

CIS 365
Part II
Lecture 20
VSAM & COBOL

SELECT PERSONNEL-FILE

ASSIGN	TO INFILE
ORGANIZATION	IS INDEXED
ACCESS MODE	IS RANDOM
RECORD KEY	IS EMPLOYEE-NUMBER
ALTERNATE RECORD KEY	IS EMPLOYEE-AGE WITH DUPLICATES
ALTERNATE RECORD KEY	IS DEPARTMENT-CODE WITH DUPLICATES
FILE STATUS	IS PERSONNEL-FILE-STATUS.

The FILE STATUS clause is available for any type of file organization and allows the programmer to distinguish between the many different types of I/O error conditions. The concept was first introduced in Chapter 6 in connection with debugging (see page 158). The operating system automatically returns a two-position field known as the *I/O status* (or file status bytes) to the data name designated in the FILE STATUS clause. The value of the file status bytes may be interrogated by the programmer, who is thus able to more closely monitor the results of any I/O operation.

Table 18.1 lists the various file status codes and their meaning. The use of file status codes is illustrated in the ensuing program to create an indexed file.

Table 18.1 File Status Codes

00	A successful input/output operation is performed with no further information available.
04	A READ is successful, but the length of the record being processed does not conform to the fixed file attributes for that file.
05	An OPEN is successful, but the referenced optional file is not present at open time.
07	An input/output statement is successful; however, for a CLOSE with NO REWIND, REEL/UNIT, or FOR REMOVAL or for an OPEN with NO REWIND the referenced file is on a nonreel/unit medium.
10	A sequential READ is attempted and no next logical record exists because (1) the end of file has been reached; or (2) an optional input file is not present.
14	A sequential READ is attempted and the number of significant digits in the record number is larger than the size of the key data item described for the file.
15	A sequential READ statement is attempted for the first time on an optional file that is not present.
21	A sequence error exists for a sequentially accessed indexed file.
22	An attempt is made to write or rewrite a record that would create a duplicate prime record key or duplicate alternate record key without the DUPLICATES phrase.
23	An attempt is made to randomly access a record that does not exist in the file, or a START or random READ is attempted on an optional input file that is not present.
24	An attempt is made to write beyond the externally defined boundaries.
25	A START statement or a random READ statement has been attempted on an optional file that is not present.
30	A permanent error exists and no further information is available concerning the input/output operation.
34	A permanent error exists because of a boundary violation; an attempt is made to write beyond the externally defined boundaries.
35	A permanent error exists because an OPEN with the INPUT, I/O, or EXTEND phrase is attempted on a nonoptional file that is not present.
37	A permanent error exists because an OPEN is attempted on a file and that file will not support the open mode specified: (1) EXTEND or OUTPUT phrase specified but not supported by the file; (2) I/O phrase is specified, but input and output operations are not supported by the file; or (3) INPUT phrase is specified, but the file will not support READ operations.
38	A permanent error exists because an OPEN is attempted on a file previously closed with a lock.
39	The OPEN is unsuccessful because a conflict has been detected between the fixed file attributes and the ones specified for that file in the program.
41	An OPEN statement is attempted for a file in the open mode.
42	A CLOSE statement is attempted for a file not in the open mode.
43	In the sequential access mode, the last input/output statement executed for the file prior to the execution of a DELETE or REWRITE statement was not a successfully executed READ statement.
44	A boundary violation exists because of an attempt to: (1) write or rewrite a record that is larger than the largest or smaller than the smallest record allowed by the RECORD IS VARYING clause of the associated file-name, or (2) rewrite a record and the record is not the same size as the record being replaced.
46	A sequential READ is attempted on a file open in the input or I/O mode and no valid next record has been established because the preceding: (1) START was unsuccessful, (2) READ was unsuccessful but did not cause an at-end condition, or (3) READ caused an at-end condition.
47	The execution of a READ or START is attempted on a file not open in the input or I/O mode.
48	The execution of a WRITE is attempted on a file not open in the I/O, output, or extend mode.
49	The execution of a DELETE or REWRITE statement is attempted on a file not open in the I/O mode.

WRITE Statement. There is only one syntax for the WRITE statement for indexed sequential files. It resembles the WRITE statement for sequential files except for the optional INVALID KEY clause.

COBOL-85

```
WRITE record-name [FROM identifier ]  
[INVALID KEY imperative-statement-1]  
[NOT INVALID KEY imperative-statement-2 ] [END-WRITE ]
```

An example is provided below.

```
WRITE PERSONNEL-RECORD  
FROM WS-PERS-RECORD  
INVALID KEY PERFORM ERROR-IN-OUTPUT.
```

Under the following circumstances, an output operation will fail and an invalid key condition will be raised:

1. Key order violation during loading
2. Record key duplication during insertion
3. Alternate key duplication
4. Attempting to write into a full file

READ Statement. An indexed file can be read in sequential or random access modes. Sequential input requires the exact same syntax of the READ statement for sequential files, with the exception of the NEXT clause. The syntax of this format is as follows.

```

READ file-name [ NEXT ] RECORD [ INTO identifier ]
[ AT END imperative-statement-1 ]
[ NOT AT END imperative-statement-2 ] [ END-READ ]

```

An example is given below.

```

READ PERSONNEL-FILE RECORD
INTO WS-PERS-RECORD
AT END PERFORM INPUT-COMPLETED.

```

If the file access mode is DYNAMIC, the sequential READ requires the use of the NEXT phrase.

```

READ PERSONNEL-FILE NEXT RECORD
INTO WS-PERS-RECORD
AT END PERFORM INPUT-COMPLETED.

