Towards Scalable Scoring for Preference-based Item Recommendation

Markus Stolze & Walid Rjaibi
IBM Research Division - Zurich Research Laboratory
CH-8803 Rüschlikon, Switzerland
{mrs, rja}@zurich.ibm.com

Abstract

Preference-based item recommendation is an important technique employed by online product catalogs for recommending items to buyers. Whereas the basic mathematical mechanisms used for computing value functions from stated preferences are relatively simple, developers of online catalogs need flexible formalisms that support the description of a wide range of value functions and map to scalable implementations for performing the required filtering and evaluation operations. This paper introduces an XML language for describing simple value functions that allow emulating the behavior of commercial preference-based item recommendation applications. We also discuss how the required scoring operations can be implemented on top of a commercial RDBMS, and present directions for future research.

1 Introduction

Online product catalogs are the Internet equivalent to the paper catalogs used by mail-order retailers and component suppliers to present their products to individual customers in business-to-business contexts. Similar to their paper counterparts, they present information about product features and possible uses. In addition to this, online catalogs can provide up-to-date information about availability and prices computed for the individual customer. A second advantage of online catalogs is that they can provide buyer decision support functionality and give recommendations for items in the catalog.

First-generation online catalogs only provide static HTLM pages arranged in a hierarchy. Here terminal pages describe individual products, and intermediary category pages provide the navigational links to lower-level product category pages and terminal product pages. First-generation online catalogs do not provide any item recommendation or decision support. On the contrary, if a buyer needs to compare a group of items that are not directly listed in a predefined category page, then the buyer has to invest a considerable amount of work to first identify all the relevant product pages, then print the pages or manually extract the relevant information into a single document, and finally compare the information to determine the best fit.

The inclusion of applications for feature-based item retrieval into online catalogs helps reduce this problem. These applications facilitate the task of identifying the relevant products. They provide buyers with a product-class specific form that lets them specify requirements regarding the item features. Thus, a user looking for a PC can specify, for example, that she is interested in buying a PC with a Pentium II processor and at least 10 GB

Copyright 2001 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.
Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

of hard-disk space. When receiving the query, the retrieval application scans the product database and returns a list of products that match the request. Some online catalogs also added side-by-side tables that list features of user-selected items in a single comparison table – thus relieving users from having to compile this information manually onto a single page. However, feature-based item retrieval is only of limited help if the number of products that meet the buyer's base requirements is large and if sorting the list according to a single item feature (such as price) is not sufficient because the buyers soft preferences consider more than a single attribute [11]. Thus, a buyer who receives a long list of PCs that all match her base requirements has to inspect each item to determine which of these computers best matches her soft preferences. An alternative to manual inspection is to go back to the original query and enter more restrictive requirements. While being faster, such a strategy can lead to the exclusion of items that would be best fits [9].

Applications for preference-based item recommendation have been developed to address this problem. In addition to allowing buyers to specify their hard requirements, they also let buyers express their "soft" preferences. The information about the requirements and preferences is then used to compile a value function that is used to evaluate and sort items. A number of commercial applications for preference-based item recommendation exist. In the next section we discuss the user experience these systems provide. We then present a simple XML representation scheme that allows specifying the range of value functions that need to be constructed by the recommendation applications discussed. We also introduce an XML-based rule language that supports the explicit representation of how user-provided information is mapped to modifications of the basic value function. Finally we discuss issues related to the scalability of the scoring mechanisms that perform the preference-based item evaluation.

2 Commercial Applications for Preference-based Item Recommendation

Commercial applications for preference-based item recommendation such as PersonaLogic (personaLogic.com), Active Buyer Guide (www.activeBuyersGuide.com), and the PuchaseSource system developed by Frictionless (www.frictionless.com) all work in three phases. After an initial phase of preference elicitation, they use the elicited information to construct the value function. This function is then used to evaluate and sort the items, and finally display the resulting list to the user. Below we discuss each of these phases in more detail.

2.1 Preference Elicitation

The basic mechanism for preference elicitation is exemplified by PersonaLogic. Here preference elicitation is performed by a sequence of dynamic web pages. Often the elicitation starts with a page prompting users for some high-level profile information. For example, the PersonaLogic recommendation application for computers[1] provides an initial dialog that elicites how buyers intend to use the computer. Buyers are asked how often they intend to use the computer for word processing, games, desktop publishing, communication, and education.

