Lusíada
Repositório das Universidades Lusíada

Universidades Lusíada

Yang, Hongji
Pinto, Paulo Jorge Gonçalves, 1956-

Foreign keys and multi-domain indexing

http://hdl.handle.net/11067/5219

Metadados

Data de Publicação 2010

Resumo Este paper mostra que todas as chaves estrangeiras podem ser transformadas em índices com o benefício de melhorar o acesso físico aos dados. Este paper mostra ainda que esta técnica é consistente com as actuais técnicas de modelação de dados, pelo que não são necessárias alterações a essas mesmas técnicas . Mostra-se ainda que a estrutura de índices, com os índices definidos como funções, podem dar suporte para o papel das relações. Ainda darão suporte para relações que envolvam mais de duas tab...

This paper shows that all foreign keys in a database can be transformed in indexes with the benefit of speeding data access. This paper also shows that this technique is consistent with actual modelling techniques, so no further changes must be done to those techniques. It will also show how the index structure, with indexes defined as functions, can provide support for relationship roles. In addition, they will provide support for more than two tables in one relationship and for supporting spec...

Palavras Chave Estruturas de dados (Informática), Bases de dados

Tipo article

Revisão de Pares Não

Coleções [ULL-FCEE] LEE, n. 10 (2010)

Esta página foi gerada automaticamente em 2024-04-23T16:34:57Z com informação proveniente do Repositório

FOREIGN KEYS AND MULTI-DOMAIN INDEXING

Paulo Jorge Gonçalves Pinto
Doutorando em Base de Dados (DeMontfort University)
Docente da Universidade Lusíada

Hongji Yang
Prof. Hongji Yang, Ph.D. (Durham), MIEEE
Professor em DeMontFort University

Resumo: Este paper mostra que todas as chaves estrangeiras podem ser transformadas em índices com o benefício de melhorar o acesso físico aos dados. Este paper mostra ainda que esta técnica é consistente com as actuais técnicas de modelação de dados, pelo que não são necessárias alterações a essas mesmas técnicas [6].

Mostra-se ainda que a estrutura de índices, com os índices definidos como funções, podem dar suporte para o papel das relações. Ainda darão suporte para relações que envolvam mais de duas tabelas, bem como definição de ordenações especiais a definir pelo utilizador.

Finalmente concluímos afirmando que, com esta nova técnica, os motores de bases de dados comerciais não perdem performance porque todas as estruturas de suporte já se encontram presentes e, inclusive, nalguns casos, poderá haver melhoria de performance.

Abstract: This paper shows that all foreign keys in a database can be transformed in indexes with the benefit of speeding data access. This paper also shows that this technique is consistent with actual modelling techniques, so no further changes must be done to those techniques [6].

It will also show how the index structure, with indexes defined as functions, can provide support for relationship roles. In addition, they will provide support for more than two tables in one relationship and for supporting special sorting order that might be needed.

Finally, we conclude stating that, with this new technique, commercial database engines should not degrade performance because all supporting structures are already there and, in some cases, a better performance could be achieved.

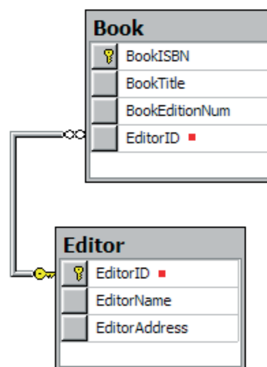
1. Foreign Keys

The Entity-Relation model, as presented by Peter Chen in the early 70's[2], stood as an independent model to represent conceptual entities and the relations amongst them. It is independent because, regardless of the technology employed, it would always be applicable.

In that model, when a relationship was defined, a role for that relation (associated with its counterpart foreign key) could always be specified. The point is that role was never properly defined outside this modeling technique.

To implement the E-R model for an information system, or more precisely, to implement relationships between entities we use foreign keys.

The use of foreign keys, in database design, is widely spread as a good practice for implementing relations amongst tables [4,3,6]. However, a foreign key is what is says: the primary key (or even a candidate one) of a table placed as an attribute on another table to enforce a relationship between those two.

