

Magiszter

Számítástechnikai szerkesztőség

MISAM

Index szekvenciális file - kezelés C nyelven,
egymunkahelyes és NOVELL hálózatos rendszerekhez

Programvédelem nélkül!

Szoftver
IBM PC, XT, AT
és
kompatibilis mikroszámítógépekre





GONDOLA—KISS—KRAFFT—SCHILL—SZEGEDI

MISAM

Index szekvenciális file-kezelés

195731

GONDOLA ZOLTÁN—KISS GYÖRGY—KRAFFT VALTER
SCHILL RÓBERT—SZEGEDI JÁNOS

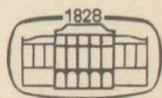
MISAM

Index szekvenciális file - kezelés

MS/PC DOS vagy NOVELL

környezetben

**MTA
KIK**



Akadémiai Kiadó, Budapest 1988

887306

MAGYAR
TUDOMÁNYOS AKADÉMIA
KÖNYVTÁRA

M. TUD. AKADÉMIA KÖNYVTÁRA
Könyvtár 14.939...../19...88...sz.

Szoftver - dokumentáció
Sorozatszerkesztő: Pintér Tibor
A kiadványt szakmailag ellenőrizte: Juhász Lehel

©Akadémiai Kiadó és Nyomda Vállalat
Gondola-Kiss-Krafft-Schill-Szegedi, 1988
Minden jog fenntartva

A szoftver - dokumentáció a $\text{m}\text{T}\text{E}\text{X}$ programcsomaggal készült
©Akadémiai Kiadó és Nyomda Vállalat · Fadgyas – Miklós

Tartalomjegyzék

1. Bevezetés	11
1.1. A MISAM rendszer általános jellemzői	12
1.2. A dBASE kompatibilis file-forma	14
1.3. A kézikönyv felépítése	16
2. A MISAM modell	17
3. A MISAM használata	19
3.1. Az állományleíró táblázat	20
3.2. Állományok létrehozása	20
3.3. Adatkezelő könyvtári függvények	20
3.4. Interaktív file editor	23
3.5. A standard adatkezelő programok	23
3.5.1. Standard adatfelvivő program	24
3.5.2. Standard lekérdező és módosító program	24
3.5.3. Standard listázó	24
3.6. Export és import segédprogramok	26
3.7. Karbantartó segédprogramok	26
3.7.1. Indexfile-ok újraépítése	26
3.7.2. Tömörítés	26
3.8. Táblázatgeneráló programok	27
3.8.1. .FDF táblából include (.H) tábla	27
3.8.2. .FDF táblából .RPD tábla	27
3.8.3. .FDF táblából .RPF tábla	27
3.9. A kézikönyvben leírt példa	28
4. Adatállományt leíró táblázat (.FDF)	29
4.1. Adattípusok és ábrázolás	29

4.2. Adatfile leírás	30
4.3. Index definíciók	31
4.4. Az adatmezők megadása	33
4.5. A teljes leíró C file esetén	34
4.6. A teljes leíró dBASE file esetén	35
5. A MISAM programozása	39
5.1. A MISAM.H header file	39
5.2. A DATE.H header file	44
5.3. Az aktuális pozíció	44
5.4. A file eleje, file vége pozíció	45
5.5. A kiválasztott index	45
5.6. Konkurens programozás	46
5.6.1. A file műveletek szinkronizálása	47
5.6.2. Rekordok módosítása, törlése	48
6. Adatállományt kezelő függvények	51
6.1. ISBUILD	51
6.2. ISCLOSE	54
6.3. ISCREAT	56
6.4. ISDATETOKEY	58
6.5. ISDELETE	59
6.6. ISFIRST	61
6.7. ISGETFDF	62
6.8. ISINFO	64
6.9. ISKEYTODATE	66
6.10. ISLAST	67
6.11. ISNETWORK	68
6.12. ISOPEN	70
6.13. ISREAD	72
6.14. ISRLOCK	76
6.15. ISRUNLOCK	78
6.16. ISSELIND	80
6.17. ISUPDATE	82
6.18. ISWLOCK	85
6.19. ISWRITE	88
6.20. ISWUNLOCK	90
6.21. Példaprogramok	93

6.21.1. Állomány listázása	93
6.21.2. Állomány karbantartása	97
7. Standard építőelemek és segédprogramok	107
7.1. CHKFDF: tábla ellenőrző	107
7.1.1. A program funkciója	107
7.1.2. A program indítása és opciói	107
7.2. CREAISAM: MISAM file létrehozó	107
7.2.1. A program funkciója	108
7.2.2. A program indítása és opciói	108
7.2.3. A kezelői párbeszéd	108
7.3. FED: interaktív file editor	108
7.3.1. A program funkciója	108
7.3.2. A program indítása és opciói	110
7.3.3. A kezelői párbeszéd	110
7.4. ENTRY: standard adatfelvivő	110
7.4.1. A program funkciója	110
7.4.2. A program indítása és opciói	112
7.4.3. A kezelői párbeszéd	112
7.5. QUERY: lekérdező és módosító	112
7.5.1. A program funkciója	113
7.5.2. A program indítása és opciói	114
7.5.3. A kezelői párbeszéd	114
7.6. LIST: állomány listázó program	119
7.6.1. A program funkciója	119
7.6.2. A program indítása és opciói	120
7.6.3. A kezelői párbeszéd	121
7.7. TODBASE: adatkonverzió dBASE formába	121
7.7.1. A program funkciója	121
7.7.2. A program indítása és opciói	122
7.7.3. A kezelői párbeszéd	122
7.8. FRDDBASE: adatkonverzió dBASE formából	123
7.8.1. A program funkciója	123
7.8.2. A program indítása és opciói	123
7.8.3. A kezelői párbeszéd	123
7.9. TODOS: adatkonverzió DOS formába	123
7.9.1. A program funkciója	124

7.9.2. A program indítása és opciói	124
7.9.3. A kezelői párbeszéd	124
7.10. FROMDOS: adatkonverzió DOS formából	124
7.10.1. A program funkciója	124
7.10.2. A program indítása és opciói	125
7.10.3. A kezelői párbeszéd	125
7.11. REBUILD: állomány újra építése	125
7.11.1. A program funkciója	125
7.11.2. A program indítása és opciói	126
7.12. COMPRESS: állomány tömörítése	127
7.12.1. A program funkciója	127
7.12.2. A program indítása és opciói	127
7.13. FDFTOH: include file generátor	128
7.13.1. A program funkciója	128
7.13.2. A program indítása és opciói	128
7.14. FDFTORPD: RPD file generátor	130
7.14.1. A program funkciója	130
7.14.2. A program indítása és opciói:	130
7.15. FDFTORPF: RPF file generátor	130
7.15.1. A program funkciója	131
7.15.2. A program indítása és opciói:	132
8. Technikai információk	133
8.1. Korlátozások	133
8.2. Externális változók	133
8.3. MISAM belső file szerkezet	135
9. FÜGGELÉK	137
9.1. A standard adatbeviteli képernyő	137
9.2. A standard ASCII file-forma	137
9.3. A listaleírás szabályai	138
9.4. A disztributív anyag ismertetése	140
Tárgymutató	143

1. fejezet

Bevezetés

A MISAM egy indexelt szekvenciális (ISAM) file-kezelő rendszernek és a köré csoportosuló adatkezelő programoknak és segédprogramoknak az együttese. IBM XT, AT (kompatibilis) számítógépeken, MS-DOS 3.xx operációs rendszer alatt működnek, a file-kezelő rendszer MICROSOFT C (v4.00) programokból hívható.

A MISAM könyvtárt használó programok futtathatók NOVELL hálózat alatt és önálló gépen egyaránt.

A MISAM fő előnyei a legtöbb forgalomban lévő file-rendszerhez képest:

- A file kezelő rendszer moduljainak magas szintje és ebből eredően a könnyű programozhatóság.
- Lehetőség dBASE III kompatibilis file-szervezésre.

A MICROSOFT C környezetben fejlesztett alkalmazások és a dBASE programok ugyanazt az adatállományt használhatják.

- A nívós, jól használható adatkezelő programok és segédprogramok.

A listázó program például olyan, hogy a felhasználó szabadon változtathatja a lista formátumát és tartalmát, beleértve az összegző sorokat is. A lekérdezés is meglehetősen kötetlen, a meglévő indexeken kívül új lekérdezési szempontokat (sorrendet) lehet megadni, s mód van minta szerinti szekvenciális keresésre is. A lekérdezés és a listázás mindig összefüggő részállományt dolgoz fel, az intervallum interaktívan jelölhető ki.

A MISAM rendszeren már több élő alkalmazói rendszer alapul. A tapasztalat szerint megbízhatósága nagy, sebessége többfelhasználós környezetben is kielégítő. Jelentőségét megnöveli az a tény, hogy MISAM háttérrel komplett adatfeldolgozási alkalmazásgenerátor működik (MICE), s annak standardizált megoldásai részekben is elérhetőek:

- Menü (és monitor) generátor,
- Kérdőív definíciós rendszer és könyvtár,
- Lekérdezések definíciós rendszere és könyvtár,
- Lista generátor,
- On - line dokumentációs rendszer.

Ezzel a fejezettel az adatfeldolgozási problémák a dBASE, FOXBASE stb. keretrendszerrel összevethető ráfordítással C nyelven is megoldhatók.

7.9.2. A program indítása és opciói	124
7.9.3. A kezelői párbeszéd	124
7.10. FROMDOS: adatkonverzió DOS formából	124
7.10.1. A program funkciója	124
7.10.2. A program indítása és opciói	125
7.10.3. A kezelői párbeszéd	125
7.11. REBUILD: állomány újra építése	125
7.11.1. A program funkciója	125
7.11.2. A program indítása és opciói	126
7.12. COMPRESS: állomány tömörítése	127
7.12.1. A program funkciója	127
7.12.2. A program indítása és opciói	127
7.13. FDFTOH: include file generátor	128
7.13.1. A program funkciója	128
7.13.2. A program indítása és opciói	128
7.14. FDFTORPD: RPD file generátor	130
7.14.1. A program funkciója	130
7.14.2. A program indítása és opciói:	130
7.15. FDFTORPF: RPF file generátor	130
7.15.1. A program funkciója	131
7.15.2. A program indítása és opciói:	132
8. Technikai információk	133
8.1. Korlátozások	133
8.2. Externális változók	133
8.3. MISAM belső file szerkezet	135
9. FÜGGELEK	137
9.1. A standard adatbeviteli képernyő	137
9.2. A standard ASCII file-forma	137
9.3. A listaleírás szabályai	138
9.4. A disztributív anyag ismertetése	140
Tárgymutató	143

1. fejezet

Bevezetés

A MISAM egy indexelt szekvenciális (ISAM) file-kezelő rendszernek és a köré csoportosuló adatkezelő programoknak és segédprogramoknak az együttese. IBM XT, AT (kompatibilis) számítógépeken, MS-DOS 3.xx operációs rendszer alatt működnek, a file-kezelő rendszer MICROSOFT C (v4.00) programokból hívható.

A MISAM könyvtárt használó programok futtathatók NOVELL hálózat alatt és önálló gépen egyaránt.

A MISAM fő előnyei a legtöbb forgalomban lévő file-rendszerhez képest:

- A file kezelő rendszer moduljainak magas szintje és ebből eredően a könnyű programozhatóság.
- Lehetőség dBASE III kompatibilis file-szervezésre.

A MICROSOFT C környezetben fejlesztett alkalmazások és a dBASE programok ugyanazt az adatállományt használhatják.

- A nívós, jól használható adatkezelő programok és segédprogramok.

A listázó program például olyan, hogy a felhasználó szabadon változtathatja a lista formátumát és tartalmát, beleértve az összegző sorokat is. A lekérdezés is meglehetősen kötetlen, a meglévő indexeken kívül új lekérdezési szempontokat (sorrendet) lehet megadni, s mód van minta szerinti szekvenciális keresésre is. A lekérdezés és a listázás mindig összefüggő részállományt dolgoz fel, az intervallum interaktívan jelölhető ki.

A MISAM rendszeren már több élő alkalmazói rendszer alapul. A tapasztalat szerint megbízhatósága nagy, sebessége többfelhasználós környezetben is kielégítő. Jelentőségét megnöveli az a tény, hogy MISAM háttérrel komplett adatfeldolgozási alkalmazásgenerátor működik (MICE), s annak standardizált megoldásai részekben is elérhetőek:

- Menü (és monitor) generátor,
- Kérdőív definíciós rendszer és könyvtár,
- Lekérdezések definíciós rendszere és könyvtár,
- Lista generátor,
- On - line dokumentációs rendszer.

Ezzel a fegyvertárral az adatfeldolgozási problémák a dBASE, FOXBASE stb. keretrendszerrel összevethető ráfordítással C nyelven is megoldhatók.

Csak a ráfordítás összevethető, a termékek nem; a C nyelven létrehozott rendszerrel ugyanis a keretrendszeres megoldások sem gyorsaságban, sem rugalmasságban nem versenyképesek.

1.1. A MISAM rendszer általános jellemzői

A MISAM rendszer három részből áll:

- a C nyelvű programokhoz szerkeszthető könyvtárból;
- az alapvető adatfeldolgozási feladatokat (adatbevitel, lekérdezés, listázás) általános szinten megvalósító programokból;
- konvertáló és karbantartó segédprogramokból.

A könyvtár az adatbázisok elérését megvalósító, C-ből hívható modulokból áll. A könyvtárnak kis, közepes és nagy (small, medium és large) memóriamodelles változata létezik SMISAM.LIB, MMISAM.LIB, LMISAM.LIB néven. A könyvtár hozzászerkesztésekor 20-35 Kbyte (kód + adat) területtel növeli meg a program méretét. A tapasztalat szerint az átlagos adatfeldolgozási alkalmazásban több ezer soros C programok is csak kis modelles könyvtárat igényelnek. Több célkönyvtárt használó programoknál a hozzászerkesztett egyéb könyvtárak miatt gyakran lehet szükség a közepes vagy a nagy modellre. Például a MI_FORM kérdőíves könyvtár, a MI_QUERY interaktív lekérdező könyvtár és a MI_REP report generátor könyvtár egyidejű használata esetén a 400 soros programok már rendre csak közepes memóriamodellben férnek el.

A MISAM rendszer tartalmazza az ENTRY.EXE, QUERY.EXE és LIST.EXE programokat. Ezek használatával programozás nélkül megvalósítható egy teljes adatfeldolgozási folyamat: adatbevitel, bevitt adatok módosítása, bevitt adatok lekérdezése és listázása. A programok általánosak és rugalmasak. Ez azt jelenti, hogy bármilyen rekordszerkezetű állományt el tudunk velük érni, a lekérdezési és listázási szempontok tetszőlegesen, szabadon változtatható a listák formája és tartalma.

A MISAM rendszer karbantartó segédprogramjai (REBUILD.EXE, COMPRESS.EXE, FED.EXE) az adatállományok javítását, rendszeres karbantartását végzik. A konvertáló programok egyrészt a MISAM rendszer programozását könnyítik meg (például C header file-ok generálása, lista leíró file-ok generálása), másrészt megteremtik az összeköttetést más rendszerekkel (adatbáziskezelő rendszerek, adatbázis átvitele más operációs rendszer alá).

A MISAM file rendszer általános jellemzői:

- B-tree szervezésű indexállományokon alapuló gyors elérés.
- Részekből összerakható kulcsok, a részek nem szükségszerűen teljes me-
zők.
- C típusokat modellező adattípusok (byte, int, long, float, double, char,
array of char, date).
- Az állományok elhelyezkedését a felhasználó szabadon változtathatja, a
programokat nem kell módosítani.
- Többfelhasználós felhasználást biztosító mechanizmus (zárás írásra és
olvasásra).
- Az egyidejűleg nyitva tartott állományok számát csak a memória fizikai
nagysága és az operációs rendszer konfigurációs paraméterei korlátozzák.
- Lehetőség dBASE kompatibilis file-forma használatára.

1.2. A dBASE kompatibilis file-forma

A kézikönyv dBASE III kompatibilitásra vonatkozó részei tudásbázisként feltételezik a dBASE III rendszer ismeretét. Aki a rendszert nem akarja dBASE kompatibilis formátumban használni, az erre vonatkozó részeket nyugodtan kihagyhatja.

A MISAM file-kezelő rendszer a C nyelvhez csatlakozó programfejlesztési eszköz. A használhatóság és hatékonyság érdekében

- támogatja a C nyelv által elfogadott típusokat,
- belső ábrázolású, bináris adatátvitelt valósít meg.

Azokat a MISAM állományokat, melyek a fent említett tulajdonságokat kihasználják, a továbbiakban C file-oknak nevezzük.

A MISAM rendszer alapvetően a C file-ok használatát támogatja. A rendszer programjai és keretprogramjai csak C file-okra használhatók. A rendszer maga, a könyvtár azonban lehetővé teszi a programozó számára C programból dBASE kompatibilis formátumú file-ok elérését. A dBASE kompatibilis formátumú file-okra a továbbiakban dBASE file-ként hivatkozunk.

A különbséget a C file-ok és a dBASE file-ok között az adatok tárolásának módja jelenti. A dBASE adatfile-ban minden adat karakteres formában tárolódik. Ezért a kiírás előtt és a beolvasás után a programozónak az adatrekordot konvertálnia kell. A dBASE indexfile-ban a nem karakteres kulcsértékek speciális belső formátumban tárolódnak, ezért az indexműveletek előtt a kulcsértékeket külön konvertálni kell. A konvertáláshoz egyrészt a C nyelv ad lehetőséget, másrészt a MISAM könyvtár tartalmaz megfelelő konverziós függvényeket.

A különböző típusok adatrekordbeli formáját a dBASE leírások tartalmazzák, ezért ezzel ebben a dokumentációban külön nem foglalkozunk. Az indexek konvertálásának problémájára a megfelelő könyvtári rutinok leírásánál térünk ki.

Ha a megfelelő adatkonverztálásokat a programban elvégezzük, akkor a MISAM függvények dBASE III kompatibilis file interface-t alkotnak. A dBASE file-ok használatánál azonban számolni kell a következőkkel:

- az adatrekord mérete -a numerikus és dátum típusú mezők számától függően- jelentősen megnövekedik a C file rekordjához képest
- az adatkonverzió többletmunkát igényel a programozótól, bonyolítja a programot és megnöveli annak méretét

- az adatkonverziók miatt a programok hatékonysága csökken
- a C típusok helyett a dBASE adattípusait kell használnunk
- az indexépítés lehetősége korlátozódik

A MISAM rendszer a dBASE által nyújtott lehetőségeket csak olyan mélységig biztosítja, amilyen mélységig feltétlenül szükséges a dBASE interface megteremtéséhez. A korlátozásokat a Technikai információk fejezet tartalmazza.

A dBASE file-ok elérésének lehetősége a MISAM rendszerben csak az azonos formátumot jelenti. Nem terjed ki a file-ok a dBASE-zel egyidejű használatára. A MISAM alapú programok egymás között tudnak konkurrensen működni. A dBASE fejlesztések egymás között szintén. Tilos azonban a file-ok egyidejű elérése MISAM és dBASE programokból.

A dBASE file-ok használatának lehetősége megteremti annak lehetőségét, hogy korábbi dBASE rendszerekhez C nyelven írt rendszereket illesszünk. Ez a felhasználók számára költségcsökkentő tényező, mert

- a C nyelven történő fejlesztésre való átállás esetén nem kell mindent feltétlenül újra írni C-ben
- C nyelven a megfelelő eszköztárak birtokában a dBASE-zel összevethető költségen hatékonyabb rendszereket lehet fejleszteni

Megjegyzés:

A MISAM rendszer export-import segédprogramjai lehetőséget nyújtanak C formátumú file-ok átvitelére más rendszerek alá és viszont. Alkalmas esetben a konvertálás elegendő lehet, nem kell a dBASE file-ok használata mellett dönteni.

1.3. A kézikönyv felépítése

A kézikönyv a MISAM rendszert írja le. Nem tér ki az indexelt szekvenciális file szervezés alapfogalmaira, a C nyelvben való programozás általános kérdéseire, s csak röviden foglalkozik a konkurrens adatelérés algoritmikus problémáival. A kézikönyv azon programfejlesztők számára készült, akik a C nyelvvel és az adatfeldolgozási téma alapfogalmaival tisztában vannak.

A kézikönyv szervezése a következő:

- A második fejezet a MISAM által megvalósított adatszerkezési modellt ismerteti.
- A harmadik fejezet a MISAM használatához szükséges áttekintést biztosítja.
- A negyedik fejezet a MISAM adatállományok leíró táblázatát elemzi.
- Az ötödik fejezet a MISAM file kezelő rendszer használatához szükséges általános információkat tartalmazza.
- A hatodik fejezet részletesen ismerteti a MISAM file kezelő könyvtárat.
- A hetedik fejezet az adatkezelő programokat és a segédprogramokat írja le.
- A nyolcadik fejezetben a MISAM belső világát ismertetjük, a felhasználáshoz esetleg szükséges mélységben.

