business rule

When expressing a rule such as "In a university, a normal staff professor cannot be recruited below a salary of xxxx$", or for an accounting application "An invoice cannot be charged on to a cost center if the budget limit is exceeded", we understand that the first rule applies to an employee as individual and the second to a single invoice. These rules have to be checked for each employee individually, and for each invoice independently.

These items define the implicit domain of a business rule: the minimal set of data to which the rule applies.

In practice, this domain is more often uniquely identified by a well-known reference, such as an employee number, an applicant number, an invoice number.

An application may have more than one domain: for instance, in our human resource database, we have defined the employee and the applicant domains. The rules which are applicable to an applicant have no sense in the context of an employee, and vice-versa.

Table of contents...

Description of the data model

Database items

As mentioned in the previous section, the tool should be application-independent. Consequently, we must tell the validation software the location of the information as well as how to fetch it (i.e. to define the application's data model). This is part of the customization process (it is not hard-coded in the tool).

ORACLE Applications introduced the concept of database items: a piece of information described by its meaning and its access path. The definition of these database items is dynamic, and they may be reused at several places (payroll formulas, quickpaint reports...).

We have adopted this idea, with some extensions : our database items may be single fields in a table (educational level of an employee), or aggregates like the current balance of an account, an annual turnover, the total number of sick leaves per year etc.

Of course, a database item may be used in several rules.

A database item acquires its meaning from a domain

In general, information (e.g. salary) only makes sense when interpreted in the context of its own domain (e.g employee) and makes no sense in other domains (e.g an applicant has no salary).

Sometimes, the same physical piece of information may have different meanings depending on the domain in which it is interpreted. In the context of ORACLE Applications, the context-dependent part of a flexfield may be stored at the same physical place, but its meaning will be different. In ORACLE Human Resources, applications for posts and contracts are stored in the same table (in spite of the fact they are managed in a completely different way).

Access methods

The way to retrieve some data - i.e. the structure of the database - is stored as a piece of SQL code that is part of the definition of the database item. The validation system makes no assumption about the format of a database item or the way to retrieve it.

Since only the information we want to check has to be defined, the tool requires a minimal effort of customization. Nevertheless, a knowledge of the underlying data model is necessary.

In brief, a database item corresponds to a user concept; a piece of SQL code tells how to fetch it.

Time dependency of data

In spite of the fact that our 'minimum salary for doctor rule' might be valid forever, the educational level of somebody might change over time. When this happens the applicable rule must be triggered to check his/her salary after the date of change.

Table of contents...

Rule definition

Triggering events

A rule involves a few database items which are built from data stored in a limited number of application tables. Only the changes to these tables may trigger the rule. Events happening on other tables are not eligible for this rule.

Applicability criteria

Rules are applicable to items satisfying certain criteria. We have already noticed that a rule only makes sense within its domain. The domain represents the elementary entity to which the rule applies. Most of the rules do not apply to all entities in this domain.

For instance, in the example of the minimal salary for doctors, the rule is applicable only to new employees having a normal staff status and having a doctorate.

A study of the business rules we want to check will probably reveal some pieces of information which are frequently used. These ones are more likely to be considered as triggering criteria for the rules.

In our example, the status of an employee may be defined as a criteria, if it is discriminant enough (i.e. if there are other statuses other than normal staff, with different regulations). The educational level may not be a criterion : there are probably not a large number of rules which refer to this information.

When defining a rule domain, the tool allows the definition of a variable number of criteria.

The values of these critical database items must be known prior to selecting the applicable rules since they have to be fetched for every event. Therefore the designer has to find a compromise between the overhead of loading the criteria for every event, and the overhead due to the execution of the rules for irrelevant events.

Time dependency of rules

Because external regulations change, rules are valid for a period of time and/or for items entered during a period of time.

For instance, for a legislation change which is only applicable to new contracts : we will have to register two different sets of rules : one for old and one for new contracts.

As mentioned previously, the execution of a rule may be delayed. In practice, the rule "an employee must have followed a safety course within two weeks" is applicable to new employees, but the tool will select the relevant events which occurred two weeks ago.

Table of contents...

User context

The program must give a pertinent feed-back to users.

A user can only modify a subset of data; his/her access is sometimes restricted to a part of the hierarchy (for instance, modify the contractual situation except salaries of everybody in a department). Therefore, such a user is only interested in:

- The consequences of his/her actions
- An error he/she may correct him/herself. Since he/she cannot modify salaries, the user should not see a pre-existing error on the salary. This can be achieved by checking user changes against a subset of rules that do not contain the rules on salaries.
- Sometimes, user responsibilities are very specific (for instance, the user is not allowed to modify salaries except for short-term contracts). The tool is able to filter data to validate according to any additional conditions on database items.

