Journal of
Information Technology & Software Engineering

Mancas and Dicu, J Inform Tech Softw Eng 2013, 3:1
DOI: 10.4172/2165-7866.1000119

Research Article Open Access

Architecture, Design, Development, and Usage of *ODBDetective* 1.0

Christian Mancas[1]* and Alina Iuliana Dicu[2]

[1]*Department of Mathematics and Computer Science, Ovidius State University, Constanta, Romania*
[2]*Department of Computers and Information Technology, Faculty of Engineering in Foreign Languages, English Stream, Polytechnic University, Bucharest, Romaina*

Abstract

"ODBDetective is an Oracle database (Oracle) metadata mining tool for detecting violations of some crucial database (db) design, implementation, usage, and optimization best practice rules (bpr). This paper presents the set of bprs that is considered by the first full version (1.0) of ODBDetective, the db axioms from which they are derived, the corresponding tool's facilities, and the essentials of its actual architecture, design, development, and usage, including the results of a case study on an Oracle production db. Moreover, even this first ODBDetective version also allows for storing semantic decision data on desired db scheme improvements, which will prove very useful to automatic improvement code generation in subsequent versions of this tool."

Keywords: ODBDetective; Oracle; Database designing

Introduction

Too often, databases (*dbs*) are very poorly designed, implemented, queried, and manipulated. As a consequence, they allow storing implausible data, sometimes even do not accept plausible data, are very slow when data accumulates, and need too much unneeded disk and memory space, as well as, especially, programming and maintenance effort.

Consequently, significant efforts were made for designing and developing panoply of tools for investigating db issues, proposing better solutions to be carried out by db administrators (*DBA*), and even for automatic db schemes improvement. They range from those provided by db relational management systems (*RDBMS*) manufacturers to third party ones.

For example, Oracle provides Automatic Workload Repositories (AWR) views [1], Automatic SQL Tuning [2], SQL Access Advisor (including Partition Advisor), Real-time SQL Monitoring (the latter three available only in Enterprise Editions), as well as other advisors. Embarcadero Technologies includes in its *DB PowerStudio for Oracle* [3] a dedicated tool called *DB Optimizer*. Details on them, as well as on other theoretical and practical related work are provided in [4] and [5].

Moreover, there are also best practice rules in this field, like, for example, Oracle's ones [6]. From a more comprehensive one [7], derived from experience, as well as from a set of db axioms [7], a relevant subset [8] was the base for selecting the 28 ones that are implemented in *ODBDetective* 1.0, according to 32 derived investigation types (see section 2). Note that three Oracle's ones (namely reduce data contention, choose indexing techniques that are best for your application, and use static SQL whenever possible) are included in *ODBDetective* too.

ODBDetective is a metadata mining tool for Oracle dbs, storing its mining and investigation results in its own Oracle db, together with additional semantic support decision data.

ODBDetective 1.0 was developed in Oracle 11*g* and MS Access 2010, as a MSc. Dissertation thesis to be publicly defended in July 3rd, 2013, at the Bucharest Polytechnic University [5].

Section 2 presents the 20 axioms, 28 considered best practice rules, and the corresponding *ODBDetective*'s 1.0 32 investigation types. Section 3 presents *ODBDetective*'s architecture. Section 4 documents its db (back-end) and application (front-end) design and implementation. Section 5 contains essentials of *ODBDetective*'s usage and the results of a case study. The paper ends with conclusion and further work, acknowledgements, and references.

Considered db Axioms, Best Practice Rules, and Corresponding Investigation Types

From all db axioms and best practice rules presented in [7], only the following small subsets (see sub-sections 2.0 and 2.1) were considered by *ODBDetective* 1.0. Consequently, it provides only the 32 investigation types listed in sub-section 2.2.

Database axioms

Design

***A1**. Data plausibility axiom*: any db instance should always store only plausible data; implausible ("garbage") data might be stored only temporarily, during updating transactions (that is, any time before start and after end of such a transaction all data should be plausible).

***A2**. Unique objects axiom*: just like, generally, for sets elements, object sets do not allow for duplicates (that is each object for which data is stored in a db should always be uniquely identifiable through its corresponding data).

***A3**. Best possible performance axiom*: db design, implementation, and optimization should guarantee obtaining the maximum possible performance–that is the overall best possible execution speed for critical queries and updates, as well as the best possible average execution speed for the non-critical ones.

***A4**. No constraints on redundant data axiom*: no constraint should be enforced on redundant data.

***A5**. Constraints discovery axiom*: for any non-trivial and non-contradictory and not implied (that is computable) restriction existing in the modeled sub-universe, any db should enforce a corresponding constraint.

***Corresponding author:** Christian Mancas, Department of Mathematics and Computer Science, Ovidius State University, Constanta, Romania; E-mail: christian.mancas@gmail.com

Received June 26, 2013; **Accepted** August 06, 2013; **Published** August 13, 2013

Citation: Mancas C, Dicu AI (2013) Architecture, Design, Development, and Usage of *ODBDetective* 1.0. J Inform Tech Softw Eng 3: 119. doi:10.4172/2165-7866.1000119

A6. *Constraints optimality axiom*: for any db scheme, no implied constraint should be enforced and the constraint set should be enforceable in the minimum possible time (that is there should not exist another equivalent constraint set whose enforcement takes less time in the given context).

Implementation

A7. *Constraints enforcement axiom*: for each fundamental table, any of its constraints whose type is provided by the target RDBMS has to be enforced through that RDBMS; all other db constraints should be enforced by all software applications built on top that db.

A8. *Rows uniqueness axiom*: Any fundamental table row should be always uniquely identifiable according to all existing uniqueness constraints in the corresponding sub-universe and it should correspond to a unique object from the corresponding object set.

A9. *No void rows axiom*: For any fundamental table, at least one semantic fundamental column should not accept null values.

A10. *Referential integrity axiom*: any foreign key value should always reference an existing value (dually, no db should ever contain dangling pointers).

A11. *Primary keys axiom*: Any fundamental table should have a surrogate integer primary key whose range cardinality should be equal to the maximum possible cardinality of the corresponding modeled object set; except for tables corresponding to subsets, primary keys should store auto generated values.

A12. *Foreign keys axiom*: Any foreign key should be simple (that is not concatenated), reference a surrogate integer primary key and have as range exactly the range of the referenced primary key.

A13. *Key Propagation Principle axiom*: any mapping (that is many-to-one or one-to-many relationship) between two fundamental object sets should be implemented according to the *Key Propagation Principle*: a foreign key referencing its co-domain (that is the "one" side) is added to the table corresponding to its domain (that is the "many" side).

A14. *No superkeys axiom*: No superkey should ever be enforced; generally, no other constraints than those declared in the corresponding conceptual db scheme should ever be enforced.

Usage

A15. *Relevant data and processing axiom*: Any query should consider and minimally process, in each of its steps, only relevant data.

A16. *Fastest manipulation axiom*: At least in production environments, data manipulations should be done with best possible algorithms and technologies, so that processing speed be the fastest possible.

Optimization

A17. *Reduce data contention*: Intelligently use hard disks, big files, multiple tablespaces, partitions, and segments with adequate block sizes, separate user and system data, avoid constant updates of a same row, etc., in order to always reduce data contention to the minimum possible.

A18. *Minimize db size*: Regularly shrink tablespaces, tables, and indexes for maintaining high processing speeds and minimal db and backup sizes.

A19. *Maximize use of RDBMS statistics*: Regularly gather and intelligently use statistics provided by RDBMS for fine-tuning dbs.

A20. *Follow RDBMS advisors recommendations*: Regularly monitor and apply all recommendations of RDBMS advisors.

Best Practice Rules

Design

BPR0 (*Data plausibility*): All actual constraints should be enforced in all fundamental (that is not temporary, not derived) tables, in order to guarantee their data instances plausibility.

BPR1 (*Surrogate primary keys*): Each fundamental table should have an associated primary surrogate key: an one-to-one integer property (range restricted according to the maximum possible corresponding instances cardinality-see BPR2 below), whose sole meaning should be unique identification of the corresponding lines (this being the reason to refer to them as *surrogate* or *syntactic keys* too).

Note that very rarely, by exception, such a surrogate key might also have a semantic meaning: for example, rabbit cages may be labelled physically too by the corresponding surrogate key values (and, obviously, no supplementary attribute should then be added for unique identification, as it would redundantly duplicate the corresponding surrogate key).

Also note that the surrogate primary key *#T* (or *TID*) of a table *T* can be thought of as the *x* for all other columns of *T* (e.g. let *T=COUNTRIES*, with *Country* (1)='U.S.A.', *Country* (2)='China', *Country* (3)='Germany', *IntlTelPrefix* (1)='01', *IntlTelPrefix* (2)='02', *IntlTelPrefix* (3)='49', etc.).

Such minimal primary keys favor optimal foreign keys (see BPR5 below) with least possible time for computing joins. If storage space is a drastic concern, you might not add them to tables that are not referenced; otherwise, it is better to always add them both for avoiding tedious supplementary DBA tasks when they will become referenced too, as well as for homogeneity reasons. Obviously, derived/computed tables, be them temporary or not, may have no primary keys. Note that, not only in the Relational Data Model (RDM) [9], but also in all RDBMS versions, any key may be freely chosen as the primary one and that, actually, as a consequence, unfortunately, the vast majority of existing dbs are using concatenated and/or semantic, not only numeric, surrogate primary keys.

BPR2 (*Instances cardinalities*): Surrogate keys should always take values in integer sets whose cardinalities should reflect maximum possible number of elements of the corresponding sets.

