

Data Access Strategies

This chapter covers the following topics:

- Efficient Use of Descriptors
 - Collation Descriptor
 - Superdescriptor
 - Subdescriptor
 - Phonetic Descriptor
 - Hyperdescriptor
 - File Coupling
 - User-Assigned ISNs
 - Using the ISN as a Descriptor
 - ADAM Usage
-

Efficient Use of Descriptors

Descriptors are used to select records from a file based on user-specified search criteria and to control a logical sequential read process. The use of descriptors is thus closely related to the access strategy used for a file. Additional disk space and processing overhead are required for each descriptor, particularly those that are updated frequently. The following guidelines may be used in determining the number and type of descriptors to be defined for a file:

- If data in certain fields needs to be resequenced before processing on the field can continue, a collation descriptor can be defined.
- The distribution of values in the descriptor field should be such that the descriptor can be used to select a small percentage of records in the file;
- Additional descriptors should not be defined to further refine search criteria if a reasonably small number of records can be selected using existing descriptors;
- If two or three descriptors are used in combination frequently (for example, area, department, branch), a superdescriptor may be used instead of defining separate descriptors;
- If the selection criterion for a descriptor always involves a range of values, a subdescriptor may be used;
- If the selection criterion for a descriptor never involves the selection of null value, and a large number of null values are possible for the descriptor, the descriptor should be defined with the null-value suppression (NU) option;

- If a field is updated very frequently, it should normally not be defined as a descriptor;
- Files that have a high degree of volatility (large number of additions and deletions) should not contain a large number of descriptors.

Collation Descriptor

A collation descriptor is used to sort (collate) descriptor field values in a special sequence based on a user-supplied algorithm. An alpha or wide field can be defined as a parent field of a "collation" descriptor.

Special collation descriptor user exits are specified using the ADARUN parameter CDXnn (CDX01 through CDX08). The user exits are used to encode the collation descriptor value or decode it back to the original field value. Each collation descriptor must be assigned to a user exit, and a single user exit may handle multiple collation descriptors.

Superdescriptor

A superdescriptor is a descriptor created from a combination of up to 20 fields (or portions of fields). The fields from which a superdescriptor is derived may or may not be descriptors. Superdescriptors are more efficient than combinations of ordinary descriptors when the search criteria involve a combination of values. This is because Adabas accesses one inverted list instead of several and does not have to 'AND' several ISN lists to produce the final list of qualifying records. Superdescriptors can also be used in the same manner as ordinary descriptors to control the logical sequence in which a file is read.

The values for search criteria that use superdescriptors must be provided in the format of the superdescriptor (binary for superdescriptors derived from all numeric fields, otherwise alphanumeric). If the superdescriptor format is binary, the input of the search value using an interactive query or report facility such as Natural may be difficult.

Subdescriptor

A descriptor that is derived from a portion of a field is called a subdescriptor. The field used to derive the subdescriptor may or may not be a descriptor. If a search criteria involves a range of values that is contained in the first 'n' bytes of an alphanumeric field or the last 'n' bytes of a numeric field, a subdescriptor may be defined from only the relevant bytes of the field. Using a subdescriptor allows the search criterion to be represented as a single value rather than a range. This results in more efficient searching, since Adabas does not need to merge intermediate ISN lists; the merged list already exists.

For example, assume an alphanumeric field AREA of 8 bytes, the first 3 of which represent the region and the last 5 the department. If only records for region '111' are desired, a search criterion of 'AREA = 11100000 thru 11199999' would be required without a subdescriptor. If the first three bytes of AREA were defined as a subdescriptor, a search criterion equal to 'REGION = 111' can be specified.

Phonetic Descriptor

A phonetic descriptor may be defined to perform phonetic searches. Using a phonetic descriptor in a Find command returns all the records that contain similar phonetic values. The phonetic value of a descriptor is based on the first 20 bytes of the field value with only alphabetic values being considered (numeric values, special characters and blanks are ignored).

Hyperdescriptor

The hyperdescriptor option enables descriptor values to be generated based on a user-supplied algorithm. Up to 31 different hyperdescriptors can be defined for a single physical Adabas database. Each hyperdescriptor must be named by an appropriate HEXnn ADARUN statement parameter in the job where it is used.

Hyperdescriptors can be used to implement n-component superdescriptors, derived keys, or other key constructs. For more information about hyperdescriptors, see the documentation on User and Hyperexits, as well as the ADACMP utility description in the Adabas Utilities documentation.

File Coupling

Using a single Find command, file coupling allows the selection of records from one file that are related (coupled) to records containing specified values in a second file. For example, assume two files, CUSTOMER and ORDERS, that contain customer and order information, respectively. Each file contains the descriptor CUSTOMER_NUMBER, which is used as the basis for relating (coupling) the files.

