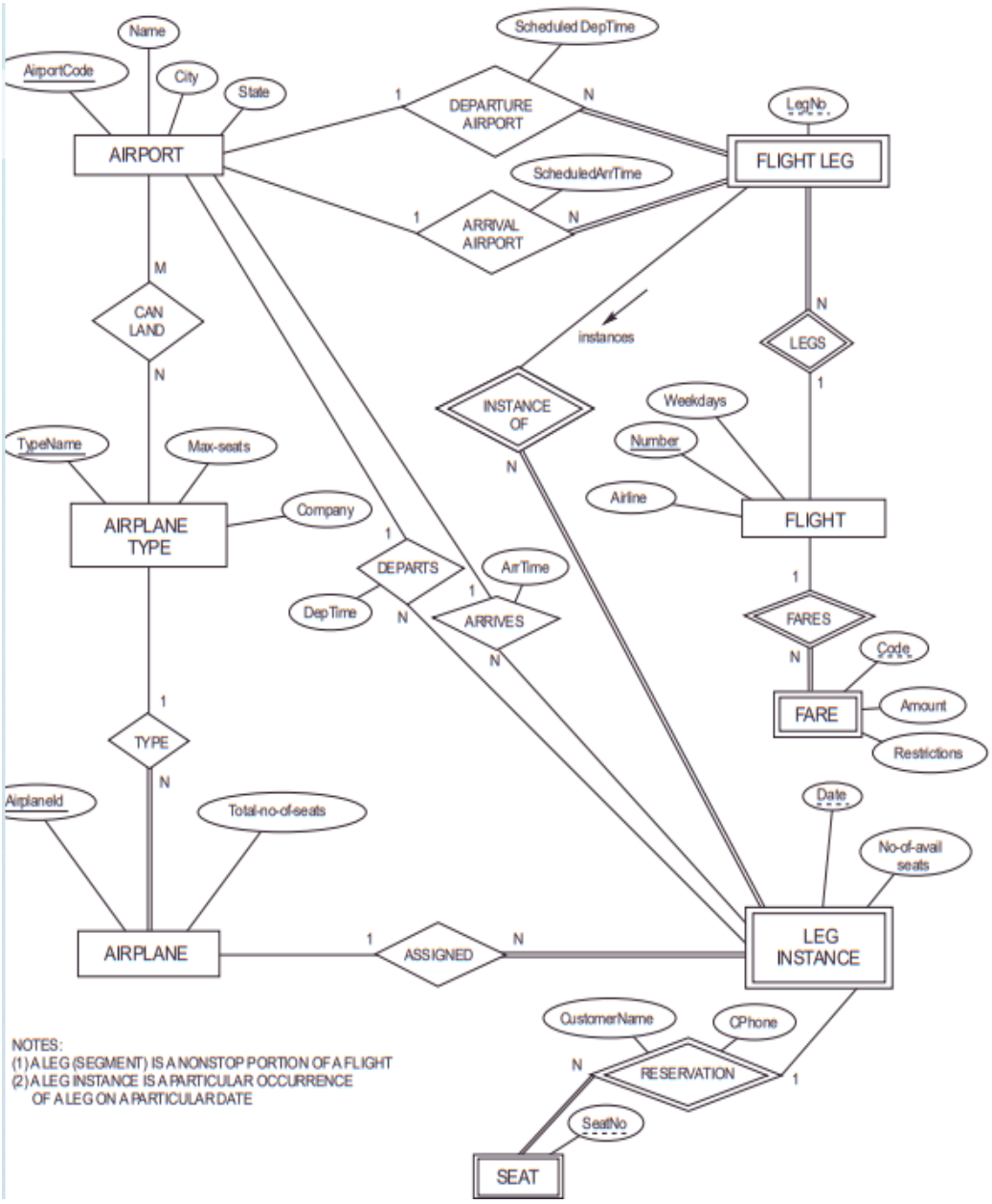
Question 1

a) Consider the ER diagram of Figure 3.16 (found on my home page. Click on “figures on Chapter 3) which shows a simplified schema for an airline reservation system. Extract from the ER diagram the requirements and constraints that produced this schema. Try to be as precise as possible in your requirement and constraint specification.



