© Springer Verlag 2004, http://www.springer.de/comp/lncs/index.html

Eder J. & Kanzian S. (2004). *Logical Design of Generalizations in Object-relational Databases*. 8th East European Conference - Advances in Databases and Information Systems (ADBIS 2004), Budapest, Hungary, LCNS3255

Logical Design of Generalizations in Object-Relational Databases

Johann Eder and Simone Kanzian

University of Klagenfurt
Institute of Informatics-Systems
eder@isys.uni-klu.ac.at skanzian@edu.uni-klu.ac.at

Abstract. The richer data models of object relational databases opens many more options for the logical design of a database schema increasing the complexity of logical database design enormously. Focusing on generalization constructs of conceptual models we explore the performance implications of the various design alternatives for mapping generalizations into the schema of an object-relational database system.

1 Introduction

Database Design is the process to transform the results of the requirements analysis into a database schema of a specific database management system (DBMS). The process starts with creating a conceptual model based on the results of the requirements analysis. The conceptual data model presents the structure of data, constraints and relationships in a high-level graphical notation, independent of the physical storage structures and independent of the particularities of the target DBMS. Popular notations for a conceptual model are the Entity Relationship model [Che76]) and the Unified Modelling Language [Obj02].

The next step is to choose a specific DBMS and derive a logical schema from the conceptual model using a well-defined set of various transformation rules. Entities are mapped to relations, primary and foreign keys are identified and relationships between data are dissolved. When making decisions between different mapping possibilities the designer must take into account the characteristics of the DBMS as well as prospective data volumes and transaction profiles. For relational DBMS this process is well understood and topic of several textbooks. Well established design procedures guide the designer to a good logical schema. For object-relational DBMS such design procedures are not yet proposed. This paper

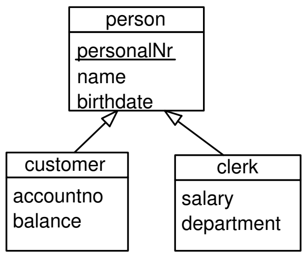


Figure 1. UML example model

is a step towards such design recommendations. Object-relational databases offer a much more complex data model including generalization/inheritance, user defined types, complex types, etc. The decision space for the designer increased tremendously because of the increased complexity of the data models. The implications of design choices on the performance of the target information systems are great but not yet well understood.

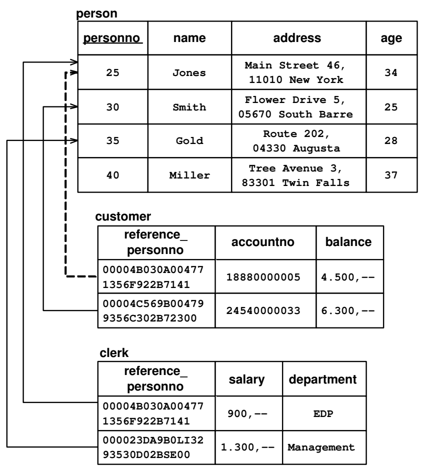
In this paper we aim at contributing to the development of a design process supporting the designer in making the decisions for mapping a conceptual schema into a logical schema of an object-relational DBMS. We focus on a specific aspect: how to map generalizations of UML diagrams to the object-relational schema. We describe the various alternatives and investigate the performance implications of the different choices through a series of benchmarks.

2 Mapping alternatives for generalization

In this section we discuss the various ways for mapping a generalization to an object-relational data model. Our running example is given in figure 1. The figure shows a sample for specialization/generalization based on the personnel structure of a bank. The superclass is a general person, with a personal number, a name and a birthdate. We have two specializations: a customer with the attributes *account* and *balance* and a clerk with the attributes *salary* and *department*.

There are seven different possibilities for mapping a generalization to relations. Most of them are similar to relational solutions except those using inheritance offered by the DBMS. A generalization can be disjoint or overlapping and total or partial. We will show 4 samples, one for each kind of generalization.