2. fejezet

A MISAM modell

Egy MISAM adatállomány egy adatfile-ból és a hozzá tartozó indexfile-okból áll. A file rendszer programozását az teszi egyszerűvé, hogy a könyvtári rutinok mindig együtt kezelik az adat- és indexfile-okat, a programozónak ezzel nem kell külön foglalkoznia. Ha például eltörlünk egy adatrekordot, akkor a törlő funkciót megvalósító rutin automatikusan törli az összes indexfile-ból a megfelelő kulcsértékeket.

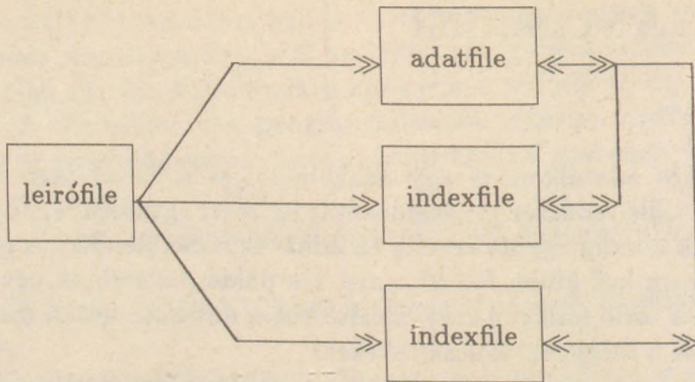
Az adat- és indexfile-ok együtt kezelése többfelhasználós környezetben is igaz: a programozónak nem kell külön gondoskodnia az adat- és indexfile-ok zárásáról, ezt a rutinok automatikusan elvégzik.

Egy MISAM adatfile-hoz akárhány indexet definiálhatunk. Egy index megadása kötelező. Az index állhat több részből, az egyes részek eltérő típusúak is lehetnek. Egy indexen belül az eltérő rendezettség nem megengedett. Az index egyes karakteres részei eltérhetnek az adatfile-ban definiált mezőktől. Például az adatrekord „főkönyvi szám” mezejére (15 elemű karaktertömb) tehetünk két indexet vagy indexrészt: az első a főkönyvi szám első karaktere szerint, a második a harmadik és negyedik karakter szerinti rendezettséget szabja meg.

A MISAM adatállományt egy leíró tábla definiálja. A tábla közönséges szövegszerkesztő programokkal létrehozható és változtatható. Minden állományhoz külön leíró tábla tartozik. A táblát tartalmazó file kiterjesztése kötelezően .FDF (File Descriptor File). Az adatállományt leíró tábla felépítését a negyedik fejezet tárgyalja.

Eltérően más, ISAM szerű file kezelő rendszerektől (például CTREE), az alapegység a MISAM rendszer programozásakor is a MISAM állomány (adat- és indexfile-ok együttese). Ez lehetővé teszi, hogy a programokból tetszőleges számú állományt nyissunk meg, egymástól függetlenül. A nyitott állományok számának a szabad memóriaterület és az operációs rendszer konfigurálása szab határt.

Az alábbi ábra a leíró file, az adatfile és az indexfile-ok közötti kapcsolatot szemlélteti.



A leíró file határozza meg az adat- és indexfile-ok felépítését. Ezek között a hatás egyirányú. Az adat- és indexfile-ok kölcsönösen hatnak egymásra: az adatrekord tartalma határozza meg az egyes kulcsértékeket; az adatrekordokat az indexeken keresztül érhetjük el. Az indexek egymásra nincsenek hatással, kezelésük párhuzamosan történik.

3. fejezet

A MISAM használata

A MISAM felhasználása általában a következő lépésekben történik:

1. Megtervezzük a MISAM állomány(oka)t.
2. Editorral elkészítjük a .FDF leíró file-okat.
3. Létrehozuk az üres állományokat.
4. Feltöltjük és használjuk az állományokat. Ehhez a MISAM adatkezelő programokat (felvivő, lekérdező, módosító, listázó) használhatjuk, de az összetettebb feldolgozások önálló C programokat igényelnek. A C programok a MISAM könyvtár funkcióinak hívásával kezelik a MISAM állományokat.
5. A MISAM állományt bármikor ellenőrizhetjük és módosíthatjuk a FED file editor segítségével. Karbantartó jellegű tevékenységekre az index file újraépítő program (REBUILD) és a tömörítő program (COMPRESS) áll rendelkezésre.
6. Feltöltéshez vagy továbbadáshoz dBASE vagy ASCII adatfile-okat is használhatunk. Az erre a célra alkalmas import és export programok a TODBASE, FRDBASE, TODOS és FROMDOS.
7. Esetenként szükség lehet C programba beépíthető adatrekord definícióra illetve a listázó igényel listaleíró állományokat. Ezeket a .FDF-ből az FDFTOH, FDFTORPD és FDFTORPF segédprogramok állítják elő.

A fejezet további része részletesebben tárgyalja a felhasználást érintő kérdéseket. A teljes részletességű tárgyalást a 4., 5., 6. fejezetek tartalmazzák, de ott már software elemekre széttagoltan történik a tárgyalás. Emiatt ott a logikai kapcsolódás már nem látszik tisztán.

3.1. Az állományleíró táblázat

A MISAM állományt egy .FDF kiterjesztésű file írja le. Célszerű, egy alkalmazás összes .FDF file-ját egy katalógusban helyezni el.

A leíró file-ban megadjuk:

- Az adatfile nevét és helyét a lemezen,
- Az indexfile(-ok) helyét és nevét,
- Az indexek felépítését,
- Az adatrekord mezőinek nevét, és jellemzőit.

A leíró file formájának részletes ismertetését a 4. fejezetben találhatjuk.

A .FDF file-t tetszés szerinti editorral létrehozhatjuk. Hibás .FDF leírás komoly tesztelési gondokat tud okozni, ezért célszerű a CHKFDF segédprogrammal meggyőződni arról, hogy az editált leírás helyes.

A CHKFDF program beolvassa a .FDF file-t, s megvizsgálja, hogy:

- léteznek-e a helynek kijelölt könyvtárak,
- léteznek-e a kijelölt file-ok,
- rendben van-e az indexleírás,
- rendben van-e az adatleírás (jók-e az offset-ek, csupa létező típusra hivatkozunk-e, stb.).

Az esetleges hibákat a program kijelzi, s ezen túlmenően képernyőn kiírja a .FDF-ből összeállított állományjellemzőket. Ezeket célszerű vizuálisan ellenőrizni.

3.2. Állományok létrehozása

Az adatfeldolgozási feladatok programozásakor bevált gyakorlat az, hogy nem a feldolgozó programokból hozzuk létre az adatállományokat, hanem külön segédprogramot használunk erre.

A CREAISAM segédprogram feladata, hogy a megadott .FDF file alapján üres MISAM állományt hozzon létre. Ha a leírás hibás, akkor különösebb hibaelemzés nélkül hibajelzéssel leáll a program. Valószínűleg elfelejtette futtatni a CHKFDF-et!

3.3. Adatkezelő könyvtári függvények

A MISAM állományok igényesebb kezeléséhez, pl. a körültekintően ellenőrzött adatbevitelhez, a kalkulatív feldolgozásokhoz C programok szükségesek. A C programokból az állományt a MISAM C könyvtár funkcióin keresztül kezeljük.

A könyvtári funkciók:

név	jelentés
isnetwork	vizsgálja, van-e NOVELL hálózat
iscreat	létrehozza a MISAM állományt
isopen	az állomány nyitása
isclose	az állomány zárása
isfirst	file elejére pozícionálás
islast	file végére pozícionálás
isread	pozícionálás és (vagy) beolvasás
iswrite	új rekord hozzáfűzése az állományhoz
isdelete	az aktuális rekord törlése
isupdate	az aktuális rekord felülírása
isselind	a kiválasztott index beállítása
isrlock	írás elleni védelem kérése
isrunlock	az írás elleni védelem feloldása
iswlock	írás - olvasás elleni védelem kérése
iswunlock	az írás - olvasás elleni védelem feloldása
isbuild	indexfile építése
isgetfdf	a .FDF file beolvasása belső táblákba
isdatetokey	karakteres dátum konverziója double-re
iskeytodate	double konvertálása karakteres dátummá

A függvények részletes ismertetését a hatodik fejezet tartalmazza.

3.4. Interaktív file editor

Az interaktív file editor (FED) segítségével tetszőleges MISAM állományhoz tudunk hozzáférni: rekordokat bevinni, lekérdezni, törölni, módosítani. Éles alkalmazásban a használata nagy körültekintést igényel, a programfejlesztés során azonban nagyon gyakran hasznos segédprogram.

A FED segítségével tetszőleges (teszt)adatokat vihetünk be az állományokba (például hibásakat is a hibakezelés ellenőrzésére), ellenőrizhetjük, hogy egy program helyesen módosítja-e az állományokat, stb.

A FED segédprogram nem csak az állományok vizsgálatára alkalmas, hanem az egyes könyvtári rutinok kipróbálására is. Így nem kell tesztprogramokat írni a különböző szituációk tanulmányozásához.

A FED kezelése egyszerű és kényelmes: az ablak technikát használva ajánlja fel a választható tételeket, gyors (kezdőbetű szerinti) kiválasztást és teljes képernyős adatbevitelt alkalmaz.

3.5. A standard adatkezelő programok

A MISAM rendszer standard adatkezelő programjaival programozás nélkül lefedhetünk egy egyszerű adatfeldolgozási ciklust: adatbevitel, adatok lekérdezése, törlése, módosítása, listázása. Ezek általánosan megírt programok:

- bármilyen szerkezetű file-ra használhatók
- működésük a kezelői párbeszéd során és a paraméterezhetőség miatt rugalmasan változtatható (például elérési szempontok megadása, listaformátumok összeállítása).

Az általánosság miatt a programok nem tartalmaznak semmiféle ellenőrzést az adatok helyességére vonatkozólag, velük nem automatizálhatók a kalkulatív feladatok és az állományok közötti kölcsönhatások. Emiatt csak egyszerűbb adatfeldolgozás esetén (például törzsállományok karbantartása) alkalmazhatók éles programként.

A standard programok segítségével programozás nélkül felállíthatjuk egy adatfeldolgozási rendszer nulladik verzióját, prototípusát. Ennek a prototípusnak a segítségével tesztelhetjük a rendszert szervezési szempontból, például helyes az adatkapcsolódás az egyes állományok között. Így kiszűrhetünk olyan rejtett hibákat is, melyek későbbi felbukkanása esetleg az egész rendszer újraírását jelentheti.

A standard programok segítségével „előredolgozhatunk”. A fejlesztés során a nulladik verzióval elkészíthetjük a megfelelő tesztállományokat. Így nem feltétlenül kell az egyes részrendszerek fejlesztésének összhangban lenni, azok külön-külön is fejleszthetők.

3.5.1. Standard adatfelvivő program

Az ENTRY program segítségével új rekordokat vihetünk be egy állományba. Az adatbevitel rekordorientált, egy ún. standard adatbeviteli képernyőn keresztül történik. A képernyő felépítését a .FDF file-ban megadott meződefiniciók szabják meg.

A standard adatbeviteli képernyő leírása a Függelékben található.

3.5.2. Standard lekérdező és módosító program

A QUERY program az .FDF leíró file alapján hármass funkciókat valósít meg:

- adatállományok gyors, interaktív lekérése a képernyőre,
- rekordok módosítása,
- rekordok törlése.

A rekordok eléréséhez tetszőleges elérési szempontot adhatunk meg. A rekordokat elérhetjük

- valamelyik, az .FDF file-ban definiált index alapján,
- új index definiálásával,
- minta megadásával.

A leggyorsabb módszer az előre definiált indexek használata. Új index definiálásakor számolni kell az indexfile felépítéséhez szükséges idővel. A minta szerinti elérés egy lassúbb szekvenciális feldolgozás. Minta szerint nem kapunk sorrendiséget, viszont tetszőleges rekordhalmaz kijelölhető.

A program a megjelenítéshez és bevitelhez a standard adatbeviteli képernyőt használja. A file-ban az elérési szempont alapján tetszős szerint tudunk előre-hátra mozogni.

3.5.3. Standard listázó

A standard listázó program, a LIST MISAM állományok listázására alkalmas. A listázandó rekordok kiválasztására a LIST ugyanazokat a lehetőségeket kínálja mint a QUERY program, kibővítve intervallum megadásának lehetőségével.

A listázó segítségével tetszőleges formájú és tartalmú listákat készíthetünk, mivel a program táblázatok alapján dolgozik. A táblázatok alapja a MISAM leíró .FDF file. A táblázatgeneráló segédprogramok segítségével (FDFTORPD, FDFTORPF) automatikusan generálhatjuk a nyers listaleíró táblákat, majd bármilyen szövegszerkesztővel kialakíthatjuk a táblázatok (és így a lista) végső formáját.

A segédprogramok és táblázatok leírását a hatodik fejezet, valamint a függelék tartalmazza.

A lista konstans részei (például a fejléc) tartalmazhatnak magyar ékezetes betűket is. Mivel ezek kinyomtatása nyomtató függő, a listázó ún. szűrőfile-okon keresztül ír a nyomtatóra. A szűrőfile-ok tulajdonképpen konverziós táblák, melyek segítségével tetszőleges típusú nyomtatón tudunk ékezetes listát készíteni.

3.6. Export és import segédprogramok

A MISAM rendszerhez tartozó export és import segédprogramok lehetőséget adnak arra, hogy a MISAM állományokat tetszőleges (file kezelő és/vagy operációs) rendszer alá átvigyük, illetve onnan adatokat hozzunk. A TODOS és FROMDOS segédprogramok a C file-okat konvertálják standard ASCII formára és vissza. (A standard ASCII file formátum leírása a függelékben található.) A konverzió alapja a MISAM leíró .FDF file. Ha tehát ASCII file-t konvertálunk C formára, akkor előbb el kell készítenünk a megfelelő .FDF file-t.

Az ASCII file konverzió önmagában elég a bármely rendszerrel történő kommunikációra. Mivel azonban az adatfeldolgozási rendszerek igen nagy részét a legelterjedtebb keretrendszer, a dBASE segítségével fejlesztették (és fejlesztik), a MISAM rendszer külön konverziós segédprogramokat tartalmaz, melyek segítségével közvetlenül konvertálhatók a C file-ok dBASE file-ra és viszont (TODBASE, FRDBASE segédprogramok). E konverzió jelentősége abban rejlik, hogy korábban dBASE alatt fejlesztett rendszerekhez – megfelelő munkaszervezés mellett – viszonylag könnyen illeszthetünk C nyelven fejlesztett rendszereket.

3.7. Karbantartó segédprogramok

3.7.1. Indexfile-ok újraépítése

A REBUILD segédprogrammal az adatfile alapján a MISAM állományhoz tartozó valamenyi indexfile-t újraépíthetjük. Így:

- Az indexfile-ok megsérülése esetén könnyen vissza tudjuk állítani a sérülés előtti állapotot.
- Mentéskor elegendő az adatfile-t menteni, visszatöltés után fel tudjuk építeni az adatfile-nak megfelelő indexfile-okat. Nagy állományok esetén jelentős hely- és időmegtakarítás csak az adatfile mentése.

3.7.2. Tömörítés

A tömörítés funkció (COMPRESS segédprogram) az állományok karbantartásának fontos része, rendszeresen ismétlődő feladat. A törölt rekordok helyét a MISAM nem allokája újra, így azok továbbra is foglalják a helyet az állományban. Emiatt sok törölt rekord esetén az állomány jóval nagyobb helyet foglalhat el a szükségesnél.

A COMPRESS segédprogram használatával eltüntethetjük a „lyukakat” az állományból, így módon javítva a helykihasználást. Ha nem tömörítjük az állományokat, akkor azok mérete nő, amíg a lemez be nem telik. A tömörítés gyakorisága az állomány használatától függ, de mindenképpen célszerű rendszeres időközönként elvégezni.

3.8. Táblázatgeneráló programok

3.8.1. .FDF táblából include (.H) tábla

A C programokból az adatfile rekordjait általában struktúraként kezeljük. A struktúrát leíró include file-t az FDFTOH programmal automatikusan létre lehet hozatni.

3.8.2. .FDF táblából .RPD tábla

A lista generátorra alapozott MISAM listázó egy .RPD típusú file-ból várja az elemi sor leírását. Az FDFTORPD program ezt a file-t automatikusan létrehozza. Felhasználja ehhez az FDF-ben meglévő, de a file rendszer szempontjából egyébként felesleges mezőszélesség és decimális pozíció adatokat.

3.8.3. .FDF táblából .RPF tábla

A listázó a lista formátumát egy editálható .RPF file-ból veszi. Kiindulásként az FDFTORPF programmal létrehozhatunk egy primitív listaleíró formátumot. Ebben fejlécnek az adatmezők nevei vannak megadva, az elemi sor kiírásában minden változó szerepel. Sorvég ellenőrzés és illesztés nincs, összegző sorok sincsenek.

A listázó a lista formátumát egy editálható .RPF file-ból veszi. Kiindulásként az FDFTORPF programmal létrehozhatunk egy primitív listaleíró formátumot. Ebben fejlécnek az adatmezők nevei vannak megadva, az elemi sor kiírásában minden változó szerepel. Sorvég ellenőrzés és illesztés nincs, összegző sorok sincsenek.

Ezt a file-t tovább módosíthatjuk, de ismernünk kell a lista leírás szabályait. Egy rövid ismertetés a Függelékben található.

3.9. A kézikönyvben leírt példa

Az egyes software elemek részletes ismertetése során a kézikönyvben végig ugyanazt a példát használjuk. A példa egy könyvelési rendszer főkönyvi törzsállományát írja le. A törzsállományt alkotó file-okról (adat- és indexfile) feltételezzük, hogy a "C:" meghajtó gyökérkönyvtárából nyíló „MIKRI-DAT” könyvtárban vannak. A törzsállomány egy adat- és egy indexfile-ból áll. Egy adatrekord 11 mezőt tartalmaz és 107 byte hosszú. A kulcs egy darabból áll, karakter típusú és 15 karakter hosszú. Az index a törzsrekord első mezője.

4. fejezet

Adatállományt leíró táblázat (.FDF)

Ebben a fejezetben a MISAM állományokat leíró .FDF file felépítését ismertetjük minden részletre kiterjedően.

A leíró file megadja:

- az adatfile nevét és helyét,
- az indexfile-ok neveit és helyeit,
- az indexek felépítését,
- az adatrekord felépítését.

A leírás editálható formájú, tetszés szerinti editorral létrehozható. Az adatok sor-orientáltak, azaz egy információ egy sorban helyezkedik el. Az információk sorrendje szigorúan kötött.

A fő információs egységek:

- adatfile leírás : 3 sor,
- indexfile-ok száma: 1 sor
- indexfile leírás: 3 sor,
- index leírás: 1 + n sor, ahol n az indexrészek száma.

Az indexfile leírást azonnal követi az index felépítésének a leírása. Az új indexfile csak ez után következhet.

- Adatmezők száma: 1 sor
- Adatmezők leírása: mezőnként egy sor.

Az információs egységeken belüli elemi információk sorrendje is kötött. Az elválasztó jel az üres köz vagy a tabulátor.

A leírást hashmark (#) kezdetű magyarázó sorokkal láthatjuk el.

Az egyes egységek (file-ok, indexek, mezők) azonosítását és értelmezését (a megjegyzés sorokon kívül) a programból is elérhető hivatkozási nevek és magyarázatok segítik. Ezek helyes megválasztása áttekinthetőbbé teszi a programot és/vagy érthetőbb üzeneteket tesz lehetővé.

A MISAM állomány definiálásához ismernünk kell a rendszer által megszabott korlátokat. Ezek a hetedik fejezetben, a technikai információk között találhatók.

4.1. Adattípusok és ábrázolás

Mielőtt a MISAM állományt leíró táblázat részletes ismertetését elkezdjük, szükséges a rekordmezők, valamint az indexek definiálásához a lehetséges adattípusok és azok tárolásának tárgyalása.

C file esetén a következő adattípusokat használhatjuk:

- byte
- unsigned byte
- integer
- unsigned integer
- long
- float
- double
- character
- character array
- date

Az utolsó két típus kivételével a típusok a C nyelv típusai, külön magyarázatra nem szorulnak. Lényeges megjegyezni, hogy a rekordhosszba mindig a C belső ábrázolásának megfelelő méret számít bele, mivel a file kezelő rendszer belső ábrázolásban várja az adatokat. Például a double típus hossza nyolc byte.

A character array típus karaktersorozatot jelent, de ez nem C string, nem kell NULL karakterrel zárni.

A „date” típus a MISAM rendszer belső ábrázolású dátuma. A dátum típus tulajdonképpen C-ben egy long érték. A karakter sorozat dátum formából (például 88/08/08) belső ábrázolású long érték konvertálásához a DATE.H header file-ban definiált DATE struktúrát használhatjuk. (A header file ismertetése a könyvtár leírásánál, a következő fejezetben található.)

4.2. Adatfile leírás

Az adatfile-ról meg kell adni:

- A hivatkozási nevet.

Az adatfile hivatkozási nevét a MISAM rendszer nem használja, de a programozó számára információtartalommal bírhat. Bevált gyakorlat, hogy az adatfile neve, a .FDF file alapneve (és a MISAM állomány neve) ugyanaz.

- A helyet, teljes ösvény megadásával.

– Jelentést leíró magyarázó sort.

