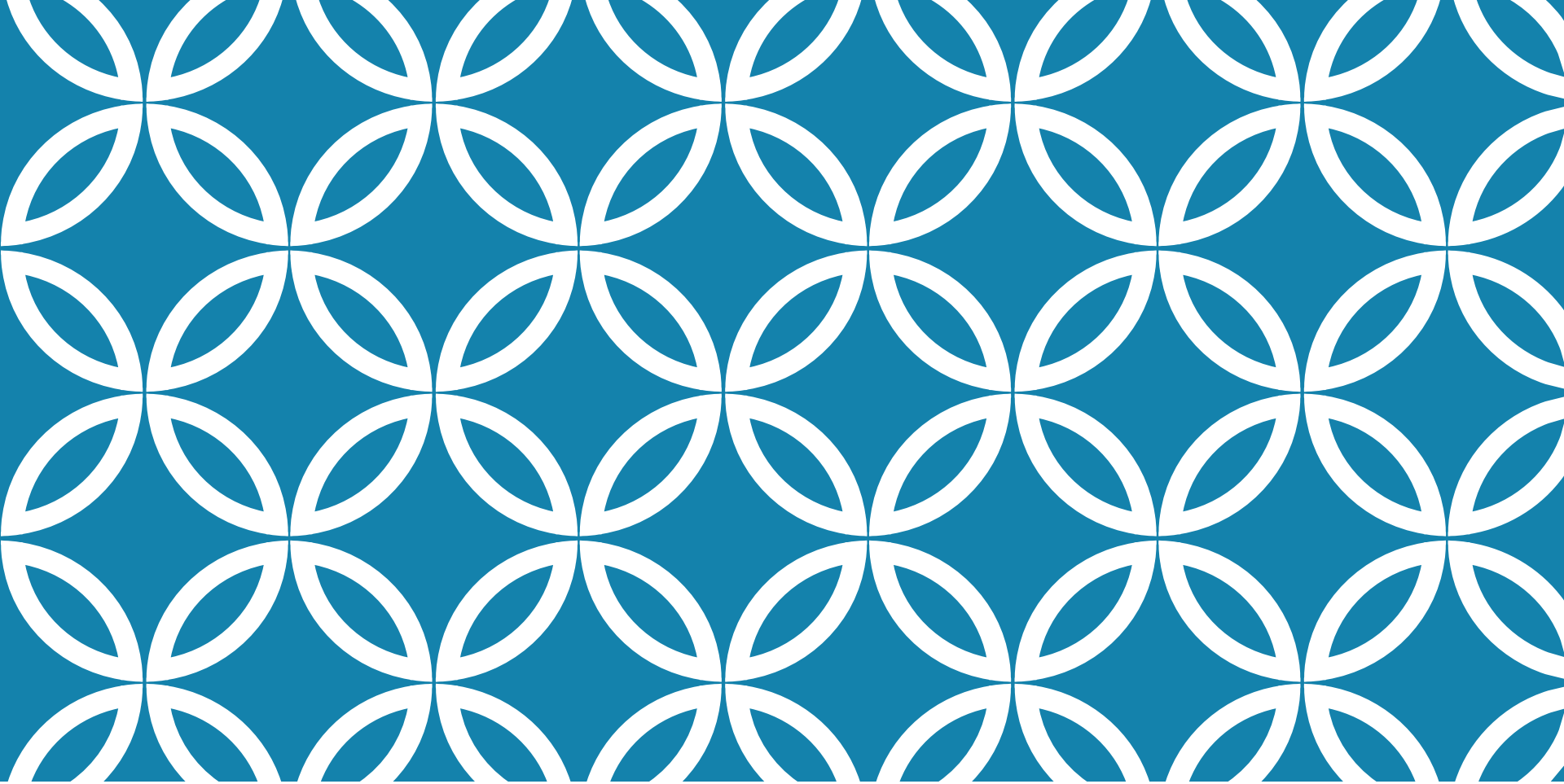


FILE ORGANISATIONS

CT230
Database
Systems I

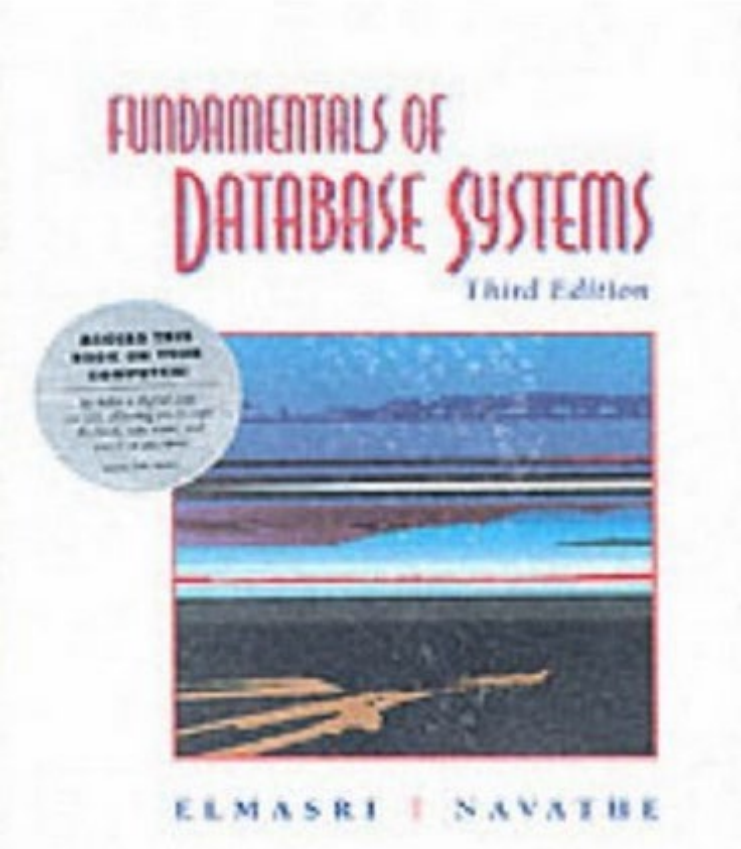
RECOMMENDED TEXT:

See:

Chapter 5

Elmasri & Navathe

(3rd Edition)



MOTIVATIONS

- Generally can assume for non-trivial relational databases, that the entire database will **not** fit in main memory (RAM)
- One of the DBMS's tasks is to manage the physical organisation (storage and retrieval) of the tuples (rows) in each table in the database
  - This is called File Organisation

NOTE:

Newer database system architectures, in-memory databases (such as SAP Hanna), manage their data through virtual memory, relying on the Operating System to manage the movement of data to and from main memory through the OS paging mechanism.

DEFINITION: FILE ORGANISATIONS

A database file organisation is the way tuples (records) from a table are physically arranged in secondary storage to facilitate storage of the data and read/write requests by users (via queries).

A number of factors to consider, including:

- Support of fast access of data – moving to/from secondary storage
- Cost
- Efficient use of secondary storage space
- Provision for table growth (when new tuples added)

Concerning the physical storage of tuples

- Options?
  - All stored together?
  - Separated in some way based on some *logical* grouping?

More definitions:

File = collection of data stored in bulk

In DBMS we have referred to these files as *tables* or *relations*

In DBMS we know that such tables contain a sequence of tuples, where each tuple contains a sequence of bytes and is subdivided into attributes or fields. Each attribute contains a specific piece of information. Associated with each attribute is a data type

In File Systems, we refer to these tuples as **records** containing **fields**

Size of records/tuples:

Fixed length: all records (tuples) in file (table) have exactly same size

Variable length: different records (tuples) in file (table) have different size

RECORDS

Each record often begins with a *header*, a fixed-length region which stores information about the record such as:

- Pointer to the database schema
- Length of the record
- Timestamp indicating the time the record was last modified or read
- Pointers to the fields of the record

File organisation issues:

How can these records be organised to:

- store in a compact manner on devices of limited capacity?
- provide convenient and quick access by programs

BLOCKS

- Different terminology used but generally,

  Block = Frame = Page

where records from a file are assigned to Blocks/Pages/Frames

- In relational DBMS use the terminology of a **block**
- Therefore, a table can also be defined as a collection of blocks where each block contains a collection of records.

