Solutions

Answer 3.1

(a) The following SQL statements create the corresponding relations.

```
CREATE TABLE Professors (   prof_ssn CHAR(10),
                            name CHAR(64),
                            age INTEGER,
                            rank INTEGER,
                            speciality CHAR(64),
                            PRIMARY KEY (prof ssn) )

CREATE TABLE Depts (        dno INTEGER,
                            dname CHAR(64),
                            office CHAR(10),
                            prof_ssn CHAR(10) NOT NULL,
                            PRIMARY KEY (dno),
                            FOREIGN KEY (prof_ssn) REFERENCES Professors
                        (prof_ssn) )
CREATE TABLE Work_Dept (  dno INTEGER,
                            prof_ssn CHAR(10),
                            pc_time INTEGER,
                            PRIMARY KEY (dno, prof_ssn),
                            FOREIGN KEY (prof_ssn) REFERENCES
                                    Professors (prof_ssn),
                            FOREIGN KEY (dno) REFERENCES Depts)
```

Observe that we would need check constraints or assertions in SQL to enforce the rule that Professors work in at least one department.

```
CREATE TABLE Project(       pid INTEGER,
                            sponsor CHAR(32),
                            start date DATE,
                            end date DATE,
                            budget FLOAT,
                            prof_ssn CHAR(10) NOT NULL,
                            PRIMARY KEY (pid),
                            FOREIGN KEY (prof_ssn) REFERENCES Professors
                        (prof_ssn))
CREATE TABLE Graduates(grad_ssn CHAR(10),
                            -- needed by the Advisor relationship
                            senior_ssn CHAR(10) NOT NULL,
                            age INTEGER,
                            name CHAR(64),
                            deg_prog CHAR(32),
                            major INTEGER NOT NULL,
                            PRIMARY KEY (grad_ssn),
                            -- needed by the Major relationship
                            FOREIGN KEY (major) REFERENCES Depts (dno),
                            -- needed by the Advisor relationship
```

```
                                  FOREIGN KEY (senior_ssn) REFERENCES Graduates
(grad_ssn)
                                  )
```
Note that the Major table is not necessary since each Graduate has only one major and so this can be an attribute in the Graduates table. Same applies for the Advisor table.
```
CREATE TABLE Work_In ( pid INTEGER,
                                  prof ssn CHAR(10),
                                  PRIMARY KEY (pid, prof_ssn),
                                  FOREIGN KEY (prof_ssn) REFERENCES Professors
                         (prof_ssn),
                                  FOREIGN KEY (pid) REFERENCES Project (pid) )
```
Observe that we cannot enforce the participation constraint for Projects in the Work_In table without check constraints or assertions in SQL.
```
CREATE TABLE Supervises (  prof_ssn CHAR(10) NOT NULL,
                                          grad_ssn CHAR(10),
                                          pid INTEGER,
                                  PRIMARY KEY (grad_ssn, pid),
                                  FOREIGN KEY (prof_ssn) REFERENCES Professors
                         (prof_ssn),
                                  FOREIGN KEY (grad_ssn) REFERENCES Graduates
                         (grad_ssn),
                                  FOREIGN KEY (pid) REFERENCES Projects (pid) )
```
Note that we do not need an explicit table for the Work Proj relation since every
time a Graduate works on a Project, he or she must have a Supervisor.

(b)
Rows of the Professors table:
"P1", "Marc Johnson", 48, 50, "Databases"
"P2", "Anna Pacson", 45, 50, "Communications"
"P3", "Paul Lonson", 44, 45, "Smart homes"

Rows of the Depts table:
1, "Computer Science", "CAMPUS1", "P1"
2, "Electrical Engineering", "CAMPUS2", "P2"
3, "Architecture", "CAMPUS3", "P3"

Rows of the Work_Dept table:
1, "P1", 50
2, "P1", 25
3, "P1", 25
2, "P2", 100
3, "P3", 100

Rows of the Project table:
1, "NSF", "01/01/2015", "01/01/2020", 800000, "P1"

Rows of the Graduates table:

"S1", "S1", 30, "John Marcson", "Postdoc", 1

Rows of the Work_In table:
None

Rows of the Supervises table:
"P1", "S1", 1

Answer 3.2

The statements to create tables corresponding to entity sets Doctor, Pharmacy, and Pharm co are straightforward and omitted. The other required tables can be created as follows:

```
CREATE TABLE Pri_Phy_Patient ( ssn CHAR(11),
                                name CHAR(20),
                                age INTEGER,
                                address CHAR(20),
                                phy_ssn CHAR(11) NOT NULL,
                                PRIMARY KEY (ssn),
                                FOREIGN KEY (phy_ssn)REFERENCES Doctor
                                (phy_ssn))
CREATE TABLE Prescription ( ssn CHAR(11),
                             phy_ssn CHAR(11),
                                date CHAR(11),
                                quantity INTEGER,
                                trade_name CHAR(20),
                                pharm_id CHAR(11),
                                PRIMARY KEY (ssn, phy_ssn, trade_name, pharm_id),
                                FOREIGN KEY (ssn) REFERENCES Patient (ssn),
                                FOREIGN KEY (phy_ssn) REFERENCES Doctor
                        (phy_ssn),
                                FOREIGN KEY (trade_name, pharm_id) References
                                        Make_Drug (trade_name, pharm_id))
CREATE TABLE Make_Drug (trade_name CHAR(20),
                                Pharm_id CHAR(11),
                                Formula CHAR(100),
                                PRIMARY KEY (trade_name, pharm_id),
                                FOREIGN KEY (pharm_id) REFERENCES Pharm_co
                        (pharm_id) ON DELETE CASCADE)
CREATE TABLE Sell ( price INTEGER,
                        name CHAR(10),
                        trade_name CHAR(10),
                        pharm_id CHAR(11),
                        PRIMARY KEY (name, trade_name, pharm_id),
                        FOREIGN KEY (name) REFERENCES Pharmacy (name),
                        FOREIGN KEY (trade_name, pharm_id) REFERENCES
                        Make_Drug(trade_name, pharm_id))
CREATE TABLE Contract ( name CHAR(20),
                                pharm_id CHAR(11),
                                start_date CHAR(11),
                                end_date CHAR(11),
                                text CHAR(10000),
                                supervisor CHAR(20),
                                PRIMARY KEY (name, pharm_id),
                                FOREIGN KEY (name) REFERENCES Pharmacy (name),
                    FOREIGN KEY (pharm_id) REFERENCES Pharm_co
                (pharm_id))
```

Answer 3.3 No. In this case, the index is unclustered, each qualifying data entry could contain an rid that points to a distinct data page, leading to as many data page I/Os as the number of data entries that match the range query. In this situation, using index is actually worse than file scan.

Answer 3.4

a. Each page is 4KB=4096 bytes. It has 96 bytes of metadata, so it has 4096-96 space for tuples and it can hold 4000/200=20 tuples. Since we have 40 * 400 = 16000 books, in total we need 16000/20=800 pages.
b. Each PAX page is organized as minipages with the following characteristics: BookID: 20*8=160 bytes, author: 20*64=1280 bytes, title: 20*100=2000 bytes, genre: 20*4=80 bytes, year: 20*8=160 bytes and price: 20*16=320 bytes.
   Since the page has a header of 48 bytes and a footer of 48 bytes, the 4KB of a PAX page have the following offsets:
   0 bytes Header
   48 (0+48) bytes BookID
   208 (48+160) bytes author
   1488 (208+1280) bytes title
   3488 (1488+2000) bytes genre
   3568 (3488+80) bytes year
   3728 (3568+160) bytes price
   4048 (3728+320) bytes Footer

1st subpage (0-1024)	Header (0-48) BookID (48-208) Author (208-1488)
2nd subpage (1024-2048)	Author (208-1488) Title (1488-3488)
3rd subpage (2048-3072)	Title (1488-3488)
4th subpage (3072-4096)	Title (1488-3488) Genre (3488-3568) Year (3568-3728) Price (3728-4048) Footer (4048-4096)

   Accessing BookID requires the 1st subpage, author requires 1st and 2nd subpage, title requires 2nd, 3rd and 4th subpage and genre, year and price require 4th subpage.
c. Note that each page contains 20 records, so the record with BookId=512, would exist in 26th page (as the 12th record: 25*20 + 12). Since BookID is the primary key, the result contains one row. However, as we need to reassemble the whole row, we need to read all the relevant subpages (recall that the unit of transfer for flash is the 1KB subpage), which would be the 1st, 3rd, and 4th page. Specifically, we have to identify the subpage where the 12th record can be located. For the 12th record, the information pertaining to BookId would start at: 48 + 11* 8 = 136th byte (1st subpage), Author would start at: 208 + 11*64 = 912th byte (1st subpage), Title would start at: 1488 + 11*100 = 2588th byte (3rd subpage), Genre, Year, Price would be present in the 4th subpage. Thus, we need to access 3 subpages * 1KB = 3KB.

d. Since the number of books published each year over the period of 40 years is constant, we need to access the data about the last 10 years only, i.e., we need the information from the quarter of the table: 800/4 = 200 pages. As we need only the price attributes, it is enough to read the fourth subpage only. In total, we need to access 200 * 1 KB = 200KB. Note that the year attribute will be just read once, to identify the first record with year = 2004, after that every record will qualify for the predicate in the query.
e. Since the number of books published each year over the period of 40 years is constant, we need to access the data about the last 20 years only, half of the table: 800/2 = 400 pages. As we need the author and title attributes, we need to access all 4 subpages, i.e., we need to read the whole pages. In total, we need to read 400 * 4 KB = 1600KB. Note that the year attribute will be just read once, to identify the first record with year = 1994, after that every record will qualify for the predicate in the query.

Answer 3.5

The main conclusion about the five file organizations is that all five have their own advantages and disadvantages. No one file organization is uniformly superior in all situations. The choice of appropriate structures for a given data set can have a significant impact upon performance. An unordered file is best if only full file scans are desired. A hash indexed file is best if the most common operation is an equality selection. A sorted file is best if range selections are desired and the data is static; a clustered B+ tree is best if range selections are important and the data is dynamic. An unclustered B+ tree index is useful for selections over small ranges, especially if we need to cluster on another search key to support some common query.

a. Using these fields as the search key, we would choose a sorted file organization or a clustered B+ tree depending on whether the data is static or not.
b. Heap file would be the best fit in this situation.
c. Using this particular field as the search key, choosing a hash indexed file would be the best.