From these initial inputs, the PersonaLogic applications derive recommendations and default settings for some of the feature-related requirements and preferences. For example, from the fact that a buyer wants to play games frequently the PersonaLogic computer recommendation application derives the requirement for a fast processor.

In the next elicitation phase (which often spans multiple screens) all user preferences and requirements relating to item features are elicited. Thus, the PersonaLogic computer recommendation application lets buyers enter information such as the minimum memory requirements and the importance of having an integrated DVD drive.

[1]This application, which was available until late 2000, currently (August 2001) is no longer accessible. PersonaLogic was acquired by AOL in mid 2000. However, at AOL a number of PersonaLogic applications still are accessible; for example a dog recommendation application, a pet recommendation application, and a career recommendation application can be accessed at www.aol.personalogic.com/?product=dogs,aolcom, www.aol.personalogic.com/?product=pets,aolcom, and www.aol.personalogic.com/?product=career,aolcom respectively. We have started an archive of screen-shots of such disappearing recommendation applications at the following URL: www.zurich.ibm.com/mrs/recArchive/

Finally, in the last elicitation step, the relative importance of the different preferences is elicited. Thus, the PersonaLogic computer recommendation application elicits the relative importance of having a bigger hard-drive, a faster processor, more RAM, a lower price, more multimedia features, and being from a preferred manufacturer. Similar strategies for preference elicitation are also used by the Frictionless recommendation component. Active Buyers Guide extends this basic preference elicitation procedure by providing additional help in setting the relative importance of preferences. Instead of asking buyers to explicitly provide information about the relative importance of preferences, buyers are presented with a sequence of pairwise comparisons of hypothetical items that only differ in two features. The buyer is then asked to provide information whether, and to which degree, she prefers one of the items.

Allowing buyers to specify base requirements and preferences is consistent with the disjunctive model of decision making observed in the marketing literature [6] that assumes consumers use some base requirements to eliminate purchase alternatives and an additive multi-attribute evaluation function to evaluate the remaining alternatives.

2.2 Value Function Construction

We were not able to find any first-hand public information that describes how commercial applications construct the value function from the elicited preference information. However, from the information elicited and the evaluation that items receive, it seems possible to emulate the evaluation behavior of these applications with a scoring mechanism that interprets a filtering expression and a simple Multi-Attribute Utility Theory (MAUT) value function [5]. The general form of a simple MAUT function that combines the evaluations of individual features of a given item I as a weighted sum is $U(I) = \sum_j W_j \cdot U_j(F_j(I))$, where $U_j(F_j(I))$ is the evaluation of feature j of item I and W_j is the weight assigned to this feature. In such a framework the elicited requirements are used to compile the filtering expression. The elicited buyer preferences determine how item features are evaluated, i.e. how values of feature j (returned by $F_j(I)$) are mapped to an evaluation between 0 and 1 by the evaluation functions $U_j(F_j(I))$. The elicited buyer preferences also determine the relative weight of feature evaluations (i.e. W_j).

2.3 Result Presentation

After the filter expression and the valuation function have been constructed, they are applied to the set of items by the scoring mechanism. The scoring mechanism inspects the feature values of each item. It determines whether the item matches any of the filtering clauses, determines the evaluation of each of the item features, and computes the overall evaluation as the weighted sum of the feature evaluations. After this operation has been performed for each item, the filtered list of evaluated items is displayed to the user. All the preference-based recommendation applications supply buyers with a list of items sorted by their evaluation. However, the applications differ in the amount of information they provide to users for determining the fit of the item with the preferences stated. PersonaLogic provides the least support in that it hides the absolute quality of fit by always assigning the best fitting item an evaluation of 100% – which might be surprising to users that find 100% items lacking with respect to some important preferences. Active Buyers Guide, on the other hand, provides an absolute score that is less than 100% for the best item if it does not completely match the stated buyer preferences. Moreover, for each item it lists the values of features the buyer has expressed special interest in. It also highlights some special plus and minus points about each of the items. This facilitates a rapid inspection of the main item features. The Frictionless PurchaseSource system provides users with the most detailed information on the fit of items to the stated requirements and preferences[2]: For each item the user can see a summary page that lists graphically how well the item satisfies each of the stated requirements and preferences.