Table of contents...

Implementation

The first section of this chapter introduces the customization features of the validation tool. The two following sections contain a description of its functioning.

Parametrisation

The first task to achieve is to make a fairly complete list of the validation rules needed. A detailed analysis of the concepts involved will lead to the correct implementation:

Task 1 : Definition of rule domains

Rules should be classified into general categories depending on the entity to which they apply (e.g. employee, applicant, invoice, etc.). This will reveal the rule domains to be defined.

Each entity to check must be identified by a unique key in our tool. This key will be called "domain key" in the following sections.

In the validation system, a rule domain is the top level concept: everything is defined for and within a domain. Consequently, defining the domains is the first task to accomplish when customizing the validation system for a new application.

Task 2 : Declare actions to log

The rules must be triggered when executing certain transactions. At the database level, these transactions trigger some inserts, updates, deletes in the application repository.

The designer must map which database actions should trigger which rule, in order to instruct the tool about what to log.

The information we collect on line is :

- who has done what and when (the username, the table touched, the database action and a timestamp),

- the time period for which the change is effective,
- the domain(s) key(s): if a salary has been changed, the employee number is stored.

The designer must tell the tool how to fetch the domain key from every table that should be audited.

Task 3 : Choose rule criteria

The analysis of all the rules should indicate the subset of ìconceptsî that are used by a majority of rules. These should be considered as the triggering criteria for the rules (e.g. employee status, job, position, etc.). A criterion is a database item which is fetched prior to the application of the rule. Because the only information the validation system knows is the domain key (logged with the event), a criterion must be fetched from this key.

Task 4 : Define database items

As introduced before, a database item is a user concept used in the rules and read from the database when the rule is executed. It is valid within the domain of the rule. The designer must write the SQL code to fetch it and carefully tune these statements. Unlike the rule criteria, a database item can be fetched from the result of another database item. This is a way to modularize the code and optimize complex rules.

Task 5 : Rule programming

Syntax

Once everything above has been put in place, rules have to be written. We have reused the flexibility of the PL/SQL syntax: a rule is a piece of PL/SQL that is dynamically executed.

There is no limitation to the PL/SQL syntax that cannot be used in the rules: a rule may call external PL/SQL routines, for instance.

The tool provides two primitives GET_VALUE(<db_item>) and CHECK_IF(<SQL condition>) to fetch database items and to check their values during time.

In addition the designer has to define error messages, help, and, if needed, the actions to be performed (e.g. feed-back mails).

Task 6 : Batch customization

The validation jobs must correspond to the users' responsibilities : we will have different programs for each group of users. Validation jobs may be customized using :

- sets of rules. This is very useful to check part of the data, whoever has entered this data.
- sets of users. This feature is mainly use for control by supervisors.
- additional restriction criteria (employee within a department...) on database items.

Table of contents...

On-line event recording

As part of the parametrisation process, the validation tool will create a simple database trigger for each table that must be audited. These triggers look at the definition of what to log in the validation tool and log corresponding events (with their domain keys).

Because the code is dynamically executed, it is possible to change the parametrisation of the validation tool while the application is running.

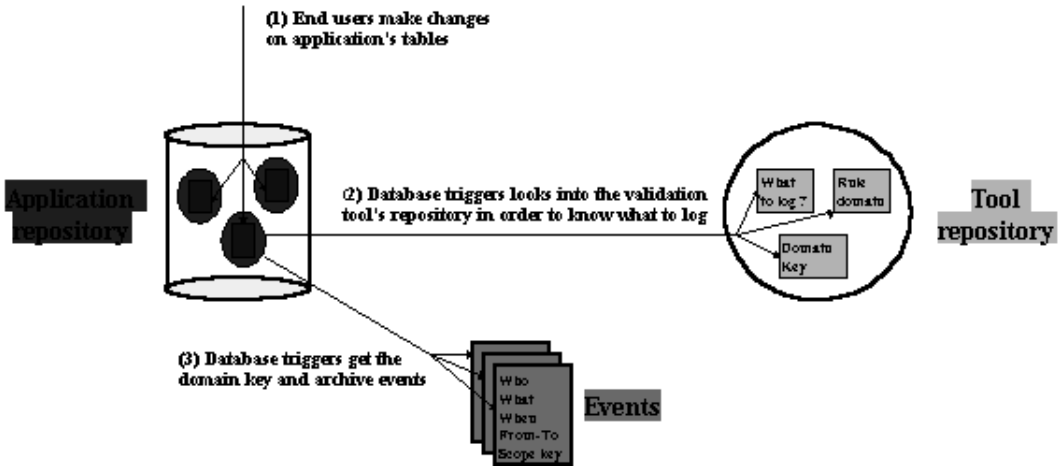


Table of contents...