For example, *#Cities* values should be between 0 and 999 for states/regions/departments/lands/etc. (e.g. in Oracle, NUMBER (3)), or 99,999 (e.g. NUMBER(5)) for countries, or 99,999,999 (e.g. NUMBER(9)) worldwide, whereas *#Countries* values should be between 0 and 250 (e.g. NUMBER(3)). Note that, for example, not specifying cardinality in Oracle (e.g. using only NUMBER), means that the system is using its corresponding maximum (i.e. NUMBER(38), which needs 22 bytes that not only wastes space, but is much slower, as it cannot be processed (e.g. for joins) by CPU arithmetic/logic units, which are the fastest ones, in only one memory cycle).

BPR3 (*Semantic keys*): Any fundamental (not temporary, not derived) table corresponding to an object set that is not a subset (of another object set) should have associated all of its corresponding semantic (candidate, ordinary) keys: either one-to-one columns or minimally one-to-one column products. With extremely rare exceptions (see the rabbit cages example in BPR1 above), any such non-subset table should consequently have at least one semantic key: any of its lines should differ in at least one non-primary key column (note that there is not only one NULL value, as, for example, Oracle and MS SQL Server erroneously assume in this context, but an infinite number of NULLs, all of them distinct!). Only subsets and derived/computed ones might not have semantic keys.

Note that any table may have more than one semantic key (and, according to a theorem from [7], combining those published in [10] and [11,12], a maximum number of keys equal to the combination of *n* taken [*n*/2] times, where *n* is the total number of semantic, non-primary key columns of the table and [*x*] is the integer part of *x*). Obviously, in order to reject implausible instances, all of them (just like all of the other existing constraints) should always be included in the corresponding conceptual models, schemes, and implementations.

For example, in any context, *COUNTRIES* has the following two semantic keys: *CountryName* (there may not be two countries having same name) and *Code* (no two countries may have the same code, those used from vehicle plates to the U.N.); *STATES* has three keys: *State* • *Country* (there may not be two states of a same country having same name), *Code* • *Country* (there may not be two states of a same country having same code), and *TelPrefix* • *Country* (there may not be two states of a same country having same telephone prefix). Derived set **ExtStates*, computed from *STATES* and *COUNTRIES* above (e.g. with an inner join between them on the object identifier *#Country* and *Country*, implemented as a foreign key in *STATES* referencing *#Country*, be it in a temporary table or in a persistent one), extending *STATES* with, say, *CountryName*, should not have any key (and no other constraint either!). Subset *IPS* of *EMPLOYEES*, storing only those employees having retention bonuses, together with their corresponding periods and amounts, should not have any semantic key either.

Note that, for all subsets tables, all of their lines may always be uniquely identified semantically too (through the subset surrogate primary key, see BPR1 above) by all of the keys of the corresponding superset table (in the above example, any important people in *IPS*, by all keys of *EMPLOYEES*). Also note that, obviously, subset tables too may have other semantic keys (not only their syntactical primary ones).

BPR4 (*No superkeys*): We should never consider superkeys (i.e. one-to-one column products that are not minimal, that is for which there is at least one column that can be dropped and the remaining sub-product is also one-to-one; equivalently, superkeys are those products that properly include keys, i.e. they include at least one key without being equal to it), either conceptually, or, especially, for implementations. We should always stick to keys (see BPR3 above).

For example, obviously, within the U.S., both *StateCode* and *StateName* are and will always be unique (one-to-one, hence keys); trivially, these two constraints should be added to any dbs including the *STATES* table (obviously, only when the sub-universe is limited to the U.S. or to any other particular country; worldwide, these two columns are not one-to one anymore–see BPR3 above).

Even if not that trivial, you should never also add either a product superkey with any of them (e.g. *StateCode* • *StatePopulation*) or, worse, of both of them (*StateCode* • *StateName)*: the result will not only be an unjustified bigger db, but especially a slower one, as it will have to enforce this superfluous constraint too for any insert or update concerning at least one of the corresponding values. Obviously, it would be even worse to replace such a key with, in fact, one of its superkeys: for example, if you do not add the two single unique constraints above, but, instead, you add the constraint that their product (*StateCode* • *StateName*) is unique, then you allow for implausible instances (e.g. that there may be several states having same name, but different codes, and/or several ones having same codes, but different names).

Note that, unfortunately, not only MS Access, SQL Server, Oracle, or MySQL do not reject superkeys!

BPR5 (*Foreign keys*): Any foreign key should always reference the primary key of the corresponding table. Hence, it should be a sole integer column: we should never use concatenated columns, neither non-integer columns as foreign keys. Moreover, their definitions should match exactly the ones of the corresponding referenced primary keys (see BPR2 above).

This rule is not only about minimum db space, but mainly for processing speed: numbers are processed by the fastest CPU sub-processors, the arithmetic ones (and the smaller the number, the fastest the speed: for not huge numbers, only one simple CPU instruction is needed, for example, in comparing two such numbers for, let's say, a join), whereas character strings need the slowest sub-processors (and a loop whose number of steps is directly proportional to the strings lengths); moreover, nearly a thousand natural numbers, each of them between 0 and nearly 4.3 billion, are read from the hard disk (the slowest common storage device) with only one read operation (e.g. from a typical index file), while reading a thousand strings of, let's say, 200 ASCII characters needs 6 such read operations.

Please note that, again, unfortunately, not only RDM, but almost all RDBMSs too allow foreign keys, be them concatenated or not, to reference not only primary keys (see BPR1 above), but anything else, including non-keys (provided that all of the corresponding columns belong to a same table).

Please also note that, most unfortunately, it is a widespread practice to declare concatenated primary keys containing concatenated foreign keys for chains of referencing tables; consequently, even if the root of such a chain has only a column as its primary key, the next table in the chain should have a primary key with arity of at least two, the third one's arity should be of at least three, etc.

For example, in very many dbs, *COUNTRIES*' primary key is *CountryName*, *STATES*' one is *Country* • *SName*, where *Country* is a foreign key referencing *CountryName*, and *CITIES*' one is *Country* • *State* • *City*, where *Country* • *State* is a foreign key referencing *Country* • *SName* of *STATES*.

BPR6 (*At least one not null non primary key column per table*): Any table should have at least one not accepting nulls column, other than its primary key: what would a line having only nulls (except for its syntactic surrogate key value) stand for?

BPR7 (*No constraints on not fundamental tables*): Temporary and derived (computed) tables should not have any constraints enforced: as they are not fundamental, they should be read-only for users; moreover, being computed from valid data with valid expressions, their instances are always plausible. Consequently, adding constraints on them would only slow down processing speed and increase db size for nothing.

BPR8 (*No superfluous fundamental tables or rows*): Fundamental tables should be useful. If, for example, the instances of such a table are always empty, then that table is superfluous. Similarly, if the set of values (the image) of a column of a non-empty fundamental table is always empty, that column is superfluous. A fundamental table on which no object depends upon, except for its triggers, might also be superfluous.

Implementation in Oracle

BPR9 (*Reduce data contention*): Critical tables, (almost) static ones, core ones (to the enterprise, being used by several applications), very large ones, temporary ones, all of the others, and indexes of any db should be placed in distinct tablespaces (and all of them also distinct from the system ones) for optimal fine-tuning (e.g. caching all frequently used small static tables in memory–see BPR10 below–, setting adequate block sizes, etc.), and thus performance.

Whenever several hard disks are available, tablespaces should cleverly exploit them all (e.g. storing critical tables tablespace on the fastest one). Create associated data files with auto extension enabled, rather than creating many small ones. Separate user data from system dictionary data to reduce I/O contention.

As data contention can substantially hurt application performance, reduce it by distributing data in multiple tablespaces and partitions, avoid constant updates of a same row (e.g. to calculate balance), and run periodic reports instead.

BPR10 (*Caching small frequently used tables*): Always cache small, especially lookup static (but not only) tables that are frequently used.

BPR11 (*Use correct data types*): Using incorrect data types might decrease the efficiency of the optimizer, hurt performance, and cause applications to perform unnecessary data conversions. Don't use strings to store dates, times or numbers. Ensure that conditional expressions compare the same data types. Do not use type conversion functions (such as TO_DATE or TO_ CHAR) on indexed columns: use instead the functions against the values being compared to a column.

BPR12 (*Not adding unnecessary indexes*): Indexes should not be added either for small instances tables or for columns containing mostly nulls in which you do not search for NOT NULL values. Obviously, there should not be more than one index on same columns (although there are RDBMSs allowing it! Fortunately, Oracle does not.).

BPR13 (*Concatenated indexes column order*): For concatenated indexes, the order of their columns should be given by the cardinality of their corresponding duplicate values: the first one should have the fewest duplicates, whereas any other one should have more duplicates than its predecessor and fewer than its successor; columns having very many duplicates or NULLs should then be either placed last, or even omitted from indexes.

For example, as in a *CITIES* table there are much more distinct *ZipCode* values (rare duplicates being possible only between countries) than *Country* ones (as, generally, there are very many cities in a country), the corresponding unique index should be defined in the order <*ZipCode*, *Country*> (not as <*Country*, *ZipCode*>).

BPR14 (*Indexes best types*): Normal (B-Trees) indexes should be used except for small range of values, when bitmap ones should be used instead. Note, however, that bitmap indexes are greatly improving queries, but are significantly slowing down updates and that they are available only for Enterprise Oracle editions and not for the standard ones too.

BPR15 (*Indexes effectiveness*): For improving indexes effectiveness, DBAs should regularly gather statistics on them. (Note that on some RDBMS–e.g. Oracle starting with 10*g*–this can be scheduled and performed automatically.)