1. AIRPORT: The database presents the information about each AIRPORT in an AIRPORT table. Each AIRPORT has its unique combination of the attributes Airport_code, Airport Name, City and State where it is located. The Airport_code is the unique identifier of this entity.
2. FLIGHT: Each FLIGHT has a unique Number attribute as an identifier. The FLIGHT entity also consists of the details of the flight, including the Airline of the flight and the Weekdays of the flight schedule.

3. FARE: Each FLIGHT has its FARE with a Code attribute as the discriminator and the FARE weak entity has its Amount and Restrictions.
4. FLIGHT_LEG: The FLIGHT_LEG is a weak entity type with Leg_No as the partial key attribute. A FLIGHT can have one or more FLIGHT_LEGs with Leg_No. Each FLIGHT_LEG can depart at DEPARTURE AIRPORT having a scheduled Departure Time (Scheduled DepTime). Similarly, each FLIGHT_LEG can arrive at ARRIVAL AIRPORT at a scheduled Arrival Time (ScheduledArrTime).
5. LEG-INSTANCE: A FLIGHT_LEG can have one or more LEG_INSTANCEs. A LEG_INSTANCE is a weak entity type with date on which flight travels as the partial key. It is used to keep track of the number of available seats and has daily records of all the AIRPLANE information including the actual Arrival Time and Departure Time at the AIRPORT. Each RESERVATION also need check with the daily LEG_INSTANCE for the number of seats available before making a seat reservation for a customer.
6. AIRPLANE: Each AIRPLANE has its unique Airplane ID as the identifier along with the attribute, total number of seats. Each AIRPLANE is assigned a LEG_INSTANCE according to its flight date and has AIRPLANE_TYPE.
7. AIRPLANE_TYPE: The type of the airplane is determined with the attributes Type Name, Fixed Maximum number of seats and the Company name who owns the AIRPLANE. Each Type Name is considered unique and is used as an identifier. Many AIRPLANE_TYPEs can land at a time at one or many other AIRPORTs.
8. SEAT: Each SEAT weak entity type has a Seat_No as a partial key. RESERVATION for seats can be made with Customer Name and Customer Phone number.

Relationship Constraints:

1. DEPARTURE AIRPORT: a 1:N relationship type between AIRPORT and FLIGHT_LEG. The participation of AIRPORT is partial whereas for FLIGHT_LEG is total.
2. ARRIVAL AIRPORT: a 1:N relationship type between AIRPORT and FLIGHT_LEG. The participation of AIRPORT is partial whereas for FLIGHT_LEG is total.
3. LEGS: a 1:N relationship type between FLIGHT and FLIGHT_LEG which is also the identifying relationship for the weak entity type FLIGHT_LEG. The participation of FLIGHT is partial, whereas that of FLIGHT_LEG is total.

4. FARES: a 1:N relationship type between FLIGHT and FARE which is also the identifying relationship for the weak entity type FARE. The participation of FLIGHT is partial and that of FARE is total.
5. ASSIGNED: a 1:N relationship type between AIRPLANE and LEG_INSTANCE. The participation of AIRPLANE is partial but that of LEG_INSTANCE is total.
6. RESERVATION: a 1:N relationship type between SEAT and LEG_INSTANCE which is also the identifying relationship for the weak entity type SEAT. The participation of SEAT is total and that of LEG_INSTANCE is partial.
7. TYPE: a 1:N relationship type between AIPLANE_TYPE and AIRPLANE. The participation of AIRPLANE_TYPE is partial and that of AIRPLANE is total as all airplanes must be of a particular type and there can be many airplanes of an AIRPLANE_TYPE.
8. CAN_LAND: a N:M relationship type between AIRPLANE_TYPE and AIRPORT. One or more Airplanes of an AIRPLANE_TYPE can land at every AIRPORT. Both participations are determined to be partial as not all types of airplanes can land at every airport.
9. INSTANCE OF: There are many instances of FLIGHT_LEG. It is the identifying relationship for the weak entity LEG_INSTANCE. The participation of FLIGHT_LEG is partial whereas it is total for LEG_INSTANCE.

