Requirements Driven Database Physical Design

Abdelkader Ouared
Ibn Khaldoun University (UIK) Tiaret, Algeria
abdelkader.ouared@univ-tiaret.dz

Fatima Zohra Kharroubi
Ibn Khaldoun University (UIK) Tiaret, Algeria
fatima.kharroubi@univ-tiaret.dz

Abstract—Designing database systems is an inherently complex process, especially with the Big Data Era and the spectacular evolution of conceptual, logical, physical models, deployment platforms, and the complexity of queries. As a result, the database (DB) design often has to compare the goodness performance of hypothetical designs without having to access the full databases and hardware of end-users. Every physical design solution is not valid for every database and workload and is not adequate for every context. To tackle this problem, we propose an approach, based on the Model Driven Architecture (MDA) to facilitate the selection of the right physical design for non-expert users. The approach is based on three different models:

(i) a requirements model based on goal-oriented modeling for representing information requirements, (ii) a Database Context model for representing database, and query which will be connected to database materialization and, (iii) a database materialization model for representing physical design details regardless of their implementation technology (e.g. hardware, disk-layout, Deployment-Platform, deployment Architecture). Together with these models, a set of transformations allow us to provides a database without requiring the setup of physical hardware, the deployment of architecture, or the need to configure the database. Finally, a case study is presented.

Keywords: database physical design, Model Driven Architecture

, User requirements.

I. Introduction

Designing database systems is an inherently complex process, especially with the Big Data Era and the spectacular evolution of conceptual, logical, physical models, deployment platforms, and the complexity of queries. This process imposes the evaluation of the quality of service (QoS) of non-functional properties such as usability, reliability, performance, etc. [17] according to criteria limiting databases (DB) to a certain number of connections or a peak level of I/O, or some other criteria. Due to the increasing needs of companies and businesses to store and analyze the deluge of data, performance becomes one of the major requirements. The satisfaction of this requirement necessitates the optimal selection of DB deployment architecture [15].

From the conceptual modeling perspective, the Conceptual-Model Programming [6] and the agile methods [16], [17] in designing databases pose new challenges on the traditional vision of database development, in which, the conceptual models are independent of underlying database materialization (e.g. hardware, disk-layout, deployment-Platform, deployment- Architecture).

A.Problem statement

In general, the DB deployment depends on a physical design vocabulary. The deployment-platform, deployment- architecture, and storage-device such as storage memory size, block size. In the DB design process, if we want to check whether ER schema is lively, we need to mappe ER schema into a relational schema and the corresponding SQL generated that represents a physical schema. To assist their users (in this case, the Database Administrator, or DBA) can start the final step of creating a database in the server. They must have to specify a few parameters such as the DB server (e.g. DBMS version, memory target) and architecture parameters (e.g. number of nodes in parallel environments). While many professionals involved in software releases are feeling the pain of DB deployments. To realize the choice of deployment architecture, we must realize a series of tests about CPU, memory, disk, buffer pool utilization based on quality metrics for this resource (e.g. throughput, response time, scalability, and others). By exploring the major state-of-art, manually or semi-automated conceptual approach can increase the cost of DB deployment and gives less time to react to problems. Currently, most of the proposed technique is based on a manual approach that is time- consuming, skill-intensive for expert designers, and requires manual matching according to the application.

B. Contribution

In this paper, we focus on the contribution of linking conceptual design to physical design by translating the conceptual model specification automatically via MDE into a physical data model (e.g. deployment and storage characteristics). The main objective of this work is to provide a DB materialization from conceptual model to the user application and reason up of the causality of its cause as mapping rules which reduce performance. For instance, the choice of a supported DBMS server hosting a DB: SQL, NoSQL (e.g. MongoDB[1] or Cassandra[2]), NewSQL (e.g. VoltDB) according to the CAP Theorem [2]), the choice of physical structure like partitioning strategy is well- suited to OLTP/ Web workloads. Consequently, we propose a framework. It is an intermediate framework between DB design and practical systems deployed that non-technical stakeholders (users, owners, clients) should be able to use this framework. In our knowledge, there is still no commercial tool that directly translates a conceptual data model into a materialization to help the user to translate the requirements of DB applications into diverse physical configuration and reduce the space of possible variants because an evaluation of the whole variant space within this work is not feasible.

C.Paper Outline

Our paper is structured as follows: Sect. II presents the database physical design domain. Section III presents our

1http://www.mongodb.org 2http://cassandra.apache.org

approach to manage the variability of DB design. Section IV presents its evaluation process. Section V discussed the related works. Paper is conducted in Sect. VI.

II. Ba ckground

In order to make this research self-contained and straight- forward, this section introduces some fundamental notions of physical design for database systems.

A.Database Physical Design al models, storage lay- outs, the wide panoply of optimization structures, Deployment Paradigm, Deployment Architecture, and Storage Platform. Indeed, it would be much more interesting if the final DB design is valued in order

1 http://www.mongodb.org
2 http://cassandra.apache.org

to help the user to choose between different designs. In Testing Hardware in the Loop, the designer must have to deploy it on a DBMS and to compare the different costs of testing. This solution spends a lot of time/money on testing activities and adopted by companies such as GAFA (Google, Apple, Facebook and Amazon). The second solution is Testing Software in the Loop specialized tools such as database cost models [12] is a tool designed to evaluate the performance of a solution without having to deploy it on a DBMS, and to compare different solutions.

