Database Generation by End-Users

Raymond G. Blackie
Analyst Programmer
Shell Canada Limited
Toronto, Ontario
M5G 1X4

Abstract

Many Database Management Systems (DBMS) set formidable barriers to the development of database applications by non-technical end-users. Users are forced to turn to programmers to have them build the databases they need. This results in a bottleneck that effectively discourages vital user participation during application development. This paper outlines techniques that have been successfully used with INQUIRE, a text DBMS, to provide end-users with the tools they need to build databases that they may use in a fashion consistent with the production environment.

La création dynamique de base de données par les utilisateurs

Certains Systèmes de Gestion de Bases de Données (SGBD) élèvent des barrières insurmontables dans le développement d'applications de bases de données par des utilisateurs à formation non technique. Ces utilisateurs sont alors obligés de consulter des programmeurs afin de construire les bases de données dont ils ont besoin. Cela aboutit souvent à une impasse qui décourage efficacement la participation vitale des utilisateurs durant le développement des applications. Cette présentation souligne les techniques qui ont été utilisées avec succès avec INQUIRE, un SGBD à textes, pour fournir aux utilisateurs les outils dont ils ont besoin pour construire des bases de données qu'ils peuvent utiliser d'une manière compatible avec la production de l'environnement.

INTRODUCTION

Most Database Management Systems are so complicated that a programmer handles the creation of the databases. The complexity arises in the attempt to be all things to all people. This in turn creates a bottleneck - the programmer - who must be consulted before you can have your database.

This applies to INQUIRE, a text DBMS, that has been used at Shell Canada for the past three years.

To establish a production INQUIRE database it is necessary to amass sixteen (16) pieces of information. These include the database name, sample data, the Field Definition Table, average record size, capacity, key length, direct access structure characteristics and space allocations for the component files.

When you put all of this information together, you end up with a database that has a direct access structure and a static Field Definition Table. The database may be searched, either sequentially or via the direct access structure, or maintained by multiple concurrent users.

However, such a database must be frequently backed up and periodically reorganized. All maintenance transactions should be logged. A recovery procedure for damaged databases must be available. User access to the system must be monitored for chargeback purposes.

To add insult to injury, the process of establishing such a database often takes weeks to complete. This includes discussing the Field Definition Table with the user, discussing standards, selecting keyed fields, appointment of a Database Administrator, paperwork, determination of the database characteristics and insertion of the database into the production system.

Few non-technical users are willing to wait and endure a rigorous analysis of their application. The Information Centre concept has shown that, given appropriate tools, users can develop their own applications. The user wants to enter data, build a database and query the database - today!

FIELD DEFINITION TABLE

The Field Definition Table, or FDT, describes the fields in the database. The description includes the name, format, length, print format and type of each field.

Field formats include fixed length text, variable length text, unpacked decimal, binary, short floating point and long floating point. The ones most relevant to our discussion are the fixed and variable length text formats and unpacked decimal. The other formats are special numeric formats that could not be entered by a user at a terminal.

Field types include scalar, repeating, subfield, element or group. Since subfields, elements and groups are merely reinterpretations of previously defined fields and do not themselves require any space in the database records, they may be excluded from our discussion. They may be added later if necessary. This leaves us scalar of repeating fields.

Scalar fields may be either unpacked decimal or fixed or variable length text fields.

Repeating fields have either a fixed or variable number of repeats. They may be either unpacked decimal or fixed or variable length text. The one restriction is that a repeating field may not be variable length text and have a variable number of repeats.

The format and type of field determine the amount of space required to hold the field.

ISI

Infodata Standard Input (ISI) format is the format in which the data is expected to be in when building an INQUIRE database.

Traditional flat files put field values in specific columns and the space for a field is reserved whether it is used or not. The length of a field must be known in advance so that columns can be assigned. This is a "horizontal" view of the data.

ISI format, however, is a "vertical" approach. In its simplest form, the field name is entered in columns 1 to 8 and the data in columns 10 to 72. Long fields may be continued on subsequent lines. Fields that are empty need not appear. Records are delineated with an 'END' statement.

ISI format promotes fielding of data and allows the data to use whatever room it requires. Fields are not tied to specific columns and there is no need for fancy line continuation schemes.

Data entry is aided by an ISI command that provides a full-screen form that the user fills in with field names and data.

SIMPLIFYING THE PROCESS

The key to providing an appropriate tool for end-user database creation is the realization that the two most difficult aspects of database creation are the determination of nine (9) characteristic numbers associated with the direct access structure and the construction of a suitable Field Definition Table for the database. Users are also notoriously poor at estimating the amount of space they need for a database.

The first step is to sacrifice the direct access structure and this has several important ramifications. It reduces the number of parameters necessary to define a database from sixteen to seven. All queries must sequentially search the database. The database cannot be maintained directly - instead, you must change your data and rebuild the database. The database does not need backup, reorganization, logging of maintenance, recovery procedures or access monitoring.

The second step is to create a Field Definition Table for the user. This requires that the data be entered in ISI format. The ISI data is analyzed to pick up the field names and various field characteristics.

The final step is to determine the disk space required to hold the database. The database usually requires less than 50% of the space required to hold the ISI data and that information is readily obtained from the system catalogue.

Now we have reduced database creation to two (2) parameters: a database name and data in ISI format. No further reduction is possible.