Példa:

```
#####
# A FOKONYVI TORZSALLOMANY DEFINICIOS LEIRASA
#
# Bazis file definicioja
#
FOKONYV
c:\mikridat\fo.dbf
A fokonyvi torzsallomany
```

A hashmarkkal (#) kezdődő sorok magyarázó sorok, ténylegesen nem vesznek részt a MISAM file leírásában.

Az első nem hashmark kezdetű sor az adatfile hivatkozási neve. A következő sor az adatfile neve a lemezen, a teljes ösvénnyel együtt. Utána a jelentést leíró magyarázó sor következik.

4.3. Index definíciók

Az indexállományok leírása az indexfile-ok számának megadásával kezdődik. Az erre szánt külön sor után következnek sorra az indexállományok leírásai, minden indexfile leírás után közvetlenül jön a hozzátartozó indexleírás.

Az indexfile leírás hasonló az adatfile leíráshoz. Meg kell adni:

- A hivatkozási nevet. A programokból az indexekre a hivatkozási névvel hivatkozhatunk.
- A helyet, teljes ösvény megadásával.
- Jelentést leíró magyarázó sort.

Az indexleírás az indexrészek számának megadásával kezdődik, ezt követi annyi sor, ahány rész van. Minden rész leírása három elemi információból áll:

- az indexrész kezdeti pozíciója az adatrekordban (0-tól számozva),
- az indexrész hossza,
- típusa.

A típusmegadás kódoltan történik, egy típusjelző karaktert kell megadni. Típus kódok C file esetén:

B : byte
 S : unsigned byte
 I : integer
 U : unsigned integer
 L : long
 M : date
 F : float
 D : double
 C : character
 A : character array (nem string, nincs lezáró jell)

Típuskódok dBASE file esetén:

C : character
 D : date
 N : numeric

Példa:

```

#####
# Index definicio
#
1
FOKONYV_I1
c:\mikridat\fo001.ndx
Kulcs a fokonyvi szam szerint
1
0      15      C
  
```

A MISAM file-hoz egy (1) indexfile tartozik. Az indexfile hivatkozási neve „FOKONYV_I1”. Az index egyetlen darabból áll. Ez az adatrekord 0. pozícióján kezdődik és 15 karakter hosszú. Az indexdarab típusa karakter.

4.4. Az adatmezők megadása

Az első információs sor az adatmezők számát adja. Ezt követi mezőnként három sor.

A mezőleírás elemei:

- mezőnév
- a mező jelentését magyarázó sor,
- a mezőt leíró sor. Megadja:
 - a relatív kezdőpozíciót (offset-et),
 - a hosszát byte-ban,
 - a dimenziót (egyelőre mindig 1!)
 - a típust, egy karakterrel kódoltan. A kódokat lásd fent.
 - az ábrázolást.

A mezőt leíró sor utolsó két eleme az ábrázolás: milyen szélességben kívánjuk a mezőt ábrázolni és abból hány tizedes pozíció legyen. Ezt az információt a standard adatkezelő, illetve a konverziós programok használják a mező megjelenítéséhez, konvertálásához.

Példa:

```
#####
# Adatrekord definicio
#
11
fokonyvi_szam
Fokonyvi számlaszám
0      15      1      A      15      0
megnevezes
Megnevezes
15     30      1      A      30      0
kulso_merleg
Kulso merleg sorszám
45     3       1      A      3       0
belso_merleg
Belso merleg sorszám
48     3       1      A      3       0
nyito_egyenleg
Nyito egyenleg
51     8       1      D     12      2
f_tartozik
Az éves kumulatív tartozik forgalom
59     8       1      D     14      2
f_kovetel
Az éves kumulatív követel forgalom
67     8       1      D     14      2
```

egyenleg_1					
Az utolsó év egyenlege					
75	8	1	D	14	2
egyenleg_2					
Az utolsóelőtti év egyenlege					
83	8	1	D	14	2
egyenleg_3					
Az utolsóelőtti-előtti év egyenlege					
91	8	1	D	14	2
f_tervezett					
A tervezett forgalom erre az évre					
99	8	1	D	14	2

Az adatrekord 11 mezőből áll. Az első négy mező karakteres szöveg. A rekordbeli hossz és az ábrázolási hosszuk megegyezik. A többi mező double típusú mező. A rekordban elfoglalt hely 8 byte, de az ábrázolás: 14 karakter szélességben, ebből 2 tizedesjegy.

4.5. A teljes leíró C file esetén

```
#####
# A FOKONYVI TORZSALLOMANY DEFINICIOS LEIRASA
#
# Basis file definicioja
#
FOKONYV
c:\mikridat\fo.dbf
A fokonyvi torzsallomany
#
#####
# Index definicio
#
1
FOKONYV_I1
c:\mikridat\fo001.ndx
Kulcs a fokonyvi szam szerint
1
0      15      C
#
#####
# Adatrekord definicio
#
```


11					
fokonyvi_szam					
Fokonyvi számlaszám					
0	15	1	A	15	0
megnevezes					
Megnevezes					
15	30	1	A	30	0
kulso_merleg					
Kulso merleg sorszám					
45	3	1	A	3	0
belso_merleg					
Belso merleg sorszám					
48	3	1	A	3	0
nyito_egyenleg					
Nyito egyenleg					
51	8	1	D	12	2
f_tartozik					
Az éves kumulatív tartozik forgalom					
59	8	1	D	14	2
f_kovetel					
Az éves kumulatív követel forgalom					
67	8	1	D	14	2
egyenleg_1					
Az utolsó év egyenlege					
75	8	1	D	14	2
egyenleg_2					
Az utolsóelőtti év egyenlege					
83	8	1	D	14	2
egyenleg_3					
Az utolsóelőtti-előtti év egyenlege					
91	8	1	D	14	2
f_tervezett					
A tervezett forgalom erre az évre					
99	8	1	D	14	2

4.6. A teljes leíró dBASE file esetén

Az állomány definiálásakor dBASE file esetén a C file-hoz képest korlátozottak a lehetőségek (adattípusok, indexek összeállítás, stb). Az erre vonatkozó információkat a hetedik fejezetben, a Korlátozások között találhatjuk.

```
#####
# A FOKONYVI TORZSALLOMANY DEFINICIOS LEIRASA
#
# Bazis file definicioja
#
FOKONYV
c:\mikridat\fo.dbf
A fokonyvi torzsallomany
#
#####
# Index definicio
#
1
FOKONYV_I1
c:\mikridat\fo001.ndx
fokonyvi_szam
1
0      15      C
#
#####
# Adatrekord definicio
#
11
fokonyvi_szam
Fokonyvi számlaszam
0      15      1      C      15      0
megnevezes
Megnevezes
15      30      1      C      30      0
kulso_merleg
Kulso merleg sorszam
45      3      1      C      3      0
belso_merleg
Belso merleg sorszam
48      3      1      C      3      0
nyito_egyenleg
Nyito egyenleg
51      12      1      N      12      2
f_tartozik
Az eves kumulativ tartozik forgalom
63      14      1      N      14      2
f_kovetel
```


Az éves kumulatív követel forgalom					
77	14	1	N	14	2
egyenleg_1					
Az utolsó év egyenlege					
91	14	1	N	14	2
egyenleg_2					
Az utolsóelőtti év egyenlege					
105	14	1	N	14	2
egyenleg_3					
Az utolsóelőtti-előtti év egyenlege					
119	14	1	N	14	2
f_tervezett					
A tervezett forgalom erre az évre					
133	14	1	N	14	2

Az index leírásában a magyarázó sor a dBASE kulcskifejezést tartalmazza. A numerikus mezőket „N” típusúnak definiáltuk és a rekordban akkora helyet foglalnak el, amekkora az ábrázolásuk szélessége. Emiatt az adatrekord mérete 107 karakterről 147 karakterre növekedett.

5. fejezet

A MISAM programozása

A fejezet a MISAM függvények használatához szükséges MISAM.H header file-t ismerteti és a MISAM programozásához szükséges fogalmakat magyarázza meg. A disztributív anyag még egy további header file-t tartalmaz, a DATE.H nevű-t. Ebben azokat a struktúrákat találhatjuk, melyek a belső ábrázolású dátum használatát könnyítik meg.

5.1. A MISAM.H header file

A könyvtári rutinok programozásához a forrásmodulokba be kell hívni (#include) kell a MISAM.H header file-t, amely a MISAM programozásához szükséges konstans, típus és struktúra definíciókat tartalmazza.

```
/*  
 *      Operation modes  
 */  
#define DB_COMPAT      0x00  
#define MSC_MODE       0x80
```

```

/*
 *      Function codes for isread call
 */
#define CURRENT      0      /* actual record          */
#define FIRST        1      /* first record           */
#define LAST         2      /* last record            */
#define NEXT         3      /* next rekord            */
#define PREVIOUS     4      /* previous record        */
#define FIND         5      /* find a key value       */
#define NEXTSAME     6      /* next with the same key value */
#define PREVSAME     7      /* prev with the same key value */

#define NOREAD      128      /* only set position, don't read*/

/*
 *      Constants to check the lock status
 */
#define READ_LOCK    0x01      /* read lock set          */
#define WRITE_LOCK   0x02      /* write lock set         */

/*
 *      Type declarations
 */
struct ISINFO
/* structure to
/* ask informations */
{
    int      openflag;
    int      formatflag;
    int      lockstat;
    long     recnum;
    char     ixname[16];
};
/* number of records */
/* name of pref. index */

typedef struct ISINFO INFO;

```



```

struct KEYPARTS          /* Description of key parts */
{
    int      start;          /* start pos. of part */
    int      length;        /* length of key part */
    char     type;          /* type of the key part */
};

```

```
typedef struct KEYPARTS KPARTS;
```

```

struct KEYDESC           /* Description of keys */
{
    char     iname[16];      /* index file ref. name */
    char     ifullname[80]; /* index file name */
    char     icomment[80];  /* description */
    char     *ixf;          /* index file descriptor */
    int      nparts;        /* number of keyparts */
    KPARTS   **iparts;      /* key part description */
};

```

```
typedef struct KEYDESC KDESC;
```

```

struct FLDDDESC          /* Description of a fields */
{
    char     fdname[16];    /* field name */
    char     fdcomment[80]; /* field description */
    int      fdstart;       /* starting position */
    int      fdlength;      /* length of field */
    int      fddim;         /* array element number */
    char     fdtype;        /* field type */
    char     fdwidth;       /* field format width */
    char     fddec;         /* field format dec. pos */
    /*
     *      For internal use only
     */
    int      fdbufpos;
    char     fdrow;
    char     fdcolfrom;
    char     fdcolto;
};

```

```
typedef struct FLDDDESC FLDDDESC;
```

```

struct FILEDESC          /* Description of MISAM files */
{
    char     dname[16];     /* data file ref. name */
    char     dfullname[80]; /* data file name */
    char     dcomment[80];  /* description */
};

```

```
int      nfields;          /* number of fields      */
FLDDDESC **fields;         /* description of fields*/
char     *df;              /* data file description*/
int      nkeys;            /* number of keys        */
KDESC    **keys;           /* description of keys   */
int      kpref;            /* the preferred key     */
long     actrec;           /* last accessed record  */
int      status;           /* status flag           */
};
typedef struct FILEDESC FDESC;
```



```

/*
 *      Return codes
 */
#define SUCCESS      0      /* successfull completion      */
#define E_MEMORY     100    /* not enough free memory      */
#define E_IO         101    /* read/write error            */
#define E_BUSY       102    /* the file is locked          */
#define E_BADFORM    103    /* bad file format, damaged file*/
#define E_BADREC     104    /* bad actual record position  */
#define E_BADARG     105    /* invalid parameter           */
#define E_EMPTY      106    /* the file is empty           */
#define E_TOPKEY     107    /* reached the beginning of file*/
#define E_ENDKEY     108    /* reached the end of the file */
#define E_GREATER    109    /* only greter key value found */
#define E_CHANGED    110    /* the data record changed     */
#define E_NETWORK    111    /* network error               */

```

```

#ifdef LINT_ARGS

```

```

/*global*/ int isclose(struct FILEDESC *);
/*global*/ int iswrite(struct FILEDESC *,char *);
/*global*/ int isbuild(struct FILEDESC *,char *);
/*global*/ int isdelete(struct FILEDESC *,char *);
/*global*/ int isfirst(struct FILEDESC *,char *);
/*global*/ int islast(struct FILEDESC *,char *);
/*global*/ int isinfo(struct FILEDESC *,struct ISINFO *);
/*global*/ int isselind(struct FILEDESC *,char *);
/*global*/ int isupdate(struct FILEDESC *,char *,char *);
/*global*/ int isnetwork(void);
/*global*/ int isrlock(struct FILEDESC *);
/*global*/ int isrunlock(struct FILEDESC *);
/*global*/ int iswlock(struct FILEDESC *);
/*global*/ int iswunlock(struct FILEDESC *);
/*global*/ int isopen(struct FILEDESC *);
/*global*/ int isdatetokey(char *,double *);
/*global*/ int iskeytodate(double *,char *);
/*global*/ int isgetfdf(char *,struct FILEDESC * *);
/*global*/ int iscreat(struct FILEDESC *,int );
/*global*/ int isread(struct FILEDESC *,char *,
                      char *,char *,int );

```

```

#else
/*global*/ int isclose();
/*global*/ int iswrite();
/*global*/ int isbuild();

```



```

/*global*/ int isdelete();
/*global*/ int isfirst();
/*global*/ int islast();
/*global*/ int isinfo();
/*global*/ int isselind();
/*global*/ int isupdate();
/*global*/ int isnetwork();
/*global*/ int isrlock();
/*global*/ int isrunlock();
/*global*/ int iswlock();
/*global*/ int iswunlock();
/*global*/ int isopen();
/*global*/ int isdatetokey();
/*global*/ int iskeyto date();
/*global*/ int isgetfdf();
/*global*/ int iscreat();
/*global*/ int isread();
#endif

```

Megjegyzés:

A belső használatra magyarázatú struktúra-elemek a programozás szempontjából érdektelenek, kizárólag a MISAM rendszer belső adminisztrációját szolgálják.

5.2. A DATE.H header file

```

struct DATE
{
    unsigned          year;           /* 0-          */
    unsigned char     month;          /* 1-12        */
    unsigned char     day;            /* 1-31        */
};

union UDATE
{
    struct DATE      s;               /* access of elements */
    long             l;               /* quick access       */
};

```


5.3. Az aktuális pozíció

Az index szerinti szekvenciális mozgás, valamint az adatrekordra vonatkozó műveletek megértéséhez ismerni kell az aktuális pozíció fogalmát. Az aktuális kulcspozíció valamely indexfile-ban az a kulcsérték, amelyre relatívan a következő kulcs, előző kulcs műveleteket értelmezzük. Az aktuális kulcsérték nem feltétlenül létezik az indexfile-ban (például törlés után). Minden indexfile-hoz külön aktuális kulcspozíció tartozik.

Az aktuális kulcspozíciót csak az `isread()` függvényhívás változtatja meg. Ha tehát egy rekordot felülírunk vagy törölünk, az aktuális kulcspozíció ettől nem változik meg.

Példa:

Tételezzük fel, hogy egy indexfile-ban a kulcsértékek a következők:

1, 2, 3, 4, 5

Az aktuális kulcspozíció a 3. Ha az adatrekordot úgy módosítjuk, hogy az adatrekordból nyert kulcsérték 3 helyett 6, akkor a módosítás után az indexfile tartalma a következő:

1, 2, 4, 5, 6

A következő kulcs funkció azonban a 4 értéket fogja elővenni, mert a módosítás nem változtatta az aktuális kulcspozíciót: a 3 kulcsérték után a következő érték a 4. Ugyanez vonatkozik a törlésre is.

5.4. A file eleje, file vége pozíció

A file eleje, file vége pozíciók az aktuális kulcspozíció speciális értékei.

A file eleje pozíció valamelyik indexfile-ban az első index előtti pozíciót jelenti. File eleje pozícióban kiadott következő kulcs művelet az első kulcsértéket adja vissza.

A file vége pozíció valamelyik indexfile-ban az utolsó index utáni pozíciót jelenti. File vége pozícióban az előző kulcs művelet az utolsó kulcsértéket adja vissza.

5.5. A kiválasztott index

Egy MISAM file egy adatfile-ból és tetszőleges számú (de legalább egy) indexfile-ból áll. Ezek közül egyet (tetszés szerint változtatható, hogy melyiket) a programban kiválasztott index-nek deklarálhatunk.

A kiválasztott index megadásának értelme a következő:

- Ha a függvényhívásoknál az index hivatkozási név paramétert NULL-nak adjuk meg, akkor a függvény a kiválasztott indexet használja.
- A törlés és felülírás funkciók mindig a kiválasztott index szerinti aktuális kulcspozíció által meghatározott rekordra vonatkoznak.

Kiválasztott index mindig van. A MISAM file nyitása után a kiválasztott index a .FDF leíró file-ban elsőként leírt index.

A kiválasztott indexet a MISAM file leíró struktúra (lásd a MISAM.H header file-t) *kpref* eleme adminisztrálja: értéke az index sorszáma, nullától kezdve. (Az indexek sorrendje a .FDF file-ban megadott sorrend!)

Példa:

Hivatkozás a kiválasztott index magyarázatára:

```

struct FILEDESC *fdp; /* mutató a MISAM file leíró blokkra*/
struct KEYDESC *kdesc; /* mutató az indexleíró blokkra */

.
.
.

kdesc = *(fdp->keys+fdp->kpref);
printf("%s\n",kdesc->icomment);
.
.
.

```


5.6. Konkurens programozás

A MISAM file kezelő rendszer lehetővé teszi NOVELL hálózat alá többfelhasználós rendszerek fejlesztését. A MISAM konkurens programozása a programozótól minimális többletmunkát igényel, viszont a hálózatosra írt programok egyfelhasználós környezetben is futtathatók.

A konkurens programozás szemprontjából a függvényhívásokat két csoportra osztjuk:

– olvasási műveletek

isopen()

isread()

isinfo()

– írási műveletek

iscreat()

isclose()

isdelete()

isupdate()

isbuild()

Az írási műveletek előtt írási zárást, az olvasási műveletek előtt olvasási (vagy írási) zárást kell kiadni. A zárásek file szintű zárásek, vagyis az egész MISAM állományt fogják. Emiatt az algoritmusokat úgy kell szervezni, hogy a lehető legrövidebb ideig legyen szükség egy állomány záráására. A MISAM rendszerben rekord szintű zárák nincs, a felülírások kivédésére más technikát ajánl. A zárák és a zárák feloldása (két kivételtől, az isopen() és isclose() függvényektől eltekintve) nem automatikus. A zárásek helyes kezelése a programozó feladata.

5.6.1. A file műveletek szinkronizálása

Az írási zárák (write lock) mindenfajta hozzáférést tilt a többi állomás számára. Az írási zárák az iswlock() hívástól az iswunlock() hívásig van érvényben. Az olvasási zárák (read lock) az írási műveleteket tiltja meg a többi állomás számára, vagyis olvasási zárák esetén egyszerre több állomás

olvashatja ugyanazt az állományt. Az olvasási zárás az `isrlock()` hívástól az `isrunlock()` hívásig van érvényben.

A zárásokkal tulajdonképpen „tranzakciókat” definiálhatunk (de nem a szó igazi értelmében). Ha több rekord felülírása és/vagy törlése szorosan összehasonlítandó és megkövetelt, hogy más munkaállomás ne férjen hozzá az állományhoz, amíg ezek mindegyikét el nem végeztük, akkor nem külön-külön kell a műveletekhez zárni. Az első művelet előtt ki kell adni az írási zárást és az utolsó művelet után kell kiadni az írási zárás feloldását.

Példa:

```

írási zárás beállítása
.
.
első rekord felülírása
.
.
utolsó rekord felülírása
.
.
írási zárás feloldása
.
.

```

5.6.2. Rekordok módosítása, törlése

A rekordok módosításával, törlésével kapcsolatban a legfontosabb annak kivédése, hogy egyik munkaállomás felülírja a másik által elvégzett módosításokat. A MISAM rendszerben erre a rekord jellegétől függően két út kínálkozik.

A rekordot törölő és módosító rutinok megadják a lehetőséget annak figyelésére, hogy a memóriában lévő rekordhoz képest a lemezen a rekord megváltozott-e. Ha megváltozott a rekord, akkor a kért műveletet nem végzik el. Ez elegendő olyan esetben, mikor a rekord teljes tartalmát felülírjuk (például egy törzsszállomány esetén), csak az a kikötés, hogy a rekord az adatbevitel közben nem változhatott meg.

Bonyolultabb a helyzet akkor, ha a módosítás eredménye a rekord tartalmától függ, egy számlálót (például halmozott forgalom) növelünk. Ilyen esetben nem vezet eredményre a változás figyelése. Ehelyett az írási zárás kiadása

után, a felülírás előtt újra be kell olvasni a rekordot, így a módosításhoz a legutóbbi változtatásokat is figyelembe vesszük.