BPR16 (*Avoid row chaining*): If *pctfree* setting is too low, updates may force Oracle to move rows (*row migration*). In some cases (e.g. when row length is greater than db block size) row chaining is inevitable. When row chaining and migration occur, each access of a row will require accessing multiple blocks, impacting the number of logical reads/writes required for each command. Hence, never decrease *pctfree* and always use largest block sizes possible.

Querying and manipulating

BPR17 (*Use parallelism*). Whenever possible, be it for querying and/or manipulating, use parallel programming!

1. Process several queries in parallel by declaring and opening multiple explicit cursors, especially when corresponding data is stored on different disks and the usage environment is a low-concurrency one.
2. Create and rebuild indexes in parallel.
3. Use PARALLEL ENABLED functions (including pipelined table ones), which allows them to be safely used in slave sessions of parallel query evaluations.

BPR18 (*Avoid dynamic SQL*): Whenever possible, avoid dynamic SQL and use views and/or parameterized stored procedures instead; when it is absolutely needed, keep dynamicity to the minimum possible (i.e. keep dynamic only what cannot be programmed otherwise and for the rest use static SQL) and prefer the dynamic native one, introduced as an improvement on the DBMS_SQL API, as it is easier to write and executes faster.

Note that DBMS_SQL is still maintained because of the inability of native dynamic SQL to perform so-called "Method 4 Dynamic SQL", where the name/number of SELECT columns or the name/number of bind variables is dynamic.

Also note that native dynamic SQL cannot be used for operations performed on a remote db. Views and stored procedures have obvious advantages: not only they are already parsed, but they also have associated optimized execution plans stored and ready to execute.

BPR19 (*Avoid subqueries*): Subqueries allow for much more elegant and close to mathematical logic queries, but are generally less efficient than corresponding equivalent join queries without subqueries (except for cases when RDBMSs optimizers replace them with equivalent subqueryless queries).

BPR20 (*Use result cache queries*): For significant performance improvement of frequently run queries with same parameter values, always use result cache queries and query fragments: as their results are cached in memory (in the result cache, part of the shared pool), there is no more need to re-evaluate them (trivially, except for the first time and then only for each time when underlying data is updated).

BPR21 (*Use compound triggers*): Always use compound triggers instead of ordinary ones, not only for new code, but also by replacing existing ordinary ones, whenever possible: they improve not only coding efficiency, but also processing speed for bulk operations.

BPR22 (*Set firing sequences*): Always use, whenever necessary, setting the triggers firing sequence (clause FOLLOWS), in order to control their execution order.

BPR23 (*Use function result cache*): Whenever appropriate, use result cache functions, which enhances corresponding code performance (e.g., according to Oracle, with at least 40% and up to 100% for most of pure Oracle PL/SQL code).

Optimization

BPR24 (*Regularly shrink tables and tablespaces*): Whenever rows are deleted from table instances, physical disk space they occupy is not freed, as, in fact, Oracle only marks them for deletion. Not only to gain storing space, but especially to speed up queries, you should regularly shrink involved tables and tablespaces.

BPR25 (*Regularly gather statistics on indexes*): For improving indexes effectiveness, DBAs should regularly gather statistics on them (by invoking stored procedures DBMS_STATS.GATHER_SCHEME_STATISTICS and DBMS_STATS.GATHER_TABLE_STATISTICS). Note that, starting with 10*g*, this can be scheduled and performed automatically by the GATHER_STATS_JOB of the Automatic Statistics Collection tool. Moreover, you might use the COMPUTE STATISTICS clause of the CREATE INDEX statement.

BPR26 (*Use key-compressed indexes*): Especially for unique multicolumn indexes, key compression should always be used for obtaining smaller and faster indexes: they eliminate repeated occurrences of key column prefix values, by sharing prefix entries among all suffix entries in index blocks. Note that the COMPRESS clause can be specified during either index creation or rebuild.

BPR27 (*Follow Oracle advisors recommendations*): Oracle advisors are almost always right, so the best thing to do is to regularly monitor and do what they are suggesting, preferably automatically; failed ones should especially be monitored and necessary actions should be taken a.s.a.p.

ODBDetective 1.0 investigation types

Design

1. Fundamental tables having no not-null columns (except for the primary key).
2. Keyless non-empty fundamental tables.
3. Fundamental tables (not corresponding to subsets) that don't have semantic keys.
4. Tables having concatenated primary keys.
5. Tables having non-numeric primary key columns.
6. Oversized surrogate primary keys.
7. Tables having superkeys.
8. Temporary tables constraints.
9. Empty fundamental tables, their associated indexes and constraints.
10. Empty columns in non-empty fundamental tables, their associated indexes, and constraints.
11. Tables on which no other interesting (i.e. not trigger only referring to the corresponding table) object (e.g. view, stored procedure, etc.) depends upon.

Implementation

12. User tables and/or indexes in system tablespaces.
13. Small (under *x* lines) non-cached fundamental tables.
14. Improperly typed foreign keys.
15. Concatenated foreign keys.
16. Foreign keys having non-numeric columns.
17. Over and under-sized foreign keys.
18. Concatenated indexes with wrong columns order.
19. Fundamental tables having migrated rows.
20. Normal (B-tree) indexes that should be of bitmap type (as their columns have less than y% distinct values or more than *z*% null ones).
21. Bitmap indexes that should be of normal (B-tree) type (as their columns have more than y% distinct values and less than *z*% null ones).
22. Tables with empty blocks.

Usage

23. Total number of sub-queries and each ones position in source code.
24. Total number of dynamic SQL executions and each ones position in source code.
25. Not parallel enabled functions and each ones position in source code.
26. Not result cache functions and each ones position in source code.
27. Not result cache queries and each ones position in source code.
28. Not compound triggers and each ones position in source code.
29. Triggers without firing sequences and each ones position in source code.

Optimization

30. Tables to shrink.
31. Indexes without recent statistics gathered on them.
32. Indexes to be compressed (on tables having less than *w* lines).

ODBDetective 1.0 Architecture

The 3-tier architecture of *ODBDetective* is presented in Figures 1 and 2:

The light GUI, developed in MS Access 2010, provides users with a simple, three levels menu and forms/reports for displaying/printing mainly investigation results, but also for accepting corresponding parameter values (e.g. the desired thresholds for caching tables or compressing indexes), as well as managing Oracle target servers and dbs (users) needed data (server names, connection strings, dbs names, users accounts and passwords, IPs, etc.), and additional semantic decision support data; it also provides radio buttons for selecting desired detection options (e.g. full list of subsection 2.2 above or only partial ones) and buttons for launching metadata mining, investigations, etc.

GUI-triggered events are handled by the VBA BL tier sub-layer, which is made up of three class forms (for servers, dbs, and investigation options respectively) and a library used by all of them. This sub-layer contains methods for adding, updating, and deleting data on targeted servers and dbs, deleting no more needed investigation data, adding (and even updating catalogue metadata) for the currently selected db, etc.

As usual, especially when extended SQL is available (PL/SQL in this case), there is also a RDBMS engine-based BL sub-layer, which, in this case, mainly includes detection and investigation views, but

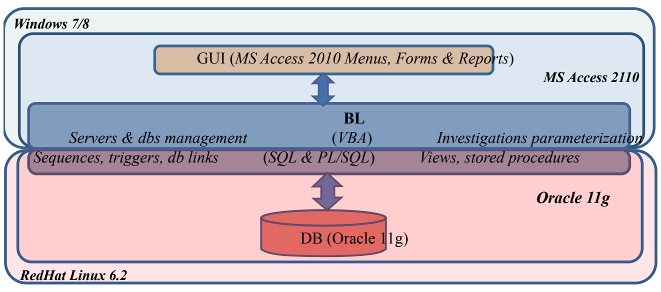


Figure 1: *OBD Detective's* overall architecture.

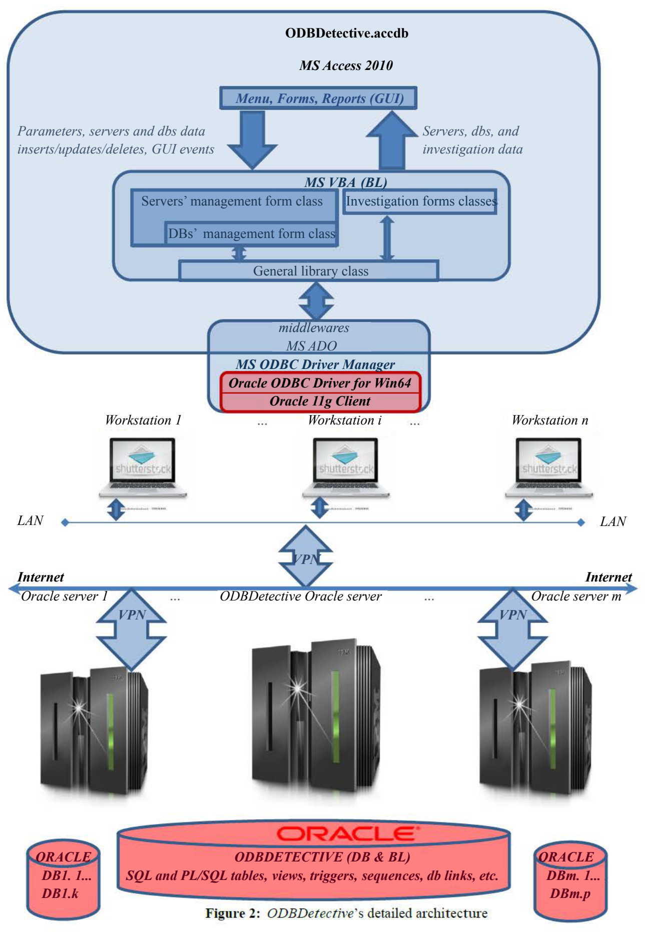

ODBDetective.accdb
MS Access 2010
Menu, Forms, Reports (GUI)
Parameters, servers and dbs data inserts/updates/deletes, GUI events
Servers, dbs, and investigation data
MS VBA (BL)
Servers' management form class
Investigation forms classes
DBs' management form class
General library class
middlewares
MS ADO
MS ODBC Driver Manager
Oracle ODBC Driver for Win64
Oracle 11g Client
Workstation 1
Workstation i
Workstation n
LAN
LAN
VPN
Internet
Internet
Oracle server 1
ODBDetective Oracle server
Oracle server m
VPN
VPN
ORACLE
ORACLE
DB1. 1...
DB1.k
ODBDETECTIVE (DB & BL)
SQL and PL/SQL tables, views, triggers, sequences, db links, etc.
ORACLE
DBm. 1...
DBm.p
Figure 2: ODBDetective's detailed architecture

Figure 2: *OBD Detective's* detailed architecture.

also triggers, sequences, stored functions and procedures, etc. Between these two BL sub-layers, a four level middleware pack is needed to bridge MS and Oracle, which comprises ADO (included in Access), MS ODBC Driver Manager, Oracle ODBC Driver for Windows, and Oracle Client 11*g*.