Alternative I - Subclasses referencing Superclass: A complex datatype is defined for the superclass consisting of the superclasses' attributes and a typed



person

<u>personno</u>	name	address	age
25	Jones	Main Street 46, 11010 New York	34
30	Smith	Flower Drive 5, 05670 South Barre	25
35	Gold	Route 202, 04330 Augusta	28
40	Miller	Tree Avenue 3, 83301 Twin Falls	37

customer

reference_ personno	accountno	balance
00004B030A00477 1356F922B7141	18880000005	4.500,--
00004C569B00479 9356C302B72300	24540000033	6.300,--

clerk

reference_ personno	salary	department
00004B030A00477 1356F922B7141	900,--	EDP
000023DA9B0LI32 93530D02BSE00	1.300,--	Management

Figure 2. Mapping Generalization using Alternative I

table is created based on the datatype. The key attribute of the superclass is the primary key. More datatypes and typed tables are defined likewise for each subclass. Each subclass-datatype contains an additional attribute for referencing the superclass. Alternative I is shown in figure 2.

Alternative II - Subclasses containing complex datatype of Superclass: A complex datatype is defined for the superclass' attributes. A table is created for each subclass containing the specialization attributes of the subclasses and one attribute defined on the superclass datatype (figure 3). Alternative II can't be used for partial generalizations. Entries of the superclass without corresponding entries in a subclass are lost, because there is no table where they fit in. With an overlapping generalization the attributes of the superclass are stored redundantly, complicating maintaining data consistency. Therefore, alternative II should not be used if the superclass has many and/or volatile attributes.

Alternative III - Single Relationship containing complex datatypes of Super-/Subclass with distinction attribute: A complex datatype is defined for the superclass' attributes as well as for the attributes of each subclass. One table is created whose attributes are defined on the complex datatypes. One additional attribute is needed in order to assign a data record to one of the subclasses. The result is shown in figure 4. Alternative III can only be used for disjoint generalization because the distinction attribute is single value. When many subclasses with many attributes are given, there will be many empty attributes in the table, because a data record can only be assigned to one subclass.

customer

person				accountno	balance
personal no	name	address	age		
25	Jones	Main Street 46, 11010 New York	34	18880000005	4.500,--
30	Smith	Flower Drive 5, 05670 South Barre	25	24540000033	6.300,--

clerk

person				salary	department
personal no	name	address	age		
25	Jones	Main Street 46, 11010 New York	34	900,--	EDP
35	Gold	Route 202, 04330 Augusta	28	1.300,--	Management

Figure 3. Mapping Generalization using Alternative II

person				customer		clerk		subclass_ type
personal no	name	address	age	accountno	balance	salary	department	
30	Smith	Flower Drive 5, 05670 South Barre	25	24540000033	6.300,--			CU
35	Gold	Route 202, 04330 Augusta	28			1.300,--	Management	CL
40	Miller	Tree Avenue 3, 83301 Twin Falls	37					NULL

Figure 4. Mapping Generalization using Alternative III

Both partial and total generalization are supported. When a data record is assigned to the superclass only, the distinction attribute is empty.

Alternative IV - Single Relationship containing complex datatypes of Super-/Subclass with boolean distinction attributes: This alternative is similar to alternative III. The difference is the definition of the distinction attribute: a complex datatype is defined with one boolean attribute representing each subclass. This datatype is assigned to the distinction attribute. The value "TRUE" must be given to every boolean attribute which represents a subclass the data record belongs to. This principle is shown in figure 5. It is possible to resolve overlapping generalizations in alternative IV, using all of the attributes of the complex distinction column. Partial generalization is valid too: the value "FALSE" is assigned to the boolean attributes for every subclass. Alternative IV is best qualified for total, overlapping generalizations. Partial and disjoint generalization is possible, but results in a table containing many NULL-Values.
Alternative V - Single Relationship containing complex datatypes of Super-/Subclass with multivalue distinction attribute: The relation of alternative V looks very much like alternative IV, except for the definition of the

person				customer		clerk		subclass_ type	
person no	name	address	age	accountno	balance	salary	department	CU	CL
25	Jones	Main Street 46, 11010 New York	34	18880000005	4.500,--	900,--	EDP	1	1
30	Smith	Flower Drive 5, 05670 South Barre	25	24540000033	6.300,--			1	0
35	Gold	Route 202, 04330 Augusta	28			1.300,--	Management	0	1
40	Miller	Tree Avenue 3, 83301 Twin Falls	37					0	0