(b) A database is being constructed to keep track of the teams and games of a sport league. A team has a number of players, not all of whom participate in each game. It is desired to keep track of the players participating in each game for each team, the positions they played in that game, and the result of the game. Try to design an ER schema diagram for this application, stating any assumption you make. Choose your favorite sport (soccer, baseball, football, …).

Answer to Question 1 Part (b)

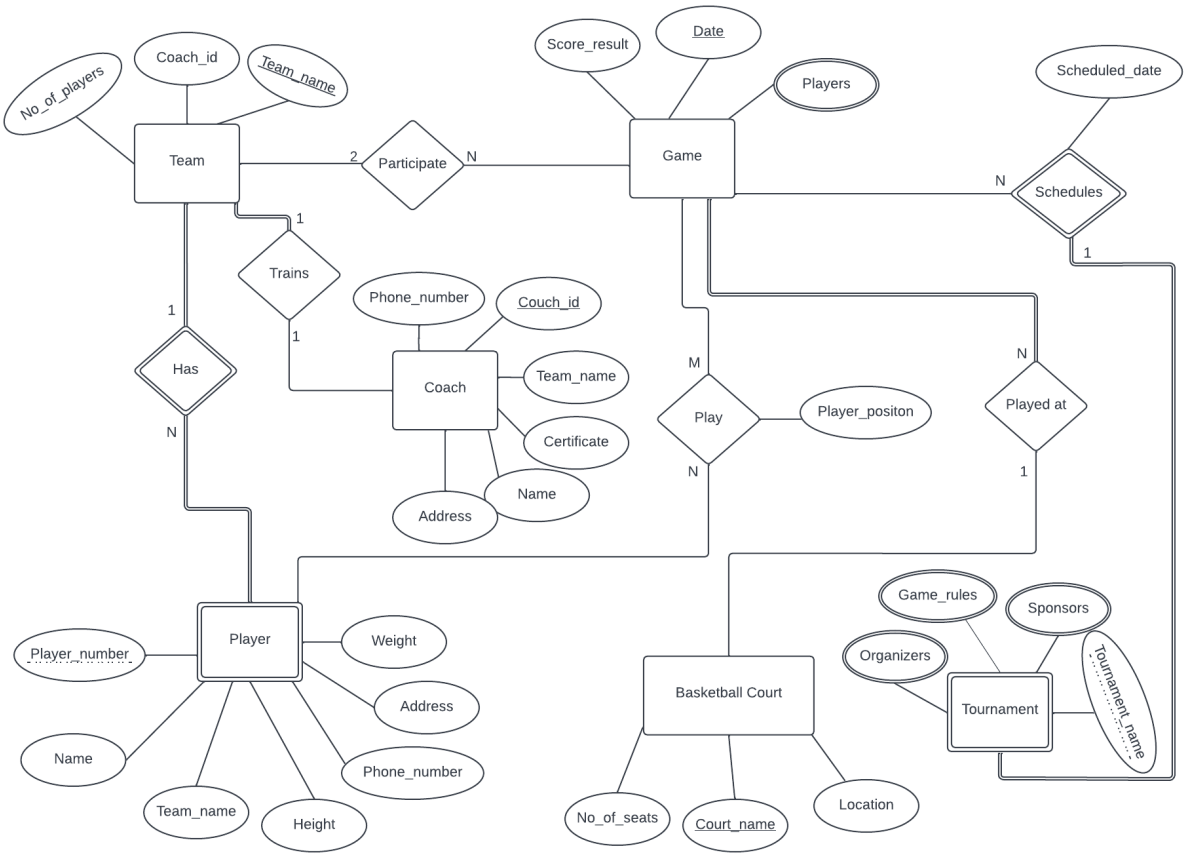


Figure: ER schema diagram for Basketball League

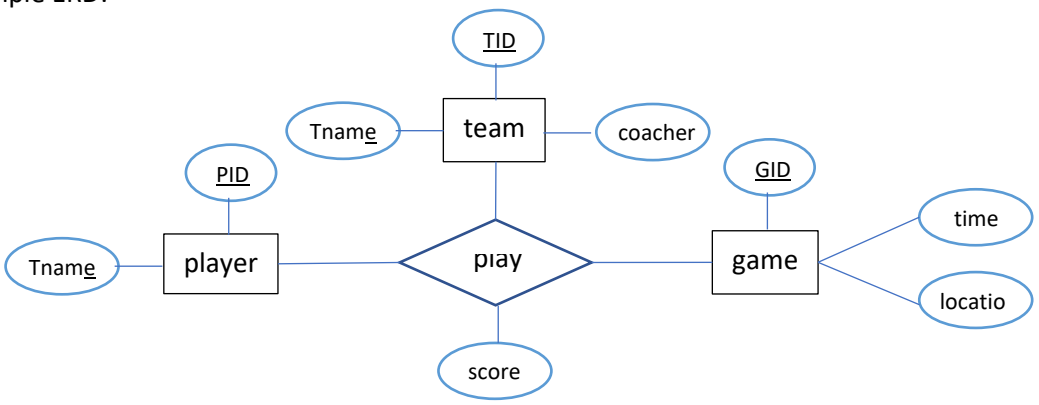
Assumptions:

- No two Games are scheduled on the same date.
- A Game might get cancelled and might happen on a different date than the scheduled date.
- A Coach can only train one Team.
- A Team must have a Coach but a Coach can decide to not train a Team
- Some Teams might not participate in a game.
- For a Team to exist, it must have Players.
- A Player must have a Team.
- A Tournament with the same tournament name can be organized again (maybe annually).
- A Game must take place at a Basketball court.
- Players from different Teams can have the same player number.

Answer to Question 2

a. SELECT FNAME, LNAME FROM EMPLOYEE, PROJECT, WORKS_ON WHERE EMPLOYEE.SSN = WORKS_ON.ESSN AND PROJECT.PNUMBER = WORKS_ON.PNO AND EMPLOYEE.DNO = 5 AND WORKS_NO.HOURS > 10 AND PROJECT.PNAME = 'Project X';