All MS Access 2010 layers of the solution are encapsulated in a pure code (except for the table which drives the menu) concurrent db called *ODBDetective.accdb*, which can be deployed on any number of workstations having (VPN) access to an Oracle 11*g* server where the *ODBDetective*'s SQL & PL/SQL BL sub-layer, as well as its DB layer reside. Through db links, *ODBDetective* can mine and import metacatalog data from any number of other Oracle servers' dbs too.

ODBDetective 1.0 Design and Implementation

Database design and implementation

ODBDetective's db design and implementation were done according to the algorithms presented in [7] and [13]. For investigating possible violations of the best practice rules presented in section 2.2 above by Oracle db schemes, *ODBDetective* has to mine in the corresponding server catalog for metadata on dbs (users), tablespaces, objects, their dependencies, tables, views, columns, constraints, indexes, their columns, triggers, and PL/SQL stored functions and procedures code. Fortunately, Oracle provides both DBA and user versions of views for all such metadata, from which *ODBDetective* extracts only relevant columns into its own corresponding db tables.

Besides the above corresponding object sets, *ODBDetective* offers some dictionary-type ones (for decrypting Oracle internal codifications), as well as a set of Oracle servers and one of dbs (just like, for example, Oracle's SQL Developer, for saving needed connections data). Figure 3 presents the *ODBDetective*'s structural Entity-Relationship (*E-R*) Diagram (*E-RD*).

Note that, for graphical reasons, canonical injections are represented as, instead of ⊂. Also note that, again for graphical reasons, although indexes are Oracle objects too, in Figure 3 this inclusion (*INDEXES* ⊂ *OBJS*) is not depicted.

Moreover, note that, except for the properties of *SERVERS* and *DBS*, all other ones (except for some few properties added for future extensions and **DB* of *OBJS*, as well as **Tablespace* of *OBJS*, which are computed) are obtained through data mining, so they are read-only: consequently, there are no restrictions on them. Note too that the *Package* property of *METHODS* (which abstracts both stored PL/SQL functions and procedures) is not compulsory (as they may be methods defined outside PL/SQL packages). Perhaps the most important thing to note is that Oracle metacatalogs are uni-server: although all of its *g* versions are supporting grids of interconnected servers, there is no centralized metacatalog: consequently, *ODBDetective* had to add to its db computed property **Server* of *TABLESPACES*.

Oracle, in fact, does not abstract PL/SQL stored function and procedures into a same methods set (so, for *ODBDetective*, *METHODS*=*FUNCTIONS* ⊕ *PROCEDURES*), neither differentiates between views and tables columns (although most of the views ones

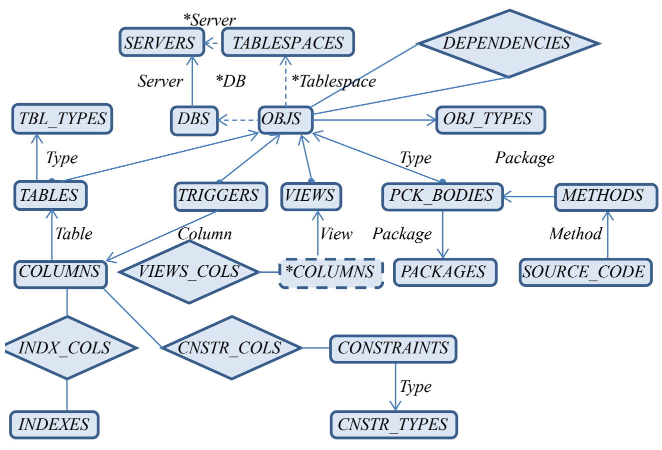


Figure 3: *OBD Detective's* structural E-RD.

are obtained from tables ones), nor, unfortunately, stores for each PL/SQL source code line the method to whom it belongs, but only the corresponding package body.

The associated restrictions list (*RL*) is the following one:

SERVERS (the set of Oracle db servers of interest)

RL1 (*servers' max. cardinality*): 200

RL2 (*servers' domain ranges*):

ServerName: ASCII(64)

HostIP: ASCII(16)

SID, *ServiceName*, *Pw*: ASCII(30)

Port: [1521, 1575] $\subseteq$ NAT(4)

SMode: {'Dedicated', 'Shared'}

Protocol: {'TCP', 'IPC'}

**Connection String*: ASCII(255), computable according to the template:

(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=*Protocol*)(HOST=*HostIP*)(PORT=*Port*)))(CONNECT_DATA=(SERVER =*SMode*)(SERVICE_NAME=*SID*/*ServiceName*)))

RL3 (*servers' mandatory properties*): *ServerName*, *HostIP*, *Port*, *SMode*, *Protocol*

RL4 (*servers' uniqueness restrictions*): *ServerName* are *HostIP* are unique (there may not be two servers with either same name or same host address); note that, generally, according to **ConnectionString* definition, *HostIP* is not a key (but *Protocol* • *HostIP* • *Port* • *SMode* • *SID* and *Protocol* • *HostIP* • *Port* • *SMode* • *ServiceName* are instead), as several connections to a same server are allowed (for example with different protocols and/or modes); obviously, *ODBDetective* only needs one per server, which is why *HostIP* is unique for it (and, consequently, *Protocol* • *HostIP* • *Port* • *SMode* • *SID* and *Protocol* • *HostIP* • *Port* • *SMode* • *ServiceName* are superkeys).

RL5 (*servers' non-existence restrictions*): whenever *SID* is null, *ServiceName* may not be null and vice-versa (i.e. whenever *ServiceName* is null, *SID* may not be null) (that is either *SID* or *ServiceName* should be specified, but not both of them)

DBS (the set of Oracle dbs of interest)

RL6 (*dbs' max. cardinality*): 20000

RL7 (*dbs' domain ranges*):

DB: {['A'-'Z'], [0-9], '_'}(30) *Prefix*(*DB*, 4) $\notin$ {"SYS_", "ORA_"} (Oracle names may not exceed 30 characters, which should be either capital letters, numbers, and/or underscore; moreover, their prefixes may not be "SYS_" or "ORA_")

Pw: ASCII(30)

**Link_Name*: ASCII (128), computable according to the template: *DB* & "#" & *Server*

RL8 (*dbs' mandatory properties*): *DB*, *Server*, *Pw* RL9 (*dbs' uniqueness restrictions*): *Server* and *DB* are unique (there may not be two dbs with same name on a same server)

OBJ_TYPES (the quasi-static set of Oracle db object types of interest)

RL10 (*object types' max. cardinality*): 32

RL11 (*object types' domain ranges*):

ObjType: {"Function", "Index", "Java Class", "Package", "Package Body", "Procedure", "Synonym", "Trigger", "Type", "View"}

RL12 (*object types' mandatory properties*): *ObjType*

RL13 (*object types' uniqueness restrictions*): *ObjType* is unique (there may not be two Oracle object types with same name)

CNSTR_TYPES (the quasi-static set of Oracle db constraint types of interest)

RL14 (*constraint types' max. cardinality*): 16

RL15 (*constraint types' domain ranges*):

CnstrCode: {'C', 'F', 'H', 'O', 'P', 'R', 'S', 'U', 'V'}

CnstrType: {"Check", "REF", "Hash", "R/O", "Primary", "Referential", "Supplemental", "Unique", "View"}

CnstrDescr: {" Check constraint on a table", "Constraint involving a REF column", "Hash expression", "Read-only view", "Primary key", "Referential integrity (foreign key)", "Supplemental logging", "Unique constraint (non-primary superkey)", "Check constraint on a view"}

RL16 (*constraint types' mandatory properties*): *CnstrCode*, *CnstrType*

RL17 (*constraint types' uniqueness restrictions*): *CnstrCode*, *CnstrType*, *CnstrDescr* are unique

(there may not be two Oracle constraint types with same code, name, or description)

TBLS_TYPES (the quasi-static set of *ODBDetective* table types)

RL18 (*table types' max. cardinality*): 8

RL19 (*table types' domain ranges*):

Code: {'B', 'C', 'G', 'N', 'R', 'U', 'Y', 'l'}

Comments: {" To be reviewed", "Critical table", "Not used anymore: to be dropped", "Rarely used", "Small table that can grow large", "Used table", "Small table to be cached", "System tables: null value"}

RL20 (*table types' mandatory properties*): *Code*, *Comments*

RL21 (*table types' uniqueness restrictions*): *Code* and *Comments* are unique (there may not be two *ODBDetective* table types with same code or comment)

Mathematical scheme

By applying the algorithms for translating E-RDs and restriction lists into mathematical schemes (see [7], [13]) and for detecting and designing constraints to the above E-RD and restrictions list, the following refined mathematical scheme is obtained:

SERVERS

S# $\leftrightarrow$ [-100, 100], total

ServerName $\leftrightarrow$ ASCII(64), total

HostIP $\leftrightarrow$ ASCII(16), total

SID $\rightarrow$ ASCII(30)

ServiceName $\rightarrow$ ASCII(30)

Pw → ASCII(30)

Port → [1521, 1575] ⊆ NAT(4), total

SMode → {'Dedicated', 'Shared'}, total

Protocol → {"TCP", "IPC"}, total

**ConnectionString* ↔ ASCII (255) (superkey, as *HostIP* is a key)

**ConnectionString* (*x*) = "(DESCRIPTION=(ADDRESS_LIST=(ADDRESS="&(PROTOCOL=" & *Protocol*(*x*) & ")(HOST=" & *HostIP*(*x*) & ")(PORT=" & *Port*(*x*) &")))(CONNECT_DATA=(SERVER =" & *SMode*(*x*) & ")(SERVICE_NAME=" & *iif*(*Isnull*(*SID*(*x*)), *ServiceName*(*x*), *SID*(*x*)) & ")))"

Initial keys (according to RL4) are {*S#*, *ServerName*, *HostIP*}. Results of applying the algorithm for keys detection are given in Table 1.