[2] Note that the Frictionless PurchaseSource application currently (August, 2001) is no longer available. We hope to include screen-shots also from this application in our recommendation applications archive.

2.4 Preference Revision

After inspecting the list of evaluated items, users have the option to return to the preference elicitation stage of the dialog, and revise some of their stated requirements and preferences. However, none of the recommendation applications discussed here support a close integration of item inspection and preference revision, as it is possible in some research systems [11]. They also do not support critiquing of returned items and their feature values as suggested for systems performing evaluation-oriented provisioning of product information [4].

3 An XML Representation for Simple Value Functions

In the preceding section, we argued that the evaluation behavior of commercial applications for preference-based item recommendation can be emulated with scoring mechanism that evaluates a database of items according to a filtering expression and a simple MAUT value function. We have devised an XML representation schema that allows such filtering expressions and simple MAUT value functions to be represented in an integrated fashion. An example XML representation of a simple value function is shown in Figure 1.

```
<criteria rootId="root">
        <criterion id="Price" importance="50" must="false" attribute="Price">
            <numEval valUnit="$" val1="0" eval1="1" val2="5600" eval2="0"/>
        </criterion>
        <criterion id="Speed" importance="25" must="false" attribute="Speed">
            <numEval valUnit="mhz" val1="300" eval1="0" val2="1600" eval2="1"/>
        </criterion>
        <criterion id="RAM" importance="25" must="false" attribute="RAM">
            <numEval valUnit="MB" val1="32" eval1="0" val2="512" eval2="1"/>
        </criterion>
        <criterion id="CDWriter" importance="5" must="false" attribute="CDWriter">
            <symEval sym="12X">1</symEval>
            <symEval sym="8X">.75</symEval>
            <symEval sym="4X">.5</symEval>
            <symEval sym="2X">.25</symEval>
            <symEval sym="none">0</symEval>
        </criterion>
        <criterion id="HD Size" importance="25" must="false" attribute="HD">
            <numEval valUnit="GB" val1="0" eval1="0" val2="50" eval2="1"/>
        </criterion>
        <criterion id="root" importance="100">
                    <subCriteria>
                            <Criterion  ref="Price"/>
                            <Criterion  ref="Speed"/>
                            <Criterion ref="RAM"/>
                            <Criterion  ref="CDWriter"/>
                            <Criterion  ref="HD Size"/>
                    </subCriteria>
        </criterion>
</criteria>
```

Figure 1: **Example of XML-based specification of a simple MAUT value function**

Using this representation language, a MAUT value function is represented as a tree of evaluation criteria. A criterion is either a terminal or a combination criterion. Terminal criteria (such as Price and CDWriter in Figure 1) have a set of evaluation specifications as sub-elements that specify how numeric and symbolic values of features should be evaluated. Numeric evaluation specifications (i.e., the *numEval* sub-elements of criteria) allow specifying how the numeric values of the corresponding item feature should be mapped to an evaluation between 0 and 1. In the *numEval* specification the “val1” value specifies the starting point of a linear function and “val2” the end-point; the “eval1” value specifies the evaluation at the starting point “val1” and “eval2” the evaluation at the end of the interval. The *symEval* sub-elements of criteria specify how symbolic values are mapped to evaluations. Thus, for example, in the criterion CDWriter in Figure 1 the evaluation specification $< symEvalval = "8X" > .75 < /symEval >$ implies that items with a value of “8X” for their CDWriter feature will receive an evaluation of 0.75 for the CDWriter criterion. In addition to the *numEval* and *symEval* evaluation specifications *numReject* and *symReject* elements can also be included. These elements (not shown in Figure 1) allow hard requirements to be specified for items. Thus, items that match these reject clauses will not be included in the result set. The scoring mechanism interpreting the XML specification has to compute the evaluation and reject values independently. Combination criteria such as "root" combine the evaluation and reject values of their sub-criteria into single evaluation and reject values. Scoring mechanisms interpreting the XML specifications compute the evaluation as the weighted sum of the sub-node evaluations using the "importance" attribute of criteria as the weighting factor. Reject information is propagated in such a way that if an item is rejected according to one sub-criterion that has the attribute "must" set to true, then the item is also rejected according to the super-criterion. Criteria where "must" is set to false only flag items as unsatisfactory without rejecting them.