In practice, the DB deployment architecture is independent of the used query language. In a nutshell, each selected hard- ware, storage layout, architecture deployment, etc. have an impact on the workload cost, as follows (in chronological order of the DB design life-cycle). Accordingly, and depending on the complexity of the database system and the deployment architecture, the possible physical configuration is multiplying diversity of DB design. As a result, database design often has to compare goodness performance of hypothetical designs without having to access the full databases and hardware of end-users, from the conceptual modeling perspective, recently papers solution in the research DB community to facilitate the use of Agile Methods in designing databases [16]. The agile methods challenge the traditional vision of database development, in which, the conceptual models are independent of underlying database materialization (e.g. hardware, disk-layout, database size) and increases the interdependencies between phases of the life cycle [17]. Other metrics such as usability, reliability, quality, security, integrity, etc. must be taken into account in the database life cycle. For instance, a Relational Cloud Environments of database-as-a-service (DBaaS) well-suited to the high performance, high availability, and elastic scalability. The choice of database deployment challenging for Database Administrators (DBAs). Moreover, this choice has a relevant impact on the database performance required by end-users.

B.Motivating Example

Let consider a scenario that designer comes with a database schema and workload to evaluate the performance of a solution by having to deploy it on a DBMS, and to compare different solutions. The generic conceptual processes of the database design are illustrated in Figure 1. In this case, we're interested in the performance criterion, for which, database design specifications become as follows: Conceptual formalism has no direct measurable impact on performance, since it is requirement- oriented, and a modeling matter more than anything else. It will be of great value if the evaluation criteria were security or understandability. The user involves metrics for measuring the quality of a conceptual schema from the point of view of its understandability. To evaluate we must follow as: Conceptual schema determines which query to be considered from the whole workload since it provides the" local" schema related

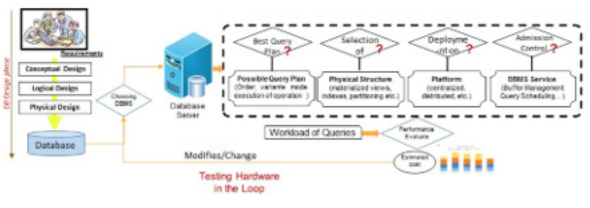

Requirements
DB Design plane
Conceptual Design
Logical Design
Physical Design
Database
Choosing DBMS
Database Server
Best Query Plan ?
Selection of ?
Deployment on ?
Admission Control ?
Possible Query Plan (Order, variante mode execution of operation...)
Physical Structure (materialized views, indexes, partitioning etc.)
Platform (centralized, distributed, etc.)
DBMS Service (Buffer Management Query Scheduling...)
Workload of Queries
Performance Evaluate
Estimated cost
Modifies/Change
Testing Hardware in the Loop

Fig.1. Process of Deploying the Database with Testing Hardware in the Loop.

to current requirements. The Logical Modeling features to determine the algebraic operators to be used. Deployment Layout features serve to provide a final rewriting of the queries. In fact, queries have to be rewritten according to the final deployment of data. Deployment Paradigm, Deployment Architecture, and Storage Platform feed the cost model with necessary parameters, such as storage information: memory size, block size, and the number of nodes in parallel environments. Optimization Structures determine the formula for each algebraic operator. Figure 1 shows the process of valuing the performance of DB design process. However, this is a very long, complex task and a manual solution, that relies on user experience and whereby he has to select the most appropriate paths to his application, and then compare their costs. the DB design process as a whole allows users to have an overall vision, consider life- cycle interdependencies, and tackle all DB design steps while increasing process automation. Moreover, in contrast to the classic DBMS-to-DB vision, according to which, comes the DBMS selection before the logical step, we see in figure 1 that the usual designer and DBA, like DB architect, analyst, and developer evaluate the design process under NFR and can customize their choices. This observation has motivated us to closely analyze the latter and spot the main variables that control the life-cycle as well as their dependencies.

III. Proposed Solution

In this section, we present our approach, which is the set of step in the approach presented in Figure 2 to reduce the gap between conceptual design and physical design. The usage

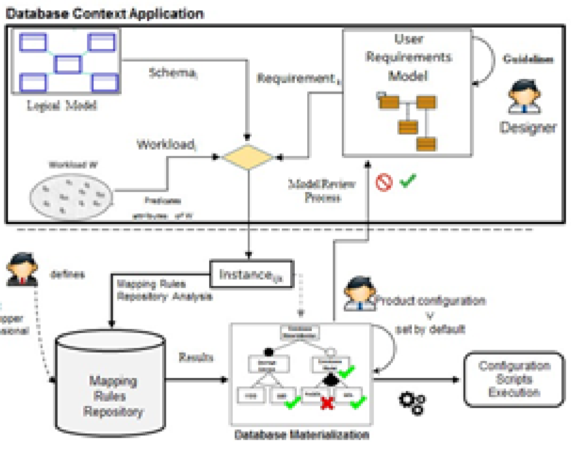


Fig.2. Overview of our Framework architecture.

scenario is organized as it is shown in Figure 2. It shows a gescenario is organized as it is shown in Figure 2. It shows a generic scenario due to the usage of our framework. This latter helps designers to analyze step by step their database designs during the design phase, which can be increasingly improved by applying the three following processes:

- Using the dbCxtDL design language for system modeling, or for refining imported models of CASE Tools (e.g. PowerAMC).
- Selection of the database materialization by helping the database designers to extract the relevant elements from the requirement of database application.
- Generating script design from database materialization using the Object Constraint Language (OCL).
- In the following we detail it step by step.