Consequently, SERVERS does not have any other keys than the three initial ones.

C5 (according to RL5): (∀*x*∈*SERVERS*) (*SID*(*x*) ∈ NULLS ⇒ *ServiceName*(*x*) ∉ NULLS ∧ *ServiceName*(*x*) ∈ NULLS ⇒ *SID*(*x*) ∉ NULLS)

DBS

D# ↔ [-10000, 10000], total

DB → {['A'-'Z'], [0-9], '_'}(30), total

Pw → ASCII(30), total

**Link_Name* → ASCII(128)

**Link_Name*(*x*) = *DB*(*x*) & "#" & *Server*(*x*)

Server: DBS → *SERVERS*, total

C7 (according to RL7): (∀*x*∈*DBS*) (*Prefix*(*DB*(*x*), 4) ∉ {"SYS_", "ORA_"})

Initial keys (according to RL9) are {*D#*, *DB* • *Server*}.

Results of applying the algorithm for keys detection are given in Table 2.

Consequently, *DBS* does not have any other keys than the initial one.

OBJ_TYPES

OT# ↔ [1, 32] ⊂ NAT(2), total

ObjType ↔ {"Function", "Index", "Java Class", "Package", "Package Body", "Procedure", "Synonym", "Trigger", "Type", "View"} ⊂ ASCII(16), total

CNSTR_TYPES

CT# ↔ [1, 16] ⊂ NAT(2), total

CnstrCode ↔ {'C', 'F', 'H', 'O', 'P', 'R', 'S', 'U', 'V'} ⊂ ASCII(1), total

CnstrType: {"Check", "REF", "Hash", "R/O", "Primary", "Referential", "Supplemental", "Unique", "View"} ⊂ ASCII(16), total

CnstrDescr ↔ {" Check constraint on a table", "Constraint involving a REF column", "Hash expression", "Read-only view", "Primary key", "Referential integrity (foreign key)", "Supplemental logging", "Unique constraint (non-primary superkey)", "Check constraint on a view"} ⊂ ASCII(64), total

TBLS_TYPES

TT# ↔ [1, 8] ⊂ NAT(1), total

Code ↔{'B', 'C', 'G', 'N', 'R', 'U', 'Y', 'I'} ⊂ ASCII(1), total

Comment ↔ {" To be reviewed", "Critical table", "Not used anymore: to bedropped", "Rarely used", "Small table that can grow large", "Used table", "Small table to be cached", "System tables: null value"} ⊂ ASCII(32), total Applying the E-RD cycles detection and analysis algorithm, the following cycles exist in Figure 3:

- The cycle having nodes *DEPENDENCIES* and *OBJS* is of commutative type, but with length = two, corresponding to the binary homogeneous relation *DEPENDENCIES*, which should obviously be acyclic (as no Oracle object may depend either directly or indirectly on itself); as this constraint is enforced by Oracle, no other explicit constraint needs to be added to the math scheme;
- The cycle having nodes *OBJS*, *DBS*, *TABLESPACES*, *SERVERS* is of commutative type, should commute, and commutes by the definition of computed mappings **Server*, **Tablespace*, **DB* (*Server* ° **DB* = **Server* ° **Tablespace*, as any Oracle db object belonging to a db should reside in a tablespace belonging to the same Oracle db server as the corresponding db): no other explicit constraint needs to be added to the math scheme;
- The cycle having nodes *TRIGGERS*, *COLUMNS*, *TABLES*, *OBJS* is of commutative type, but should not commute, as, trivially, any trigger attached to a table column is a different object than that table: no other explicit constraint needs to be added to the math scheme;

Candidate	*Key?*	*Prime?*	*Proof*
SID	No	No	There may be any number of servers having same *SID; SID* Cannot take part in any *SERVERS* key.
ServiceName	No	No	There may be any number of servers having same ServiceName; ServiceName cannot take part in any *SERVERS* key.
Pw	No	No	There may be any number of servers having same *Pw; Pw* cannot take part in any *SERVERS* key.
SMode	No	No	There may be any number of servers having same *SMode; SMode* cannot take part in any *SERVERS* key.
Protocol	No	No	There may be any number of servers having same *Protocol; Protocol* cannot take part in any *SERVERS* key.

Table 1: Applying the algorithm for keys detection according to RL4.

Candidate	*Key?*	*Prime?*	*Proof*
DB	No	Yes	There may be any number of dbs having same names (on different servers); according to RL9, *DB* takes part in the *Server* DB key.
Pw	No	No	There may be any number of dbs having same *Pw* (even on a same server); *Pw* cannot take part in any *DBS* key.
Server	No	No	There may be any number of dbs on a same server; according to RL9, *Server* takes part in the *Server DB* key.
DB Server	Yes		According to RL9, there may not be two dbs having a same name on a same server.

Table 2: Applying the algorithm for keys detection according to RL9.

- The cycle having nodes *TRIGGERS*, *COLUMNS*, *VIEWS_COLS*, **COLUMNS*, *VIEWS*, *OBJS* is of commutative type, but should not commute, as, trivially, any trigger attached to a table column is a different object than any view whose columns are based on that table column: no other explicit constraint needs to be added to the math scheme;
- The cycle having nodes *TABLES*, *COLUMNS*, *VIEWS_COLS*, **COLUMNS*, *VIEWS*, *OBJS* is of commutative type, but should not commute, as, trivially, any table is a different object than any view whose columns are based on that table columns: no other explicit constraint needs to be added to the math scheme;
- The cycle having nodes *INDEXES*, *INDX_COLS*, *COLUMNS*, *TABLES*, *OBJS* (not depicted, but existing)
- is of commutative type, but should not commute, as, trivially, any index (built on a table's columns) is a different object than any view whose columns are based on that table's columns: no other explicit constraint needs to be added to the math scheme;
- The cycle having nodes *INDEXES*, *INDX_COLS*, *COLUMNS*, *TRIGGERS*, *OBJS* (not depicted, but existing)
- is of commutative type, but should not commute, as, trivially, any index (built on a table's columns) is a different object than any trigger attached to a column of that table: no other explicit constraint needs to be added to the math scheme. The corresponding relational scheme is presented in [4] and [5]; its associated non-relational constraints list is the following:

C5 (according to RL5): $(\forall x \in SERVERS)$ $(SID(x) \in \text{NULLS} \Rightarrow ServiceName(x) \notin \text{NULLS} \wedge ServiceName(x) \in \text{NULLS} \Rightarrow SID(x) \notin \text{NULLS})$

C7 (according to RL7): $(\forall x \in DBS)$ $(Prefix(DB(x), 4) \notin \{$"SYS_", "ORA_"$\})$.

These non-relational constraints are implemented in *ODBDetective* in VBA, which can provide users with context sensitive and immediate error messages (whilst Oracle is returning only context independent error messages, which are not immediate, at cell level, but always delayed at the row one).

All *ODBDetective's* tables, constraints, sequences, triggers, db links, stored procedures, and views were implemented according to the algorithm [7,13] for translating relational schemes and associated non-relational constraint lists into Oracle 11*g* dbs, by using Oracle SQL Developer. Using Oracle's db links, DBA and user views, as well as parameterized stored procedures (the parameters being db links), *ODBDetective* is filling its db with all needed metadata for investigation, thus impacting very few (only in read-only mode, only from the metacatalogs, only once) on the customers' investigated Oracle remote servers, dbs, and network bandwidth

(typically, the size of the *ODBDetective*'s db is only some 200MB). Mined metadata is investigated by *ODBDetective* with the aid of (hierarchies of) views, built upon its db.

Note that, unfortunately, for tables *SERVERS* and *DBS* whose instances are managed by the *ODBDetective* MS Access application, their constraints had to be also implemented in VBA: in order to avoid getting the Oracle context-insensitive corresponding error messages, VBA is enforcing all these constraints too.

Also note that if you declare NOT NULL constraints in Oracle, when users are trying to leave such a column cell null, unfortunately, corresponding VBA *BeforeUpdate* trigger-type methods are not invoked, as ADO is first passing nulls to Oracle, which is rejecting them. Consequently, the only way to give users context sensitive error messages in such cases (and, moreover, to help them by undoing accidentally emptied cells) is not to enforce NOT NULL constraints in Oracle, but only in VBA. This is why *ODBDetective*'s NOT NULL constraints are not enforced in Oracle.