ANALYZING THE ISI DATA

Historically, this was the first part of the process to be developed. It was originally intended to pinpoint inefficiently stored fields. Part of its function was to propose an alternative FDT that would minimize the storage necessary for each field. It would also identify unused or under-utilized fields.

The first step in the analysis involves gathering information on each field in the ISI data for each record. We need to know the field name, the total length of data over all repeats in the field, number of repeats, and length of the longest repeat. Is the field numeric? If so, how may decimal places does it have? is the field empty?

From this, we can continue on to establish some overall characteristics of each field. We are now in a position to calculate the maximum number of repeats, the longest length of a field value and the average number of repeats. We now also know how many records we are dealing with. We can also answer our earlier questions regarding blank and numeric fields. If the field is numeric, the field length must be adjusted to minimize the loss of decimal places.

The final step is to decide which of the various possible field configurations actually minimizes storage for each field. Once the format and type have been decided for a field, an entry can be generated for the FDT.

FDT entries consist of a field name, format, length, print format, type and possibly the number of repeats.

GENERATING THE SEQUENTIAL DATABASE

As outlined in the introduction, it is only necessary for the user to supply 2 pieces of information in order to build a sequential database.

In the system, as I have set it up, the user would type: DATABASE PROJ to create a sequential database named PROJ. The user would then be prompted for the name of the dataset containing the ISI data. Once this information has been entered, a batch job is created and submitted.

This is what the user sees - let us briefly look at what goes on behind the scenes.

After the user enters the ISI dataset name, the system catalogue is checked for the size of that dataset. The size of the database is then estimated at less than 50% of that size. (If it turns out that the database actually requires more room than the estimated size, additional smaller chunks of space will be provided.)

The batch job includes steps to delete any currently existing database under that name, delete any currently existing FDT under that name, analyze the ISI data and generate an FDT, validate the FDT and load the ISI data into the sequential database. (The database and FDT generated are specific to one user so multiple users may use the same database names.)

This system could be run online but I have chosen to run it in batch because the reports that it produces may be too long to be displayed on a screen.

QUERYING THE SEQUENTIAL DATABASES

Production databases are accessed through the commands INQUIRE (online) and INQUIREB (batch). Users may generate reports, control certain parameters, create datasets on disk or tape, print multiple copies of a report, use special forms, print reports in simplex or duplex and perform maintenance on their databases through the query language.

All of these capabilities, except for the performance of maintenance, exist for the sequential databases. This is an extremely important point! Users should not have to learn a multitude of systems just to get at their database. The path from development to production should be consistent and similar. The only additional information that the user need provide is the fact he is working with a test sequential database.

At the command level, production databases use either direct access or sequential access to retrieve records. Sequential databases are restricted to sequential access (and hence the name).

Since the maintenance commands of the query language rely on direct access, sequential databases cannot be maintained through the query language.

APPLICATIONS

Applications for sequential databases break down into three general classes: personal, design and review.

Sequential databases are not particularly useful for long-term applications or large applications where many people need fast access to some of the records. Applications that require even a moderate amount of maintenance may be unsuitable.

Personal applications include calendaring, project management and a phone book among others. These applications tend to be highly individualistic and involve very few records.

Designing new applications becomes much simpler because the user can do the development on his own. The user may gain new insights into his application before putting it into production thus saving himself and the programming staff a lot of work in the long run.

Reviewing applications is important. Over the lifetime of a database, fields may need to be added, others will no longer be used or the nature of the data changes. Users are prompt to ask for new fields but are often unaware that certain fields are unused or are could be structured more efficiently. The report generated by the analysis of the ISI data provides sufficient information for investigating deletion or alternative storage configurations for each field. To review a production application, it must be dumped into a dataset in ISI format.

EXTENSIONS

The most obvious extension is to have the DATABASE command accept an FDT from the user. If the user provides the keyword NOGEN after the database name, the user will be prompted for the dataset name containing the FDT. The batch job will not delete the current FDT and it will not analyze the ISI data.

This leads to the second extension - verification of the FDT. If users are to develop FDTs, they should be able to validate them without the necessity of loading records into a database. If the keyword NOGO is supplied after the database name, the FDT will be validated and no database will be created.

The third extension is to allow users to specify the amount of space for the database. The parameter TRKS(17) will provide 17 tracks for the database.

The final extension is to permit the loading of flat files into sequential databases. Flat file formats are impossible to analyze without knowledge of the field layout. Since the user knows the field layout he must provide the FDT. Since the user is providing the FDT, analysis of the data is bypassed. The only restriction with flat files is that there may only be one (1) variable type field in the FDT. The DATABASE command will still estimate the database size but it uses a different formula because flat files are assumed to be denser in data than ISI files.

SUMMARY

The system, as I have designed it, consists of five (5) pieces.

First, the user requires a method for entering the data. The ISI command provides a screen on which the user enters field names and data.

Secondly, the user may wish to create his own FDT. Screens are provided for this purpose in the FDT command.

The key component is the DATABASE command as it brings together the ISI data and an FDT to form a sequential database. The FDT is often generated by the DATABASE command through analysis of the ISI data. The DATABASE command also often determines the space necessary for the sequential database.

The final two (2) components provide the query environment for the sequential databases. These are the commands INQUIRE and INQUIREB. Any differences or restrictions pertaining to the sequential databases are due to the nature of sequential databases and are not artificially imposed. The lessons the user learns here are applicable to databases in the production environment.

Sequential databases as I have described them should be viewed as augmenting the production environment to provide a smooth and consistent path into production for end user applications.