```

Random-access input requires a different syntax for the READ statement.

```

READ file-name RECORD [ INTO identifier ]
[ KEY IS data-name ]
INVALID KEY imperative-statement-1 ]
NOT INVALID KEY imperative-statement-2 ]
END-READ ]

```

```

.....
ENVIRONMENT DIVISION.
.....
SELECT PERSONNEL-FILE
    ASSIGN                TO INFILE
    ORGANIZATION          IS INDEXED
    ACCESS                IS RANDOM
    RECORD KEY            IS EMPLOYEE-NUMBER
    ALTERNATE RECORD KEY IS EMPLOYEE-AGE WITH DUPLICATES
    ALTERNATE RECORD KEY IS DEPARTMENT-CODE WITH DUPLICATES
.....
PROCEDURE DIVISION.
.....
***** RANDOM RETRIEVAL BY PRIMARY KEY, "EMPLOYEE-NUMBER"

    MOVE "1200" TO EMPLOYEE-NUMBER.

    READ PERSONNEL-FILE
    KEY IS EMPLOYEE-NUMBER
    INVALID KEY PERFORM ERROR-IN-INPUT.
.....
***** RANDOM RETRIEVAL BY ALTERNATE KEY, "EMPLOYEE-AGE"

    MOVE 29 TO EMPLOYEE-AGE.

    READ PERSONNEL-FILE
    KEY IS EMPLOYEE-AGE
    INVALID KEY PERFORM ERROR-IN-INPUT.
.....

```

REWRITE Statement. The purpose of the REWRITE statement for indexed files is to update a record. Syntactically, it resembles the WRITE statement.

```
REWRITE record-name [FROM identifier]
  [INVALID KEY imperative-statement-1]
  [NOT INVALID KEY imperative-statement-2]
  [END-REWRITE]
```

```
.....
MOVE "1200" TO EMPLOYEE-NUMBER.

READ PERSONNEL-FILE
  INVALID KEY PERFORM ERROR-IN-INPUT.

MOVE "SHEILA GORDON" TO EMPLOYEE-NAME.
ADD 1 TO EMPLOYEE-AGE.
.....
REWRITE PERSONNEL-RECORD
  INVALID KEY PERFORM ERROR-IN-REWRITE.
```

If the file access mode is RANDOM, a random READ should be used to access a record. An invalid key condition will be raised with REWRITE under two circumstances.

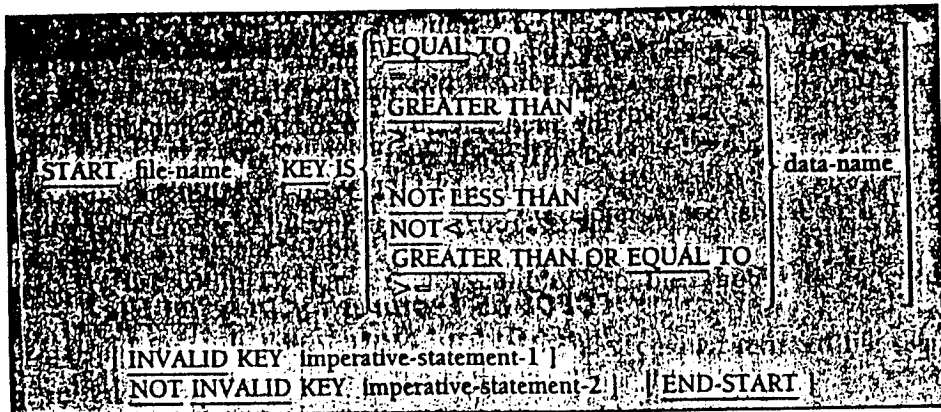
1. The primary key value of the record that has been read most recently is changed prior to rewriting.
2. An alternate key value of the record has been changed such that rewriting violates its uniqueness. This happens if the alternate key has been defined without a WITH DUPLICATES clause.

DELETE Statement. This statement is used to access a record either sequentially or directly and delete it from the file. Syntactically, the DELETE statement is defined as follows.

```
DELETE file-name RECORD  
    [ INVALID KEY imperative-statement-1 ]  
    [ NOT INVALID KEY imperative-statement-2 ] [ END-DELETE ]
```

An example is provided below.

```
.....  
MOVE "1200" TO EMPLOYEE-NUMBER.  
  
DELETE PERSONNEL-FILE  
    INVALID KEY PERFORM ERROR-IN-DELETE.
```



Here is another example.

```

.....
MOVE "1200" TO EMPLOYEE-NUMBER.

START PERSONNEL-FILE
  KEY IS GREATER THAN EMPLOYEE-NUMBER
  INVALID KEY PERFORM ERROR-IN-START.

READ PERSONNEL-FILE
  AT END MOVE "YES" TO END-OF-FILE-FLAG.
.....

```

The START statement above locates the first record whose primary key value is larger than "1200", and the READ statement accesses that record.

```

.....
MOVE "1200" TO EMPLOYEE-NUMBER.

START PERSONNEL-FILE
  INVALID KEY PERFORM ERROR-IN-START.

READ PERSONNEL-FILE
  AT END MOVE "YES" TO END-OF-FILE-FLAG.
.....

```

In this case, the START predicate is implicit and is equivalent to `KEY IS EQUAL TO EMPLOYEE-NUMBER`. Therefore, the READ statement will access the record whose primary key value is equal to "1200".

An invalid key condition will be raised if the START predicate cannot be satisfied within the scope of the key space for the file. For example, if the START predicate contains `IS EQUAL TO` and there is no record in the file whose key value matches that of the current value stored in the data name in the predicate, the invalid key condition is true. As another example, if the START predicate is `KEY IS GREATER THAN EMPLOYEE-NUMBER`, the current value of `EMPLOYEE-NUMBER` is 1200, and the largest key value in the file is 1100, again an invalid key condition will be raised.