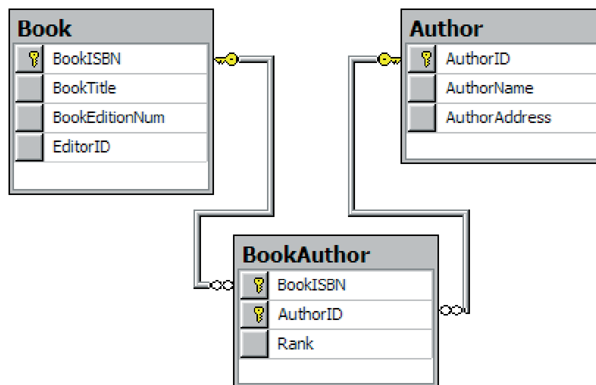


Picture 1: A Relationship enforced with a foreign key

Nevertheless, we should only place a foreign key in a table if we have a relation in which each tuple of the target table matches only one of the referred tables.

As we know when we have multiple associations between tuples (for

instance in a many to may relationship such as authors and books, in which an author can write many books and a book can be written by several authors) we have to adopt another strategy by building a new table with both keys. In this new table, we have our relation "dictionary" [3], because we have the references to the tuples in the original tables that should match.



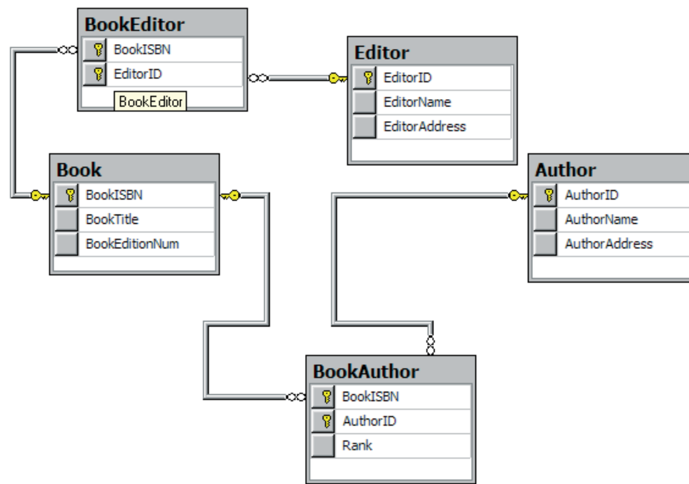
Picture 2: A Relationship enforced with the aid of an additional table

In addition, we can build these "dictionaries" for any kind of relationship. This includes those that we use to create foreign keys directly (such as the costumer id on an invoice).

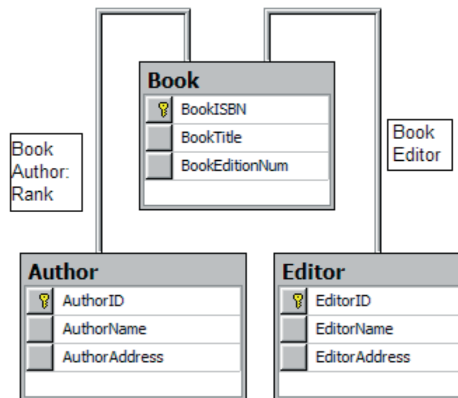
We can think we can loose some performance doing more tables than necessary since we are now using a table for the relation, but what can we surely gain?

The relation table is built with rules, and we can have in a more clear way what rules connect to instances of data together. This would show in a more meaningful way in which data relate amongst them.

We also would eradicate foreign attributes from target tables (no longer a Editor ID in an Book record), so tables could mirror their conceptual counterpart.



Picture 3: Full diagram with no foreign keys in the main tables



Picture 4: Diagram with no connection tables

Sometimes relationships have attributes (for instance the attribute “rank” which is the relative position of an author within the group of authors that wrote the book: 1st, 2nd, and so forth).

The attribute should be declared when the relationship is build and should have the very same rules as an attribute in a base table. It will be provided when a particular instance of a relationship is made, through the SQL keyword “SET” in the context of that relation[1].