Dually, but for the same reason of displaying context sensitive error messages and undoing implausible data updates, as well as because of the fact that Oracle, ODBC, and ADO error trapping in VBA is problematic, the rest of the *ODBDeetective*'s relational type constraints are enforced both in Oracle and VBA.

For example, there are times when Oracle or ODBC errors are not triggering ADO and/or Access corresponding errors; much worse, there are even contexts (e.g. connecting to an Oracle db) when, dually, although there is no Oracle, ODBC, or Access error, ADO is however reporting an error!

Application design and implementation

ODBDetective's application design was done according to the principles and methodologies presented in [13-17].

It was decided that the best software life cycle that suits *ODBDetective*'s needs is the *Prototype Software Development* one, as there is only one developer, so that *Incremental Prototyping* methodology was used.

The only human actor that will interact with the *ODBDetective* application is the *User*, who analyzes db schemes for detecting anomalies of their design, implementation, and usage, and provides statistics and decision support data, for improving dbs performance by correcting their schemes in order to eliminate best practices violations. Figure 4 presents the *ODBDetective* system's use cases:

ODBDetective's menu is a very simple one, consisting only of three levels, with a main menu page, a sub-menu one, and over 60 forms (some of them used as sub-forms on other three hierarchical levels of embedding), implemented by using MS Access standard Switchboard Manager. Most of the forms (over 40) are read-only, as they only

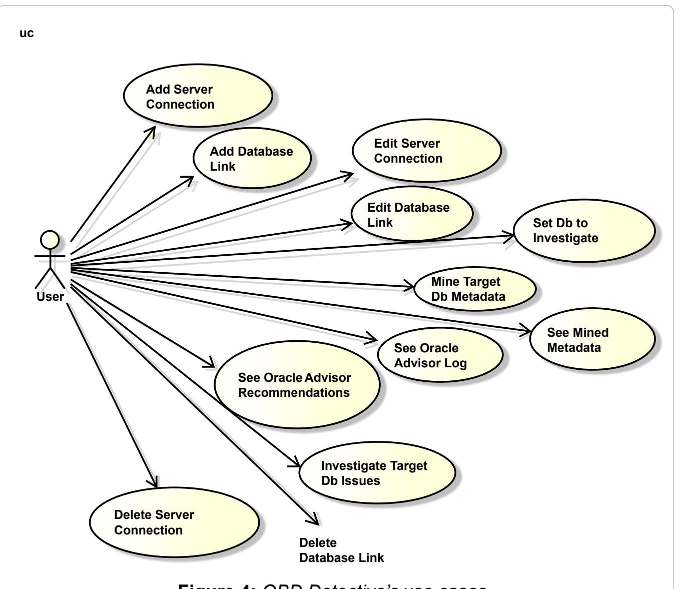


Figure 4: *OBD Detective's* use cases.

display mining and investigation data. The rest of them also allow for writing additional semantics and decision support data, so they also have associated VBA classes for enforcing corresponding constraints.

In order to avoid hard-coding and re-definitions, all commonly used constants and methods are grouped in a *General* VBA library.

Please note that, generally, VBA programming for ODBC-connected back-ends is more demanding than for MS Jet or ACME, or even SQL Server ones: for example, re-querying and even refreshing are not automatic (but need explicit programming), ADO, so Oracle too, are taking precedents in the events chain over VBA trigger-like methods (which, normally, is not the same).

Oracle table schemes default values are not automatically copied into Access forms, VBA object properties like *NewRecord* and *OldValue* are not available, etc.; moreover, as Oracle does not provide auto-numbering, you should first define corresponding Oracle sequences and triggers and then you should carefully program in VBA all corresponding methods, as these Oracle trigger generated surrogate key values are only generated just before inserting new rows in Oracle tables (and are null before), whereas Access auto-numbering generates them before saving any new data in virtual memory, long before saving it to disk, in dbs.

Odbdetective 1.0 Usage and Case Study

Installing *ODBDetective* 1.0 is very easy, as it only needs executing a SQL script for creating and initializing its Oracle 11*g* db and copying the ODBDetective.accdb MS Access 2007-2013 file in any folder of a pc running Windows 7/8. Configuring Access to remotely link to the Oracle db needs installing the Oracle 11g client (freely downloadable from the Oracle website), which also includes the corresponding Oracle ODBC driver for Windows (32 or 64bit, depending on your actual platform).

Figure 5 presents *ODBDetective* 1.0 main menu and Figure 6 its submenu. Figure 7 shows the *Manage known servers and users* window, with which users may add, update, and/or delete both Oracle db servers connection and db links data. Figure 8 displays the *Investigations* window. Figure 9 presents the *Global Statistics* window, which displays totals on both mined data and investigation results. Figure 10 shows the *Table details* form, which displays essential mined table, columns, triggers, constraints, and indexes data. Other similar forms display metadata on tablespaces, objects and dependencies between them, views, sequences, and PL/SQL source code lines, as well as Oracle's advisors log and recommendations for the investigated db. Figure 11 displays an example of detailed investigation result (tables violating

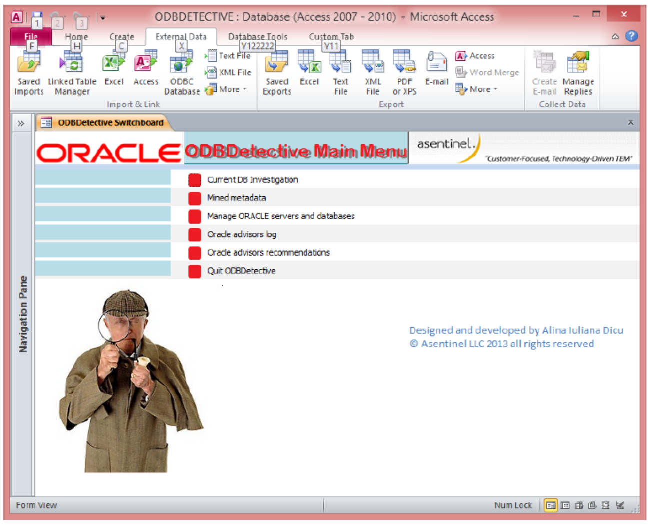

ODBDETECTIVE : Database (Access 2007 - 2010) - Microsoft Access
ODBDetective Switchboard
ORACLE ODBDetective Main Menu
asentinel. "Customer-Focused, Technology-Driven TEM"
Current DB Investigation
Mined metadata
Manage ORACLE servers and databases
Oracle advisors log
Oracle advisors recommendations
Quit ODBDetective
Designed and developed by Alina Iuliana Dicu
© Asentinel LLC 2013 all rights reserved
Navigation Pane
Form View

Figure 5: *OBD Detective's* application main switchboard menu window.

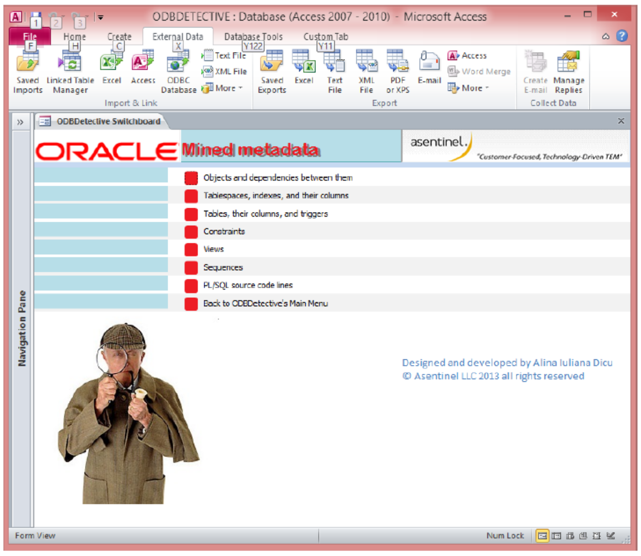
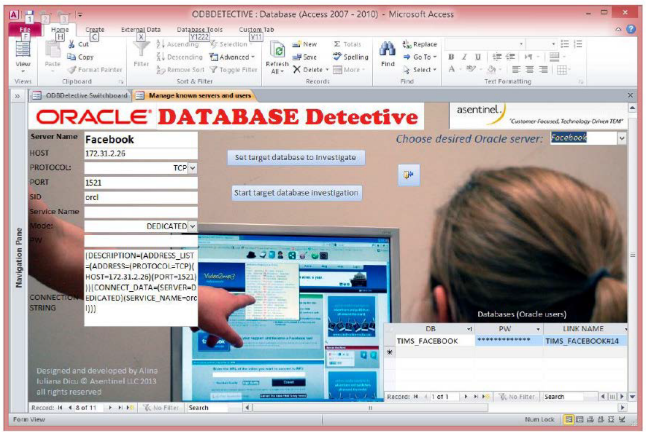

ODBDETECTIVE : Database (Access 2007 - 2010) - Microsoft Access
ODBDetective Switchboard
ORACLE Mined metadata
asentinel. "Customer-Focused, Technology-Driven TEM"
Objects and dependencies between them
Tablespaces, indexes, and their columns
Tables, their columns, and triggers
Constraints
Views
Sequences
PL/SQL source code lines
Back to ODBDetective's Main Menu
Designed and developed by Alina Iuliana Dicu
© Asentinel LLC 2013 all rights reserved
Navigation Pane
Form View

Figure 6: *OBD Detective's* application *Mined metadata* submenu.