b. SELECT FNAME, LNAME FROM EMPLOYEE, DEPENDENT WHERE EMPLOYEE.SSN = DEPENDENT.ESSN AND DEPENDENT.DEPENDENT_NAME = EMPLOYEE.FNAME;
c. SELECT emp1.FNAME, emp1.LNAME FROM EMPLOYEE emp1, EMPLOYEE emp2 WHERE emp2.FNAME = 'Franklin' AND emp2.LNAME = 'Wong' AND emp2.SSN = emp1.SUPERSSN;
d. SELECT PNAME, SUM(HOURS) FROM PROJECT, WORKS_ON GROUP BY PNAME HAVING PROJECT.PNUMBER = WORKS_ON.PNO;
e. SELECT FNAME, LNAME FROM EMPLOYEE AS e WHERE NOT EXISTS (SELECT * FROM PROJECT AS p WHERE NOT EXISTS(SELECT * FROM WORKS_ON AS w WHERE w.ESSN = e.ESSN AND w.PNO=p.PNO));
f. SELECT FNAME, LNAME FROM EMPLOYEE AS e, PROJECT AS p WHERE NOT EXISTS (SELECT * FROM WORKS_ON AS w WHERE w.ESSN = e.SSN AND w.PNO = p.PNUMBER);

A simple ERD:



Question 2

2. (15) Specify the following queries in SQL (see Fig. 7.5 on my home page).

a. Retrieve the name of all employees in department 5 who work more than 10 hours per week on the 'Product X' project.

b. List the names of all employees who have a dependent with the same first name as themselves.

c. Find the names of all employees who are directly supervised by 'Franklin Wong'.

d. For each project, list the project name and the total hours per week (by all employees) spent on that project.

e. Retrieve the names of all employees who work on every project.

f. Retrieve the names of all employees who do not work on any project

a)

Query:

```
SELECT FNAME,MINIT,LNAME
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE PNAME = 'Product X' AND PNO = PNUMBER AND HOURS > 10
AND SSN = ESSN AND DNO=5
```