A.A Model-Based Approach

Our framework relies on the database context application that composes of three metamodels in green (the Requirement Database Application Metamodel, the DB schema Metamodel, and the Query Metamodel) with database materialization using Context Mapping Relation, as depicted by Figure 3, pink and orange respectively.

B.Database Context Metamodel

Our approach starts from our Database Context Language Metamodel named dbCxtDL. By instantiating dbCxtDL language, a designer with can specify the conceptual model of his/her application (DB), a workload (Q) and Requirement Model (RM) (requirements of DB applications). Consider all instances on database context dbCxtDLi that are instances of dbCxtDL. One of these dbCxtDLi must be instantiate by database context Design Language Model like the query and the database metamodel extracted from the Common Warehouse Metamodel 3 . dbCxtDL =<DB, Q, RM> where: dbCxtDL characterized by the DB, workload and a set of element of DB application. Due to the lack of space, we are unable to present all metamodels.

- Database: An instance of the Database class is composed of conceptual entities and their attributes. In addition, links between entities are also represented via associations. We also represent several semantically restrictions, such as primary and foreign keys.
- Query: An instance of the Query class takes as input a set of concepts which are used to perform a set of algebra operations (join, union, etc.) or data manipulation operations (update, insert, etc.). An algebra operation can be a unary or a binary function. The result of the query can be restricted by a set of predicates using logical and arithmetic operators or textual operators.
- Requirement Model: Our approach bases on a goal- oriented requirements model that allows us to capture information needs, such as criteria involved by stake- holders. To describe the coordinates required to build a database context (Goal, Resource, Task) we follow the specification to automate database materialization in the deployment platform given. Our metamodel shown in Fig. 4. For instance, (i) if user needs to focus on high-availability and high-scalability (the CAP Theorem [2]), NoSQL is recommended then SQL database, (ii) if ratio of [Qread/Qupdate] > 1 and Application Server > 2, Data Replication is generally recommended then data distributed, and (iii) Bitmap indexing is advantageous if low-cardinality domains, join operation, and aggregation operations.

C.Database Materialization

Database materialization composes of a set of parameters that are related to different categories. Figure 5 gives only a brief view of our meta-model classes that are organized on four categories of parameters:

- Database parameters: elements of this category are related to the database and different functionalities that have to be provided by the DBMS. Through this category3www.omg.org/cwm/ we precise context parameters of storage systems (for instance relational or non-relational), buffer management (e.g. the buffer pool size) and database schema (e.g. tables/columns, partitioning table).
- Hardware parameters: generally, the hardware context parameters define device characteristics, such as processing device (e.g. CPU, Graphical Processing Units GPU, etc.), different storage device (e.g. Main-Memory, Solid State Drives SSD or Hard Disk Drive HDD) and communication device.
- Query parameters: the query parameters mean concepts used to perform a set of algebra operations (join, select, etc.). The operation can be unary or binary function. The result of the query can be restricted by a set of predicates. These operators should perform as fast as possible by exploiting underlying an access method (e.g., index methods or in-memory access methods [8]).
- Architecture parameters: elements of this category contain the deployment architecture such as: distributed or parallel database systems, database clusters, or cloud environments. These category parameters feed the context on the type of the system architecture (e.g. shared memory, shared disc, etc.) and their parameters like the number of nodes in parallel environment.

The view is expressed as a feature tree in SPL. In our feature model of the DB materialization, we have approximately 151 features. By using online automated analysis tool for feature models. S.P.L.O.T4 computed 4522 possible variants. Since we are not able to evaluate such a high number of possible variants, we have to limit our considerations to a selected number of features. We must reduce of the space of possible variants, because an evaluation of the whole variant space within this work is not feasible.

D.Mapping Rules Elicitation and Formalization

The DBA Standards and recommended practices for database has been identified by academicians and industrials. In deployment database (disk, flash, etc.) development domain, engineers and designers have a great experience, and they can easily identify the shared concepts between devices, their re- placement policies, etc. A number of computer science problems such as selecting optimization structure, storage model choice, document intelligent retrieval would benefit from the capability to model the absolute meaning of a given domain. This experience gave us a motivation to propose a Mapping Repository MRrp for database materialization. The MRrp contains rules which has impact on performance and help to choose the physical database configuration context corresponding to the design model

1)Formalization: We define a mapping rules as ECA type (Event, Condition - Action). A mapping rules are built on the following pattern: a list of at least one condition (connected by a Logical Operator if more than one) which, if satisfied, implies what is defined within the action. Both conditions and the action are Operations, which are applied on Constrainable Elements, that is, a feature or an attribute of database materialization:

A rule RxCP is defined by (NRx, SRx, ERx, CRx, ARx) with:

- NRx is the name identifying the rule,
- SRx designates the component kind of the rule. Rules can be associated with either an attribute value (AVi), either to a features (Fi), either to a hierarchy (variants, children) (FHj) of DB materialization.
- ERx is the manipulation operations triggering the rule and that are applied to the input required (DB, Q, RM) for implementing the database:

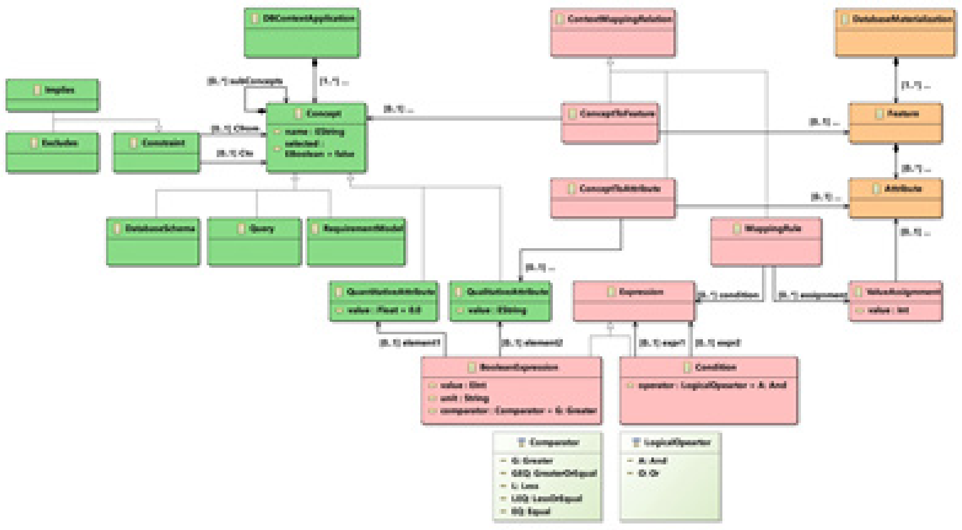


Fig.3. Database Context Metamodel connected to the Database Materialization

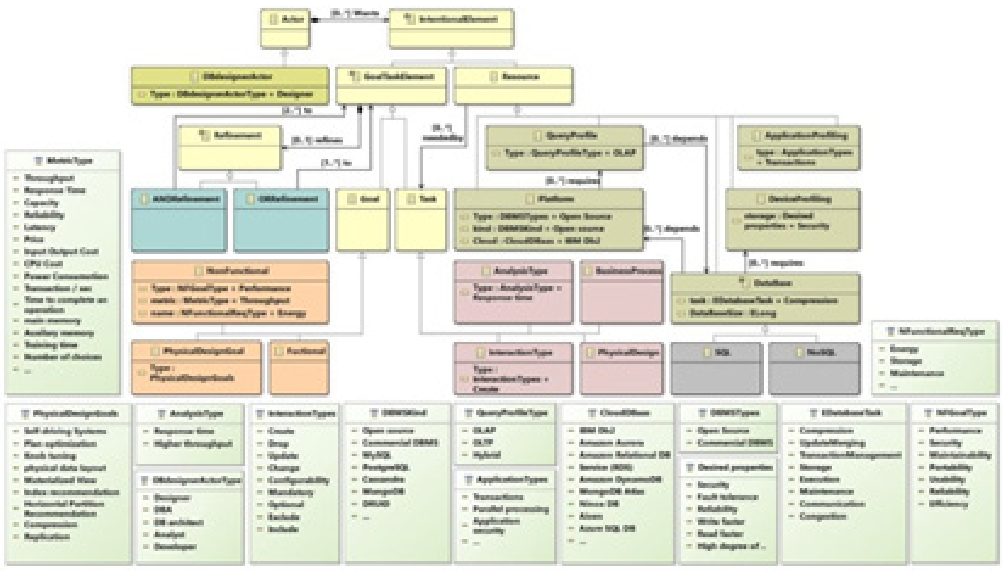


Fig.4. User requirements metamodel of DB applications.

E^{Rx}=<exclude|require|recommend|recommendnot|Cardinal-Attribute >. These operations are controlled by guideline, meaning that all mandatory children have to be selected whenever this parent is selected, at least one feature must be selected, exactly one of these children has to be selected.

- C^{Rx} is a condition defining if the rule is triggered through a current E^{Rx}.
- condition1 δ ... δ conditionn → action, with δ∈ {∧, ∨}. Note that the Abstract Operation is only for modeling purpose, as a condition is either a Value Operation or a simple Operation. Each conditioni is defined as: conditioni= Element.attribute θ Value, with θ ∈ {<;<=; >;>=} and Element of dbCxtDL <DB, Q, RM>.
- A^{Rx} is the sequence of actions triggered. The actions apply to the components (feature, child-feature, attribute, etc.) of the database materialization. Actions should be included or excluded from a database materialization.
- In the following, we describe this transformation:
- R1: each concept or class c ∈ dbCxtDL is mapped to a features (Fi), either to a hierarchy (variants, children) (FHj) of DB materialization.
- R2: each attribute a ∈ attributes are mapped to a property definition (an attribute value (AVi)) of DB materialization,
- where their value can be constrained to be in a given range, greater, lower and/or equal to a given value, or compared with another value.
- R3: each association between classes c1, c2…, cn of dbCxtDL is mapped to either an attribute value (AVi), either to a features (Fi), either to a hierarchy (variants, children) (FHj) of DB materialization.
- example we define OCL constraints, which represent basic deployment of database rules. The first one checks that the server number of a database is always greater than 1, the second one verifies that the Types of Non-functional requirement are high consistency ∧ availability, and ensures a client has Application Type is Transaction Profile. In this case, the relational data model is recommended.

context Goal. NFRequirement -> includesAll (high consistency and availability)

context Resource . Database valid Server : self.