Példa:

írási zárás kiadása

aktuális rekord beolvasása

összegzés elvégzése

aktuális rekord felülírása

írási zárás feloldása

6. fejezet

Adatállományt kezelő függvények

6.1. ISBUILD

Funkció: Lokális indexfile építése

Használat: `int isbuild(fdp,ixname)`
 `struct FILEDESC *fdp;           mutató file leíróra`
 `char                    *ixname;   index hivatkozási név`

Leírás: Az `isbuild()` függvény kettős funkciót lát el:

- már létező index újraépítése
- lokális indexfile építése

A lokális indexfile lehetőséget ad arra, hogy a többi állomástól függetlenül, speciális elérési szempontot definiáljunk. *fdp* mutató a MISAM file leíró blokkjára (struktúrára), *ixname* pedig az építendő (újraépítendő) index hivatkozási neve. Ha az *ixname* paraméter NULL, akkor a függvény a kiválasztott indexet építi újra.

Ha lokális indexfile-t építünk, akkor a MISAM file-t leíró struktúrát az építendő indexfile leírásával módosítani kell:

- össze kell állítani az indexet leíró struktúrát
- az index leírásával ki kell bővíteni az indexek dinamikus tömbjét
- növelni kell az indexek számát

Ha lokális indexfile-t építünk, akkor annak eltörléséről a MISAM file zárása után külön kell gondoskodni.

Ha már létező indexfile-t építünk újra, akkor a MISAM állományt az építés idejére írásra kell zárni.

Visszatérés:	SUCCESS	sikeres művelet
	E_BADARG	hibás paraméter
	E_IO	írási vagy olvasási hiba
	E_MEMORY	nincs elég szabad memória

Lásd még: `isselind()`, `isread()`, `iswlock()`, `iswunlock()`

Példa: A példában egy lokális indexfile-t építünk a főkönyvi törzsállományhoz. Az index két részből áll. Az első rész karakter típusú, a külső mérleg sorszáma. A második indexdarab double típusú, az éves tartozik forgalom. Az index segítségével külső mérlegsorszám, azon belül éves tartozik forgalom szerinti sorrendben érhetjük el a számlákat.

```
#include <MISAM.H>
:
struct FILEDESC *fdbp;
struct KEYDESC *keyp;
struct KEYPARTS *partp;
:
```



```
/*
 *      Helyet allokálunk az indexleírónak és
 *      inicializáljuk
 */

keyp = (struct KEYDESC *)malloc(
        sizeof(struct KEYDESC));
memset(keyp, '\0', sizeof(struct KEYDESC));
strcpy(keyp->iname, "LOC_IDX");
strcpy(keyp->ifullname, "C:\\\\WORK\\WORK.IDX");
strcpy(keyp->icomment, "Ez egy lokális \
        indexfile");

keyp->nparts = 2;

/*
 *      Helyet allokálunk a kulcsdarabok
 *      leírójának és inicializáljuk
 */

partp = (struct KEYPARTS *)malloc(
        sizeof(struct KEYPARTS));
partp->istart = 45;
partp->ilength = 3;
partp->itype = 'C';

keyp->iparts = malloc(sizeof(struct KEYPARTS *));
*keyp->iparts = partp;

/*
 *      Második kulcsdarab
 */

partp = (struct KEYPARTS *)malloc(
        sizeof(struct KEYPARTS));
partp->istart = 59;
partp->ilength = 8;
partp->itype = 'D';

keyp->iparts = realloc(keyp->iparts,
        2*sizeof(struct KEYPARTS *));
*(keyp->iparts+1) = partp;
```

```
/*  
 *      Az indexleírást beillesztjük  
 *      a MISAM file-t  
 *      leíró struktúrába  
 */  
  
fdp->keys = realloc(fdp->keys,  
                    (fdp->nkeys+1)*sizeof(struct KEYDESC *));  
*(fdp->keys+fdp->nkeys) = keyp;  
fdp->nkeys++;  
:  
if (isbuild(fdp,"LOC_IDX") != SUCCESS)  
    printf("Hiba\n");
```


6.2. ISCLOSE

Funkció: MISAM file zárása

Használat: `int isclose(fdp)`
`struct FILEDESC *fdp;        mutató a file leíróra`

Leírás: Az `isclose()` függvény egy korábban, az `isopen()` függvénnyel megnyitott állományt zár le. A program befejeződése anélkül, hogy a megnyitott MISAM file-okat lezárnánk, a file-ok megsérüléséhez vezethet. Ezért a program befejeződése előtt kötelező az `isclose()` függvény meghívása minden nyitott file-ra.

Az *fdp* paraméter mutató egy MISAM file leíró blokkra.

Mivel a zárás írási jellegű művelet, az `isclose()` függvény a zárás idejére automatikusan írásra zárja a file-t.

Ha építettünk lokális indexfile-t, akkor a MISAM file zárása után ezt külön el kell törölnünk.

Visszatérés:	SUCCESS	sikeres művelet
	E_BUSY	a file-t más munkaállomás írásra foglalja
	E_IO	írási hiba
	E_MEMORY	nincs elég szabad memória

Lásd még: `isopen()`

Példa: Konkurens környezetben MISAM file zárása

```
#include <MISAM.H>
.
struct FILEDESC *fdp;
int rc;
.
if ((rc = isclose(fdp)) == SUCCESS)
    exit(0);
if (rc == E_IO)
{
    printf("Irasi hiba !!!\n");
    exit(2);
}
if (rc == E_BUSY)
{
    printf("Foglalt allomany !!!\n");
    exit(3);
}
```


6.3. ISCREAT

Funkció: MISAM állomány létrehozása

Használat: `int iscreat(fdp,mode)`
`struct FILEDESC *fdp;` mutató a file leíróra
`int mode;` dBASE kompatibilis

Leírás: Az `iscreat()` függvénnyel üres MISAM állományokat tudunk létrehozni. A függvény nem vizsgálja a file-ok létezését, a létező file-okat felülírja. Ezért a függvényt kellő óvatossággal használjuk.

Az `fdp` paraméter mutató a MISAM file-t leíró stuktúrára. A file-t leíró struktúrának inicializáltnak kell lennie egy korábbi `isgetfdf()` függvényhívás által, de a file-nak zártnak kell lennie. Nyitott file-ra tilos `iscreat()` függvényt meghívni.

A `mode` paraméter két értéket vehet fel:

- `DB_COMPAT`, dBASE III formátumú file
- `MSC_MODE`, C formátumú file

A `mode` értékei a MISAM.H header file-ban találhatók.

Konkurrens környezetben írásra zárással biztosítani kell a file kizárólagos használatát.

A létrehozás után a file lezárt, a további hozzáférések előtt meg kell nyitni egy `isopen()` hívással.

Visszatérés: `SUCCESS` sikeres művelet
`E_IO` írási hiba

`E_BADARG` hibás paraméter (például rossz

	mezőtípus)
E_MEMORY	nincs elég szabad memória

Lásd még: isgetfdf(), isopen(), iswlock(), iswunlock()

Példa: A példában konkurrens környezetben hozzuk létre az fdp által mutatott MISAM file-t. A párhuzamos létrehozás elkerülése miatt az iscreat() függvény hívása előtt az állomány írásra (kizárólagos használatra) zárjuk.

```
#include <MISAM.H>
:
struct FILEDESC *fdp;
int rc;
:
if (iswlock(fdp) != SUCCESS)
{
    printf("Foglalt allomany\n");
    exit(0);
}

rc = iscreat(fdp, MSC_MODE);
iswunlock(fdp);

if (rc != SUCCESS)
    printf("Hiba a file letrehozasa \
        kozben\n");
```


6.4. ISDATETOKEY

Funkció: Dátum konverziója kulcs formára

Használat: `int isdatetokey( date, numdate)`
 `char` `*date;` a karakter formájú dátum
 `double` `*numdate;` a belső ábrázolású dátum

Leírás: A függvényt csak dBASE III formátumú file-ok elérése esetén kell használni. Az `isdatetokey()` a dBASE formátumú karakteres dátumot (ééééhnn) konvertálja dBASE index formátumra.

`isdatetokey()`-t az `isread()` függvény előtt kell meghívni, és az `isread()`-nek a konvertált dátumot kell átadni kulcsértékként.

Visszatérés: **SUCCESS** sikeres művelet

Lásd még: `iskeytodate()`, `isread()`

Példa:

```
#include <MISAM.H>
:
char    key[] = "19880809";
double  numkey;
:
isdatetokey(key, &numkey);
isread(fdp, NULL, &numkey, buffer, FIND);
```

6.5. ISDELETE

Funkció: rekord törlése az állományból

Használat: `int isdelete(fdp, bufptr)`
 `struct FILEDESC *fdp;` mutató a file leíróra
 `char *bufptr;` rekord buffer címe

Leírás: Az `isdelete()` függvénnyel rekordokat törölhetünk a MISAM file-ból. A függvény az adatfile-lal párhuzamosan az összes indexfile-t karbantartja (törli a megfelelő kulcsértékeket). A függvény a kiválasztott index aktuális kulcspozíciója alapján határozza meg a törlendő rekordot.

Az `fdp` paraméter egy megnyitott MISAM file leíró blokkjára mutat. A `bufptr` paraméter lehet NULL, vagy mutató egy rekordbufferre. Ha `bufptr` NULL, akkor a függvény nem vizsgálja, hogy a rekord a file-ban változott-e. Ha `bufptr` nem NULL, akkor a törlés előtt a lemezen levő rekordot összehasonlítja a `bufptr` által mutatott rekorddal. A függvény a törlést csak akkor végzi el, ha a két rekordtartalom azonos.

Konkurrens környezetben a törléshez a file kizárólagos használatát az írási zárás beállításával biztosítani kell.

A törölt rekordok helye az állományban nem allokálódik újra, a file-ban lyukak maradnak. Ezért a file rendszeres tömörítéséről ne feledkezzünk el.

Visszatérés:	SUCCESS	sikeres művelet
	E_IO	írási vagy olvasási hiba
	E_CHANGED	a rekord megváltozott
	E_EMPTY	üres állomány
	E_MEMORY	nincs elég szabad memória
	E_BADREC	a rekord már nem létezik az állomány-

ban, vagy a pozíció a file eleje-vége
pozíció

Lásd még: `isread()`, `isopen()`, `iswlock()`, `iswunlock()`

Példa: A példában konkurrens környezetben töröljük az `fdp` által mutatott állomány aktuális rekordját. A törlést csak akkor végezzük el, ha sikerült írásra zárni az állományt és a rekord a lemezen nem változott meg.

```
#include <MISAM.H>
:
struct FILEDESC *fdp;
int rc;
char *key, *bufptr;
:
if (iswlock(fdp) != SUCCESS)
{
    printf("Foglalt allomany\n");
    :
}

rc = isdelete(fdp, bufptr);
iswunlock(fdp);

if (rc == E_CHANGED)
    printf("A rekord tartalma \
        megvaltozott\n");
```

6.6. ISFIRST

Funkció: File elejére pozícionálás

Használat: `int isfirst(fdp,izname)`
 `struct FILEDESC *fdp;` mutató a file leíróra
 `char *izname;` index hivatkozási neve

Leírás: Az `isfirst()` függvénnyel a megadott index szerint beállíthatjuk a file eleje pozíciót. File eleje pozícióban a következő kulcs kérés az első kulcsot adja vissza.

Az *fdp* paraméter egy megnyitott MISAM file leíró blokkjának mutatója. *izname* lehet NULL, vagy egy index hivatkozási nevére mutat. Ha értéke NULL, akkor a pozícionálás a kiválasztott index szerint történik.

Visszatérés: `SUCCESS` sikeres művelet
 `E_BADARG` hibás paraméter (hivatkozási név)

Lásd még: `isread()`, `islast()`

Példa: A file elejére pozícionálás a kiválasztott index alapján:
`#include <MISAM.H>`
 `...`
`isfirst(fdp, NULL);`

6.7. ISGETFDF

Funkció: A file leíró blokk inicializálása

Használat: `int isgetfdf(name,fdpp)`
 char **name*; .FDF file neve
 struct FILEDESC ***fdpp*; leíróblokk mutató címe

Leírás: Az `isgetfdf()` függvény beolvassa a *name* által meghatározott .FDF leíró file-t. Allokálja és feltölti a MISAM leíró blokkot (struktúrát). Az allokált struktúra címét beleteszi az *fdpp* változóba. A könyvtári rutinok paraméterként inicializált leíró blokkot várnak, ezért az `isgetfdf()` hívásnak meg kell előznie minden más MISAM függvény hívását (kivéve az `isnetwork()` függvényt).

name-nek tartalmaznia kell a teljes ösvényt, a .FDF leíró file nevét és a kiterjesztést is.

A file leíró blokk beolvasása konkurrens környezetben nem igényel többlet programozást.

Visszatérés: **SUCCESS** sikeres művelet
 E_IO olvasási hiba
 E_MEMORY nincs elég szabad memória

Lásd még: `isopen()`, `iscreat()`

Példa:

```
#include <MISAM.H>
:
struct FILEDESC *fdp;
:
if (isgetfdf("C:\\FILEDESC\\FOKONYV.FDF",
            &fdp) != SUCCESS)
{
    printf("Hibas leiro file\\n");
    exit(3);
}
```


6.8. ISINFO

Funkció: MISAM file állapotának lekérdezése

Használat: `int isinfo(fdp,infoptr)`
 `struct FILEDESC *fdp;           mutató a file leíróra`
 `struct ISINFO    *infoptr;   output buffer`

Leírás: Az `isinfo()` függvénnyel lekérdezhetjük a MISAM file állapotát. *fdp* mutató egy file leíró blokkra. A file-nak nem kötelező megnyitottnak lenni, de a leíró blokkot az `isinfo()` hívás előtt inicializálni kell az `isgetfdf()` függvénnyel.

infoptr egy ISINFO típusú struktúra címe. Ha a file meg van nyitva, akkor a struktúra `openflag` eleme egy és a többi struktúra tag fel van töltve a megfelelő értékekkel. Ha a file még nincs megnyitva, akkor az `openflag` elem értéke nulla és a többi struktúra tag értéke meghatározatlan.

A `recnum` elem az adatfile allokált rekordjainak a számát jelenti. Ebbe az értékbe beletartoznak a törlés után keletkezett lyukak is.

A `lockstat` mező értéke:

- `READ_LOCK`, ha olvasási zárás van beállítva
- `WRITE_LOCK`, ha írási zárás van beállítva
- `READ_LOCK+WRITE_LOCK`, ha a file-t írásra is és olvasásra is zártuk

A konstansok a MISAM.H header file-ban találhatók.

A `formatflag` mező értéke nulla, ha a file C file és egy, ha a file dBASE III formátumú file.

Visszatérés: SUCCESS

sikeres művelet

Lásd még: iscreat()

Példa: A kiválasztott index hivatkozási nevének lekérdezése:

```
#include <MISAM.H>
:
struct FILEDESC *fdp;
struct ISINFO  infobuff;
:
isinfo(fdp,&infobuff);
printf("Hivatkozási nev: %s\n",infobuff.ixname);
```


6.9. ISKEYTODATE

Funkció: Kulcs konverziója dátum formára

Használat: `int iskeytodate(numdate,date)`
 double **numdate*; a belső ábázolású dátum
 char **date*; a karakter formájú dátum

Leírás: A függvényt csak dBASE III formátumú file-ok elérése esetén kell használni. iskeytodate() a dBASE index formátumú dátumot konvertálja dBASE karakteres (ééééhhnn) formátumra.

iskeytodate()-t az isread() függvény után kell meghívni, hogy az isread() által visszaadott belső ábrázolású dátumot karakteres formára konvertáljuk. A visszaadott karaktersorozat nem string, vagyis nincs NULL karakterrel lezárva.

Visszatérés: SUCCESS sikeres művelet

Lásd még: `isdatetokey()`, `isread()`

```
Példa:      #include <MISAM.H>
              .
              .
              char    key[] = "19880809";
              double   numkey;
              .
              .
              isdatetokey(key,&numkey);
              isread(fdp,NULL,&numkey,buffer,FIND);
              iskeytodate(&numkey,key);
              key[8] = '\0';
              printf("A megalalt kulcs: %s\n",key);
```

6.10. ISLAST

Funkció: File végére pozícionálás

Használat: `int islast(fdp, ixname)`
 `struct FILEDESC *fdp;        mutató a file leíróra`
 `char                        *ixname; index hivatkozási név`

Leírás: Az `islast()` függvénnyel a megadott index alapján beállíthatjuk a file vége pozíciót. File vége pozícióban az előző kulcskérés az utolsó kulcsot adja vissza.

Az `fdp` paraméter egy megnyitott MISAM file leíró blokkjának mutatója. `ixname` lehet NULL, vagy egy index hivatkozási nevére mutat. Ha értéke NULL, akkor a pozícionálás a kiválasztott index szerint történik.

Visszatérés: **SUCCESS** sikeres művelet
 E_BADARG hibás paraméter (hivatkozási név)

Lásd még: `isread()`, `isfirst()`

Példa: A file végére pozícionálás a lokálisan felépített indexfile alapján:

```
#include <MISAM.H>
:
struct FILEDESC *fdp;
:
islast(fdp, "LOCAL_IDX");
```


6.11. ISNETWORK

Funkció: Konkurens környezet vizsgálata

Használat: `int isnetwork()`

Leírás: Az `isnetwork()` függvény kötelezően a program első MISAM függvényhívása. A függvény megvizsgálja, hogy az operációs rendszer kompatibilis-e a NOVELL NetWare 4.6 vagy magasabb verzióval. Egyben inicializálja az operációs rendszert a file zárások számára. Ha a függvény nem talál NOVELL hálózatot, akkor ezt jelzi a könyvtári rutinok számára. Így a hálózatosra írt programok egyfelhasználós környezetben ugyanolyan hatékonyan működnek, mintha kihagytuk volna a file zárásokat.

Visszatérés: 0, ha az operációs rendszer egyfelhasználós
1, ha NOVELL hálózat működik

Lásd még: `isrlock()`, `iswlock()`, `isrunlock()`, `iswunlock()`

Példa: #include <MISAM.H>
 :
 main(argc,argv)
 int argc;
 char *argv[];
 {
 if (isnetwork())
 printf("NOVELL halozat\n");
 else
 printf("Egyfelhasznalos kornyezet\n");
 :
 }

6.12. ISOPEN

Funkció: MISAM file megnyitása

Használat: `int isopen(fdp)`
`struct FILEDESC *fdp;` mutató a file leíróra

Leírás: A MISAM file-okhoz történő hozzáférések előtt a file-t az `isopen()` függvénnyel meg kell nyitni. *fdp* egy MISAM file leíró blokk mutatója. A file leíró blokkot az `isopen()` hívás előtt inicializálni kell az `isgetfdf()` függvénnyel. A függvény megnyitja az adatfile-t és az összes indexfile-t.

Mivel a file nyitás olvasási művelet, konkurens környezetben a függvény automatikusan gondoskodik az olvasási zárás beállításáról.

Az `isopen()` függvény hívás után a kiválasztott index a .FDF leíró file-ban elsőként definiált index. Az aktuális kulcspozíció a file eleje pozíció.

Visszatérés:	SUCCESS	sikeres művelet
	E_BUSY	foglalt állomány
	E_MEMORY	nincs elég memória
	E_IO	olvasási hiba
	E_BADFORM	az adat- vagy valamelyik indexfile nem MISAM formátumú

Lásd még: `isgetfdf()`, `isrlock()`, `isrunlock()`, `isclose()`

Példa: `#include <MISAM.H>`
`struct FILEDESC *fdp;`
`if (isopen(fdp) != SUCCESS)`
`{`
`printf("Hiba a file nyitasa kozben\n");`
`exit(3);`
`}`

6.13. ISREAD

Funkció: Adatrekord olvasása

Használat: `int isread(fdp, izname, key, buffer, mode)`

<code>struct FILEDESC</code>	<code>*fdp;</code>	mutató a file leíróra
<code>char</code>	<code>*izname;</code>	az index hivatkozási neve
<code>char</code>	<code>*key;</code>	kulcsérték
<code>char</code>	<code>*buffer;</code>	adatrekord címe
<code>int</code>	<code>mode;</code>	műveleti kód

Leírás: Az `isread()` függvény kettős funkciót tölt be:

- a MISAM file pozícionálása
- adatrekordok olvasása

`fdp` mutató egy nyitott file leíró blokkjára. Mind a pozícionálás, mind az adatrekordok olvasása valamelyik index szerint történik. Ha `izname` NULL, akkor a függvény a kiválasztott index alapján hajtja végre a funkciót. Ha `izname` egy index hivatkozási neve, akkor a függvény a megadott indexet használja.

`key` a kulcsérték `buffer`-nek címe. Az `isread()` függvény inputként várja a kulcs értéket, ha a `mode`:

- FIND
- NEXTSAME
- PREVSAME

A `key` paraméter a választott funkciótól függetlenül mindig output paraméter is. Az `isread()` függvény felülírja a kulcsbuffer eredeti tartalmát. A kulcsértéket mindig az index leírásának megfelelően kell összeállítani. Ha a file C file, akkor a függvény minden kulcsdarabot belső ábrázolásúnak tételez fel. Ha a file dBASE III formátumú file, akkor a numerikus kulcsértékeket double értékke kell konvertálni, a dátum tí-

pusú kulcsot az `isdatetokey()`, `iskeytodate()` függvényekkel szintén konvertálni kell. Az `isread()` által visszaadott kulcsértékek mindig az input formátumnak megfelelők. (Tehát dBASE file esetén a dátum és a numerikus kulcsérték konvertáltak.)

buffer az adatrekord címe. Ha a funkció adatrekord olvasása, akkor az `isread()` a *buffer* által mutatott területre tölti az adatrekordot. Ha a visszatérési érték nem SUCCESS és nem E_GREATER, akkor a *buffer* által mutatott terület nem változik.