Figure 5. Mapping Generalization using Alternative IV

distinction attribute. Alternative V has a single multivalue distinction attribute. One value is assigned to the distinction attribute for each subclass the record belongs to. The resulting table is shown in figure 6. When selecting data all values of the distinction attribute must be checked to find out if the data record belongs to a specific subclass. Partial, total, overlapping and disjoint generalization is possible, but partial and overlapping generalization should be avoided due to the great number of NULL-values.

Alternative VI - Inheritance offered by DBMS: Alternative VI uses the inheritance enhancements offered by the DBMS. A complex datatype is defined for the superclass. For every subclass a subordinate complex datatype to the superclass is defined. Every attribute, method and function is inherited from the supertype to the subtype. Typed tables are defined on the various complex types. The resulting hierarchy is shown in figure 7. The resulting tables correspond to the relations shown in figure 3, with one additional typed table, defined on the complex type of the superclass. The difference between alternative II and alternative VI is the point in time at which inheritance takes place. With alternative II, the attributes of the superclass are assigned to the subclass table when the column based on the complex datatype of the superclass is created. In alternative VI the superclass attributes are already inherited by the DBMS at the definition of the subordinate complex datatype, before creating any table. Total, partial, overlapping and disjoint generalizations are supported. A partial generalization has no entries in the typed tables of the subclass, as well as a total generalization has no entries in the superclass table.

Alternative VII - Inheritance offered by DBMS with not-instantiable root type: Alternative VII is the same as alternative VI, but the complex type of the superclass is created as "not instantiable" meaning that no typed table can be defined on it. The subordinate datatypes still inherit all attributes and

person				customer		clerk		subclass_set
personno	name	address	age	accountno	balance	salary	department	
25	Jones	Main Street 46, 11010 New York	34	18880000005	4.500,--	900,--	EDP	CU, CL
30	Smith	Flower Drive 5, 05670 South Barre	25	24540000033	6.300,--			CU
35	Gold	Route 202, 04330 Augusta	28			1.300,--	Management	CL
40	Miller	Tree Avenue 3, 83301 Twin Falls	37					NULL

Figure 6. Mapping Generalization using Alternative V

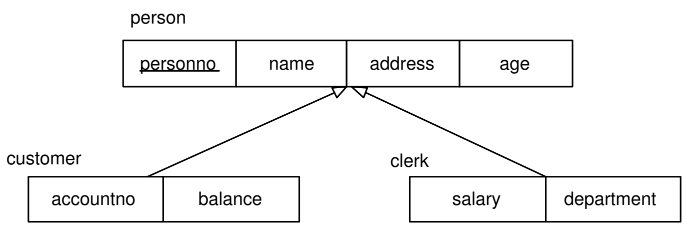


Figure 7. Mapping Generalization using Alternative VI

methods from the not-instantiable supertype. The resulting tables are the same as in alternative VI, without the typed table for the superclass. Partial generalization is not supported, because a data record which doesn't belong to a subclass doesn't fit into one of the defined tables.
These seven alternatives can be used on every object-relational DBMS provided that it offers complex datatypes, multivalued attributes and structural inheritance. In a relational system there are only four alternatives for mapping generalization. The designer has to decide which alternative to choose. The choice is somewhat limited by the above mentioned characteristics of the generalization.