ODBDETECTIVE : Database (Access 2007 - 2010) - Microsoft Access
ODBDetective Switchboard
Manage known servers and users
ORACLE DATABASE Detective
asentinel. "Customer-Focused, Technology-Driven TEM"
Set target database to investigate
Start target database investigation
Designed and developed by Alina Iuliana Dicu © Asentinel LLC 2013 all rights reserved
Navigation Pane
Form View

Server Name: Facebook
HOST: 172.31.2.26
PROTOCOL: TCP
PORT: 1521
SID: orcl
Service Name:
Mode: DEDICATED
PW:
CONNECTION STRING: (DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=172.31.2.26)(PORT=1521)))(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=orcl)))

Choose desired Oracle server: Facebook

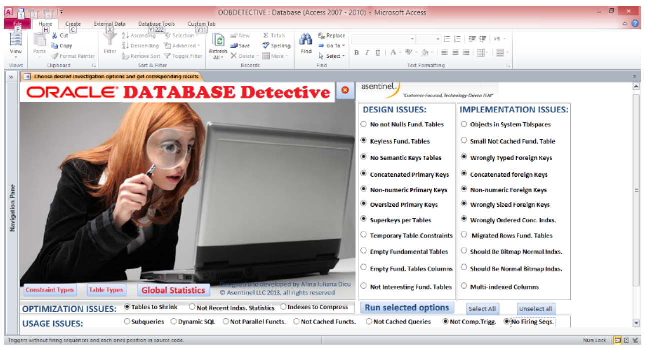
Databases (Oracle users)

DB	PW	LINK NAME
TIMS_FACEBOOK	************	TIMS_FACEBOOK#14

Record: 1 of 1

Record: 8 of 11

Figure 7: *OBD Detective's* application *Manage known servers and users* window.


ODBDETECTIVE : Database (Access 2007 - 2010) - Microsoft Access
Choose desired investigation options and get corresponding results
ORACLE DATABASE Detective
asentinel. "Customer-Focused, Technology-Driven TEM"
Constraint Types
Table Types
Global Statistics
Designed and developed by Alina Iuliana Dicu © Asentinel LLC 2013, all rights reserved
Run selected options
Select All
Unselect all
Navigation Pane

DESIGN ISSUES:
- No not Nulls Fund. Tables
- [✓ Keyless Fund. Tables]
- [✓ No Semantic Keys Tables]
- [✓ Concatenated Primary Keys]
- [✓ Non-numeric Primary Keys]
- [✓ Oversized Primary Keys]
- [✓ Superkeys per Tables]
- Temporary Table Constraints
- Empty Fundamental Tables
- Empty Fund. Tables Columns
- Not Interesting Fund. Tables

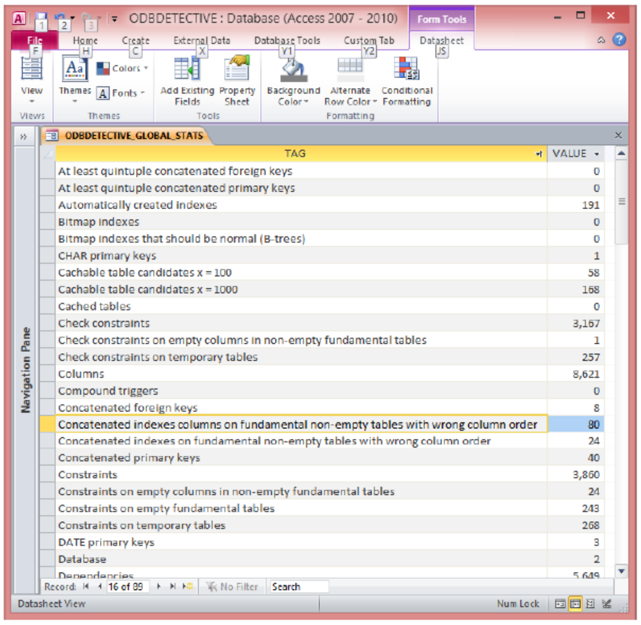
IMPLEMENTATION ISSUES:
- Objects in System Tblspaces
- Small Not Cached Fund. Table
- [✓ Wrongly Typed Foreign Keys]
- [✓ Concatenated foreign Keys]
- [✓ Non-numeric Foreign Keys]
- [✓ Wrongly Sized Foreign Keys]
- [✓ Wrongly Ordered Conc. Indxs.]
- Migrated Rows Fund. Tables
- Should Be Bitmap Normal Indxs.
- Should Be Normal Bitmap Indxs.
- Multi-indexed Columns

OPTIMIZATION ISSUES: [✓ Tables to Shrink] Not Recent Indxs. Statistics Indexes to Compress

USAGE ISSUES: Subqueries Dynamic SQL Not Parallel Functs. Not Cached Functs. Not Cached Queries [✓ Not Comp.Trigg.] [✓ No Firing Seqs.]

Triggers without firing sequences and each ones position in source code.

Figure 8: *OBD Detective's* application *investigations* window.

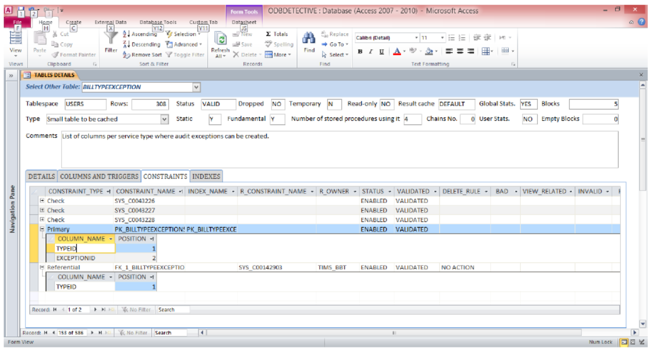


ODBDETECTIVE_GLOBAL_STATS

TAG	VALUE
At least quintuple concatenated foreign keys	0
At least quintuple concatenated primary keys	0
Automatically created indexes	191
Bitmap indexes	0
Bitmap indexes that should be normal (B-trees)	0
CHAR primary keys	1
Cachable table candidates x = 100	58
Cachable table candidates x = 1000	168
Cached tables	0
Check constraints	3,167
Check constraints on empty columns in non-empty fundamental tables	1
Check constraints on temporary tables	257
Columns	8,621
Compound triggers	0
Concatenated foreign keys	8
Concatenated indexes columns on fundamental non-empty tables with wrong column order	80
Concatenated indexes on fundamental non-empty tables with wrong column order	24
Concatenated primary keys	40
Constraints	3,860
Constraints on empty columns in non-empty fundamental tables	24
Constraints on empty fundamental tables	243
Constraints on temporary tables	268
DATE primary keys	3
Database	2
Dependencies	5,649

Record: 16 of 89 No Filter Search

Figure 9: *OBD Detective's* application *Global Statistics* window.



TABLES DETAILS

Select Other Table: BILLTYPEEXCEPTION

Tablespace	USERS	Rows:	308	Status	VALID	Dropped	NO	Temporary	N	Read-only	NO	Result cache	DEFAULT
Global Stats.	YES	Blocks	5										
Type	Small table to be cached	Static	Y	Fundamental	Y	Number of stored procedures using it	4	Chains No.	0	User Stats.	NO	Empty Blocks	0

Comments: List of columns per service type where audit exceptions can be created.

DETAILS | COLUMNS AND TRIGGERS | CONSTRAINTS | INDEXES

CONSTRAINT_TYPE	CONSTRAINT_NAME	INDEX_NAME	R_CONSTRAINT_NAME	R_OWNER	STATUS	VALIDATED	DELETE_RULE	BAD	VIEW_RELATED	INVALID
Check	SYS_C0043226				ENABLED	VALIDATED				
Check	SYS_C0043227				ENABLED	VALIDATED				
Check	SYS_C0043228				ENABLED	VALIDATED				
Primary	PK_BILLTYPEEXCEPTION!	PK_BILLTYPEEXCE			ENABLED	VALIDATED				
Referential	FK_1_BILLTYPEEXCEPTIO		SYS_C00142903	TIMS_BBT	ENABLED	VALIDATED	NO ACTION			

Primary:

COLUMN_NAME	POSITION
TYPEID	1
EXCEPTIONID	2

Referential:

COLUMN_NAME	POSITION
TYPEID	1

Record: 1 of 2 No Filter Search

Record: 153 of 586 No Filter Search

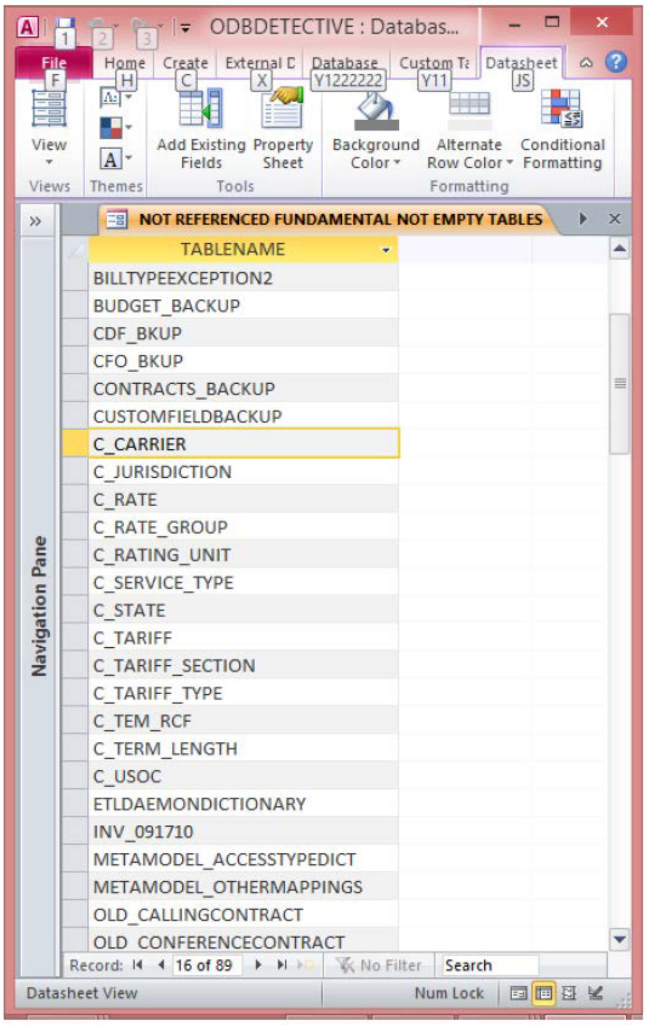
Figure 10: *OBD Detective's* application *Table details* window.

one best practice rule).