Physical Coupling

The files are physically coupled using the ADAINV utility, which creates a pair of additional indices in the inverted list indicating which records in the CUSTOMER file are related (coupled) to records in the ORDERS file (that is, have the same customer number) and vice versa. Once the files have been coupled, a single Find command containing descriptors from either file may be constructed, for example:

```
FIND CUSTOMER WITH NAME = JOHNSON
      AND COUPLED TO ORDERS
      WITH ORDER-MONTH = JANUARY
```

Physical coupling may be useful for information retrieval systems in which file volatility is very low, or the additional overhead of the coupling lists is deemed insignificant compared with the ease with which queries may be formulated. It may also be useful for small files which are primarily query-oriented.

Physical coupling may involve a considerable amount of additional overhead if the files involved are frequently updated. The coupling lists must be updated if a record in either of the files is added or deleted, or if the descriptor used as the basis for the coupling is updated in either file.

Physical coupling requires additional disk space for the storage of the coupling indices. The space required depends on the number of records that are related (coupled). The best case is where the coupling descriptor is a unique key for one of the files. This means that only a few records in one file will be coupled to a given record in the other file. The worst case is when a many-to-many relationship exists between the files. This will result in a large number of records being coupled to other records in both files.

A descriptor used as the basis for coupling should normally be defined with the null suppression option so that records containing a null value are not included in the coupling indices.

See the Adabas Utilities documentation, the ADAINV utility, for additional information on the use of coupling.

Logical Coupling

A multiframe query may also be performed by specifying the field to be used for interfile linkage in the search criteria. This feature is called logical coupling and does not require the files to be physically coupled.

Although this technique requires read commands, it is normally more efficient if the coupling descriptor is volatile because it does not require any physical coupling lists. It should also be noted that the user program need only specify the search criteria and the field to be used for the soft-coupling link. *Adabas performs all necessary search, read and internal list matching operations.*

User-Assigned ISNs

The user has the option of assigning the ISN of each record in a file rather than having this done by Adabas. This technique permits later data retrieval using the ISN directly rather than using the inverted list technique. This requires that the user develop his own method for the assigning a unique ISN to each record. The resulting ISNs must be within the range of the MINISN and MAXISN parameter values specified by the ADALOD utility when the file is loaded.

Using the ISN as a Descriptor

The user may store the ISN of related records in another record in order to be able to read the related records directly without using the Inverted Lists.

For example, assume an application which needs to read an order record and then find and read all customer records for the order. If the ISN of each customer record (for more than one customer per order, a multiple-value field could be used) were stored in the order record, the customer records could be read directly since the ISN is available in the order record.

Storing the customer record ISNs avoids having to issue a FIND command to the customer file to determine the customer records for the order. This technique requires that the field containing the ISNs of the customer records be established and maintained in the order record, and assumes that the ISN assignment in the customer file will not be changed by a file unload and reload operation.

ADAM Usage

The Adabas direct access method (ADAM) facility permits the retrieval of records directly from Data Storage without access to the inverted lists. The Data Storage block number in which a record is located is calculated using a randomizing algorithm based on the ADAM key of the record. The use of ADAM is completely transparent to application programs and query and report writer facilities.

The ADAM key of each record must be a unique value. The ISN of a record may also be used as the ADAM key.

While accessing ADAM files is significantly faster, adding new records to and loading of ADAM files is slower than for standard files because successive new records will not generally be stored in the same block.

If an ADAM file is to be processed both randomly and in a given logical sequence, the logical sequential processing may be optimized by using the bit truncation feature of the ADALOD utility. This feature permits the truncation of a user-specified number of bits from the rightmost portion of each ADAM key value prior to its usage as input to the randomizing algorithm. This will cause records of keys with similar leftmost values to be stored in the same Data Storage block.

It is important not to truncate too many bits, however, as this may increase the number of overflow records and degrade random access performance. The reason is, overflow records which cannot be stored in the blocks located using the ADAM key are stored elsewhere using the standard inverted list process; overflow records must also be located using the inverted list. The only other way to minimize overflow is to specify a relatively large file and padding factor size.

ADAM will generally use an average of 1.2 to 1.5 I/O operations (including an average of overflow records stored under Associator control in other blocks of the file), rather than the three to four I/O operations required to retrieve a record using the inverted lists. Overflow records are also retrieved using normal Associator inverted list references.

The variable factors of an ADAM file that affect performance are, therefore, the amount of disk space available (the more space available, the fewer the overflow records which need to be located with an inverted list), the number of bits truncated from the ADAM key, and the amount of record adding and update activity. The ADAMER utility may be used to determine the average number of I/O operations for various combinations of disk space and bit truncation. See the Adabas Utilities documentation for additional information.