Ha a funkció csak pozícionálás, akkor *buffer* lehet NULL is.

A *mode* paraméter megadja az `isread()` által végrehajtandó funkciót. *mode* értéke a következők valamelyike lehet:

- CURRENT, rekord olvasása az aktuális kulcspozíció szerint
- FIRST, index szerint az első rekord olvasása
- LAST, index szerint az utolsó rekord olvasása
- NEXT, index szerint a következő (file vége felé eső) rekord olvasása
- NEXTSAME, index szerint a következő, a megadottal megegyező kulcsú rekord olvasása
- PREV, index szerint az előző (file eleje felé eső) rekord olvasása
- PREVSAME, index szerint az előző, a megadottal megegyező kulcsú rekord olvasása

- **FIND**, az adott kulccsal első megegyező kulcsú rekord olvasása (keresés)

A *mode* paraméterhez hozzáadva a **NOREAD** konstanst, az **isread()** függvény nem olvassa be az adatrekordot, csak az **indexfile**-t pozicionálja. A konstansok a **MISAM.H** header file-ban találhatók.

Az **isread()** függvény minden esetben állítja az aktuális kulcspozíciót.

Konkurrens környezetben gondoskodni kell az olvasási záras beállításáról.

Visszatérés:	SUCCESS	sikeres művelet
	E_BADARG	hibás paraméter (pl. hivatkozási név)
	E_GREATER	a keresetnél nagyobb kulcsú rekordot talált, vagy NEXTSAME , PREVSAME funkcióknál a rekord kulcsa nem egyezik a megadottal
	E_TOPKEY	nincs előző rekord
	E_ENDKEY	nincs következő rekord, vagy a keresetnél csak kisebb kulcs létezik
	E_BADARG	az aktuális rekordpozíció meghatározatlan
	E_EMPTY	üres állomány
	E_BADFORM	sérült állomány
	E_IO	olvasási hiba
	E_MEMORY	nincs elég szabad memória

Lásd még: **isrlock()**, **isrunlock()**, **isselind()**, **isfirst()**, **islast()**

Példa: Konkurens környezetben annak ellenőrzése, hogy egy adott kulcsú rekord létezik-e az állományban:

```
#include <MISAM.H>
.
struct FILEDESC *fdp;
char             *key;
int              rc;
.
if (isrlock(fdp) != SUCCESS)
{
    printf("Foglalt allomany\n");
    .
}
.
rc = isread(fdp, NULL, key, NULL, FIND+NOREAD);
isrunlock(fdp);
switch (rc) {
case SUCCESS:
    printf("Letezik\n");
    break;
case E_GREATER:
case E_EMPTY:
case E_ENDKEY:
    printf("Nem letezik\n");
    break;
default:
    printf("Hiba az allomany olvasasa koz\
        ben\n");
    break;
}
```


6.14. ISRLOCK

Funkció: Olvasási zárás beállítása

Használat: `int isrlock(fdp)`
`struct FILEDESC *fdp;` mutató a file leíróra

Leírás: Az `isrlock()` függvénnyel egy MISAM file-ra beállíthatjuk a file szintű olvasási zárást (read lock). Konkurens környezetben az olvasási műveletek csak akkor működnek helyesen, ha a file-ra az adott állomás által beállított olvasási zárás van érvényben.

fdp mutató egy inicializált (`isgetfdf()`) MISAM file leíró blokkra. A file-nak a záráshoz nem kötelező megnyitottnak lennie.

A file szintű olvasási zárás a többi munkaállomás számára tiltja az adott file-hoz az írási jellegű hozzáférést (például rekord törlése, állomány zárása, stb.). Olvasási zárás egy file-ra egyidőben több munkaállomásról is lehet érvényben, vagyis a file-t egyidőben több munkaállomásról lehet olvasni.

Egy munkaállomás egy file-ra egyszerre kiadhat olvasási és írási zárást is. Ilyen esetben a többi munkaállomás felé az erősebb írási zárás érvényes.

Az `isrlock()` függvény által kiadott olvasási zárás a file-ra kiadott `isrunlock()` hívásig van érvényben. Mivel a zárás file szintű, ezért a programozás során arra kell törekedni, hogy a zárás a lehető legrövidebb ideig legyen érvényben.

A zárások csak akkor működnek helyesen, ha a program első MISAM hívása az `isnetwork()` függvényhívás. A konkurens környezetre felkészített programok egyfelhasználós környe-

zetben is futtathatók, az egyfelhasználásra írt programokkal megegyező hatékonysággal.

Az `isrlock()` függvény 10 másodpercig várakozik az olvasási zárás sikeres beállítására. Ez az érték elég nagy ahhoz, hogy normális esetben még erősen leterhelt hálózaton is sikeres legyen a művelet. Ezért nem helyes programozási technika a zárást beállító függvényeket ciklusban hívni a sikeres zárásig. Ehelyett külön le kell kezelni azt az esetet, ha a zárás nem sikerül.

Visszatérés:	SUCCESS	sikeres művelet
	E_BUSY	a file más munkaállomás által írásra zárt
	E_NETWORK	hálózati hiba (például túlterhelt a hálózat)
	E_BADARG	a munkaállomás korábban már beállított egy olvasási zárást
	E_BADFORM	sérült MISAM file

Lásd még: `isrunlock()`, `iswlock()`, `iswunlock()`, `isnetwork()`, `isopen()`, `isclose()`, `isinfo()`

Példa:

```
#include <MISAM.H>
:
struct FILEDESC *fdp;
:
switch (isrlock(fdp)) {
case SUCCESS:
case E_BADARG:
    break;
case E_BUSY:
    printf("Foglalt allomany\n");
    exit(1);
case E_NETWORK:
    printf("Halozati hiba\n");
    exit(2);
}
```


6.15. ISRUNLOCK

Funkció: Olvasási zárás feloldása

Használat: `int isrunlock(fdp)`
`struct FILEDESC *fdp;` mutató a file leíróra

Leírás: Az `isrunlock()` függvénnyel egy korábbi `isrlock()` hívással beállított olvasási zárást oldhatunk fel. Az *fdp* paraméter mutató egy inicializált MISAM file leíró blokkra. Ha a file-ra egyszerre írási és olvasási zárás is érvényben volt, akkor az `isrunlock()` hívás után az írási zárás továbbra is érvényben marad.

Visszatérés: **SUCCESS** sikeres művelet
E_BUSY az adott munkaállomás által olvasási
zárás nem volt beállítva
E_NETWORK hálózati hiba
E_BADARG a munkaállomás nem állított be olvasási
zárást

Lásd még: `isrlock()`, `iswlock()`, `iswunlock()`, `isnetwork()`, `isopen()`, `isclose()`, `isinfo()`

Példa:

```
#include <MISAM.H>
:
struct FILEDESC *fdp;
:
switch (isrunlock(fdp)) {
case SUCCESS:
    break;
case E_BUSY:
    printf("Nem volt olvasasi zaras\n");
    break;
case E_NETWORK:
    printf("Halozati hiba\n");
    break;
}
```


6.16. ISSELIND

Funkció: Kiválasztott index beállítása

Használat: `int isselind(fdp,ixname)`
`struct FILEDESC *fdp;` mutató a file leíróra
`char *ixname;` index hivatkozási neve

Leírás: Az `isselind()` függvénnyel a kiválasztott indexet jelölhetjük ki. Az index paramétert váró függvények a kiválasztott index alapján dolgoznak, ha az index paraméterük NULL.

Az *fdp* paraméter mutató egy megnyitott MISAM file leíró blokkjára. *ixname* egy létező index hivatkozási neve. Ha *ixname* NULL, akkor a kiválasztott index a .FDF file-ban elsőként megadott index lesz.

Visszatérés: **SUCCESS** sikeres művelet
E_BADARG az adott hivatkozási nevű index nem létezik

Lásd még: `isopen()`, `isread()`, `isbuild()`, `isfirst()`, `islast()`

Példa: A lokálisan felépített index kiválasztása, majd pedig visszatérés az elsőként megadott indexhez:

```
#include <MISAM.H>
:
struct FILEDESC *fdp;
:
isselind(fdp,"LOCAL_IDX");
:
isselind(fdp,NULL);
```


6.17. ISUPDATE

Funkció: Adatrekord felülírása

Használat: `int isupdate(fdp,buff,oldbuff)`
`struct FILEDESC *fdp;` mutató a file leíróra
`char *buff;` mutató az új rekordra
`char *oldbuff;` mutató a régi rekordra

Leírás: Az `isupdate()` függvénnyel adatrekordot módosíthatunk. A módosítandó rekordot a kiválasztott index aktuális kulcspozíciója jelöli ki.

fdp egy megnyitott MISAM file leíró blokkjának címe. *buff* annak a rekordbuffernek a címe, amely a módosított adatokat tartalmazza. *oldbuff* értéke lehet NULL, vagy egy másik rekordbuffer címe. Ha *oldbuff* NULL, akkor a függvény a felülírás előtt nem végez semmiféle ellenőrzést. Ha nem NULL, akkor az általa mutatott buffernek a módosítás előtti rekordot kell tartalmaznia. Ebben az esetben a függvény a felülírás előtt összehasonlítja a lemezen levő rekordot az *oldbuff* által mutatott rekorddal. A felülírást csak akkor végzi el, ha a két rekord tartalma megegyezik.

Az *oldbuff* paraméter a párhuzamos felülírások kivédéséhez ad egy lehetőséget. Egy másik lehetőség az, hogy a rekordot a módosítás előtt mindig újra beolvassuk. Erre a módszerre általában akkor van szükség, ha valamelyik rekordmező számlálóként funkcionál.

A függvény az adatrekorddal párhuzamosan az indexfile-okat is karbantartja.

Konkurrens környezetben a függvény hívása előtt a file-ra ki kell adni az írási zárást.

Az `isupdate()` függvény nem változtatja az aktuális kulcs-
pozíciót.

Visszatérés:	E_SUCCESS	sikeres művelet
	E_IO	írási vagy olvasási hiba
	E_MEMORY	nincs elég szabad memória
	E_BADREC	az aktuális rekordpozíció meghatá- rozatlan
	E_CHANGED	a rekord tartalma már megváltozott

Lásd még: `iswlock()`, `iswunlock()`, `isread()`

Példa: A példa konkurens környezetben módosítja a rekordot, el-
lenőrizve az esetleges változást:

```
#include <MISAM.H>
:
struct FILEDESC *fdp;
char             *buff;
char             *newbuff;
char             *key;
int              rc, reclen;
:
:
```



```
if (isrlock(fdp) != SUCCESS)
{
    printf("Foglalt allomany\n");
    .
    .
}
rc = isread(fdp,NULL,key,buff,FIND);
isrunlock(fdp);
if (rc != SUCCESS)
{
    printf("Nem letezo kulcs\n");
    .
    .
}
memcpy(newbuff,buff,reclen);

/*
 *      Rekord tartalmának módosítása
 */

if (iswlock(fdp) != SUCCESS)
{
    printf("Foglalt allomany\n");
    .
    .
}
rc = isupdate(fdp,newbuff,buff);
iswunlock(fdp);
switch (rc) {
case SUCCESS:
    break;
case E_CHANGED:
    printf("A rekord valtozott\n");
    break;
case E_BADREC:
    printf("Hibas rekordpozicio\n");
    break
default:
    printf("Hiba a rekord modositasa \
        kozben\n");
    break;
}
```

6.18. ISWLOCK

Funkció: Írási zárás beállítása

Használat: `int iswlock(fdp)`
`struct FILEDESC *fdp;` mutató a file leíróra

Leírás: Az `iswlock()` függvénnyel egy MISAM file-ra beállíthatjuk a file szintű írási zárást (write lock). Konkurens környezetben az írási műveletek csak akkor működnek helyesen, ha a file-ra az adott állomás által írási zárás van érvényben.

fdp mutató egy inicializált (`isgetfdf()`) MISAM file leíró blokkra. A file-nak a záráshoz nem kötelező megnyitottnak lennie.

A file szintű írási zárás a többi munkaállomás számára tiltja az adott file-hoz történő (írási és olvasási) hozzáférést. Írási zárás egy file-ra egyidőben csak egy munkaállomásról lehet érvényben, vagyis a file-t egyidőben csak egy munkaállomásról lehet írni.

Egy munkaállomás egy file-ra egyszerre kiadhat olvasási és írási zárást is. Ilyen esetben a többi munkaállomás felé az erősebb írási zárás érvényes.

Az olvasási zárás helyett bármikor használhatunk írási zárást is, ebben az esetben azonban számolnunk kell az egyidőben történő olvasás lehetőségének kizárásával.

Az `iswlock()` függvény által kiadott írási zárás a file-ra kiadott `iswunlock()` hívásig van érvényben. Mivel a zárás file szintű, ezért a programozás során arra kell törekedni, hogy a zárás a lehető legrövidebb ideig legyen érvényben.

A zárások csak akkor működnek helyesen, ha a program első MISAM hívása az isnetwork() függvényhívás. A konkurrens környezetre felkészített programok egyfelhasználós környezetben is futtathatók, az egyfelhasználásra írt programokkal megegyező hatékonysággal.

Az iswlock() függvény 10 másodpercig várakozik az írási zárás sikeres beállítására. Ez az érték elég nagy ahhoz, hogy normális esetben még erősen leterhelt hálózaton is sikeres legyen a művelet. Ezért nem helyes programozási technika a zárást beállító függvényeket ciklusban hívni a sikeres zárásig. Ehelyett külön le kell kezelni azt az esetet, ha a zárás nem sikerül.

Visszatérés:	SUCCESS	sikeres művelet
	E_BUSY	a file más munkaállomás által olvasásra vagy írásra zárt
	E_NETWORK	hálózati hiba (például túlterhelt a hálózat)
	E_BADARG	a munkaállomás már korábban állított be írási zárást
	E_BADFORM	sérült MISAM file

Lásd még: iswunlock(), isrlock(), isrunlock(), isnetwork(), isclose(), isinfo()

Példa: #include <MISAM.H>
 :
 struct FILEDESC *fdp;
 :
 switch (iswlock(fdp)) {
 case SUCCESS:
 break;
 case E_BUSY:
 printf("Foglalt allomany\n");
 exit(1);
 case E_NETWORK:
 printf("Halozati hiba\n");
 exit(2);
 }

6.19. ISWRITE

Funkció: Új rekord kiírása

Használat: `int iswrite(fdp,buffer)`
`struct FILEDESC *fdp;` mutató a file leíróra
`char *buffer;` rekordbuffer címe

Leírás: Az `iswrite()` függvénnyel új rekordokat írhatunk ki a MISAM állományba. Az új adatrekordok mindig az adatfile végére kerülnek, az azonos kulcsú rekordokat a kiírás sorrendjében érhetjük el.

A függvény az adatrekorddal párhuzamosan az összes index-file-t is karbantartja, vagyis beszúrja a megfelelő kulcsértékeket.

fdp egy megnyitott MISAM file leíró blokkjának címe. *buffer* az adatrekordra mutat. Az adatrekordot a kiírás előtt szükség esetén konvertálni kell (például dBASE file esetén a numerikus értékeket karakteres formára).

Konkurrens környezetben az `iswrite()` függvény hívása előtt gondoskodni kell a file kizárólagos használatáról, vagyis ki kell adni az írási zárást.

A rekord kiírása nem változtatja az aktuális kulcspozíciót.

Visszatérés: **SUCCESS** sikeres művelet
E_IO írási hiba
E_MEMORY nincs elég szabad memória

Lásd még: iswlock(), iswunlock()

Példa: #include <MISAM.H>

```
struct FILEDESC *fdp;
char             *buffer;
int              rc;
                :
if (iswlock(fdp) != SUCCESS)
{
    printf("Foglalt allomany\n");
    :
}
rc = iswrite(fdp,buffer);
iswunlock(fdp);
if (rc != SUCCESS)
    printf("Hiba a file irasa kozben\n");
```


6.20. ISWUNLOCK

Funkció: Írási zárás feloldása

Használat: `int iswunlock(fdp)`
`struct FILEDESC *fdp;` mutató a file leíróra

Leírás: Az `iswunlock()` függvénnyel egy korábbi `iswlock()` hívással beállított írási zárást oldhatunk fel. *fdp* mutató egy inicializált MISAM file leíró blokkra. Ha a file-ra egyszerre írási és olvasási zárás is érvényben volt, akkor az `iswunlock()` hívás után az olvasási zárás továbbra is érvényben marad.

Visszatérés:	SUCCESS	sikeres művelet
	E_BUSY	az adott munkaállomás által írási zárás nem volt beállítva
	E_NETWORK	hálózati hiba
	E_BADARG	a munkaállomás nem állított be írási zárást
	E_IO	írási hiba

Lásd még: `iswlock()`, `isrlock()`, `isrunlock()`, `isnetwork()`, `isclose()`, `isinfo()`

Példa:

```
#include <MISAM.H>
...
struct FILEDESC *fdp;
...
switch (iswunlock(fdp)) {
case SUCCESS:
    break;
case E_BUSY:
    printf("Írasi záras nem volt \
          beallitva\n");
    break;
case E_NETWORK:
    printf("Hálózati hiba\n");
    break;
}
```


6.21. Példaprogramok

A fejezet hátralevő része a függvények programozását bemutató példaprogramokat tartalmazza. A példaprogramok a korábbi példákban bemutatott állományt, a főkönyvi törzsállományt dolgozzák fel. Mivel az állomány felépítését a korábbi fejezetek tartalmazzák, itt ennek ismertetésére külön nem térünk ki.

A példaprogramok elsődleges célja a MISAM függvények programozásának bemutatása. Ezért a példaprogramok egyes részei (például a felhasználóval történő kommunikáció) csak olyan mélységig finomítottak, amilyen mélységig a programok futtatásához erre szükség volt.

6.21.1. Állomány listázása

Az első példaprogram egy egyszerű listázó program. Paraméterként a .FDF file nevét várja (teljes ösvénnyel és kiterjesztéssel). A program beolvassa a .FDF file-t, megnyitja a MISAM állományt és az összes rekord tartalmát kilistázza a képernyőre.

```
/*  
*      MISAM példa program:  
*          állomány listázása kulcs szerint  
*      Hívás:  
*          LIST <.FDF file neve>  
*/
```

```
#include <MISAM.H>

/*
 *      A könnyebb megértés miatt a .FDF file-ból
 *      generált header file-t nem includoljuk, hanem
 *      bevágjuk a szövegbe
 */

/* FOKONYV.H */

struct FOKONYV {
    char fokonyvi_szam[15];
    char megnevezes[30];
    char kulso_merleg[3];
    char belso_merleg[3];
    double nyito_egyenleg;
    double f_tartozik;
    double f_kovetel;
    double egyenleg_1;
    double egyenleg_2;
    double egyenleg_3;
    double f_tervezett;
};

#define L_fokonyvi_szam 15
#define L_megnevezes 30
#define L_kulso_merleg 3
#define L_belso_merleg 3
#define L_nyito_egyenleg 8
#define L_f_tartozik 8
#define L_f_kovetel 8
#define L_egyenleg_1 8
#define L_egyenleg_2 8
#define L_egyenleg_3 8
#define L_f_tervezett 8

/*
 *      Generált header file vége
 */

main(argc,argv)
int     argc;
char    *argv[];

{
    char                *buffer;
    struct FILEDESC *fdp;
```



```
if (argc != 2)
{
    printf("Hasznalat: LIST <fdf file neve>\n");
    exit(0);
}
buffer = malloc(sizeof(struct FOKONYV));
if (buffer == NULL)
{
    printf("Nincs eleg szabad memoria\n");
    exit(3);
}

/*
 *      Hálózati levezsgálása
 *      zárások inicializálása
 */
isnetwork();

/*
 *      Beolvassuk a .FDF file leíróit, inicializáljuk
 *      a MISAM leíró struktúrát
 */
if (isgetfdf(argv[1], &fdf) != SUCCESS)
{
    printf("Hiba a leíró file olvasása közben\n");
    exit(1);
}

/*
 *      Megnyitjuk a file-t. Az isopen() automatikusan
 *      zárja magának a file-t, így ezzel nem kell
 *      törődni.
 */
if (isopen(fdp) != SUCCESS)
{
    printf("Hiba a file nyitása közben\n");
    exit(1);
}

/*
 *      Nyitás után a kiválasztott index az első
 *      -most egyetlen- index. Az aktuális kulcspozíció
 *      a file eleje pozíció.
 */
while (nextrec(fdp, buffer) == SUCCESS)
    putrec(buffer);
if (isclose(fdp) != SUCCESS)
```

```

        {
            printf("Hiba a file zarasa kozben\n");
            exit(1);
        }

    exit(0);
}

int nextrec(fdp, bufptr)
/*
 *      A következö rekord olvasása.
 *      A file-t olvasás előtt zárja.
 */
struct FILEDEF *fdp;
char            *bufptr;

    {
        char    key[16];
        int      rc;

        /*
         *      Az olvasás előtt olvasásra zárjuk a file-t.
         */
        rc = isrlock(fdp);
        if (rc != SUCCESS)
        {
            printf("Foglalt allomany\n");
            return(rc);
        }

        /*
         *      Beolvassuk a kiválasztott index szerint a
         *      következö rekordot.
         *      Olvasás után rögtön feloldjuk az olvasási
         *      zárást.
         */
        rc = isread(fdp, NULL, key, bufptr, NEXT);
        isrunlock(fdp);
        if (rc != SUCCESS)
            if (rc != E_ENDKEY && rc != E_EMPTY)
                printf("Hiba a file olvasasa kozben\n");

        return(rc);
    }

int putrec(bufptr)
/*
 *      Adatrekord kiirasa a kepornyore

```



```

*/
char      *bufptr;

{
    struct FOKONYV *recptr;

    recptr = (struct FOKONYV *)bufptr;

    printf("fokonyvi_szam %15.15s\n", recptr->fokonyvi_szam);
    printf("megnevezes %30.30s\n", recptr->megnevezes);
    printf("kulso_merleg %3.3\n", recptr->kulso____merleg);
    printf("belso_merleg %3.3\n", recptr->belso_merleg);
    printf("nyito_egyenleg %f\n", recptr->nyito_egyenleg);
    printf("f_tartozik %f\n", recptr->f_tartozik);
    printf("f_kovetel %f\n", recptr->f_kovetel);
    printf("egyenleg_1 %f\n", recptr->egyenleg_1);
    printf("egyenleg_2 %f\n", recptr->egyenleg_2);
    printf("egyenleg_3 %f\n", recptr->egyenleg_3);
    printf("f_tervezett %f\n\n", recptr->f_tervezett);
}

```

6.21.2. Állomány karbantartása

A második példaprogram a MISAM állományok karbantartására példa. A karbantartó program szintén a FŐKÖNYV állományt dolgozza fel. Egy egyszerű funkcióválasztó eljárással választhatunk a program által felajánlott funkciók közül:

- új rekord bevitele
- már létező rekord módosítása
- már létező rekord törlése

Az utóbbi két funkció esetén a program bekéri a rekord kulcsát, felolvassa a rekordot a file-ból és kiírja annak tartalmát a képernyőre. Módosítás esetén bekéri a rekord új tartalmát (a teljes rekordot meg kell adni), majd elvégzi a megfelelő funkciót.