Relational Algebra Notation

$EP10 \leftarrow \pi_{ESSN}(\sigma_{HOURS>10}(WORKS\_ON \bowtie_{PNO=PNUMBER}(\pi_{PNUMBER}(\sigma_{PNAME='ProductX'}(PROJECT))))$

$\pi_{FNAME,MINIT,LNAME}(\sigma_{DNO=5}(EMPLOYEE \bowtie_{SSN=ESSN}(EP10))$

b)

Query:

```
SELECT FNAME,MINIT,LNAME
FROM EMPLOYEE, DEPENDENT
WHERE FNAME = DEPENDENT_NAME AND SSN = ESSN
```

Relational Algebra Notation

$\pi_{FNAME,MINIT,LNAME}(EMPLOYEE \bowtie_{FNAME = DEPENDENT\_NAME\ AND\ SSN=ESSN}(DEPENDENT))$

c)

```
SELECT E.FNAME,E.MINIT,E.LNAME
FROM EMPLOYEE E, EMPLOYEE S
WHERE S.FNAME = 'Franklin' AND S.LNAME = 'Wong'
AND E.SUPERSSN = S.SSN
```

Relational Algebra Notation

$\pi_{FNAME,MINIT,LNAME}$ (EMPLOYEE $\bowtie_{SUPERSSN=SSN}(\pi_{SSN}(\sigma_{FNAME='Franklin' AND LNAME='Wong'}$(EMPLYOEE))))

d)

```
SELECT PNAME, SUM(HOURS)
FROM PROJECT, WORKS_ON
WHERE PNO=PNUMBER
GROUP BY PNAME
```

Relational Algebra Notation

PNAME, $\mathcal{F}_{SUM\ HOURS}$(PROJECT $\bowtie$PNO=PNUMBER(WORKS_ON))

e)

```
SELECT E.FNAME,E.MINIT,E.LNAME
FROM EMPLOYEE E
WHERE NOT EXISTS
(SELECT * FROM PROJECT AS P WHERE NOT EXISTS
(SELECT * FROM WORKS_ON AS W WHERE W.ESSN=E.SSN AND W.PNO=P.PNUMBER))
```

Relational Algebra Notation

P_MISSED $\leftarrow \pi_{ESSN}$ (PROJECT$\bowtie_{PNUMBER!=PNO}(\pi_{ESSN,\ PNO}$(WORKS_ON$\bowtie_{ESSN=SSN}$(EMPLYOEE))))

$\pi_{FNAME,MINIT,LNAME}$(EMPLOYEE $\bowtie_{SSN\ !=\ ESSN}$ (P_MISSED))

The above query gets all the employes who didn't missed any project to work on i.e they worked on all porjects

f)

```
SELECT E.FNAME,E.MINIT,E.LNAME
FROM EMPLOYEE E
WHERE E.SSN NOT EXISTS
(SELECT W.ESSN FROM WORKS_ON AS W WHERE W.ESSN=E.SSN)
```

Relational Algebra Notation

$\pi_{FNAME,MINIT,LNAME}$ (EMPLOYEE $\bowtie_{SSN\ !=\ ESSN}(\pi_{ESSN}$(WORKS_ON$\bowtie_{ESSN=SSN}$(EMPLYOEE))))

Question 3

3. (25) Linear Hashing

- collision resolution strategy: chaining
- split rule: when load factor > 0.7
- initially M = 4 (M: size of the primary area)
- hash functions: $h_i(key) = key \bmod 2^i \times M$ (i = 0, 1, 2, …)
- bucket capacity = 4

Trace the insertion process of the following keys into a linear hashing file:

32, 44, 36, 9, 14, 18, 10, 25, 5, 30, 31, 35, 7, 11, 43, 37, 29, 50

Phase 0:
Insert using: h_0 = key mod M, Split using h_1 = key mod 2M
Initially M=4

Insertion Process: n=0 (split point)

32 44 36	9 25 5	14 18 10 30	31 35
0	1	2	3

When we inserted the 12th value we have load factor (12/16) = 0.75 > 0.70. So bucket 0 needs to be split using h_1 and and n will be incremented to 1. So, we get