If there are still alternatives left after classifying the generalization, the transaction profile and datavolume tables are consulted. Unfortunately, this often doesn't narrow the selection of alternatives, because it's difficult to rate a logical access to a table with object-relational enhancements. The actual physical structure is different for every DBMS and it cannot be assumed that logical writes and reads are a suitable approximate value for their object-relational physical counterparts. A performance test can help to make the final decision. The next section shows the results of a performance test, made for the different alternatives on the DBMS Oracle 9.2.0.
Oracle offers one additional alternative (alternative VIII). Like alternative VI a type hierarchy of complex datatypes is build. But only one typed table, de-

fined on the superclass type, is neccessary. Data records belonging to a subclass type can be inserted into this table using the corresponding type constructor. Selections must state the datatype of the result set. The DBMS offers the two functions "VALUE" and "TREAT" which are used for this purpose.

3 Benchmarks

The eight possibilities offered by Oracle are now investigated with regard to their costs and relative runtimes. The experiments were made with the following conditions: we were only interested in the relative performance of the different variants, so we do not show absolute values. We did not tune the system for improving the different variants but used the general settings, and we did not elaborate on the physical level. To decide which of the several alternatives is suitable for a specific problem we made a small performance test. The system was a PC with an Athlon XP 1800+ processor, 256 MB RAM with Windows 2000. The DBMS was the Oracle 9.2.0 standard installation without any physical tuning measures. From a fictive transaction profile we chose six transactions, which we run against the database:

- T1: selects data from the superclass and exactly one subclass, e.g. name, account number and balance of a customer
- T2: selects data from the superclass and both subclasses (only possible for overlapping generalization), e.g. name, balance, salary and department of clerks, that also have an account
- T3: selects data from the superclass only, e.g. customers older than 30 years
- T4: selects data with an aggregate function, e.g. average balance
- T5: adds a new customer (who is no clerk)
- T6: modifies an existing customer (who is no clerk)

We compared costs and relative runtimes of the reading transactions T1 to T4 and the writing transactions T5 and T6 separately. The tests were made with different datavolumes, beginning with 40.000 persons. 10.000 are customers, 10.000 are clerks, 10.000 are both customers and clerks and 10.000 are neither customer nor clerk. This scheme was repeated four times, adding 40.000 persons each time until we reached 160.000 persons.

alternative I, IV, V and VI	exactly one subclass connected with the superclass		several subclasses connected with the superclass		superclass alone		query with aggregate function	
	few datarecords	many datarecords	few datarecords	many datarecords	few datarecords	many datarecords	few datarecords	many datarecords
minimal cost	VI	VI	IV, VI	IV, VI	I, IV	I, IV	I	I
minimal runtime	I	VI	I	I	I	I	I	I

Figure 8. Results of alternatives I, IV, V and VI with regard to minimal cost and runtime

3.1 Alternative I, IV, V and VI

The first comparison was made for overlapping, disjoint, partial and total generalization, which leaves the alternatives I, IV, V and VI to investigate. For every transaction we measured the costs of the access plan and the runtime in seconds. Alternative I was realised in Oracle with object tables and references. Alternative IV used a single relation with a complex datatype as distinction attribute. Alternative V was implemented with a VARRAY as distinction attribute. Alternative VI uses the structual inheritance offered by Oracle. The queries T1, T2, T3 and T4 where run against the different relations. An overview of the results, which can be used as a help for choosing the best alternative, is shown in figure 8. For details about cost and runtime, check figure 11 in the appendix.

T1 - Superclass and exactly one subclass Transaction T1 was the first to be measured. Whichever metric is more important, minimal cost or minimal runtime, another alternative has better values. For minimal costs, alternative VI has to be chosen, regardless of the amount of data. With a large quantity of data records it also has the best runtimes. However, alternative I has significant better runtimes for small data sets, but it also requires the highest cost. Alternative IV is the middle for both cost and runtime. The costs of alternative V where extremly high due to accessing the VARRAY and it also had the worst runtimes.