A pplication Server > 1

context Resource . A pplication Profile -> select (Transaction Profile | OLTP)

Listing 1. Textual Constraints

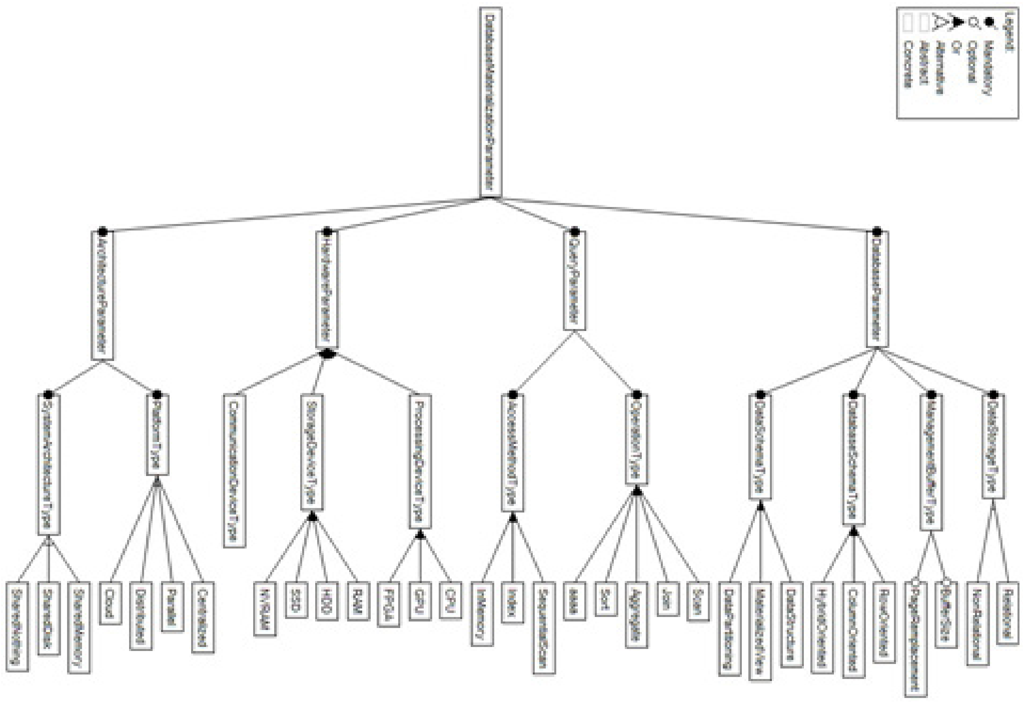


Fig.5. Excerpt of database materialization parameters meta-model: core entities

2)Instantiation of the MR Analysis Repository: We can reuse the mapping rules of similar works to reduce their total cost of testing. We propose an analysis repository MRrp, which

is a reorganization of concepts tackled in Section III-D1. So, for factoring the number of assumptions and to guarantee a good scalability of the analysis repository, we suggest that the analysis repository contains also a set of identification rules R. They will help for identifying correctly the closest database context matching the design model (dbCxtDL) which requests analysis.

Let Sr be a function for specifying database contexts in the analysis repository. where, Sr(dbCxtDLi) = R^{undef} ∪ R^{true} ∪ R^{false}= R, such that:

R_i^{true} is a set of rules which have to be satisfied, for every $r_i \in R_i^{true}$, dbCxtDL |= r_i ;

R_i^{false} is a set of rules whose complements have to be satisfied, for every $r_i \in R_i^{false}$, dbCxtDL |= $\overline{(r_i)}$;

$R_i^{undef} = R \backslash (R_i^{true} \cup R_i^{false})$

Database Schema and Query instance

To illustrate the different transformation steps of our frame-

To illustrate the different transformation steps of our framework we introduce as a running example the conceptual schema

presented in Figure 6 representing a simple excerpt of an

application. User can using a Conventional Database Schema

(e.g. TPC-H,SSB benchmark) or import the XMI file generated

from CASE Tool like PowerAMC. This schema is specified

using the ER notation that describe the SSB schema 5. The

schema contains a fact table Line order, and four dimension

tables: Customer, Supplier, Part and Dates. The OLAP queries

are download file which is stored in a folder. Note that every

query instance has to conform to dbCxtDL design language.

We propose an analysis repository, which is a reorganization as set of rules: R (NRx, SRx, ERx, CRx, ARx). In this section, we surround some capabilities of the MRrp Repository by instantiating its elements, we have formalized this problem as follow: Cxt =< DB, W, RM > where: Cxt characterized by the DB, the workload consists of a set of queries W = q1... qm and the set of the element of requirements DB applications. MRrp is a mapping rules repository characterized by the set of Rules. Then, MRrp = < NRx, SRx, ERx, CRx, ARx > where:

X the context that represents all assumptions related to the

nature of the application.