4 Explicit Representation of Value Function Adaptation Rules

Value function construction from user input can be done by a program that interprets the user input and generates the value function specification. However, we believe that it is advantageous to make explicit the relationship between user input and configuration of the value function. This way the relationship remains easier to understand and easier to maintain. We have therefore created an XML-based rule language that allows specifying how the base value function should be adapted based on user input. Figure 2 shows an example of a set of modification rules. The proposed language for criteria modification includes actions for setting, increasing, and decreasing criteria importance. It also includes means to add reject conditions to criteria. In our experiments with this language in the context of utility-based generation of sales interviews [10], we found it sufficiently expressive to represent how a basic value function is to be adapted to reflect the preferences of the individual buyer. In particular, in our examples we did not encounter a need to dynamically change the hierarchy of criteria that make up the value function.

```
<rules>
    <rule ID="games">
        <if><answer question="gamer">yes</answer></if>
        <then><incImportancePerc criterion="Graphics Card">50</incImportancePerc></then></rule>
    <rule ID="price1">
        <if><answer question="price">over $300</answer></if>
        <then><setImportance criterion="Price">0</setImportance></then></rule>
    <rule ID="price2">
        <if><answer question="price">up to $1500</answer></if>
        <then><addNumValueReject criterion="Price" from="1501" to="5600"/></then></rule>
</rules>
```

Figure 2: **Example of XML representation of value function adaptation rules**

5 Towards Scalable Scoring for Preference-based Item Recommendation

As the set of input items to which the scoring mechanism has to apply the evaluation function becomes larger, the performance of the scoring mechanism becomes increasingly critical. Users are not willing to wait too long before they can access the list of recommended items being generated. Stand-alone scoring mechanisms are likely to opt for implementing sophisticated algorithms and techniques to address this performance issue. Given that the set of input items is likely to be stored in a relational database a question arises. Can the scoring functionality (or at least some part of it) be delegated to the database system (DBMS)? We argue that the answer to this question is an unequivocal yes. Most commercial database systems such as IBM's DB2 UDB allow users to write their own extensions to SQL. In DB2 UDB a user-defined function (UDF) can be used to write extensions to SQL. By expressing the evaluation function as a UDF (or a set of UDFs), the scoring mechanism can be expressed as an SQL query and therefore take full advantage of the DBMS' query processing engine.

Below we give a simple example to illustrate how UDFs can be used to help express the scoring functionality as an SQL query. Suppose that the Price, Speed, RAM, CDWriter, and HDSize information are represented in a database table called "Product" and that a user is interested in obtaining a set of PC recommendations based on the preferences specified in Figure 1.

To express the evaluation function for the “Price” attribute as a UDF two simple steps are required. The first is to write the actual code that implements the evaluation function. In an example the code is written in JAVA but other programming languages such as C and C++ are also supported. The second step is to register the UDF to the database and associate the JAVA code (written in the first step) with it. This is easily accomplished using a CREATE FUNCTION SQL statement [14]. When the UDF is invoked by the database system, the code written in step 1 is executed. Figure 3 lists the piece of JAVA code that implements the evaluation function of the Price attribute according to the preferences set in Figure 1.

```
import COM.ibm.db2.app.*;
class EvaluationUDF extends UDF
{
    public double PriceEval (double price, double val1, double val2)
    {
        /*
        * Get the slope and end point of the line segment that includes the
        * 2 points (val1, 1) and (val2, 0)
        * The line segment equation is y = a * x + b;
        */
        double a = 1 / (val1 − val2);
        double b = −val2 / (val1 − val2);
        /*
        * Return the evaluation for this price
        */
        double eval = a * price + b;
        return (eval);
    }
}
```

Figure 3: **Evaluation function implementation as a JAVA UDF for the Price attribute**

The evaluation functions for the Speed, RAM, CDWriter, and HDSize attributes can be expressed as UDFs in the same way as the Price attribute. Let SpeedEval, RAMEval, CDWriterEval, and HDSizeEval denote the UDFs for the Speed, RAM, CDWriter, and HDSize attributes respectively. Having expressed the evaluation functions as UDFs, the scoring functionality can be easily expressed using the following SQL query :