DEFINITION: Blocks

- A block is the unit of data transfer between secondary storage and memory
- The block size **B** is fixed
- Records of a file must be allocated to blocks. Typically, the block size is larger than the record size, so each block will contain a number of records
- Some files may have very large records that cannot fit in one block so **span** records over a number of blocks
- A number of blocks is typically associated with a table

BLOCKS

Blocks also have header information holding information about the block such as:

- Links to one or more blocks associated with the table
- Which table (in the schema) the blocks belong to
- Timestamp of last access to block (read or write)

Example: Records assigned to blocks for the table with block header shown:

`dept_locations(dnumber, dlocation)`

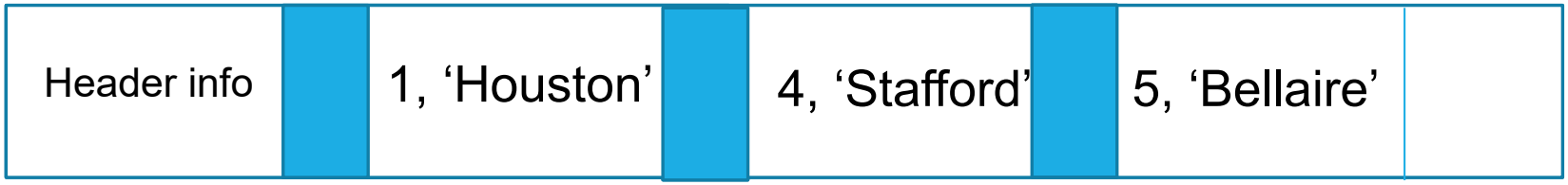
Block 1

Block 2

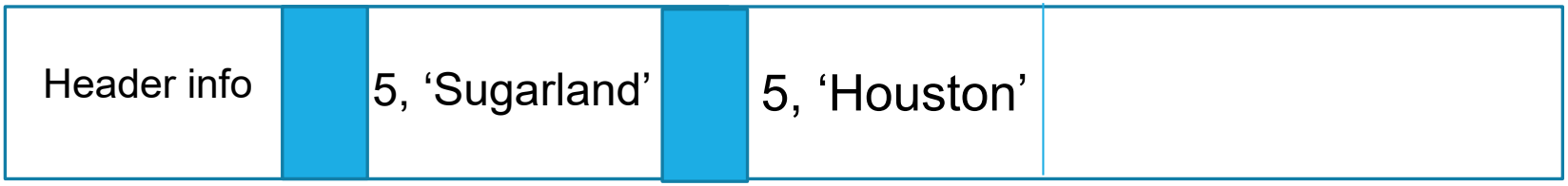
Example: Records assigned to blocks for the table with block and record header shown:

`dept_locations(dnumber, dlocation)`

Block 1



Block 2



DEFINITION: Blocking factor

- Blocking factor is the average number of records that fit per block
- Given block size B (in bytes), and record size R (in bytes), then with B >= R, can fit **floor(B/R)** records per block.
- Must ensure that the header information is also accounted for

Spanned vs Unspanned organisations:

Spanned organisation - records can span more than one block

Un-spanned - records are not allowed to cross block boundaries

So can only use when **B >= R**

(i.e., block size is greater than record size)

NOTE:

Block size and record size measured in bytes.

e.g., with unspanned memory organisation and

B = 1024 Bytes (once header information stored)

R = 100 Bytes and of fixed length

The blocking factor is:

```
floor(10.24) = 10
```

Why use blocking?

Say we need to retrieve a file with 1000 records …

- If not blocked then would need **1000 data transfers**
- If blocked with a blocking factor of 10, and records are stored one after another in blocks, then the same operation requires **100 data transfers**

EXAMPLE 1: A table has 20000 fixed-length STUDENT records

Schema:

student(name, studentID, address, mobphone, birthdate, gender, degreeCode, currentYear)

Each field is the following size:

```
name (30 bytes),
studentID (9 bytes),
address (40 bytes),
mobphone (10 bytes),
birthdate (10 bytes),
gender (1 byte),
degreeCode (8 bytes),
currentYear (4 bytes)
```

The file is stored on disk, in blocks, with 20 bytes required for header information per record.