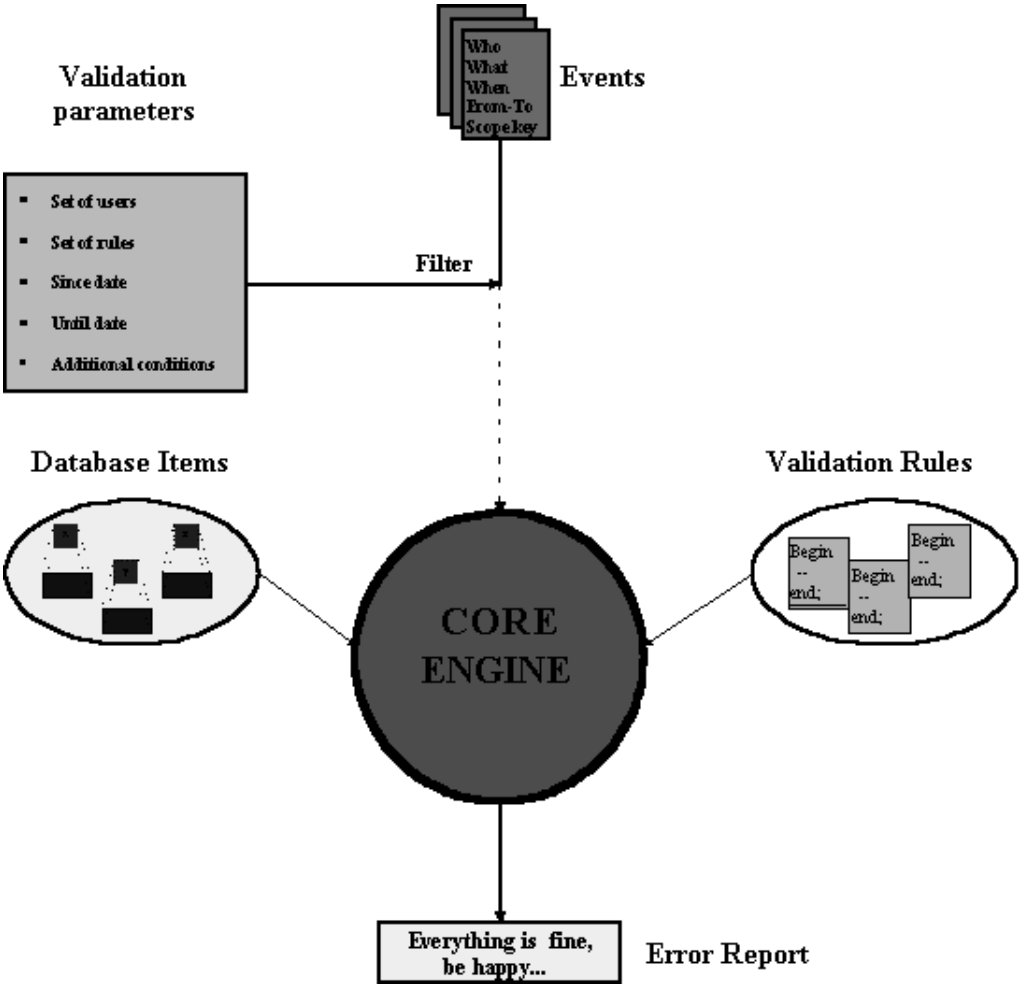
Off-line validation

Selection of events

According to the parameters passed, the relevant events are selected. They are filtered by:

- User(s) who made the change
- Modification date
- Changes on application tables which need to be audited. This is implicitly given by the set of rules.
- Additional restriction criteria (changes for an employee in a given department...)

An accurate selection of the relevant events must be performed in order to limit the impact on performances.



Rule selection

Another filter is applied to the rules :

For each remaining event, the program fetches the criteria which are defined for the current rule domain.

Applicable rules are then selected depending on :

- The table on which the event occurred,
- The database operation (insert, update, delete)
- The period of time for which the change is effective : changing a salary for 1997 should not trigger the execution of an obsolete rule valid until the end of 1996.
- The values of the criteria.