A módosítás során a program ellenőrzi, hogy az újonnan megadott főkönyvi szám egyedi-e a törzsellományban. A módosítást csak akkor végzi el, ha a főkönyvi szám egyedi.

A törlést végző függvény vizsgálja, hogy a rekord a felolvasástól a törlés tényleges elvégzéséig nem változott-e meg. Ha a rekord megváltozott, akkor a program a törlést csak megerősítés esetén végzi el.


```
/*
 *      MISAM példa program:
 *          állomány karbantartása
 *      Hívás:
 *          KARB <.FDF file neve>
 */

#include <MISAM.H>

/*
 *      A könnyebb megértés miatt a .FDF file-ból
 *      generált header file-t nem includoljuk, hanem
 *      bevágjuk a szövegbe
 */

/* FOKONYV.H */

struct FOKONYV {
    char fokonyvi_szam[15];
    char megnevezes[30];
    char kulso_merleg[3];
    char belso_merleg[3];
    double nyito_egyenleg;
    double f_tartozik;
    double f_kovetel;
    double egyenleg_1;
    double egyenleg_2;
    double egyenleg_3;
    double f_tervezett;
};

#define L_fokonyvi_szam 15
#define L_megnevezes 30
#define L_kulso_merleg 3
#define L_belso_merleg 3
#define L_nyito_egyenleg 8
#define L_f_tartozik 8
#define L_f_kovetel 8
#define L_egyenleg_1 8
#define L_egyenleg_2 8
#define L_egyenleg_3 8
#define L_f_tervezett 8

/*
 *      Generált header file vége
 */

main(argc,argv)
```



```
int      argc;
char     *argv[];

{
char      *buffer;
struct FILEDESC *fdp;
int       code;

if (argc != 2)
{
printf("Hasznalat: KARB <fdf file neve>\n");
exit(0);
}

buffer = malloc(sizeof(struct FOKONYV));
if (buffer == NULL)
{
printf("Nincs eleg szabad memoria\n");
exit(3);
}

/*
 * Hálózati levezsgálása
 * zárások inicializálása
 */
isnetwork();

/*
 * Beolvassuk a .FDF file leíróát, inicializáljuk
 * a MISAM leíró struktúrát
 */
if (isgetfdf(argv[1], &fdp) != SUCCESS)
{
printf("Hiba a leíró file olvasása közben\n");
exit(1);
}

/*
 * Megnyitjuk a file-t. Az isopen() automatikusan
 * zárja magának a file-t, így ezzel nem kell
 * törödni.
 */
if (isopen(fdp) != SUCCESS)
{
printf("Hiba a file nyitása közben\n");
exit(1);
}
```

```

/*
 *      A fő ciklus: funkció választás, a funkciónak
 *      megfelelő rutin meghívása
 */

while ((ocde = getfunc()) != 9)
    switch (code) {
        case 0:
            newrec(fdp,buffer);
            break;
        case 1:
            updrec(fdp,buffer);
            break;
        case 2:
            delrec(fdp,buffer);
            break;
        default:
            printf("Hibas kod\n");
            break;
    }

if (isclose(fdp) != SUCCESS)
{
    printf("Hiba a file zarasa kozben\n");
    exit(1);
}

exit(0);
}

int getfunc()
/*
 *      Beolvassa a funkció kódját.
 *      Visszatérési értéke a megadott funkció
 */

{
    int    c;

    printf("\n\n1. Uj rekord felvitele
           \n2. Rekord modositasa\n");
    printf("3. Rekord torlese\n\n9. Kiszallas\n\n\n");
    printf("Adja meg a funkcio kodjat");
    scanf("%d",&c);
    return(c);
}

```



```
    }

int newrec(fdp,bufptr)
/*
 *      >j rekord felvitele az állományba.
 */
struct FILEDESC *fdp;
char              *bufptr;

{
    int      rc;

    /*
     *      Adatrekord beolvasása
     */
    inputrec(bufptr);

    /*
     *      Az írási művelet előtt az állományra ki kell
     *      adni egy írási zárást. Csak akkor írjuk ki a
     *      rekordot, ha a zárási sikerült.
     */
    if (iswlock(fdp) != SUCCESS)
    {
        printf("Foglalt allomany\n");
        return;
    }
    rc = iswrite(fdp,bufptr);
    iswunlock(fdp);
    if (rc != SUCCESS)
    {
        printf("Hiba az allomany irasa kozben\n");
        isclose(fdp);
        exit(3);
    }
}

int updrec(fdp,bufptr)
/*
 *      Adatrekord felülírása
 */
struct FILEDESC *fdp;
char              *bufptr;

{
    char      key[100];
    int      rc;
```

```
/*
 *      A módosítandó rekord kulcsának beolvasása
 */
scanf("%15s",key);

/*
 *      Adatrekord megkeresése és kiírása a képernyőre
 */
if (getrec(fdp,key,bufptr) != SUCCESS)
    return;
printrec(bufptr);

/*
 *      Beolvassuk az új adatrekordot és visszaírjuk az
 *      állományba. Az állományban a főkönyvi számnak
 *      egyedinek kell lennie.
 *      Az írási művelet előtt az állományra ki kell
 *      adni egy írási zárást. Csak akkor írjuk ki a
 *      rekordot, ha a zárás sikerült.
 */
inputrec(bufptr);

if (iswlock(fdp) != SUCCESS)
{
    printf("Foglalt allomany\n");
    return;
}

/*
 *      A kulcs egyediségének ellen-
 *      őrzése. Mivel az adatrekordot
 *      nem olvassuk be, az aktuális
 *      rekordpozíció nem változik.
 */
memcpy(key,bufptr,15);
rc = isread(fdp,NULL,key,NULL,FIND+NOREAD);
if (rc == SUCCESS)
{
    iswunlock(fdp);
    printf("A megadott számlaszám már letezik\n");
    return;
}
else if (rc != E_GREATER && rc != E_ENDKEY &&
        rc != E_EMPTY)
{
    iswunlock(fdp);
}
```



```
    printf("Hiba a file olvasasa kozben\n");
    isclose(fdp);
    exit(3);
}

/*
 *      A rekord változását nem figyeljük, mivel a
 *      harmadik paraméter NULL.
 */
rc = isupdate(fdp,bufptr,NULL);
iswunlock(fdp);
if (rc != SUCCESS)
{
    printf("Hiba az allomany modositasa kozben\n");
    isclose(fdp);
    exit(3);
}
}

int delrec(bufptr,buffer)
/*
 *      Rekord törlése az állományból
 */
struct FILEDESC *fdp;
char             *bufptr;

{
    char    key[100];
    int     rc,c;

    /*
     *      A törlendő rekord kulcsának beolvasása
     */
    scanf("%15s",key);

    /*
     *      Adatrekord megkeresése és kiírása a képernyőre
     */
    if (getrec(fdp,key,bufptr) != SUCCESS)
        return;
    printrec(bufptr);

    /*
     *      Az írási művelet előtt az állományra ki kell
     *      adni egy írási zárást. Csak akkor írjuk ki a
     *      rekordot, ha a zárással sikerült.
     */
    inputrec(bufptr);
```

```

if (iswlock(fdp) != SUCCESS)
{
    printf("Foglalt allomany\n");
    return;
}

/*
 *      A rekord változását figyeljü,
 *      a második paraméter nem NULL.
 */
rc = isdelete(fdp,bufptr);
iswunlock(fdp);
if (rc == E_CHANGED)
{
    /*
     *      Változás esetén megerősítést kérünk.
     *      A felhasználóval történő párbeszéd
     *      idejére az állományt nem zárjuk
     */
    {
        printf("A rekord megváltozott. Toroljuk ? (1/0)");
        scanf("%d",&c);
        if (c != 1)
            return;

        if (iswlock(fdp) != SUCCESS)
        {
            printf("Foglalt allomany\n");
            return;
        }

        rc = isdelete(fdp,NULL);
        iswunlock(fdp);
    }

    if (rc != SUCCESS)
    {
        printf("Hiba az allomany torlese kozben\n");
        isclose(fdp);
        exit(3);
    }
}

int printrec(bufptr)
/*

```



```

*      Adatrekord kiirása a képernyőre
*/
char   *bufptr;

{
    struct FOKONYV *recptr;

    recptr = (struct FOKONYV *)bufptr;

    printf("fokonyvi_szam %15.15s\n",recptr->fokonyvi_szam);
    printf("megnevezes %30.30s\n",recptr->megnevezes);
    printf("kulso_merleg %3.3\n",recptr->kulso_merleg);
    printf("belso_merleg %3.3\n",recptr->belso_merleg);
    printf("nyito_egylenleg %f\n",recptr->nyito_egylenleg);
    printf("f_tartozik %f\n",recptr->f_tartozik);
    printf("f_kovetel %f\n",recptr->f_kovetel);
    printf("egylenleg_1 %f\n",recptr->egylenleg_1);
    printf("egylenleg_2 %f\n",recptr->egylenleg_2);
    printf("egylenleg_3 %f\n",recptr->egylenleg_3);
    printf("f_tervezett %f\n\n",recptr->f_tervezett);
}

int getrec(fdp,key,bufptr)
/*
*      Adott kulcsú rekord olvasása.
*      A file-t olvasás előtt zárja.
*/
struct FILEDEF *fdp;
char            *key;
char            *bufptr;

{
    int         rc;

    /*
     *      Az olvasás előtt olvasásra zárjuk a file-t.
     */
    rc = isrlock(fdp);
    if (rc != SUCCESS)
    {
        printf("Foglalt allomany\n");
        return(rc);
    }

    /*
     *      Beolvassuk a kiválasztott index szerint

```



```

*      a rekordot.
*      Olvasás után rögtön feloldjuk az olvasási
*      zárást.
*/
rc = isread(fdp, NULL, key, bufptr, FIND);
isrunlock(fdp);

if (rc != SUCCESS)
    if (rc == E_ENDKEY || rc == E_EMPTY ||
        rc == E_GREATER)
        printf("A keresett rekord nem letezik\n");
    else
        printf("Hiba a file olvasasa kozben\n");
return(rc);
}

int inputrec(bufptr)
/*
*      Adatrekord beolvasása
*/
char    *bufptr;
{
    struct FOKONYV *recptr;
    recptr = (struct FOKONYV *)bufptr;

    printf("fokonyvi_szam");
    scanf("%15.15s", recptr->fokonyvi_szam);
    printf("megnevezes");
    scanf("%30.30s", recptr->megnevezes);
    printf("kulso_merleg");
    scanf("%3.3", recptr->kulso_merleg);
    printf("belso_merleg");
    scanf("%3.3", recptr->belso_merleg);
    printf("nyito_egyenleg");
    scanf("%f", &recptr->nyito_egyenleg);
    printf("f_tartozik");
    scanf("%f", &recptr->f_tartozik);
    printf("f_kovetel");
    scanf("%f", &recptr->f_kovetel);
    printf("egyenleg_1");
    scanf("%f", &recptr->egyenleg_1);
    printf("egyenleg_2");
    scanf("%f", &recptr->egyenleg_2);
    printf("egyenleg_3");
    scanf("%f", &recptr->egyenleg_3);
    printf("f_tervezett");
    scanf("%f\n", &recptr->f_tervezett);
}

```


7. fejezet

Standard építőelemek és segédprogramok

Ebben a fejezetben a MISAM file rendszerhez csatlakozó adatkezelési programokat és segédprogramokat írjuk le, a felhasználáshoz kellő mélységben. Ezek a programok csak C file-ra használhatók, vagyis az adatokat a file-ban belső ábrázolásban kell tárolni. A dBASE file-okra a programok által megvalósított funkciót közvetlenül a dBASE-zel valósíthatjuk meg.

7.1. CHKFDF: tábla ellenőrző

7.1.1. A program funkciója

A program formai szempontból ellenőrzi a .FDF leíró file-t és szöveges formában kiírja a file paramétereit. Hibás sor esetén a program kiírja magát a sort és elegendő magyarázatot a hiba megtalálásához és kijavításához.

A program mindig csak az első hibás sorig vizsgálja a file-t. A hiba kijavítása után a programot újra le kell futtatni.

Ha a .FDF file formai szempontból helyes, akkor a program összegzésképpen szöveges formában kiírja a MISAM file paramétereit:

- file-ok nevei
- definiált indexek száma
- rekordmezők száma és típusa

7.1.2. A program indítása és opciói

Indítása:

CHKFDF <.FDF file neve>

Paraméterek:

A file névénél csak a nevet kötelező megadni, az fdf kiterjesztést a program feltételezi.

7.2. CREAISAM: MISAM file létrehozó

7.2.1. A program funkciója

A programmal a .FDF leírás alapján üres MISAM állományokat (adat- és indexfile-ok) hozhatunk létre.

7.2.2. A program indítása és opciói

Indítása:

CREAISAM [-d] <.FDF file neve>

Paraméterek:

A file névben csak a nevet kötelező megadni, kiterjesztésként a program feltételezi az fdf kiterjesztést.

-d opcionális paraméter. Megadása esetén a program nem kér megerősítést létező állományok felülírásához.

7.2.3. A kezelői párbeszéd

Ha a programot a -d opció nélkül indítottuk, akkor a létező állományok felülírása előtt megerősítést kér. Az állományt csak igenlő válasz esetén írja felül.

7.3. FED: interaktív file editor

7.3.1. A program funkciója

A file editor segítségével a MISAM könyvtári függvények által biztosított szolgáltatások majd mindegyikét elvégezhetjük párbeszédese formában. Ezáltal:

- Függvények leírását gyakorlatban kipróbálva, de tesztprogramok írása nélkül érthetjük meg. Ez megkönnyíti a MISAM programozásának elsajátítását.
- Megvizsgálhatjuk a file tartalmát, ellenőrizhetjük a bevitt adatok helyességét. Ez a programok tesztelését jelentősen lerövidítheti.
- Módosíthatjuk a file tartalmát (teszt)adatokat vihetünk be. A tesztadatok segítségével a fejlesztés alatt álló programot minden szituációban tesztelhetjük (például hibás értékek, foglalt állomány esete, stb.).

7.3.2. A program indítása és opciói

Indítása:

FED [(<.FDF file neve>)]

Paraméterek:

A .FDF file neve opcionális paraméter. Ha megadjuk, csak a név kötelező, az fdf kiterjesztést a program feltételezi. Ha a file-név tartalmaz ösvényt is, akkor a továbbiakban ezt a program automatikusan hozzáilleszti a .FDF file nevéhez. A paraméter megadása esetén a programban nem kell .FDF leíró file-t beolvasnunk.

7.3.3. A kezelői párbeszéd

A file editorral menüválasztó ablakokon és a standard adatbeviteli képernyőn keresztül kommunikálhatunk. A program alap ablaka a funkcióválasztó ablak.

Az ablak a file editorból meghívható függvények listáját tartalmazza. A funkciók között a cursor billentyűvel mozoghatunk és az ENTER billentyűvel választhatunk. A funkciókat természetesen csak helyes sorrendben tudjuk meghívni, nem tudunk file-t nyitni a .FDF file beolvasása előtt.

A file editor itt is és a többi menüválasztó ablaknál is megjegyzi a választást, így a funkciót csak az ENTER billentyűvel meg tudjuk ismételni.

A funkcióválasztás után a felhasználói párbeszéd funkcióként más és más. Vagy további -a file-től függő ablakokban- paramétereket választhatunk (például indexek közötti választás), vagy a standard adatbeviteli képernyő jelenik meg adatok megjelenítése vagy bevitele céljából.

Az ablakok és képernyők mindig elegendő információt tartalmaznak a megfelelő választáshoz.

A funkciók végrehajtása során a sikeres műveletről nem kapunk üzenetet, csak a hibaüzenetek jelennek meg a képernyőn.

7.4. ENTRY: standard adatfelvivő

isnEtwork	– hálózat vizsgálata
Get_fdf	– .FDF file beolvasása
iscreaT	– file létrehozása
isOpen	– file nyitása
isClose	– file zárása
isrlock	– olvasási zárás beállítása
isrunlock	– olvasási zárás feloldása
iswlock	– írási zárás beállítása
iswunlock	– írási zárás feloldása
isselInd	– kiválasztott index megadása
isFirst	– file elejére ugrás
isLast	– file végére ugrás
isRead	– rekord olvasása
isWrite	– új rekord bevitele
isDelete	– rekord törlése
isUpdate	– rekord módosítása

7.4.1. A program funkciója

A program feladata új rekordok bevitele MISAM állományokba. A bevétel a standard adatbeviteli képernyőn keresztül történik. A képernyőt a program a .FDF file-ban megadott információk alapján generálja. A program tetszőleges állomány felviteléhez használható, még akkor is, ha a rekordmezők száma a képernyő sorainak számát meghaladja.

7.4.2. A program indítása és opciói

Indítása:

ENTRY [.FDF file neve]

Paraméterek:

A program egy opcionális paramétert vár, a .FDF file nevét. A .FDF kiterjesztést nem kötelező megadni (ezt a program feltételezi), de a névnek tartalmaznia kell a teljes ösvényt.

Ha indításkor nem adunk paramétert, akkor a program először egy külön ablakban lekérdezi a .FDF file nevét. A kiterjesztés megadása itt sem kötelező.

Ha nem létező leíró file-t adunk meg, vagy a leíró file hibás, akkor a program futása hibaüzenet kiírásával befejeződik.

7.4.3. A kezelői párbeszéd

Ha helyes .FDF file-t adtunk meg, akkor indulás után megjelenik a .FDF file-ból a program által generált adatbeviteli képernyő. A főkönyvi törzslomány példájánál maradva az adatbeviteli képernyő a következő:

A képernyő bal oldalán láthatók .FDF file-ban megadott mezőnevek. A mezőnévvel egy sorban találhatjuk a mezőhöz az ún. input mezőt, amelynek segítségével a mezőnek értéket adhatunk.

A képernyő kezdetben üres és a cursor az első mezőn áll. A képernyőn egyszerre mindig csak egy rekord adatait tudjuk bevinni.

A rekordot az F2 billentyűvel írhatjuk ki a file-ba. A program csak minimális ellenőrzést végez az adatok helyességére vonatkozóan, így akár üres rekordot is kiírhatunk az állományba. Amíg az F2 billentyűvel nem írtuk ki a rekordot, addig bármelyik mező tetszés szerint változtatható.

A program két ellenőrzést végez el:

- a dátum típusú adatokat csak akkor fogadja el, ha a megadott érték tényleges dátum
- a numerikus mezőknél csak a szám jellegének megfelelő karaktereket fogadja el (például egész szám esetén csak számjegyeket és előjelet).

Általános adatfelvivő	
fokonyvi.szam	_____
megnevezes	_____
kulso_merleg	---
belso_merleg	---
nyito_egyenleg	_____
f_tartozik	_____
f_kovetel	_____
egyenleg_1	_____
egyenleg_2	_____
egyenleg_3	_____
f_tervezett	_____
F1: info F2: kiírás ESC: kiszállás	

A rekord kiírása után a képernyő törlődik, a cursor visszaáll az első mezőre és a program készen áll a következő rekord bevitelére.