As we have presented earlier in this paper, our goal is to orient designers to choose the appropriate cost model. Thus, the difficulty for database designers is to choose the most suitable DBMS and database platform which matches the characteristics of the designed system. Once features have been identified, not all possible combinations of them correspond to feasible SPL instances. Thanks to the ContextMappingRelation class (see Figure 4).

Case studies: proof of concepts

To stress our approach and to proof how it is useful and helpful, this section is devoted to present a global usage scenario of our framework. In parallel, technical implementations are highlighted.

A.Database Schema and Query instance

To illustrate the different transformation steps of our frame-

To illustrate the different transformation steps of our framework we introduce as a running example the conceptual schema

presented in Figure 6 representing a simple excerpt of an

application. User can using a Conventional Database Schema

(e.g. TPC-H,SSB benchmark) or import the XMI file generated

from CASE Tool like PowerAMC. This schema is specified using the ER notation that describe the SSB schema 5. The schema contains a fact table Line order, and four dimension tables: Customer, Supplier, Part and Dates. The OLAP queries are download file which is stored in a folder. Note that every query instance has to conform to dbCxtDL design language.

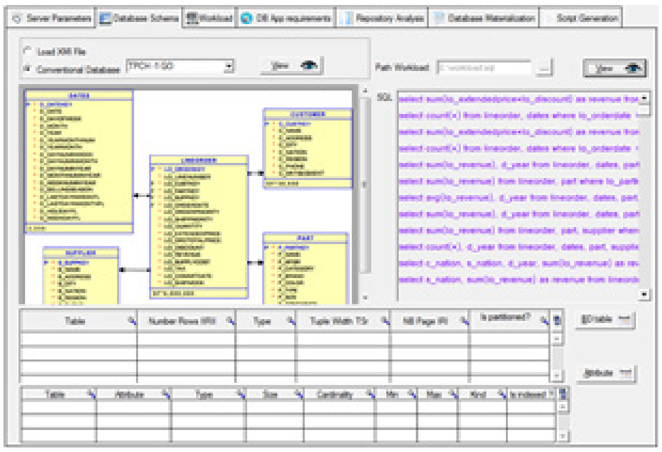


Fig.6. Import Database Schema and Query instance

B. Specifying Database Application Requirements

We assume that a design office of a company comes up with a database application with its components (database schema, workload, requirement of database application) and looks for a relevant DBMS that fulfills its requirements. This need has to be expressed via the seeker interface by using the meta-model. An instance of the meta-model dedicated to express this need is called a Manifest. Therefore, the engineers of this company may access our mapping repository to get the most appropriate DBMS that corresponds their manifest. To clarify the utility of the Manifest concept, Figure 7 represents an example of the DB application requirements instance, where Application Server configuration are missing.

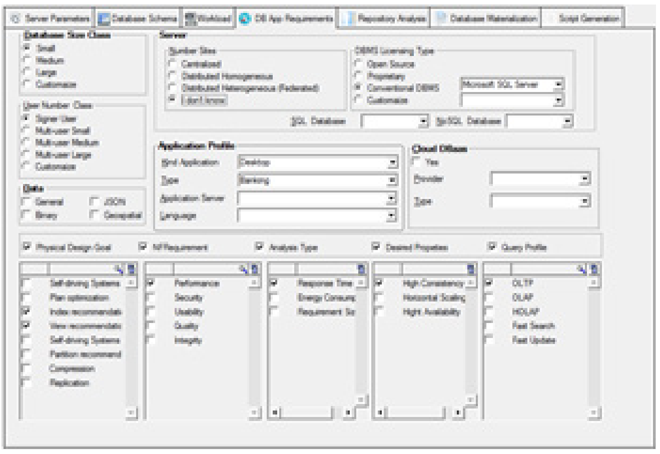


Fig.7. Example of the DB application requirements

C.Instantiation of the identification rules

Every identification rule is specified through its properties in particular the description property. For instance, the proposition" 'the hardware storage device is HDD"' is represented by the identification rule StorageDeviceHDD. The description of this latter means that if the evaluation result is true, hence the hardware architecture is HDD. Otherwise, the hardware architecture is not HDD. Figure 8 shows the identification rules that we have created in the repository. They correspond to the following propositions:

- The database schema is SQL
- The Deployment Architectures is Centralized
- The Processing Device is CPU
- The Optimization Structures is Index
- The Primary Storage is RAM
- The Storage Device is HDD
- The Integrated Persistent Buffer is SSD
- The Storage Model is RowStore
- The Processing Model is Tuple-at-a-time
- The deployment architecture is Sharednothing Infrastructure;
- The Optimization Structure Mode selection is Combined
- The DBMS Licensing Type is Open Source
- The rules DBMSKind is available or Undefined because Unknown information in requirement model instance or repository content. In this case user can choose a Product Configuration Option:" 'Manually customize"'or"'Set by default"'.