T2 - Superclass and serveral subclasses The same applied to transaction T2. The costs of alternative V were about 300 times higher than the costs of the other alternatives, the runtimes where average. Alternative IV and VI needed the same costs. Alternative I started with low costs for few data records but the costs exploded with the number of records. Therefore alternative IV or VI should be chosen for minimal costs. For minimal runtime, alternative I is best

alternative II and VII	exactly one subclass connected with the superclass		several subclasses connected with the superclass		superclass alone		query with aggregate function	
	few datarecords	many datarecords	few datarecords	many datarecords	few datarecords	many datarecords	few datarecords	many datarecords
minimal cost	II	II	II	II	II	II	II	II
minimal runtime	II, VII	II, VII	II, VII	VII	VII	VII	II, VII	II, VII

Figure 9. Results of alternatives II and VII with regard to minimal cost and runtime

suited, especially when dealing with small amounts of data, needing about half the time of the other alternatives.

T3 - Superclass only For transaction T3 the decision is a hard one when considering minimal costs. The cost of alternative I and IV are nearly the same whereas alternative VI needs significant higher costs especially for large data sets. Suprisingly the costs of alternative V are average for T3 and they stay constant regardless of the amount of data. Considering runtimes, alternative I and VI need about half the time of alternative IV and V for small data sets. For large relations alternative I has the best runtimes combined with very low costs.
T4 - Transaction with aggregate function Transactions with an aggregate function favour alternative I, with both minimal costs and runtimes. The costs for alternative V remain constant regardless of the amount of data, but are very high from the beginning. Alternative IV and VI need average costs. Runtimes of alternative V are slightly worse compared with alternative I. The runtimes of alternative IV and VI are much worse and they are highly influenced by the number of data sets. The performance tests showed, that for minimal runtimes alternative I is mostly the right choice, except when using transactions, which select from the superclass combined with exactly one subclass, on large data sets. In this case alternative IV should be used. If it's most important to minimize costs, alternative IV should be favoured for different transaction categories.

3.2 Alternative II and VII

If partial generalization is prohibited the choice should be made with alternative II and alternative VII. Alternatives I, IV, V and VI can also be used. In this case the application is responsible to prohibit partial generalization. The queries of T1, T2, T3 and T4 were run against the alternatives. Detailed information regarding cost and relative runtimes are shown in figure 12 in the appendix.

T1 - Superclass and exactly one subclass For T1, accessing the superclass and exactly one subclass, the alternatives II, VI and VII nearly have the same costs, with a linear cost trend. It begins with low costs for a small amount of data and rises continually with the number of records. For minimal runtime alternative I has the best value for small datasets. For large datasets the runtime of alternative II, IV, VI and VII is the same.

T2 - Superclass and serveral subclasses Queries like T2 have higher costs for all alternatives, due to the additional joins that must be made. Alternative I has the lowest costs for small datasets, but the highest one for large datasets. Alternative V has immense costs and is therefore not recommended. The remaining alternatives basically all need the same cost with minimal difference. Considering minimal runtimes, alternative I is the best in all cases.

T3 - Superclass only For transactions, that access the superclass only, alternatives I and IV provide minimal costs regardless of the amount of data. For minimal runtimes alternative VII should be chosen for small datasets, for large datasets alternative I has best runtimes.

T4 - Transaction with aggregate function The results are somewhat different for transactions using an aggregate function (T4). The costs rise linear depending on the amount of data for all alternatives, but alternative I needs the lowest costs of all. Regarding minimal runtimes alternative I or II should be chosen for small datasets, for large datasets alternative I or V is recommended, but it should be considered that alternative V needs extremely high costs.

The analysis shows a similar cost and runtime behaviour for alternatives II and VII, which is not very suprising. The structure of the tables of both alternatives is identical, but the tables are constructed using different object-relational facilities, which explains the similar cost and runtimes. In many cases the alternatives II and VII only have average results. For minimal costs the choices vary depending on the kind of transaction, for minimal runtimes alternative I is always recommended, except for transaction T1 with large datasets.