EXAMPLE 1 QUESTIONS:

Each field is the following size:

```
name (30 bytes),
studentID (9 bytes),
address (40 bytes),
mobphone (10 bytes),
birthdate (10 bytes),
gender (1 byte),
degreeCode (8 bytes),
currentYear (4 bytes)
```

The file is stored on disk, in blocks, with 20 bytes required for header information per record.

What is the record size? (adding in the header information also)

30+9+40+10+10+1+8+4+20 = 132 bytes

Given a block size of 512 Bytes what is the blocking factor? (unspanned memory organisation)

512/132 = 3.87 ⇒ blocking factor = 3

How many blocks are required to store all 20000 records if each block is filled before another block is used (remember records are fixed-length)

20000/3 = 6666.67 ⇒ 6667 blocks needed

Operations performed on a file

All the operations we have been performing with SQL code:

- Scan or fetch all records
- Search records that satisfy an equality condition (i.e., find specific records)
- Search records where a value in the record is between a certain range
- Insert records
- Delete records

Steps to search for a record on a disk:

1. Locate relevant blocks
2. Move these blocks to main memory buffers
3. Search through block(s) looking for required record
4. At worst (*the worst case*), may have to retrieve and check through all blocks for the record

Generally, when accessing records:

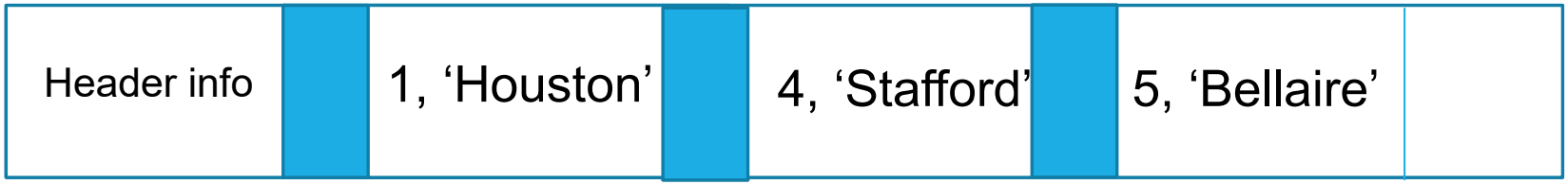
To support record level operations, must:

- keep track of the *blocks* associated with a file
- keep track of *free space* on the blocks
- keep track of the *records* on a block

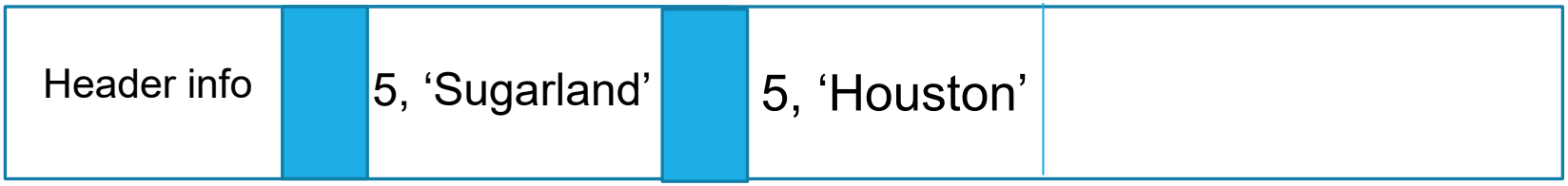
Recall example again: Records assigned to blocks for the table:

`dept_locations(dnumber, dlocation)`

Block 1



Block 2



Options for organising records?

- Heap file organisation (unordered)
- Sequential file organisation (ordered)
- Hashing/hashed file organisation
- Indexed file organisation (Primary, Clustered, B-Trees, B+ Trees)

HEAP FILE ORGANISATION

Approach: Any record can be placed in any block where there is space for the record (no ordering of records)

Insertion: last disk block associated with file (table) copied into buffer and record is added; block copied back to disk

Searching: must search all blocks (linear search)