D.Design tool for instantiating of database materialization

To facilitate the identification of database materialization parameters, we based on the description of three models: schema description model and query description model and architecture parameters (see Figure 9). We present in Figure 9

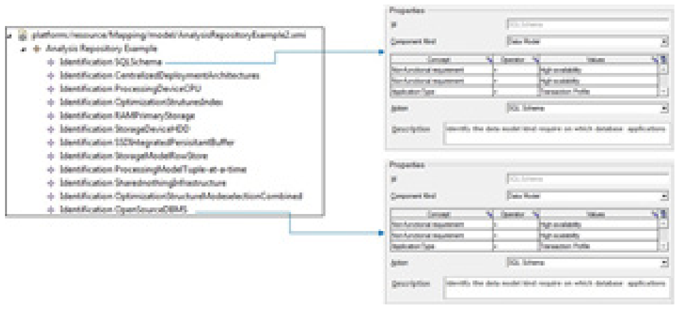

platform:/resource/Mapping/model/AnalysisRepositoryExample2.xmi
Analysis Repository Example
Identification SQLSchema
Identification CentralizedDeploymentArchitectures
Identification ProcessingDeviceCPU
Identification OptimizationStructuresIndex
Identification RAMPrimaryStorage
Identification StorageDeviceHDD
Identification SSDIntegratedPersistentBuffer
Identification StorageModelRowStore
Identification ProcessingModelTuple-at-a-time
Identification SharednothingInfrastructure
Identification OptimizationStructureModeselectionCombined
Identification OpenSourceDBMS

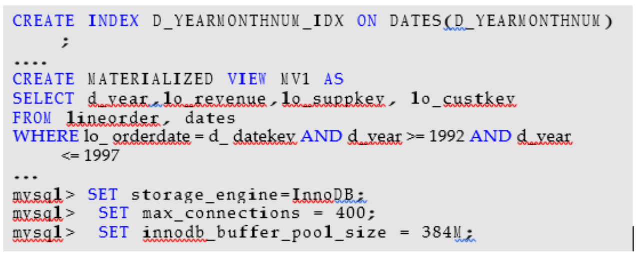
Fig.8. Identification rules and their properties

an excerpt of the database materialization instance generated by a set of identification rules.

Figure 9 shows the instantiation of some parameters of the database materialization corresponding to a manifest of Figure7. These parameters is done by using dbCxtDL language. These parameters are described by several characteristics related to the Deployment Platform, Storage Device, and Deployment Architecture.

E.Generating Script

The last step in MDE processes is a PSM-to-code trans- formation, which generates the target database structures (e.g. database schema, index, materialized view, configuration files...). Thanks to the interoperability facilitated by model- driven engineering, we have developed a user-friendly tool to exploit the parameters of materialization related of database, query, physical structure, device and storage layout as XMI files to a script by using Acceleo language. Listing 2 presents an excerpt of the DDL statement generated from the database materialization for each physical structure (e.g. Index, Materialized view) and database system properties (e.g. parallel max servers, memory target, sga max size) require for ACID databases.



```
CREATE INDEX D_YEARMONTHNUM_IDX ON DATES(D_YEARMONTHNUM)
    ;
....
CREATE MATERIALIZED VIEW MV1 AS
SELECT d_year,lo_revenue,lo_suppkey, lo_custkey
FROM lineorder, dates
WHERE lo_ orderdate = d_ datekey AND d_year >= 1992 AND d_year
    <= 1997
...
mysql> SET storage_engine=InnoDB;
mysql>  SET max_connections = 400;
mysql>  SET innodb_buffer_pool_size = 384M;
```

Listing 2. Script of Physical Structure and DB system properties

IV. Related work

The BD research community has proposed model-driven approaches to DB development process that support automatic code generation (see [4] for survey). Many practitioners have no formal education in DB design and are unfamiliar with query optimization techniques and database deployment. By examining the current BD design life-cycle, we found that works consider dependencies between 3-phase design: conceptual (CD), logical (LD) and physical (PD) and other focusing only on a part of the BD design process. For instance, In et al. [14] proposed a prototype of a DBMS product line, focusing only on PD ,LD schemas normalized. The automatic design process via a code generator consider dependencies between 3- phase design [9] and increases the interdependencies between phases of the life cycle [17], [16]. Several solutions (e.g. [11], [13], [7]) provide transformations from ER and UML models to relational DB then translated to an SQL definition before generating a server-side API (e.g. CASE Tools Rational Rose,