As noted, the relation must have a name and a definition, and, together with it, a set of relationship attributes.

Perhaps the reader is thinking that this can reduce overall performance. We will see that that is not quite true, as it might seem.

These pairs of keys are only pointers to data, so this new structure is no more that a multi-domain index. It points to two pieces of data and can be effectively built as an index. We already have indexes for foreign keys in order to "speed up" the verification of referential integrity, so no extra overhead is required. And if it is built by rules we could rebuilt them by applying the very same set of rules we had. In this manner, we could effectively implement the idea of a role in this relationship.

Obviously that some kind of data recording should be done, but this would only be done at the database engine level, not in the conceptual level.

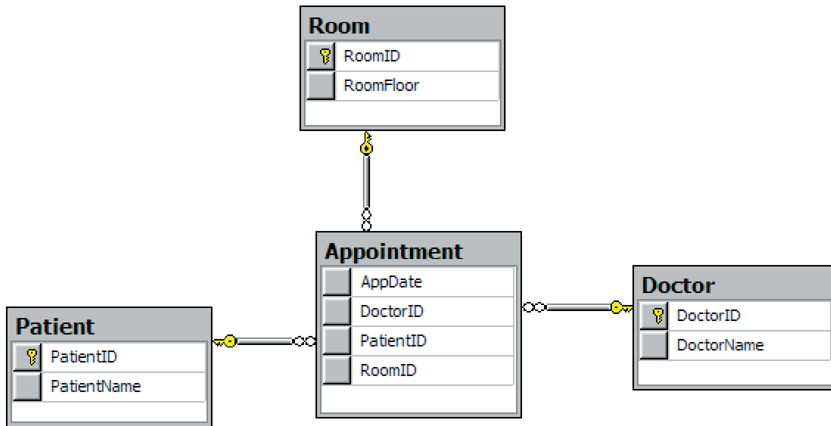
We would have to state our rules in a more precise way, we could have these rules building the relations between data and we can free ourselves out of the foreign keys.

In the present database engines implementations, if we have a one to many relationship and want to change it to a many to many relationship, we have to build the new table, copy data to it, and change every view or stored procedure that accesses that data to accommodate the new table and provide the very some results as before.

If we had a model where all relationships between tables were build with multi-domain indexes, we would only have to change the rules how data can be paired and nothing else.

Besides, there is nothing in this model that prevents the accommodation of three, four or more keys in a relation providing true associations between more than two tables (as opposite to modern relational database engines that allows us only to define a single relationship with just two tables).

With this solution, the classical clinic problem to associate patients to appointments and attending doctors could be eased. This is because we could associate all three keys (DoctorID, PatientID and AppointmentHour) and establish as a rule that we could not have duplicates in DoctorID & AppointmentHour and in PatientID & AppointmentHour. With no further restrictions, the model can validate all the main issues in this situation: Not to appoint more than one patient for hour for the same doctor, not to appoint more than one doctor for hour for the same patient. Notice that although we are dealing with the same relation, the pair DoctorID and PatientID can have duplicates.

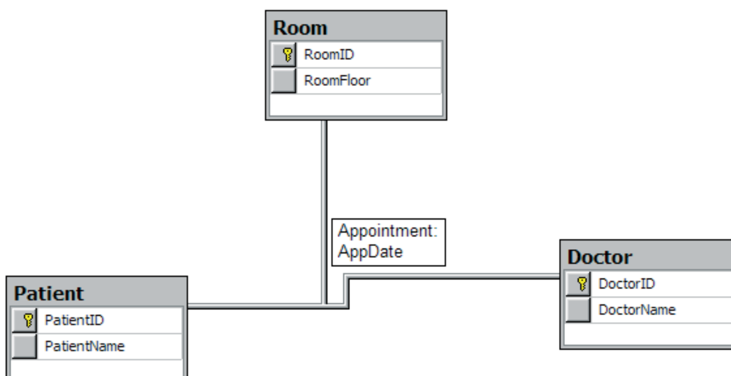


Picture 5: The Patient/Doctor/Room/Appointment problem

To create such relations we should provide an SQL statement like CREATE RELATION ON Doctor,Patient,Room WITH (AppDate Datetime NOT NULL) CONSTRAINT UNIQUE DoctorID,AppDate, UNIQUE PatientID,AppDate

This statement should create an internal table with the attributes DoctorID, PatientID, RoomID and an extra AppDate. It would also create unique indexes for the pairs DoctorID,AppDate and PatientID,AppDate. This structure would implement the conceptual relation among these entities.