Note that, for example, form *Table details*, between others, also allows for storing decision support data. Also note that, unfortunately, the only Oracle recommendation type is, in this case too, to perform shrinking: a useful, but not at all an enough one [4] also presents in detail the results of a *ODBDetective*'s case study: investigating a production Oracle db consisting of 3,860 user objects (and 5,649 dependencies between them)–with only 116 being temporary and 191 automatically generated indexes ones–, out of which 586 tables (totaling 8,621 columns and 129,956,314 rows) interconnected by 415 foreign keys, with 66 temporary ones, and 2,842 indexes (out of which 2,651 are explicitly defined); their instances' plausibility is enforced by 3,860 constraints, out of which 278 are unique ones, with 229 primary and 49 semantic keys, 3,167 are check constraints, the remaining 415 being referential integrities; there are also one view, 151 sequences, 19 triggers, 127 packages (containing 1,647 procedures and 467 functions) totaling 129,730 PL/SQL lines.

Running all of the 32 *ODBDetective* 1.0 options, discovered were the following:

- ✓ 180 empty non-temporary tables, on which there were defined 211 indexes and 243 constraints;
- ✓ 89 fundamental not empty tables on which no interesting object (table, PL/SQL code, etc.) depends on;
- ✓ no cached table, although there were 168 such candidates for *x*=1,000 and 58 for *x*=100;
- ✓ 268 constraints on temporary tables (257 check, 10 primary, and one semantic keys);
- ✓ 2 tables having migrated rows;
- ✓ 311 empty columns, out of which 161 were not VARCHAR (1 BLOB, 2 CLOB, 3 CHAR, 49 DATE, and 106 NUMBER), and on which defined were 24 constraints (1 check, 1 unique, and 22 foreign keys) and 21 indexes;
- ✓ 292 keyless non-empty fundamental tables;
- ✓ 40 tables having concatenated primary keys (3 quaternary, 8 ternary, and 29 double);
- ✓ 15 not numeric primary keys (11 VARCHAR2, 1 CHAR, and 3 DATE);



NOT REFERENCED FUNDAMENTAL NOT EMPTY TABLES

TABLENAME
BILLTYPEEXCEPTION2
BUDGET_BACKUP
CDF_BKUP
CFO_BKUP
CONTRACTS_BACKUP
CUSTOMFIELDBACKUP
C_CARRIER
C_JURISDICTION
C_RATE
C_RATE_GROUP
C_RATING_UNIT
C_SERVICE_TYPE
C_STATE
C_TARIFF
C_TARIFF_SECTION
C_TARIFF_TYPE
C_TEM_RCF
C_TERM_LENGTH
C_USOC
ETLDAEMONDICTIONARY
INV_091710
METAMODEL_ACCESSTYPEDICT
METAMODEL_OTHERMAPPINGS
OLD_CALLINGCONTRACT
OLD_CONFERENCECONTRACT

Record: 16 of 89 No Filter Search

Figure 11: OBD Detective's application example of investigation results.

- ✓ all 239 surrogate primary keys were oversized (and, what is worse, to the maximum possible NUMBER(38) value!);
- ✓ 8 concatenated foreign keys (4 ternary and 4 double);
- ✓ 63 foreign keys having non-numeric columns; all 415 foreign keys were oversized (again, at the maximum possible: NUMBER (38)!);
- ✓ 273 improperly typed foreign keys;
- ✓ 576 normal indexes that should have been bitmap instead (for *y*=10 and *z*=90);
- ✓ 56 indexes to be compressed for *w*=100 and other 37 for *w*=1,000;
- ✓ 24 concatenated indexes (out of which 14 were unique ones—with 6 for primary keys!) totalizing 80 columns on non-empty tables having wrong columns order;
- ✓ 740 subqueries;
- ✓ 317,542 dynamic SQL statements;
- ✓ unfortunately too, there were no statistics gathered on indexes, no parallel enabled functions, no result cache queries or functions, no compound triggers, and no triggers with firing sequences either;
- ✓ fortunately, there were no tables or indexes in system tablespaces, no table having no not-null columns, no table to shrink, no superkeys, and no bitmap indexes that should have been normal (as there were no bitmap indexes at all).

According to this data (that you may see in the *Global Statistics* form, see 6.5.1 above), the customer decided to apply, in a first step, the following changes to its db scheme:

- downsizing primary keys to corresponding tables' double of maximum cardinals
- replacing concatenated primary keys with simple surrogate ones
- replacing concatenated foreign keys with single ones
- replacing all non-numeric primary keys with surrogate ones
- replacing all non-numeric foreign keys with corresponding numeric ones
- eliminating migrating rows
- shrinking all tables
- regularly gathering statistics on indexes.

Consequently, performance of its application improved by more than 2.5 times, out of which, shrinking, the only Oracle advisors recommendation, brought only some 5%.

Conclusion and Further Work

Unfortunately, most of the existing dbs are poorly designed, implemented, fine-tuned, and used. For asserting and, especially, improving their quality, several sets of best practice rules were proposed in this field. Considering a kernel crucial subset of the ones introduced in [7,8], we have architectured, designed, developed, tested, used, and documented *ODBDetective* 1.0, a metadata mining tool for detecting violations by Oracle dbs of corresponding dbs design, implementation, and usage best practice rules. Implementation (hence documentation, testing, and usage) was done in Oracle 11 *g* and MS Access 2010, under RedHat Linux 6.2 and 64-bit Windows 7/8 (on Dell servers and notebooks).

A few of the *ODBDetective* features are also provided by Oracle tools, but only available for it's very ex pensive Enterprise Editions. *ODBDetective* is somewhat similar, for example, to *DB Optimizer* [3], but has very many powerful additional features.

Automatic violations detection can obviously be based only on syntactic criteria. In order to make correct decisions for db schemes enhancements, its users should analyze *ODBDetective*'s output and, based on semantic knowledge, decide whether or not to correct each of the syntactically discovered possible violations (e.g. analyzed current instance may be a non-typical one, empty tables may be legitimately be empty as they actually are temporary ones, some tables instances might grow much larger in certain contexts, etc.).

ODBDetective already allows for users to store some data on decisions they took after analyzing its findings, thus greatly simplifying the task to correct corresponding detected violations.

Further improvements will include:

- ✓ adding supplementary best practice rules [7,8]
- ✓ allow for more decision data to be stored
- ✓ backup and restore investigation and decision data
- ✓ automatically generate as much SQL scripts as possible for improving Oracle dbs performances based on investigation and decision data
- ✓ extensions to other RDBMSs than Oracle.

Acknowledgements

This work has been partly sponsored by Asentinel Intl. srl Bucharest, a subsidiary of Asentinel LLC, Memphis, TN, who owns the copyright of *ODBDetective*.

References

1. Immanuel C (2008) Oracle Database Performance Tuning Guide 11g Release 1 (11.1).
2. Peter B, Sergey K, Jack R (2007) DBA's New Best Friend: Advanced SQL Tuning Features of Oracle Database 11g.
3. Embarcadero Technologies Inc. (2012) DB PowerStudio for Oracle.
4. Mancas C, Dicu AI (2013) ODBDetective–a metadata mining tool for detecting violations of some Oracle database design, implementation, querying, and manipulating best practice rules, Ovidius State University, Constanta, Romania.
5. Dicu AI (2013) ODBDetective–a metadata mining tool for detecting violations of some crucial Oracle database design, implementation, usage, and optimization best practice rules, Polytechnic University, Bucharest, Romania.
6. Oracle Corp. (2010) Guide for Developing High-Performance Database Applications.
7. Mancas C (2013) Conceptual Data Modelling and Database Design: Analysis, Implementation and Optimization. A Fully Algorithm Approach, Apple Academic Press, NJ, USA.
8. Mancas C (2013) Best practice rules. Technical Report TR0-2013. Asentinel Intl srl, Bucharest, Romania.
9. Codd EF (1970) A Relational Model of Data for Large Shared Data Banks. CACM 13: 377-387.
10. Mancas C, Dragomir S (2003) An Optimal Algorithm for Structural Keys Design, Marina del Rey, CA, USA.
11. Mancas C, Crasovschi L (2003) An Optimal Algorithm for Computer-Aided Design of Key Type Constraints, Aristotle Macedonian University, Thessaloniki, Greece.

12. Mancas C (2002) On Knowledge Representation Using an Elementary Mathematical Data Model. University of the US Virgin Islands, St. Thomas, USA.
13. Mancas C (2012) Advanced Database Systems. Polytechnic University, Bucharest, Romania.
14. Goga N (2012) Software Engineering. Polytechnic University, Bucharest, Romania.
15. Serbanati DL (2012) Programming Paradigms. Polytechnic University, Bucharest, Romania.
16. Dimo P (2012) Human Computer Interaction. Polytechnic University, Bucharest, Romania.
17. Mancas C (2012) Architecture, Design, and Development of Database Software Applications. Ovidius University, Constanta, Romania.