Az adatbevitelt az ESC billentyűvel fejezhetjük be.

7.5. QUERY: lekérdező és módosító

7.5.1. A program funkciója

A program az állományok rekordjainak interaktív lekérdezésére, módosítására és törlésére alkalmas. A program a standard adatbeviteli képernyőt használja az adatok megjelenítéséhez és beviteléhez. A képernyőn egyszerre csak egy rekord adatai láthatók, de a rekordok között tetszés szerint mozoghatunk előre hátra. A rekordok elérésének sorrendjét szabadon választhatjuk meg.

7.5.2. A program indítása és opciói

Indítása:

QUERY [.FDF file neve]

Paraméterek:

A program egy paramétert vár, .FDF file nevét. A kiterjesztést nem kötelező megadni (a .FDF kiterjesztést a program feltételezi), de a névnek tartalmaznia kell a teljes ösvényt. Ha nem adunk paramétert, akkor a program induláskor egy külön ablakban lekérdezi a .FDF file nevét. Itt ugyanolyan formában adhatjuk meg, mint paraméterként. Ha a .FDF file nem létezik vagy hibás, akkor a program futása hibaüzenet kiírásával befejeződik.

7.5.3. A kezelői párbeszéd

Először a rendezettséget (elérési szempontot) kérdezi le a program. A főkönyvi törzsállományra a következő képernyő jelenik meg:

Az új index azt jelenti, hogy a rekordmezőkből a .FDF file-ban nem definiált indexet szeretnénk összeállítani. A következő sortól kezdve fel vannak sorolva a .FDF file-ban definiált indexek, mindegyik a magyarázattal azonosítva. Az index szerinti elérésen kívül lehetőség van még a szekvenciális minta szerinti elérés választására is.

Az ablak sorai között a cursor vezérlő billentyűkkel mozoghatunk. Egy keresési szempontot a megfelelő sorra pozicionálással és az ENTER billentyű leütésével tudunk kiválasztani.

A legegyszerűbb eset az, amikor valamelyik, a .FDF file-ban definiált index szerinti keresést választjuk. Ebben az esetben a program futása a kulcsérték megadásától folytatódik.

Keresési szempontok

Új index

Kulcs a fokonyvi szám szerint

Minta szerint

Ha új indexet szeretnénk összeállítani, akkor a kulcsérték megadása előtt definiálni kell az indexet és fel kell építeni a megfelelő indexfile-t. Az indexet definiáló képernyő a következő:

A képernyő két ablakra van osztva. Az alsó ablak a rekordmezőkhöz tartozó magyarázatokat tartalmazza. Ebben az ablakban lehet az index részeit kiválasztani. A mezők között a cursor billentyűvel pozícionálhatunk és az indexbe a mezőt az ENTER billentyűvel illeszthetjük be. Ha már nem akarunk több mezőt kijelölni, akkor az index összeállítását az ESC billentyűvel fejezhetjük be. Az ESC hatására a cursor a felső ablakba kerül.

A felső ablak, amely kezdetben üres, az index részeit tartalmazza. Egy indexdarabot egy sor ír le. Mivel az indexdarab típusa a mező típusával megegyezik, ezért csak az adatrekordbeli kezdőpozíció és a hossz van feltüntetve. A sorok között ismét a cursor billentyűvel mozoghatunk. Az INS billentyűvel az aktuális sor elé szúrhatunk be újabb indexdarabot, a DEL billentyűvel pedig törölhetjük azt.

Az indexdarab kezdőpozíciója és hossza változtatható, így nem csak teljes mezőt, hanem mező egy részét is kijelölhetjük indexdarabként. A megadott értékek nem léphetik át a mező határát és természetesen csak karakter típusú indexdaraboknál szabad ezeket az értékeket megváltoztatni.

Példánkban a megnevezés, külső és belső mérlegsorszám mezőket jelöltük ki indexdarabként.

A felső ablakból az ESC billentyűvel visszatérhetünk a keresési szempontok megadásához, az F2 billentyűvel pedig megadhatjuk az összeállított index még hiányzó információit.

a rekordokat szekvenciálisan érhetjük el. (Emiatt a keresés lényegesen lassúbb.) Index és kulcsérték helyett azoknak a rekordmezőknek kell értéket adnunk, amelyek a mintában szerepelnek. A minta szerinti lekérdezést általában akkor alkalmazzuk, ha az index túlságosan bonyolult volna, és/vagy csak néhány rekordról van szó, ezért ez a módszer gyorsabb az indexfile építésénél.

A standard adatbeviteli képernyőn ki kell tölteni azokat a rekordmezőket, melyek a mintának részei. Csak azok a rekordok fognak a képernyőn megjelenni, melyek megfelelő mezői megegyeznek a mintában megadott értékekkel. A ki nem töltött mező bármilyen értékkel megegyezik, vagyis ha nem adunk meg egyetlen értéket sem, akkor az összes rekord bele fog tartozni a mintába.

Az F2 billentyűvel tudjuk a keresést indítani. ESC-el a keresési szempontok megadásához térünk vissza.

Ettől a ponttól kezdve a program futása egységesen folytatódik, függetlenül a keresési szemponttól. A standard adatbeviteli képernyőn megjelenik az első, a keresési szempontnak megfelelő rekord.

A PgDn billentyűvel a következő, a file vége felé eső rekordra pozícionálhatunk. A PgUp billentyű hatására a file eleje felé mozgunk. Ha az adott irányban nincs több rekord, a program erről üzenetet küld. F7-tel az első, F8-cal az utolsó rekordra ugorhatunk.

A képernyő bármelyik mezője szabadon módosítható. A módosított rekordot az F2 billentyűvel írhatjuk ki az állományba. Az F3 billentyűvel a rekordot törölhetjük az állományból.

Hálózat esetén előfordulhat, hogy amíg mi a képernyőn módosítjuk a rekordot, addig egy másik állomásról azt módosítják vagy törlik. Ilyen esetben a mi módosításunk vagy törlésünk sikertelen lesz.

A keresést az ESC billentyűvel elhagyva új kezdő kulcsot vagy mintát adhatunk meg.

Standard query

fokonyvi_szam:	222
megnevezes:	Keszletek
kulso_merleg:	23
belso_merleg:	24
nyito_egyenleg:	2459.00
f_tartozik:	3445.00
f_kovetel:	6456.00
egyenleg_1:	3245.00
egyenleg_2:	4536.00
egyenleg_3:	134.00
f_tervezett:	456.00

F1:Info PGDN:Következő rekord PGUP:Előző rekord ESC:Új érték

7.6. LIST: állomány listázó program

7.6.1. A program funkciója

A LIST a MISAM adatállomány listázására alkalmas. A lista adattartalma és formája tetszés szerint változtatható.

Az adattartalmat illetően befolyásolni lehet:

- a listázás rendezettségét,
- a listázásra kerülő rekordok alsó és felső korlátját,
- a kiíratásra kerülő adatmezőket.

A listázás rendezettségét egy, a QUERY programnál ismerttetett párbeszéd-ben adhatjuk meg. Kérhetünk meglévő kulcs szerinti rendezettséget, kijelölhetünk új kulcsot, vagy előírhatunk szekvenciális, minta szerinti kiválasztási módszert. A QUERY-ben ismerttetett módon kérdezi le a program a kiválasztást behatároló alsó és felső korlátokat.

A kiíratásra kerülő adatmezőket és a listázás formáját egy .RPF listaleíró file-ból veszi a program. Ebben az editálható file-ban a Függelékben ismer-tetett módon definiálni lehet a listázás összes jellemzőjét.

A .RPF file-on kívül szükséges egy .RPD leíró file is, amely megadja a lista egy elemi sorának a felépítését. A .RPD file-t a FDFTORPD segédprogram segítségével automatikusan előállíttathatjuk a MISAM leíró (.FDF) file-ból. Ez is editálható file, de ezt a felhasználónak tilos változtatni, mivel a változtatást a LIST programban is át kellene vezetni.

A .RPF file egy nyers verziójának a létrehozására szolgál a FDFTORPF segédprogram.

A LIST indítása előtt ne felejtsek el tehát:

- az FDFTORPD programmal legeneráltatni a .RPD file-t,
- az FDFTORPF programmal legeneráltatni a .RPF file-t,
- editorral kialakítani a kívánt .RPF formát.

Amennyiben a .RPD és .RPF file-ok már léteznek, és a .RPF file-ban leírt listázási forma megfelelő, akkor természetesen a LIST azonnal indítható.

A lista konstansai (fejléc stb.) használhatunk ékezetes karaktereket is, DOG formában, azaz külön, előre ütött ékezetekkel. Az ékezetes karaktereket nyomtatónként más és más karaktersorozat kiírásával tudjuk megjeleníteni. Ezért a listázó szűrő (konverziós) file-okon keresztül nyomtat. A kiíratáshoz szükséges konverziós file-t a DOG rendszerben található CE-DIT segédprogrammal lehet létrehozni. A MISAM rendszerhez négy kész szűrő állományt mellékelünk:

- ASCII.CNV: kiíratás képernyős ékezetes karakterformára,
- WALTERS.CNV: kiíratás WALTERS (Videoton) 21200 típusú nyomta-tóra,
- EPSON.CNV: kiíratás EPSON RX 80 nyomtatóra,
- DUMMY.CNV: üres szűrőfile.

A megadott szűrőfile-ok egyben mintaként szolgálhatnak további konverziós file-ok definiálásához.

7.6.2. A program indítása és opciói

Indítása:

LIST <MISAM leíró file> [[<opciók>]

Paraméterek:

<MISAM leíró file> a .FDF leíró file nevét jelenti. A kiterjesztés a program által kikényszerítetten .FDF, nem is kell megadni.

A lehetséges opciók:

-r <listaleíró file neve>

A listaleíró file típusa kikényszerítetten .RPF, nem is kell megadni. A .RPD file-t ugyanolyan alapon és ugyanolyan ösvényen feltételezi a program. Ha nem adunk meg -r opciót, akkor a program a .FDF file nevéből képzi a listaleíró file-ok neveit.

-w <munkakönyvtár>

Az új kulcsállomány létrehozásához és a file-ba küldött listákhoz a program egy munkakönyvtárat használ. Ha nem adjuk meg, akkor az ideiglenes indexek és a listafájl-ok az aktuális könyvtárban képződnek.

-c <a konverziós file könyvtára>

A szűrőfájl az ékezetes karakterek és a betűtípus állítása miatt szükséges. Ha nem adunk meg könyvtárat, akkor a program az aktuális könyvtárban keresi a PRINTERTYPE-ban megadott alapnevű .CNV típusú szűrőfájl-t.

-h Az opciókat magyarázó help szöveg jelenik meg a képernyőn.

7.6.3. A kezelői párbeszéd

A LIST program interaktív része a listázandó rekordok kijelölésére vonatkozik, s mint ilyen, csaknem teljesen megegyezik a QUERY programban ismertetett párbeszéddel. Az egyetlen különbség, hogy a LIST-ben teljes intervallum kijelölés lehetséges, tehát felső határt is meg lehet adni.

7.7. TODBASE: adatkonverzió dBASE formába

7.7.1. A program funkciója

A program a C formátumú MISAM file-okból képez dBASE III által kezelhető adatfile-t. Az indexfile-okat külön, a dBASE nyújtotta funkciókkal kell létrehozni.

Mivel a MISAM file-ok felépítése dBASE kompatibilis, a konverter program használata csak az adat ábrázolásbeli eltérések miatt szükséges.

A program először létrehozza az .FDF file alapján, a megadott néven a dBASE adat állományt. A rekordmezők nevei az .FDF file-ban megadott nevek, tíz hosszúra csonkolva. Ezért figyelni kell arra, hogy az .FDF file-ban megadott nevek már tíz karakteren egyértelműen azonosítsák a mezőt.

A karakter típusú mezők a dBASE állományban is karakteresek lesznek az .FDF-ben megadott hosszúságban.

A dátum típusú mezők a MISAM négy byte hosszú ábrázolásából a dBASE nyolc byte hosszú karakteres dátum típusára konvertálódnak.

A numerikus típusú mezők a belső ábrázolásban tárolódnak. A konverzió során ezek a mezők a .FDF-ben megadott szélesség és tizedes pont utáni helyek száma alapján karakterláncként kerülnek a dBASE állományba, a mezők típusa a dBASE által kívánt numerikus (N) lesz.

7.7.2. A program indítása és opciói

Indítása:

TODBASE <.FDF file neve> <dBASE .FDF file neve>

Paraméterek:

Az file neveknél a kiterjesztését nem kell megadni, a program fdf kiterjesztést feltételez.

7.7.3. A kezelői párbeszéd

Ha a dBASE file létezik, akkor a program megkérdezi, hogy felülírja-e. Igenlő válasz esetén az állomány régi tartalma felülíródik. Ha nem írjuk felül az állományt, akkor a program megkérdezi, hogy indulhat-e a konvertálás. Igenlő válasz esetén a rekordok a dBASE állomány végéhez íródnak.

Futás közben a képernyő alján láthatjuk a már feldolgozott rekordok számát.

7.8. FRDBASE: adatkonverzió dBASE formából

7.8.1. A program funkciója

A program dBASE III állományokból képez MISAM C file típusú állományokat. A feldolgozás a .FDF leíró file alapján történik. A program a dBASE adatfile-ból a MISAM adatfile-t, valamint a hozzá tartozó indexfile-okat hozza létre.

Fontos megkötés, hogy a dBASE rekord nem tartalmazhat MEMO és LOGIC típusú mezőket. A LOGIC típusú mezőket célszerű egy hosszúságú karakter típusú mezővel kiváltani.

A karakteres típusú mezők változatlan formában kerülnek át a MISAM file-ba, de .FDF file-ban megadott hosszúságban.

A dBASE dátum típusú mezői-ből négy byte-os belső ábrázolású értékek lesznek.

A numerikus mezők a .FDF file-ban megadott típusoknak megfelelő belső ábrázolású értékékké konvertálódnak.

7.8.2. A program indítása és opciói

Indítása:

FRDBASE <.FDF file neve> <dBASE .FDF file neve>

Paraméterek:

Az file neveknél a kiterjesztését nem kell megadni, a program fdf kiterjesztést feltételez.

7.8.3. A kezelői párbeszéd

Ha a MISAM file létezik, akkor a program megkérdezi, hogy felülírja-e. Igenlő válasz esetén a MISAM állomány régi tartalma felülíródik. Ha nem írjuk felül az állományt, akkor a program megkérdezi, hogy indulhat-e a konvertálás. Igenlő válasz esetén a rekordok a MISAM file végéhez íródnak. Futás közben a képernyő alján láthatjuk a már feldolgozott rekordok számát.

7.9. TODOS: adatkonverzió DOS formába

7.9.1. A program funkciója

A program a MISAM állományokból képez standard ASCII formátumú szövegfájl-t, a .FDF leírás alapján. Az így képzett szövegfájl szabadon átvihető akár más operációs rendszer alá is.

7.9.2. A program indítása és opciói

Indítása:

TODOS [-v"<sepchar>"] [<.FDF file neve>] [<ASCII file neve>]

Paraméterek:

A fájl-neveknél a kiterjesztését nem kell megadni. A program a .FDF-nél fdf, az ASCII fájl-nál txt kiterjesztést feltételez.

-v "<sepchar>"

Opcionális paraméter, a rekordmezőket elválasztó karaktert adhatjuk meg. Az elválasztó karakter alapértelmezése a vessző (,) karakter.

7.9.3. A kezelői párbeszéd

Ha az ASCII fájl létezik, akkor a program megkérdezi, hogy felülírja-e. Igenlő válasz esetén az állomány régi tartalma felülíródik. Ha nem írjuk felül az állományt, akkor a program megkérdezi, hogy indulhat-e a konvertálás. Igenlő válasz esetén a rekordok az állomány végéhez íródnak.

Futás közben a képernyő alján láthatjuk a már feldolgozott rekordok számát.

7.10. FROMDOS: adatkonverzió DOS formából

7.10.1. A program funkciója

A program standard ASCII szövegfile-t konvertál MISAM állományba a .FDF file alapján. A program létrehozza az adatfile-t és indexfile-okat is.

Az ASCII rekord mezőinek és a MISAM rekord mezőinek páronként egyezniük kell. Eltérés esetén a program futása hibaüzenet kiírásával befejeződik. Az ilyen esetben létrehozott MISAM állomány sérült, nem használható.

7.10.2. A program indítása és opciói

Indítása:

FROMDOS [-v"<sepchar>"] <.FDF file neve> <ASCII file neve>

Paraméterek:

Az file neveknél a kiterjesztését nem kell megadni. A program a .FDF-nél fdf, az ASCII file-nál txt kiterjesztést feltételez.

-v "<sepchar>"

Opcionális paraméter, a rekordmezőket elválasztó karaktert adhatjuk meg. Az elválasztó karakter alapértelmezése a vessző (,) karakter.

7.10.3. A kezelői párbeszéd

Ha a MISAM file létezik, akkor a program megkérdezi, hogy felülírja-e. Igenlő válasz esetén az állomány régi tartalma felülíródik. Ha nem írjuk felül az állományt, akkor a program megkérdezi, hogy indulhat-e a konvertálás. Igenlő válasz esetén a rekordok az állomány végéhez íródnak.

Futás közben a képernyő alján láthatjuk a már feldogozott rekordok számát.

7.11. REBUILD: állomány újra építése

7.11.1. A program funkciója

A program a MISAM állomány adatfile alapján újraépíti a .FDF file-ban definiált indexeket. Az indexépítést csak akkor szabad elindítani, ha a file-t más munkaállomás nem használja. Az indexfile építés alatt az állomány írásra zárt, vagyis a többi munkaállomás nem tudja elérni.

7.11.2. A program indítása és opciói

Indítása:

REBUILD ⟨.FDF file neve⟩

Paraméterek:

A file-névnél elég csak a nevet megadni, a program az fdf kiterjesztést automatikusan feltételezi.

7.12. COMPRESS: állomány tömörítése

7.12.1. A program funkciója

A rekordok törlése során a MISAM állomány adatfile-jában "lyukak" keletkeznek, mivel a MISAM a törölt adatrekordok helyét csak akkor allokalja újra, ha az az utolsó aktív rekord után van. A COMPRESS program úgy rendezi át az adatrekordokat, hogy a törölt rekordok a file végére kerülnek. A tömörítés után a file méret nem csökken. A program futtatásával azt érhetjük el, hogy új rekord felvitele esetén újra allokalódnak a törölt rekordok helyei, ezért a file mérete nem növekszik.

Az adatrekordok átmozgatása miatt a program újraépíti az indexfile-okat is.

A tömörítést csak akkor szabad elindítani, ha másik munkaállomás nem használja a file-t. A tömörítés alatt a file írásra zárt, vagyis a más munkaállomás nem tudja elérni.

7.12.2. A program indítása és opciói

Indítása:

COMPRESS <.FDF file neve>

Paraméterek:

A file-névnél elég csak a nevet megadni, a program az fdf kiterjesztést automatikusan feltételezi.

7.13. FDFTOH: include file generátor

7.13.1. A program funkciója

A MISAM leíró .FDF file-ból egy, a C programokban használható struktúra-definíciót és hosszkonstansokat állít elő.

Példa:

A főkönyvi állomány .FDF file-jából generált header file a következő:

```
/* FOKONYV.H */
```

```
struct FOKONYV {  
    char fokonyvi_szam[15];  
    char megnevezes[30];  
    char kulso_merleg[3];  
    char belso_merleg[3];  
    double nyito_egyenleg;  
    double f_tartozik;  
    double f_kovetel;  
    double egyenleg_1;  
    double egyenleg_2;  
    double egyenleg_3;  
    double f_tervezett;  
};  
#define L_fokonyvi_szam 15  
#define L_megnevezes 30  
#define L_kulso_merleg 3  
#define L_belso_merleg 3  
#define L_nyito_egyenleg 8  
#define L_f_tartozik 8  
#define L_f_kovetel 8  
#define L_egyenleg_1 8  
#define L_egyenleg_2 8  
#define L_egyenleg_3 8  
#define L_f_tervezett 8
```

A struktúra használatánál figyeljünk arra, hogy a MICROSOFT C fordító csak külön opció esetén tárolja tömören (határra igazítás nélkül) a struktúraelemeket. A MISAM könyvtári rutinok viszont tömör adatrekordokat feltételeznek. Emiatt a programokat a **Zp** opció megadásával kell fordítani.

7.13.2. A program indítása és opciói

Indítása:

FDFTOH <MISAM leíró file neve> [opciók]

Paraméterek:

<MISAM leíró file> a .FDF leíró file nevét jelenti. A kiterjesztés a program által kikényszerítetten .FDF, nem is kell megadni.

A lehetséges opciók:

- o <include file neve>

Az include file típusa kikényszerítetten .H, nem is kell megadni.

Ha nem adunk meg -o opciót, akkor a program a .FDF file nevéből képi az include file nevét.

- h Az opciókat magyarázó help szöveg jelenik meg a képernyőn.

7.14. FDFTORPD: RPD file generátor

7.14.1. A program funkciója

A MISAM definíciós file segítségével egy elemi sor leíró .RPD file-t állít elő. A .RPD file a lista kiíráshoz szükséges. A felhasználó ezt a file-t nem változtathatja! A .FDF-től elkülönült .RPD file egyébként azért szükséges, mert a lista háttérét képező listagenerátor potenciálisan nem csak file bufferként létrejövő elemi sorok kiírására alkalmas.