Deletion: find block with record (linear search); delete link to record

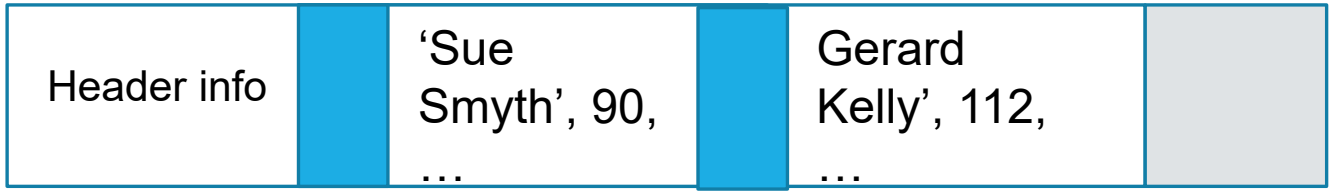
EXAMPLE 2: Given a blocking factor of 2, and the student schema from example 1, sketch the placement of the following student records, in the order given, using heaped file organization

```
('Jane Casey', 111, '34 hazel park, newcastle, galway',
'087123456', '17-05-2001', 'F', 'GY101', 1)

('Jack Walsh ', 91, '13 college road, galway',
'086654321', '01-09-2000', 'M', 'GY350', 3)

('Sue Smyth ', 90, 'Maree, Oranmore, Co. Galway',
'087111222', '25-07-1999', 'F', 'GY406', 3)

('Gerard Kelly', 112, 'Main Street, Oughterard, Co.
 Galway', '087121212', '30-12-2002', F, GY414, 1)
```

```
('Jane Casey', 111, '34 hazel park, newcastle,
 galway', '087123456', '17-05-2001', 'F', 'GY101', 1)

('Jack Walsh ', 91, '13 college road, galway',
 '086654321', '01-09-2000', 'M', 'GY350', 3)

('Sue Smyth ', 90, 'Maree, Oranmore, Co. Galway',
 '087111222', '25-07-1999', 'F', 'GY406', 3)

('Gerard Kelly', 112, 'Main Street, Oughterard, Co.
 Galway', '087121212', '30-12-2002', F, GY414, 1)
```

Block 1

Block 2



How are the following supported in heaped file organisation (using example 2)?

1. Inserting a new tuple:

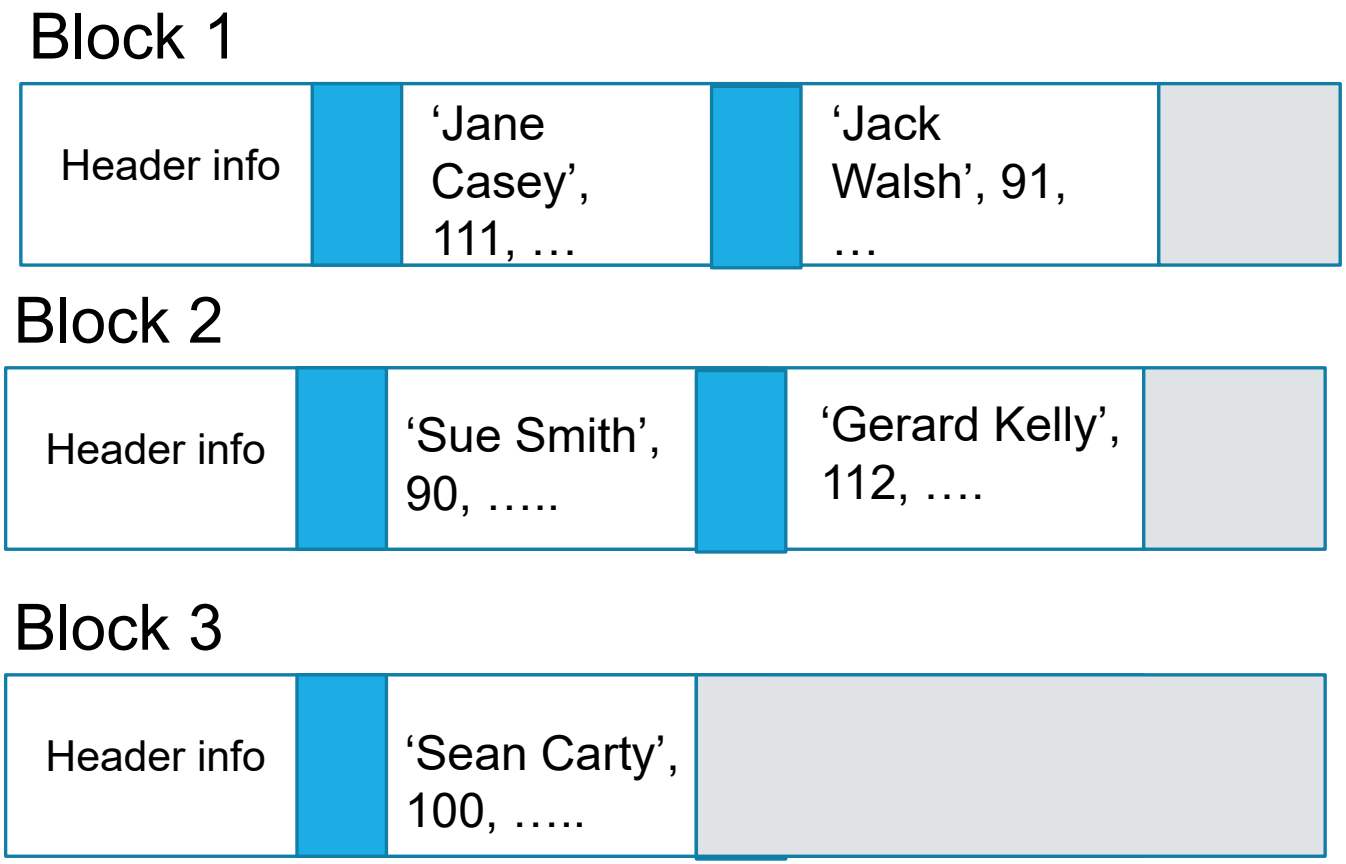
```
('Sean Carty', 100, '23 Ocean view, Salthill,
Galway', '087222333', '24-10-2002', 'M', 'GY101', 3)
```