Rule execution

Like a rule, a change (event) is always effective for a period of time, even if it is from big-bang to big-crunch. The database items are fetched for the same period of time. Database items may be dated or "datetracked", thus the result of the query will be an array of the form `date_from, date_to, DB_item_value` (1).

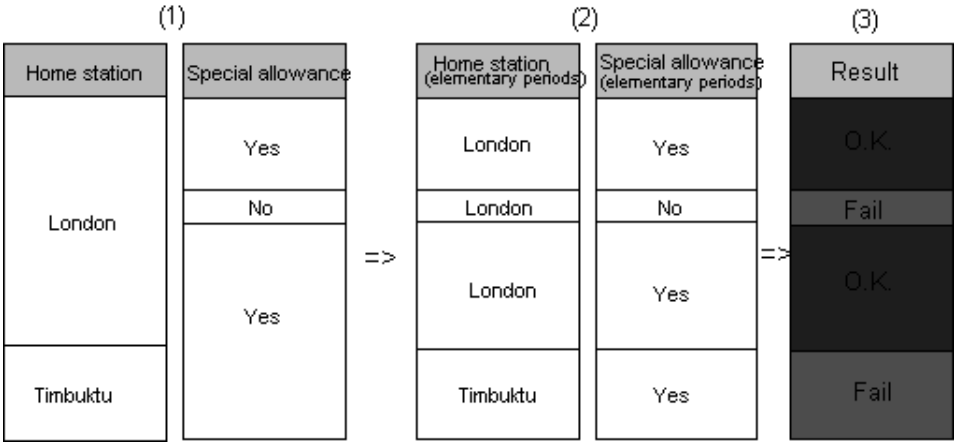
Prior to applying the check condition of a rule, the validation engine reconstructs a stack of elementary periods within which the value of all database items are constant (2).

Then, the SQL check condition is applied to every elementary period; if one of these fails, an error is raised (3).

The diagram below shows the execution of the rule "Any employee who lives in London area should receive a special allowance", between 01-JAN-92 and today.

The code of the rule looks like :

```
BEGIN r_home_station

-- Fetch home station

VALIDATION.Get_Value('HOME_STATION', 'address');

-- 'address' is an alias to be used in check conditions or as input

-- for other database items

-- fetch special allowance

VALIDATION.Get_Value('SPECIAL_ALLOWANCE', 'is_there_an_allowance');

-- check condition

IF VALIDATION.Check_If(

'<address> = "LONDON" AND <is_there_an_allowance> = "No"

OR <address> != "LONDON" AND <is_there_an_allowance> = "Yes"',

<err_no>); -- error message

THEN

RETURN FALSE;

END IF;

RETURN TRUE;

END r_home_station;
```



Error reporting

An external module (which can be the application's normal reporting mechanism, if the validation jobs are launched from within the application) will have to report errors to users and, optionally, take extra actions (E-mails to concerned people, register audit information...).

Table of contents...

Results

What it can do and what it cannot do

The system checks that the data is correct with respect to the companyís business rules. It takes a snapshot at the moment it validates. The tool will not check that some illegal operation has been performed, of which the result is 'business rule wise' correct, once the 'steady state' has been reached. For instance, if a contract has been suppressed by mistake, the tool will not detect the error if no business rules are violated in the system as a result of this action.

In our implementation, the validation system is used as a reporting tool: it makes no automatic change in the application repository. If somebody would like to take this responsibility, it would be possible to associate actions (other than sending feed-back mails) to errors. These actions would then be part of an external module.

Performance

Our Human Resources application runs on a SUN ULTRA-4000 with 4 CPUs, 640Mb of memory, with an average number of concurrent users of 150 and more than 100.000 people registered in the database.

The on-line event recording mechanism has very little impact on performance. On our machine, the recording of a single event never takes more than 0.10 seconds.

We use around 4h of CPU time per day (1 CPU) to run all the validation batches. The extra memory needed is, in our case, around 64M.

Our conclusion

Our tool has been introduced in production August 1996 on our Human resource database. Rules have been gradually added; today, we have implemented 150 rules on our Human Resources database. Some of them are very complex, but we have not yet encountered a rule which we cannot implement.

The tool is easy to customize, but requires a good knowledge of the application repository, and a good understanding of the validation system concepts.

In general the tool has been appreciated by end-users. Since they are able to quickly detect and correct their errors themselves, they feel more confident about their work.

When introducing this tool, we found very important not to give the user the impression that their work will be closely monitored by the system.

Our users' estimate that the level of accuracy of the data has increased by a factor of ten.

We are now considering using this tool to enforce business rules across multiple databases.

Table of contents...