3.3 Alternative III and VIII

Alternative III and VIII do not support overlapping generalizations. Therefore only transactions that access the superclass and exactly one subclass or the superclass alone and transactions with an aggregrate function where investigated. Alternative III is a single relation containing the super- and subclass attributes

alternative III and VIII	exactly one subclass connected with the superclass		superclass alone		query with aggregate function	
	few datarecords	many datarecords	few datarecords	many datarecords	few datarecords	many datarecords
minimal cost	III	III	III	III	III	III
minimal runtime	III	III	VIII	VIII	III, VIII	III, VIII

Figure 10. Results of alternatives III and VIII with regard to minimal cost and runtime

as complex datatype. Alternative VIII makes use of the possiblity offered by Oracle to insert records of a subtype into a table defined on the supertype. The alternatives I, IV, V and VI can also be used, in this case the application is responsible for forbidding overlapping generalization. An overview of the benchmark results is shown in figure 10. For detailed graphical representation of the cost behaviour and for a table with relative runtimes, refer to figure 13.

T1 - Superclass and exactly one subclass For queries accessing the superclass and exactly one subclass the cost trend is linearly rising with the amount of data for all alternatives. For small datasets the costs are nearly the same for all alternatives, for large amounts of data alternative VI should be chosen. Alternative III clearly has the best runtimes regardless to the amount of data, followed by alternative VIII. It should be taken into account that queries for alternative VIII are complex to write and difficult to read, due to the use of the functions *VALUE* and *TREAT*.

T3 - Superclass only Transactions that select data only from the superclass also have a linear cost trend depending on the amount of data. Alternatives III and VIII provide minimal costs. For small datasets alternative VIII also has the best runtimes, whereas for large amounts of data alternative I minimizes runtimes.

T4 - Transaction with aggregate function For transactions with aggregate functions alternative I should be clearly favoured with regard to minimal costs. For small datasets alternative I and III have minimal runtimes and for large datasets alternatives I and V.

T5, T6 - Inserting and Updating If data creation and modification are the most frequent transactions, their cost and runtimes should be considered for deciding between various alternatives. The runtimes are so insignificant, that they are not further considered.

If the data is frequently modified alternative I should never be chosen. The costs are much higher compared to the other alternatives. The same is valid if a data record is created, only alternative V has similar high costs, due to accessing the VARRAY. The bad results of alternative I are explained by the way a data record is created. First, an entry is created in the superclass table. After that the OID of the new record must be selected and transformed into a reference, before the corresponding subclass-entry can be created. When updating a record, the reference between subclass and superclass has to be resolved in order to find the correct subclass-entry, so both inserting and modifying are timeconsuming and expensive operations when using alternative I. Alternative I had good results for querying data, but if data is frequently changed, alternative VI should be chosen for minimal costs.

Alternative II or VII are valid, if partial generalization is not allowed, but the alternatives I, IV, V and VI can also be used. Alternatives II, VI and VII have the best costs for inserting data. For Updating the costs for all alternatives are minimal, except for alternative I which has been explained before.

For disjoint generalization, alternative III and VIII are suitable, but the alternatives I, IV, V and VI can also be used. Just like before the costs for updating data are the same for all alternatives, except for alternative I. For inserting alternative VI is best suited, followed by alternative III and VIII.

4 Conclusions

We reported on an experimental analysis of design choices for mapping generalization to the logical schema of an object-relational database with the aim to assist database designers in achieving logical data models with good performance characteristics.

We provided a set of rough recommendations for a specific database management system but in particular we showed that the differences between logical designs are surprisingly high and that the transaction and data volume profile is very important for designing an object-relational database. Different transaction categories require different alternatives for optimal results with regard to costs and runtime. Specific benchmarks are often necessary to avoid poor performance. There will not be an easy set of rules for logical design as for relational models.

From these experiments and another set of experiments on a different database management system we can conclude the following more general results:

– The performance differences of the different choices are great - sometimes in orders of magnitude.
– The performance champions change considerably with the size of relations.
– There are great differences in the performance of the same mapping alternatives in different DBMS.
– Mapping alternatives perform quite differently for different types of queries.