*SELECT Product_ID, Price, Speed, RAM, CDWriter, HDSize,(50 * PriceEval (Price, 0, 5600) + 25 * SpeedEval(Speed, 1600, 300) + 25 * RAMEval(RAM, 512,32) + 5 * CDWriterEval(CDWriter) + 25 * HDSizeEval(HDSize)) AS Score*
FROM product
ORDER BY Score

The above query shows that the score of each product is computed as a weighted sum aggregation of the Price, Speed, RAM, CDWriter and HDSize attributes. The weight of each attribute is as specified by the “Importance” field in Figure 1. Rather than having the weights hard-coded in the above query, a better alternative is to store the weights in a separate database table (called ATTRIBUTE_WEIGHT for example) and modify the above query to read these weights from the table ATTRIBUTE_WEIGHT. Having the weights stored in a separate table provides more flexibility because users are likely to modify these weights while searching for the product that suits them best. The ATTRIBUTE_WEIGHT table can be implemented using just two columns: A string to indicate the fully qualified attribute name and an integer to represent the weight of that attribute. The following UDF (written in SQL this time) can be used to return the weight of a given attribute :

CREATE FUNCTION weight (attribute-name char(30)) RETURNS integer
LANGUAGE SQL READS SQL DATA NO EXTERNAL ACTION DETERMINISTIC
RETURN SELECT weight FROM attribute_weight WHERE attribute_weigth.attribute-name = weight.attribute-name

The SQL query that implements the scoring functionality can then be modified as follows to use this new UDF :

*SELECT Product_ID, Price, Speed, RAM, CDWriter, HDSize,(weight("product.Price") * PriceEval (Price, 0, 5600) + weight("product.Speed") * SpeedEval(Speed, 1600, 300) + weight("product.RAM") * RAMEval(RAM, 512,32) + weight("product.CDWriter") * CDWriterEval(CDWriter) + weight("product.HDSize") * HDSizeEval(HDSize)) AS Score*
FROM product
ORDER BY Score

By expressing the evaluation function as a UDF (or a set of UDFs) the scoring mechanism can be expressed as an SQL query and therefore take full advantage of the DBMS' query processing engine. The performance benefit of using this approach can be significant when the set of input items to which the scoring mechanism needs to apply the evaluation function becomes large. However, the benefit of improved performance has to be weighted against the actual effort needed to create these UDFs, register them, and write the appropriate SQL statements to call them. Therefore, we are currently investigating ways to automatically perform the described compilation steps from the XML criteria specification given.

Also, other authors have developed schema for preference-based item scoring and discussed how to map them onto relational database queries. In [1], Agrawal and Wimmers proposed a mechanism for representing and combining preferences that is based on a relational representation. Here preferences are described in database tables that mirror the structure of item tables and have an additional "score" column containing the evaluation of items. A wildcard symbol can be used instead of an item feature value to facilitate a concise description of preferences that are independent of certain item feature values. A special symbol (instead of a score between 0 and 1) is used to indicate the rejection of items. Combination of preferences is done through a combination function that is implemented as external function. However, in our opinion, the proposed mechanism for preference representation is not sufficiently expressive to handle the range of value functions used by preference-based recommendation applications. In particular, it seems unclear how numeric evaluation rules should be represented in an efficient manner. Also, it seems that in the proposed scheme an item either receives an evaluation or is rejected. However, most recommendation applications compute rejection and evaluation information independently from each other. The work of Hristidis et al. [7] extends the work of Agrawal and Wimmers [1] by illustrating how materialized views can be used to increase the efficiency of DB-based scoring. Although the idea of using materialized views is certainly attractive, their preference representation schema suffers from the same limitations of expressiveness as that of Agrawal and Wimmers [1].