With a multi domain index, all the rules should be on that relationship and it would look like this:



Picture 6: Doctor/Patient/Room with an Appointment Relation

To relate/un-relate data, we should use the plain INSERT / UPDATE / DELETE SQL statement applied to the relation. Examples of SQL to manage relations follow:

INSERT Appointment (PatientID, DoctorID, RoomID, AppDate) VALUES (100,23,4,#5-May-2008#)

UPDATE Appointment SET AppDate = #11-May-2009# WHERE DoctorID = 21 AND AppDate = #5-May-2008#

DELETE Appointement WHERE DoctorID = 21 AND AppDate = #10-May-2008# AND PatientID = 95

As you see, there is no need to further keywords in SQL. They are all appied n the context of one relationship (Appointment). This is because internally the Appointment relation should be build as a table (as all the indexes are), so we can manipulate it from outside, as it was a regular table.

The multi domain indexes can also be used to implement some hierarchy among data, because an index (whether it be a single or multi-domain index) will eventually order data in some way, adding a little more meaning to the relation it applies. If we have, for instance, an employee-manager relationship, besides its semantic we can add a job hierarchy to this just by ordering it properly. And this can be achieved because this structure is an index structure and we can have this additional meaning added just as simply defining how that index should order its data.

It seems clear that the information about these relationships no longer resides on the tables, but instead, on the index structure of the database. This also means that in the backup strategy of the table these indexes must also be kept in order to reconstruct all the data.

On the other hand, building these structures as indexes can lead to have them permanently in memory (as databases engines already do that for indexes) [3] and reduce in a significant way accesses to related data because their physical pointers are already in memory, hence improving global database performance. Access to raw data can then be performed by one of the traditional techniques like (hash tables, clustered keys, etc.) as they are right now.

This approach is consistent with Peter Chen's definition of primary keys as functions that would return the all set of data (row) for each one given [2]. This is not only true for primary keys, but for all indexes in general. Even when we have duplicate indexes, we can add to the index the primary key (or one of the candidates) and make it unique, even if it is only in an internal database engine procedure.

Since an index is meant to sort data, if we add a function that can associate a unique order to a unique key (and that does seems achievable), we can add some more meaning to the relation between two (or more) data tables.

The meaning resides on the utility of such sort order. We can then define not only what the participants in a relation are, but also how they related among themselves and what its role in that relation is. The index can be built as a linked

list so that a particular sort order could be followed if necessary.

All indexes on a database can be built with this logic, because all the "regular" indexes fall into this document definition of indexes.

The use of indexes in a general way of speaking is a good practice for ordering and finding data amongst large databanks. If we cannot sort data, it might become very difficult or time consuming to find it. This is particularly relevant if we are dealing with foreign keys because we must see if we are violating or not any integrity rule and find efficiently the data in the referred table. This would be very hard to achieve if there were no indexes on a database.

We still have a step to take. How can this be implemented in a database engine? When we index we declare a certain number of fields to index and if they are ordered ascending or descending. This is called the "natural" or "system" ordering (according to Wilfred Ng[5]), since it only takes the binary value of data being sorted. And because we need more ways to sort data than this simple method, database engine builders created what they call "database sort-order" or "database code-page" which are no more than rules to sort out national characters in a way that matches the culture in which they are used.

For instance, the words "Abelardo", "Álvaro" and "Berardo" should be sorted in this very order although the symbol "Á" has a higher binary code than the "B" symbol. This is achieved by database engines by defining rules how to sort this kind of symbols.