Logical design for object-relational databases is much more complex than for relational databases. Differences between DBMS and the actual transaction profile and data volumes have a greater effect on performance. Detailed performance analysis will be needed for logical design to reach high performance.

References

[CD96] M. J. Carey and D. J. DeWitt. Of Objects and Databases: A Decade of Turmoil. In *Proceedings of the 22nd VLDB Conference*, 1996.

[Che76] P. Chen. The Entity-Relationship Model - Toward a Unified View of Data. In *ACM Transactions on Database Systems*, June 1976.

[DD95] H. Darwen and C.J. Date. The third manifesto. *ACM SIGMOD Record*, 24(1):39–49, March 1995.

[Dev01] R. S. Devarakonda. Object-Relational Database Systems - The Road ahead. *Crossroads*, 7(3):15–18, 2001.

[Mir97] S. Miranda. Object Relational Datamodels of the Future. In *3rd Basque International Workshop on Information Technology (BIWIT '97)*, 1997.

[MP01] W. Y. Mok and D. P.Paper. On Transformations from UML Models to Object-Relational Databases. In *Proc. of the 34th HICSS*, 2001.

[MS00] W. Mahnke and H.-P. Steiert. Zum Einsatzpotential von ORDBMS in Entwurfsumgebungen. In *Tagungsband der Fachtagung CAD 2000*, 2000.

[Obj02] Object Management Group - OMG. The UML Ressource Page. 2002.

[SBM99] M. Stonebraker, P. Brown, and D. Moore. *Object-relational DBMSs - Tracking the Next Great Wave.* Morgan Kaufmann, 1999.

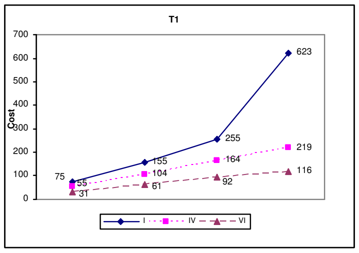
[Sch00] C. Schahczenski. Object-oriented databases in our curricula. *The Journal of Computing in Small Colleges*, 16(1), 2000.

[Sto97] M. Stonebraker. Architectural Options for Object-Relational DBMS. *Informix Software, CA*, 1997.

[ZR00] W. Zahng and N. Ritter. Measuring the Contributions of (O)RDBMS to Object-Oriented Software Development. In *2000 Int. Database Engineering and Applications Symposium (IDEAS'00)*, 2000.

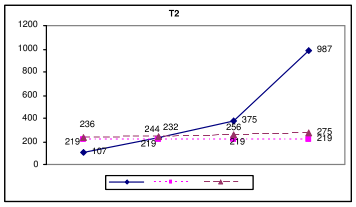
[ZR01] W. Zahng and N. Ritter. The Real Benefits of Object-Relational DB-Technology for Object-Oriented Software Development. In *Proc. 18th British National Conference on Databases (BNCOD 2001)*, 2001.

A Detailed Information about Cost and relative Runtimes



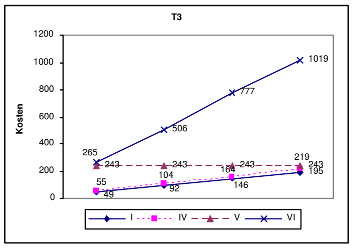
runtime

records	I	IV	V	VI
40.000	1	3,01	3,09	2,09
80.000	2,06	3,05	9,05	3,04
120.000	5,06	7,0	16,03	5,05
160.000	14,03	8,08	19,06	8,07