6 Beyond Simple Value Functions

In the preceding section we discussed how the proposed XML language for simple MAUT value functions could be mapped to scalable scoring mechanism using relational database techniques. We also argued that other proposals for implementing preference functions on top of relational databases are not sufficiently expressive to cover the range of value functions needed by applications for preference-based item recommendation in online product catalogs. However, the introduced XML preference function representation language is also limited in its expressiveness. Although we believe that it covers the value functions that most preference-based item recommendation applications need to construct, we have already experienced practical situations that required custom extensions. For example, extensions are needed to allow preferences regarding values other than numbers and symbols (e.g. dates) to be expressed. Extensions are also needed to deal flexibly with situations in which items have multiple feature values. Finally, the simple scoring scheme also does not cover situations that require items to be classified into more than the two standard classes of “not rejected” and “rejected” items. We have started to experiment with extended value function representations to find the best way to combine easy specification of simple scoring functions with expressive means for extending and customizing the base scoring functionality.

Eventually such extensible value function representations and the associated scoring mechanisms should also be able to deal with more complex value functions that have been explored by recent research on item representation and evaluation. For example, in [8] Lukacs studied mechanism for scoring items with under-specified item values (such as a travel time that “might be between 20 and 30 minutes”). FIPA [13] and Willemot et al. [12] propose to describe spaces of configurable items as constraint satisfaction problems. In addition, Chajewska et al. [2], and Chin and Porage [3] explored mechanisms for dealing with situations in which no reliable information is available regarding the preferences (i.e. weights and feature evaluation functions). The development of description languages that support easy and expressive specification of such complex value functions remains future work. Increasing the scalability of a scoring mechanism by performing the scoring operations within the framework of a relational database system is likely to remain a valid approach to increase the scalability of scoring mechanisms even for these more complex situations. Experiments and empirical evaluations will be needed to further explore this space.

References

[1] R. Agrawal and E. Wimmers. A Framework for Expressing and Combining Preferences. In *Proc. of the ACM-SIGMOD 2000 Conference on Management of Data*, pages 297–396, Dallas TX, May 2000.

[2] U. Chajewska, D. Koller and R. Parr. Making Rational Decisions using Adaptive Utility Elicitation. In *Proc. of the 17th National Conference on Artificial Intelligence AAAI-1999*, pages 363–369, Austin TX, August 2000.

[3] D. N. Chin and A. Porage. Acquiring User Preferences for Product Customization. In *Proc. of the 8th International Conference on User Modeling UM-2001*, pages 95–104, Sonthofen Germany, July 2001.

[4] A. Jameson, R. Schafer, J. Simons and T. Weis. Adaptive Provision of Evaluation-Oriented Information : Tasks and Techniques. In *Proc. of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1886–1893, Montreal, August 1995.

[5] R. T. Clemen. Making Hard Decisions : An Introduction to Decision Analysis. *Wadsworth Publishing Company*, Belmont CA, 1996.

[6] P. E. Green and Y. Wind. Multiattribute Decisions in Marketing : A Measurement Approach. *Dryden*, Hinsdale IL, 1975.

[7] V. Hristidis, N. Koudas and Y. Papakonstantinou. PREFER : A System for the Efficient Execution of Multi-Parametric Ranked Queries. *ACM SIGMOD*, pages 259–270, 2001.

[8] G. Lukacs. Decision Support under Imperfections in Electronic Commerce. *DEXA 2000 Second International Workshop on Logical and Uncertainty Models for Information Systems, Greenwich UK, IEEE Computer Society Press*, pages 538–542, September 2000.

[9] P. Steiger and M. Stolze. Effective Product Selection in Electronic Catalogs. In *Proc. of the Human Factors in Computing Systems*, pages 291–292, Atlanta GA, 2000.

[10] M. Stolze and M. Ströbel. Utility-based Decision Tree Optimization: A Framework for Adaptive Interviewing. In *Proc. of the 7th International Conference on User Modeling UM-2001*, Sonthofen, Germany, July 2001.

[11] M. Stolze. Soft Navigation in Electronic Product Catalogs. *International Journal on Digital Libraries*, 3(1):60–66, 2000.

[12] S. Willmott, M. Calisti, B. Faltings, S. Macho-Gonzalez, O. Belakhdar and M. Torrens. CCL: Expressions of Choice in Agent Communication. In *Proc. of the 4th International Conference on MultiAgent Systems ICMAS-2000*, Boston MA, July 2000.

[13] FIPA. FIPA CCL Content Language Specification. *http://www.fipa.org/specs/fipa00009/XC00009A.html*, 2000.

[14] D. Chamberlin. Using the New DB2: IBM's Object-Relational Database System. *Morgan Kaufmann Publishers*, 1996.