Példa:

A főkönyvi törzsszállomány alapján generált .RPD file:

**** FOKONYV.RPD ****

*

DETAILFIELDS=11

FIELD=fokonyvi_szam	0	15	1	A	15	0	L	0	0
FIELD=megnevezes	15	30	1	A	30	0	L	0	0
FIELD=kulso_merleg	45	3	1	A	3	0	L	0	0
FIELD=belso_merleg	48	3	1	A	3	0	L	0	0
FIELD=nyito_egyenleg	51	8	1	D	12	2	R	0	0
FIELD=f_tartozik	59	8	1	D	14	2	R	0	0
FIELD=f_kovetel	67	8	1	D	14	2	R	0	0
FIELD=egyenleg_1	75	8	1	D	14	2	R	0	0
FIELD=egyenleg_2	83	8	1	D	14	2	R	0	0
FIELD=egyenleg_3	91	8	1	D	14	2	R	0	0
FIELD=f_tervezett	99	8	1	D	14	2	R	0	0

7.14.2. A program indítása és opciói:

Indítása:

FDFTORPD <MISAM leíró file neve> <opciók>

Paraméterek:

<MISAM leíró file> a .FDF leíró file nevét jelenti. A kiterjesztés a program által kikényszerítetten .FDF, nem is kell megadni.

A lehetséges opciók:

-o <az output file neve>

Az output file típusa kikényszerítetten .RPD, nem is kell megadni.

Ha nem adunk meg -o opciót, akkor a program a .FDF file nevéből képi az output file nevét.

-h Az opciókat magyarázó help szöveg jelenik meg a képernyőn.

7.15. FDFTORPF: RPF file generátor

7.15.1. A program funkciója

A MISAM definíciós file segítségével egy kiindulási .RPF file-t állít elő. A .RPF file nem véglegesnek szánt, a felhasználónak nagy valószínűséggel módosítani kell. A fejléc az összes adatnevet tartalmazza, az elemi sorban az összes elemi adat kiíródik. Fejléc és adat illesztés nincs. A nyomtatás tömörített írásmódban, WALTERS típusú lokális nyomtatóra történik.

Példa:

A főkönyvi törzsállományból generált nyers .RPD file:

```
***** FOKONYV.RPF *****
```

```
*
```

```
DEVICE=LOCALPRINTER
```

```
PRINTMODE=CONDENSED
```

```
PAGELength=65
```

```
PRINTERTYPE=WALTERS
```

```
*
```

```
HEADERS=\
```

```
FOKONYV ALLOMANY LISTA\
```

```
#date#
```

```
#pageno#.lap\n\n\
```

```
fokonyvi_szam \
```

```
megnevezes \
```

```
kulso_merleg \
```

```
belso_merleg \
```

```
nyito_egyenleg \
```

```
f_tartozik \
```

```
f_kovetel \
```

```
egyenleg_1 \
```

```
egyenleg_2 \
```

```
egyenleg_3 \
```

```
f_tervezett \
```

```
\n
```

```
*
```

```
DETAIL=\
```

```
@fokonyvi_szam@ \
```

```
@megnevezes@ \
```

```
@kulso_merleg@ \
```

```
@belso_merleg@ \
```

```
@nyito_egyenleg@ \
```

```

@f_tartozik@ \
@f_kovetel@  \
@egyenleg_1@ \
@egyenleg_2@ \
@egyenleg_3@ \
@f_tervezett@ \
\n
*
```

A .RPF file leírása a függelékben található.

7.15.2. A program indítása és opciói:

Indítása:

FDFTORPF <MISAM leíró file neve> <opciók>

Paraméterek:

<MISAM leíró file> a .FDF leíró file nevét jelenti. A kiterjesztés a program által kikényszerítetten .FDF, nem is kell megadni.

A lehetséges opciók:

-o <az output file neve>

Az output file típusa kikényszerítetten .RPF, nem is kell megadni.

Ha nem adunk meg -o opciót, akkor a program a .FDF file nevéből képzi az output file nevét.

-h Az opciókat magyarázó help szöveg jelenik meg a képernyőn.

8. fejezet

Technikai információk

8.1. Korlátozások

C file esetén:

- Egy MISAM állomány minimálisan egy adat- és egy indexfile-ból áll.
- Az adatrekord mezőinek száma maximum 128.
- Az adatrekord maximális mérete 4000 byte.
- Az indexek maximális hossza 100 byte.
- Ahivatkozási név maximum 15 értékes karakter lehet. A hosszabb neveket a file rendszer csonkolja.
- Afile nevekhez, stb. fűzött magyarázatok hossza maximálisan 79 byte.

dBASE file esetén:

- Egy MISAM állomány minimálisan egy adat- és egy indexfile-ból áll.
- Az adatrekord mezőinek száma maximum 128.
- Mező típusa a következők valamelyike lehet:
 - o dBASE karakter típus, kódja 'C'
 - o dBASE numerikus típus, kódja 'N'
 - o dBASE dátum típus, kódja 'D'
- Az adatrekord maximális mérete 4000 byte.
- Egy adatfile-hoz maximálisan hét indexet definiálhatunk.
- Az indexek maximális hossza 100 byte.
- Az index csak karakteres típus esetén állhat több részből.
- Dátum vagy numerikus kulcs esetében az index csak egy darabból állhat.
- Ahivatkozási név maximum 10 értékes karakter lehet. A hosszabb neveket a file rendszer csonkolja.
- Afile nevekhez, stb. fűzött magyarázatok hossza maximálisan 79 byte.
- Az index leírásánál a magyarázat helyén a dBASE szintaktikának megfelelő kulcskifejezést kell megadni.

8.2. Externális változók

A MISAM könyvtár az alábbi globális neveket foglalja. Ezek a felhasználói C programok számára tiltott szimbólumok.

Függvénynevek:

- isgetfdf
- isbuild
- isclose
- iscreat
- isdatetokey
- isdelete
- isfirst
- isinfo
- iskeyto date
- islast
- isnetwork
- isopen
- isread
- isrlock
- isrunlock
- isselind
- isupdate
- iswlock
- iswrite
- iswunlock
- MISAM_blkio
- MISAM_chkeycomp
- MISAM_closeix()
- MISAM_clrix()
- MISAM_creatix()
- MISAM_dbgenkey
- MISAM_delix
- MISAM_flushix
- MISAM_genkey
- MISAM_getblk
- MISAM_insix
- MISAM_keycomp
- MISAM_keycopy
- MISAM_memcpy
- MISAM_memset
- MISAM_nkeycomp
- MISAM_openix()
- MISAM_readdhead
- MISAM_recio

MISAM_release
MISAM_selind()
MISAM_sysdate
MISAM_wrdhead

A MISAM kezdetű függvénynevek nem alkotják részét a programozói interface-nek. Ezek MISAM belső használatú függvények.

Globális változók:

MISAM_dbflag
MISAM_kdptr

8.3. MISAM belső file szerkezet

A MISAM állomány adat- és indexfile-jainak szerkezete a dBASE file-ok szerkezetével megegyezik. A különbség a C file-ok és dBASE file-ok között, hogy az előbbiek az adatokat az adatrekordban a MICROSOFT C 4.00 verzió belső ábrázolásának megfelelő formában várják. A dBASE file-okban rekordjaiban minden adat karakteres formában van jelen.

Az adatfile esetében:

- ha a file C file, akkor az első byte 0xFF
- ha a file dBASE file, akkor az első byte 0X03

Az első byte-ot követi az adatfile fejblokk majd a rekordmezők leírása. Az első adatrekord pozíciója a rekordmezők számától függ.

Az indexfile teljesen dBASE szervezésű, de az adatok C file esetében itt is belső ábrázolásban tárolódnak. Létrehozáskor a kulcskifejezés helyére a .FDF file-ban az index magyarázataként megadott szöveg kerül. C file esetében mindegy, hogy mi a kulcskifejezés, de dBASE file esetén a kulcskifejezésnek a dBASE számára elfogadható kifejezésnek kell lennie.

9. fejezet

FÜGGELÉK

9.1. A standard adatbeviteli képernyő

A standard adatbeviteli képernyő teljes képernyős kényelmes adatbevitelt és lekérdezést tesz lehetővé. Egy képernyő mindig egy rekord. A képernyő szövegkonstansokból és mezőkből áll (lehetnek védett és módosítható mezők). A módosítható mezők között a cursor vezérlő billentyűkkel mozoghatunk. A funkcióbillentyűkhöz funkciókat rendelhetünk.

A mezők közötti mozgás:

- következő mező: le nyíl vagy ENTER
- előző mező: fel nyíl
- első mező: HOME
- utolsó mező: END

Egy mező értékének megadásához vagy módosításához be kell lépni a mezőbe. Ez a jobb nyíl billentyűvel, vagy az érték megadásának elkezdésével történhet.

A dátumok megadása ellenőrzéssel történik, csak helyes dátumot tudunk megadni. A dátum formája: éé/hh/nn.

A numerikus értékeknél csak a mező típusának megfelelő karaktereket tudunk megadni. (Például egész érték esetén számjegyek és előjel.)

A kiszállás az ESC billentyűvel történik.

9.2. A standard ASCII file-forma

A standard ASCII formátumú file felépítése:

- Akarakteres mezőket kettős aposztrófok vesznek közre. Ha a karakterlánc aposztróf karaktert tartalmaz, akkor ez duplán jelenik meg. (a" a, "a" "a")
- Adátum típusú mezők éé/hh/nn formájúak.

- Anumerikus mezők az .FDF ben megadott szélesség és tizedesjegyek számának megfelelően karakteres formájúak.
- Amezőket elválasztó karakter választja el (általában vessző).
- Minden rekord kötelezően új sorban kezdődik.
- Egy rekord több sorba is eltörhető. A törés mezőhatárnál vagy karakterlánc közben lehetséges.

Példa:

"11	", "Ingatlanok	",
23456773.00,88/02/23		
"311	", "Elszamolasi betet	",
2596000.00,88/03/24		
"314	", "Hazi penztar	",
2345.00,88/05/12		

Egy rekord két karakteres, egy double és egy dátum típusú mezőt tartalmaz.

9.3. A listaleírás szabályai

Az alábbiakban röviden ismertetjük a .RPF file-okban megadott listaleírás szabályait.

A leírás megadja:

- Alistázó eszköz paramétereit,
- Afejléc formáját és adattartalmát,
- Az elemi sor formáját és adattartalmát,
- Az összegző sorok formáját, adattartalmát valamint azt, hogy a rendszer mikor írassa ki az összegeket (a kontrollváltási ismérv).

A leírás formája:

Szöveges (editált) file, amelyben a

#PAGENO#

(aktuális lapszám),

#LINENO# (aktuális sorszám),

#DATE# (rendszerdátum)

rendszerváltozók hivatkozhatók, s amelyben az elemi sor és felhasználói buffer adatai a @adatnev@-vel adhatók meg. Az adathivatkozások a megadott sorpozícióban értelmeződnek. Az összegző (total) adatok a @total(szint,adatnev)@ formájában hivatkozhatók. A régi elemi sor adatainak hivatkozása a @old.adatnev@ formában történik.

A formátum megadásakor a <backslash>n új sort jelez, a <backslash> folytatósort.

A definíciós kulcsszavak:

DETAIL=

az elemi sor formátumának a leírása

HEADER=

a fejléc formátumának a leírása

TOTAL=<n>

az n-edik összegző sor leírásának kezdete, n=1-től

kezdődhet, folyamatosan felfelé, a GRANDTOTAL a 10-es.

A GRANDTOTAL-hoz nem kell folyamatosság.

DEVICE=LOCALPRINTER,GLOBALPRINTER,filenév

A nyomtatási eszközt adja meg. Munkaállomáshoz kötött nyomtató esetén nem használ spool-t a kiíráshoz. File-ba küldött lista munkakönyvtárban fog képződni. A munka-

könyvtár nevét opcióként lehet megadni.

PRINTMODE=NORMAL,ELITE,CONDENSED

A betűnagyságot és ezen keresztül az egy sorba íratható karakterek számát állíthatjuk be.

PAGELNGTH=laphossz

A lap hosszának a megadása.

PRINTERTYPE=WALTERS,EPSON,...

A nyomtató típusát a nyomtatóhoz tartozó szűrőfile nevével adjuk meg. A MISAM-ban megadott szűrőfile-ok: WALTERS, EPSON, ASCII, DUMMY. A DOG rendszer (egy MAGISZTER által forgalmazott dokumentáció generátor) CEDIT segédprogramjával tetszés szerinti szűrőfile-t hozhatunk létre.

A TOTAL képzés leírása is a lista jellemzője. A TOTAL line-ok leírásához tartoznak, ott kell megadni őket.

CONTROL_ON=

a kontrollváltozó neve

TOTAL_ON=

az összegzendő adat neve. A név után, vesszővel elválasztva megadható a kiírási mező szélesség, a dec. jegyek száma, igazítás (R: jobbra, L: balra).

Akárhány TOTAL_ON lehet egy TOTAL-on belül. A GRANDTOTAL-ban a CONTROL_ON nem értelmes.

A felsőbb szinthez tartozó TOTAL-ok nem képeznek automatikusan alsóbb szintű TOTAL-okat. Minden megadott TOTAL külön kumulálódik, a TOTAL sor kiírásakor nullázódnak.

A TOTAL sorok láncreakcióval íródnak ki, tehát pl. a 4-es szintű TOTAL kiírási feltételének bekövetkezésekor az alsóbb szintű TOTAL-ok (1,2,3) is kiíródnak.

9.4. A disztributív anyag ismertetése

A disztributív anyag három lemezből áll.

Első lemez:

MISAM.H

header file

DATE.H

header file

SMISAM.LIB

small memória modelles könyvtár

MMISAM.LIB

medium memória modelles könyvtár

LMISAM.LIB

large memória modelles könyvtár

CHKFDF.EXE

.FDF file ellenőrző

CREAISAM.EXE

állományt létrehozó program

FED.EXE

interaktív file editor

COMPRESS.EXE

file tömörítő

REBUILD.EXE

indexfile újraépítő

Második lemez:

ENTRY.EXE

standard adatfelvivő

QUERY.EXE

standard lekérdező és módosító

LIST.EXE

standard listázó

TODBASE.EXE

konverzió dBASE formátumra

FRDBASE.EXE

konverzió dBASE formátumról

Harmadik lemez:

TODOS.EXE

konverzió ASCII formára

FROMDOS.EXE

konverzió ASCII formáról

FDFTOH.EXE

C header file generátor

FDFTORPD.EXE

lista leíró generátor

FDFTORPF.EXE

lista leíró generátor

WALTERS.CNV

WALTERS (VT21200) nyomtató

konverziós file-ja

EPSON.CNV

EPSON (RX 80) nyomtató

konverziós file-ja

ASCII.CNV

szűrőfile képernyőre íráshoz

DUMMY.CNV

üres szűrőfile

Tárgymutató

A

ASCII file 19, 26, 124, 125, 137
 adatbeviteli képernyő 24, 110, 112,
 113, 118, 137
 aktuális kulcspozíció 45, 46, 60, 71,
 73, 83, 89
 aktuális rekord 22

B

B-tree 13
 bináris adatátvitel 14

C

C file 35
 CEDIT 120, 139
 CHKDFDF 20
 COMPRESS 12, 19, 20, 27, 127
 CREAISAM 20
 C file 14, 15, 26, 31, 57, 65, 73, 107,
 122, 123, 133, 135
 C típus 13, 14, 30

D

DATE.H 30, 39
 Descriptor File 17, 19
 DOG 120, 139
 dBASE file 7, 11, 13, 14, 15, 19, 26,
 32, 35, 57, 59, 65, 67, 73, 89,
 122, 123, 133, 135
 dBASE típus 14, 123, 133

E

ENTRY 12, 24

F

FDF 7, 17, 19, 20, 22, 24, 25, 26, 27,
 29, 30, 46, 63, 71, 81, 107, 108,
 110, 120, 137
 FDFTOH 19, 27
 FDFTORPD 19, 25, 27, 120
 FDFTORPF 19, 25, 27, 120
 FED 12, 19, 23
 FRDBASE 19, 26
 FROMDOS 19, 26

file eleje pozíció 45, 62, 71

file vége pozíció 45, 68

H

hivatkozási név 30, 31, 46, 51, 62,
68, 73, 81

I, Í

írási művelet 47, 86

írási zárás 13, 47, 48, 51, 57, 60, 77,
79, 83, 86, 89, 91, 125, 127

isbuild 22, 47, 51, 81

isclose 22, 47, 55, 71, 78, 91

iscreat 22, 57, 58, 63, 65

isdatetokey 22, 59, 67, 73

isdelete 22, 47, 60

isfirst 22, 62, 68, 75, 81

isgetfdf 22, 57, 58, 63, 65, 71, 77, 86

isinfo 47, 65, 78, 79, 87, 91

iskeytodate 22, 59, 67, 73

islast 22, 62, 68, 75, 81

isnetwork 22, 63, 69, 77, 78, 79, 86,
87, 91

isopen 22, 47, 55, 57, 58, 61, 63, 71,
78, 79, 81

isread 22, 45, 47, 52, 59, 61, 62, 67,
68, 73, 81, 84

isrlock 22, 47, 69, 71, 75, 77, 79, 87,
91

isrunlock 22, 47, 69, 71, 75, 77, 78,
79, 87, 91

isselind 22, 52, 75, 81

isupdate 22, 47, 83

iswlock 22, 47, 52, 58, 61, 69, 78, 79,
84, 86, 90, 91

iswrite 22, 89

iswunlock 22, 47, 52, 58, 61, 69, 78,
79, 84, 86, 87, 90, 91

K

kiválasztott index 22, 46, 51, 60, 62,
68, 71, 73, 81, 83

L

LIST 12, 25, 119, 120, 121

LMISAM.LIB 12

leíró file 25, 26

lokális indexfile 51, 52, 55

M

MICE 11

MISAM állomány 14, 16, 17, 19, 20,
21, 22, 23, 25, 26, 29, 30, 47,
108, 124

MISAM file 13, 46

MISAM.H 39, 46, 57, 65, 73

MLFORM 12

MLQUERY 12

MLREP 12

MMISAM.LIB 12

minta szerinti keresés 11

N

NOVELL 11, 22, 46, 69

O

RPD 7, 27, 120

RPF 7, 27, 120, 138

olvasási művelet 47, 71

olvasási zárás 13, 47, 71, 73, 77, 79,
86, 91,

Q

QUERY 12, 24, 25, 117, 120, 121

R

REBUILD 12, 19, 26

S, Sz

SMISAM.LIB 12

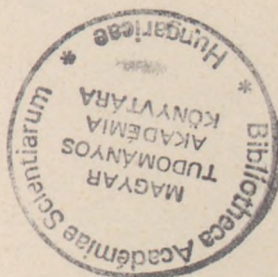
szűrőfile 25, 120, 139

T

Típuskód 31, 32

TODBASE 19, 26

TODOS 19, 26



AKADÉMIAI KIADÓ ÉS NYOMDA VÁLLALAT

Felelős kiadó: Hazai György főigazgató

Felelős szerkesztő: Bencsik Ágnes

A borítót Szabadi Zoltán tervezte

ISBN 963 05 5245 0

ISSN 0238-3896

A kézirat lezárva: 1988. szeptember 15.

MISAM programcsomag

Index szekvenciális file kezelés C nyelven,
egymunkahelyes és NOVELL hálózatos rendszerekhez.

Az adatfeldolgozási alkalmazások létfontosságú alapeleme az indexelt szekvenciális file rendszer. Számos változatuk található a piacon. A MISAM ezek közül az alábbi tulajdonságokkal tűnik ki:

- A MISAM-ban az adatfile és a hozzátartozó indexfile-ok képeznek egy "ISAM"-ot. Az ISAM összetevői mindig együtt mozognak. Egyszerre több ISAM lehet megnyitva.
- A MISAM-ot használó programok újrafordítás nélkül futtathatók egymunkahelyes vagy NOVELL hálózatos gépeken.
- Kedvező áron kínál a drága rendszerekkel egyenértékű (vagy jobb) programozási komfortot, igen egyszerű és hatékony ISAM funkciói révén.
- A felhasználói segédprogramok gazdag gyűjteménye áll rendelkezésre. Minden segédprogram NOVELL alatt is futásképes. A segédprogramok kezelési komfortja kiemelkedően jó. Több segédprogram (pl. a lekérdező ill. a listázó) átlagon felüli szolgáltatásokat biztosít.

A MISAM rendszer két részből tevődik össze:

1. A MISAM könyvtár

Tartalmazza az indexelt szekvenciális file kezelő modulokat és több, a C nyelvi programozást segítő illesztő rutint.

2. MISAM alsó és felső szintű segédprogramok

A segédprogramok gazdag HELP támogatással, egy vezérlőprogram által összefogottan, könnyen kezelhetően állnak rendelkezésre.

Opcionálisan elérhető a file-rendszer memóriarezidens változata is.

HARDVER: IBM PC/XT,AT vagy velük kompatibilis mikroszámítógép.
Garanciális szolgáltatások. Szoftverkövetés.