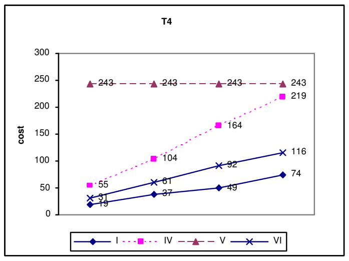
runtime

records	I	IV	V	VI
40.000	1	2,95	1,96	3,27
80.000	2,3	4,26	2,97	4,25
120.000	3,62	6,56	5,90	5,56
160.000	6,24	7,52	7,54	7,22



runtime

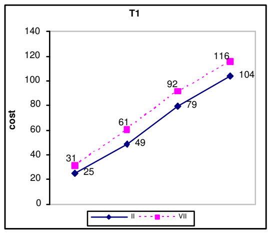
records	I	IV	V	VI
40.000	1	2,05	2,10	1,05
80.000	2,67	5,38	4,14	2,76
120.000	3,10	6,14	6,19	2,48
160.000	4,19	10,05	8,29	16,62



runtime

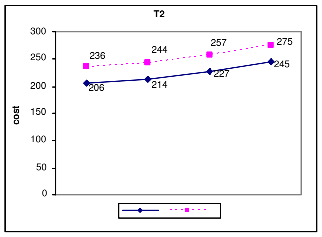
records	I	IV	V	VI
40.000	0	0,67	2,67	0,67
80.000	0	1,33	1,67	1,33
120.000	1	201	1,33	100,3
160.000	1,67	168,3	1,67	102,7

Figure 11. Cost and relative runtime for I, IV, V and VI



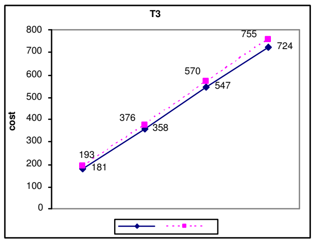
runtime

records	II	VII
40.000	1	0,68
80.000	1	1
120.000	1,66	1,66
160.000	2,64	2,65



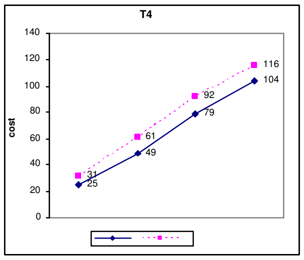
runtime

records	II	VII
40.000	1	1,01
80.000	1,3	1,31
120.000	2,01	1,7
160.000	2,6	2,21



runtime

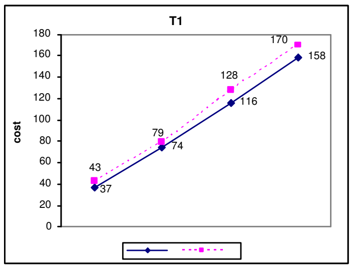
records	II	VII
40.000	1	0,53
80.000	2,67	1,33
120.000	3,28	1,81
160.000	5,27	5,32



runtime

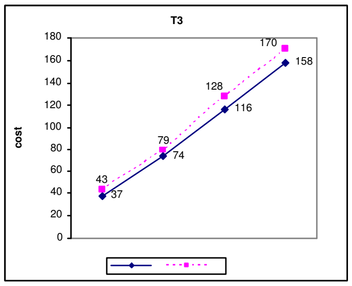
records	II	VII
40.000	0	1
80.000	1	1,5
120.000	100	104
160.000	103	102,5

Figure 12. Cost and relative runtime for II and VII



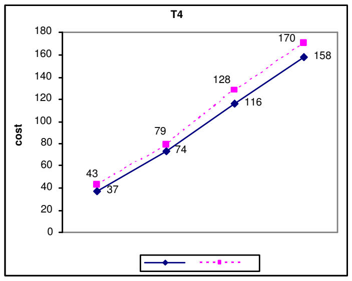
runtime

records	III	VIII
40.000	1	50,5
80.000	2,5	52
120.000	200,5	203,5
160.000	300,5	401



runtime

records	III	VIII
40.000	1	50,5
80.000	2,5	52
120.000	200,5	203,5
160.000	300,5	401



runtime

records	III	VIII
40.000	1	2
80.000	3	3
120.000	403	403
160.000	505	502

Figure 13. Cost and relative runtime for III and VIII