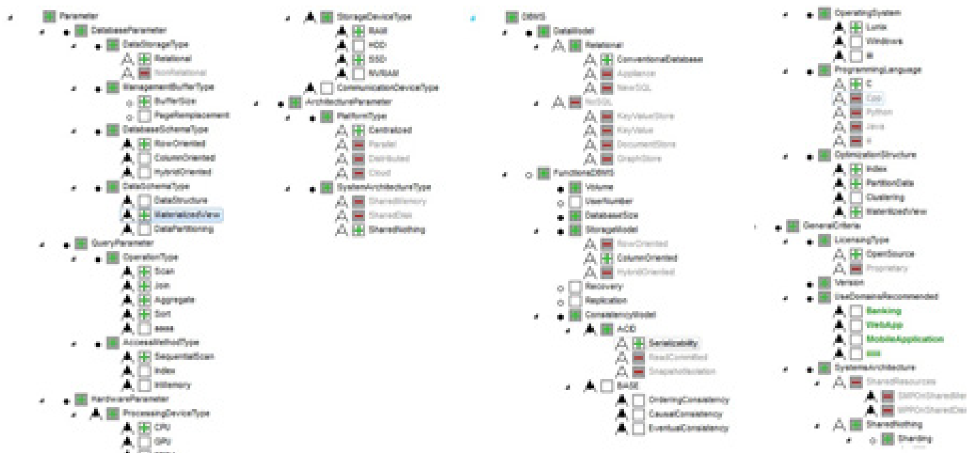


Fig.9. Excerpt of the database materialization instance

PowerAMC). More specific to NoSQL DB with different data models, such as key-values and document stores [1], the NoSQL Schema Evaluator [10] generates query implementation plans from a conceptual schema and workload definition. In [3], author focus on generating NoSQL DB (including key-value stores and document databases) from UML models. Gwendal Daniel et al. propose the UMLtoGraphDB framework [5]. There have been very interesting industrial and commercial solutions of code generation from CD. However, this solution does not take into account constraints specified in the deployment model. Moreover, the conceptual/logical models are independent of underlying DB materialization (e.g. hardware, disk-layout, DB size). According to the known literature, this is the first framework that improve the way designers select DB deployment architecture

V. Conclusion

The main objective of this work is to reduce the gap between the model and practical systems. We propose approach for conceptual Model to DB materialization to improve the way designers evaluate their system design. In this paper, we have presented the MR Analysis Repository. Each instantiation of the MR Analysis Repository offers a repository based on a set of identification rules, their evaluation can lead to the database context of the design model. Moreover, the meta-model of the MR Analysis Repository is flexible and can be enriched by analysts in order to add other characteristics. In the next paper, we link the MR analysis repository with the dbCxtDL design language in order to obtain a complete design framework. This enables the design at the same time as to orient designers (especially modelers) to choose the most suitable database physical configuration. We have presented an example showing the instantiation of the MR analysis repository. At this stage, the identification rules have been presented in an informal way.

The core of identification rules and how they check the design model. Yet, how model transformation through the analysis repository instance generate the script of database physical design. As for future work, we intend to perform case studies in real scenarios and to provide formal descriptions for the solution proposed.

References

[1] [1] A. Benelallam, A. G´omez, G. Suny´e, M. Tisi, and D. Launay.Neo4emf, a scalable persistence layer for emf models. In European Conference on Modelling Foundations and Applications, pages 230–241. Springer, 2014.

[2] J. Browne. Brewer's cap theorem. J. Browne blog, 2009.

[3] F. Bugiotti and Other. Database design for nosql systems. InInternational Conference on Conceptual Modeling, pages 223–231. Springer, 2014.

[4] A. R. Da Silva. Model-driven engineering: A survey supportedby the unified conceptual model. Computer Languages, Systems& Structures, 43:139–155, 2015.

[5] G. Daniel, G. Suny´e, and J. Cabot. Umltographdb: mappingconceptual schemas to graph databases. In International Conference on Conceptual Modeling, pages 430–444. Springer, 2016.

[6] D. W. Embley, S. W. Liddle, and O. Pastor. Conceptualmodel programming: a manifesto. In Handbook of ConceptualModeling, pages 3–16. Springer, 2011.

[7] J.-L. Hainaut. The transformational approach to databaseengineering. In International Summer School on Generative andTransformational Techniques in Software Engineering, pages143. Springer, 2005.

[8] S. Manegold, P. A. Boncz, and M. L. Kersten. Optimizingdatabase architecture for the new bottleneck: memory accessvolume 9, pages 231–246, 2000.

[9] E. Marcos and v. n. p. y. p. Other, journal=Software andSystems Modeling. A methodological approach for objectrelational database design using uml.

[10] M. J. Mior and Other. Nose: Schema design for nosql applications. IEEE Transactions on Knowledge and Data Engineering,29(10):2275–2289, 2017.

[11] A. Murolo, S. Ehrensberger, Z. Asani, and M. C. Norrie.Scaffolding relational schemas and apis from content in webmockups. In International Conference on Conceptual Modeling,pages 149–163. Springer, 2017.

[12] A. Ouared et al. Costdl: a cost models description language forperformance metrics in database. In Proceedings of the 21STIEEE ICECCS. IEEE, 2016.

[13] A. Rosa, I. Gon¸calves, and C. E. Pantoja. A mda approachfor database modeling. Lecture Notes on Software Engineering,1(1):26, 2013.

[14] M. Rosenm¨uller, N. Siegmund, H. Schirmeier, J. Sincero,S. Apel, T. Leich, O. Spinczyk, and G. Saake. Fame-dbms:tailor-made data management solutions for embedded systems.In Proceedings of the 2008 EDBT workshop on Software engineering for tailor-made data management, pages 1–6. ACM,2008.

[15] T. Z¨aschke, S. Leone, T. Gm¨under, and M. C. Norrie. Optimising conceptual data models through profiling in objectdatabases. In ER, pages 284–297, 2013.

[16] T. Z¨aschke, S. Leone, T. Gm¨under, and M. C. Norrie. Optimising conceptual data models through profiling in objectdatabases. In International Conference on Conceptual Modeling, pages 284–297. Springer, 2013.

[17] T. Z¨aschke, S. Leone, T. Gm¨under, and M. C. Norrie. Improvingconceptual data models through iterative development. DataKnowl. Eng., 98:54–73, 2015.