2. Deleting an existing tuple:

```
('Jack Walsh ', 91, '13 College road, Galway',
'086654321', '01-09-2000', 'M', 'GY350', 3)
```

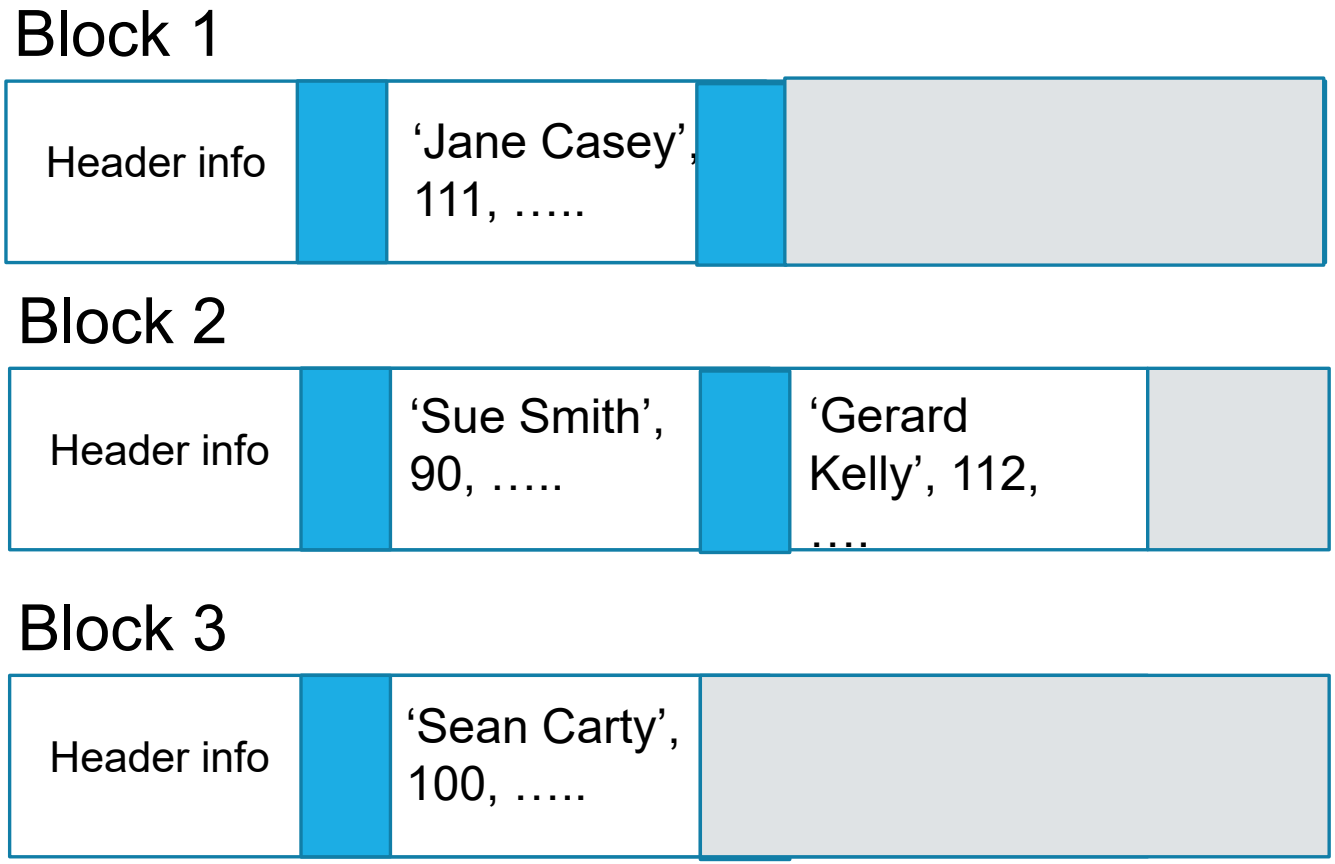
1. Inserting a new tuple:

```
('Sean Carty', 100, '23 Ocean view, Salthill, Galway',
'087222333', '24-10-2002', 'M', 'GY101', 3)
```