32	9 25 5	14 18 10 30	31 35	44 36
0	1	2	3	4

Now load factor 12/20 = 0.6, we will continue to insertion

32	9 25 5	14 18 10 30	31 35 7 11	44 36
0	1	2	3	4
			43	Overflow

Now load factor is 15/24 = 0.75 > 0.70
We will use h1 to split bucket 1 and increment n to 2. We get the following result:

0	1	2	3	4	5
32	9 25	14 18 10 30	31 35 7 11	44 36	5

43 Overflow

Now we will continue

0	1	2	3	4	5
32	9 25 37 29	14 18 10 30	31 35 7 11	44 36	5

43 Overflow

Upon inserting the 17th value the load factor is 17/24 = 0.708 > 0.70

We will use h1 to split bucket 2 and increment n to 3. We get the following result:

0	1	2	3	4	5	6
32	9 25 37 29	18 10	31 35 7 11	44 36	5	14 30

43 Overflow

Now we will continue

0	1	2	3	4	5	6
32	9 25 37 29	50 18 10	31 35 7 11	44 36	5	14 30

43 Overflow

All the values have been inserted.

Question 4

4. (25) Apply the following algorithm to the B+-tree shown in Fig. 1 to store it in a data file (each node in a B+-tree is stored as a page in the data file) In the algorithm, a stack is used to explore a B+-tree in the depth-first manner. Each entry *X* in the stack has three data fields:

   X.data – to store all the key values in a node,
   X.address-of-parent – to record the address (in the file) of the parent of a node,
   X.position – used to indicate, for a node, what child of its parent the node is (i.e., whether it is the first, second, …, child of its parent.)

   Trace the computation process.

   Algorithm:

```
push(root, -1, -1);
while (S is not empty) do
{    x := pop( );
     store x.data in file F;
     assume that the address of x in F is ad;
     if x.address-of-parent ≠ -1 then {
          y := x.address-of-parent;
          z := x.position;
          write ad in page y at position z in F;
     }
     let x1, …, xk be the children of x;
     for (i = k to 1) {push(xi, ad, i)};
```

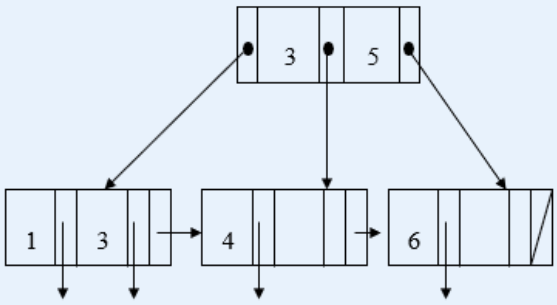
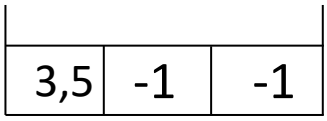
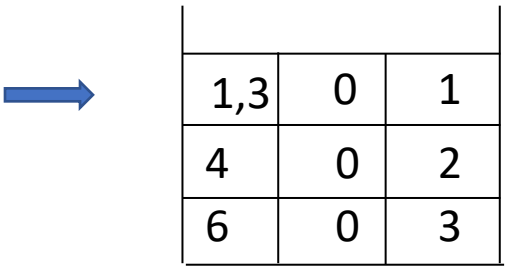


Fig. 1:

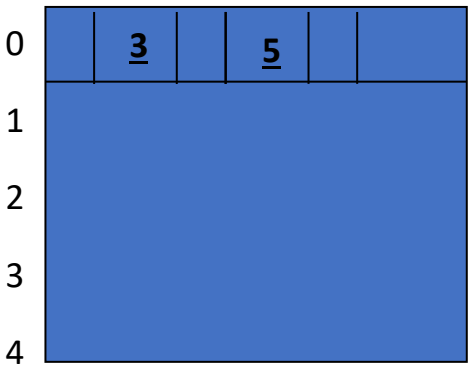
Stack:



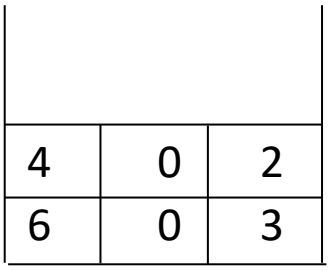


1,3	0	1
4	0	2
6	0	3

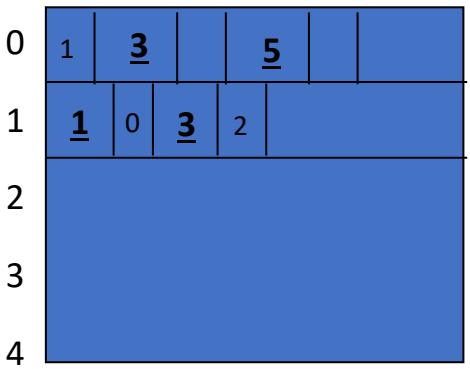
B+-tree stored in a file:

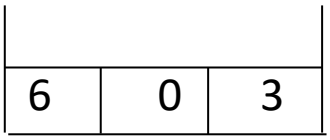


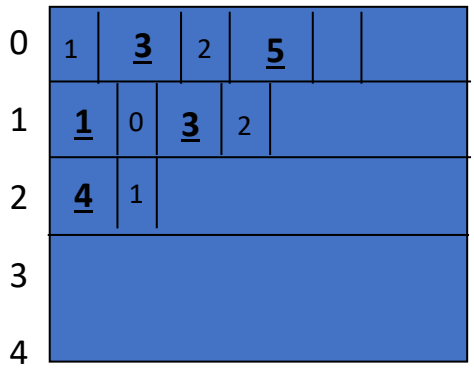
Stack:

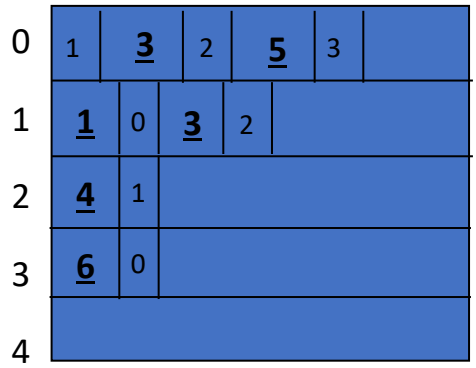


B+-tree stored in a file:

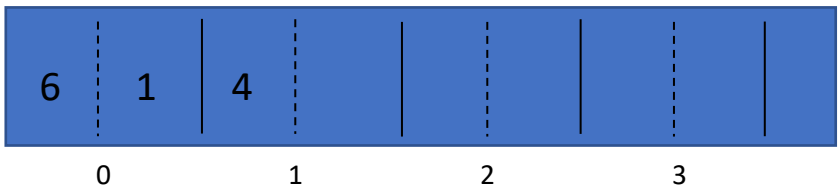








Data File



Question 5

5. (20) Assume that for an attribute of a table, *m* bit maps (bit vectors) are created and then compressed using the following procedure:

a. Decompose each bit vector into a series of runs such that each contains a set consecutive 0's followed by a 1.
b. compress each run with i 0's as below:
part 1: i expressed as a binary number, denoted as $b_1(i)$.
part 2: Assume that b(i) is j bits long. Then, part 2 is a sequence of (j – 1) 1's followed by a 0, denoted a $b_2(i)$.
c. The compressed bit string is set to be $b_2(i)\ b_1(i)$.

Let s_j be the jth compressed bit vector. Putting all the compressed bit vectors together, we get a bit string:

$$s = s_1 s_2 \ldots s_j \ldots s_m$$

Please design a method to uncompress any compressed bit vector efficiently.

We need to maintain an array A[ ] of integers, in which each A[*j*] is the beginning position of s_j in *s*. Then, to uncompress s_j, (*j* = 1, …, *m*), execute the following procedure:

1. Let A[*j*] = a. Let A[*j* + 1] = b.
2. *t* := *s*[a .. b - 1].
3. Scan *t* from beginning to find the first 0.
4. Let the first 0 appears at position *j*. Check the next *j* bits. The corresponding value is a run.
5. Remove all these bits from *s*. Go to (3).