The answer for our model resides on building functions (such as database sort order) that we can use for building indexes and sorting data.

The database engine should accommodate an area where sort orders could be defined and maintained so indexes could be created accordingly to the rules defined on such functions.

This, of course, would have to bring changes how the database language (SQL, primarily)[7] accesses data, so an adjustment to the language should be made.

We should be able to define any sort order, and access data in that order. We would have to define a new function that allows us to access any data in a particular order (not just the first or last row but also next and previous rows).

Now with an index formally defined as a function that returns data in a particular order, we can add more functions to datasets.

We can now add the concept of first, last, previous and next and access data from the first row, last row, next row and previous row, without have to define a procedural access to that.

Some database engines do implement a function called row_number based in an SQL ORDER BY clause, which is no more than an implementation of a singular function that ranks results based on a specified ORDER BY clause.

From the beginning of databases, indexes were considered an implementation option but not a conceptual issue. Over the times, the use has proved that indexes are one of the data access foundations. Some researchers began the study of

Ordered Relations [5] but what was defined was the semantic of those domains. On the other hand relations has been defined as a foundation to the relational model [1], but the relations have been implemented by foreign keys which are no more than pointer to data, meaning that they are no more than indexes.

Putting together these two realities together, we have come out with an index structure that must provide support for relationships amongst tables, provide the notion of role within a relation and behaves like a function to return the raw data involved, given a key (whether is one table, a pair related or a multi-related junction). Additionally, if these indexes are made out of rules, we can also have a semantic meaning for each relation is a database.

2. Conclusions

In conclusion, we think that deriving an E-R model to a database where relations between entities are mapped to indexes could give us a clearer view of the data involved by not having superfluous attributes (the foreign keys) and by stating clearly what are the rules to join such entities. The role of the database indexes is now bigger because they are no just an access path, as they use to be, but also a "dictionary" to the relation itself.

Database performance should not be degraded because all the supporting structures already exist in commercial database engines. In some cases, since indexes usually resides in memory during access operations, and they all will have a pointer to the raw data, some accesses should be faster than before. The question here is to improve their use in order to achieve more meaningful metadata, such as data role in a relationship.

References

[1] CODD, Edgar F.: "A Relational Model of Data for Large Shared Data Banks", Comm. of the ACM 13, No. 6 (June 1970)

[2] CHEN, Peter P-S.: "The Entity-Relationship Model - Toward a Unified View of Data", ACM, Transactions on Database Systems (1976)

[3] MARTIN, James: "Principles of Database Management", Prentice-Hall (1976, 1989).

[4] DATE, Christopher J.: "An Introduction to Database Systems - 8th Edition", Addison-Wesley (2003)

[5] NG, Wilfred K.: "An Extension of the Relational Database Model to Incorporate Ordered Domains", ACM, Transactions on Database Systems, Vol. 26, No. 3 (September 2001).

[6] REED, Paul: "The Unified Modeling Language Takes Shape", DBMS 11, Nº 8 (July 1998)

[7] ISO/IEC 9075-*: 2003, Information technology — Database Languages — SQL (2003~2006)
[8] – CODD, Edgar F.: "Domains, Keys, and Referential Integrity on Relational Databases", InfoDB3, Nº 1 (Spring 1988)
[9] – DATE, Christopher J.: "Referential Integrity", Proc.7th Int. Conference on Very Large Data Banks, Cannes, France (September 1981)
[10] – HALL, P. OWLETT, J. and TODD, S. J. P.: "Relations and Entities" in G. M. Nijssen (ed.) Modeling in Data Base Management Systems, Amsterdam, The Netherlands: North-Holland/New York, N. Y.: Elsevier Science (1975)
[11] – CODD, Edgar F.: "Data Models in Database Management", Proc. Workshop on Data Abstraction, Databases and Conceptual Modeling, Pingree Park, Colo (June 1980)
[12] – CHAUDHURI, Surajit and SHIM, Kyuseok: "Optimizations of Queries with User-defined Predicates", Proceedings 22nd International Conference on Very Large Data Bases, Mumbai (Bombay), India (September 1996)