2. Deleting an existing tuple:

```
('Jack Walsh ', 91, '13 College road, Galway',
 '086654321', '01-09-2000', 'M', 'GY350', 3)
```



HEAP FILE ORGANISATION

Advantages: Insertion efficient and easy - last disk block copied into buffer and record is added; block copied back to disk

Disadvantages:

1. Searching inefficient - must search all blocks (linear search)

2. Deleting inefficient - search first; delete and then leave unused space in block if using 'easy' insert approach

SEQUENTIAL FILE ORGANISATION

Approach: Records are stored in *sequential* order, based on the value of some key of each record – often primary key

Usually use an *index* with sequential file organisation

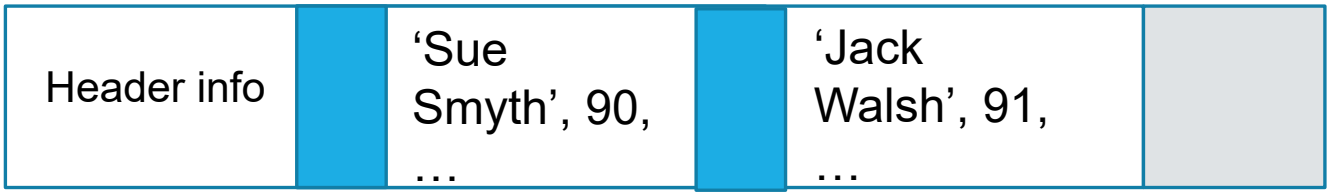
Allows records to be read in sorted order

EXAMPLE 3: Using a blocking factor of 2, and the schema from example 1, sketch the placement of the following student records using a sequential file organisation ordered on the `studentID`:

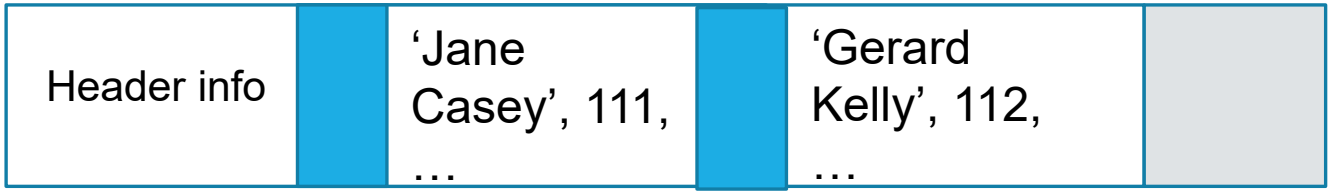
```
('Jane Casey', 111, '34 hazel park, Newcastle,
 Galway', '087123456', '17-05-2001', 'F', 'GY101', 1)


('Jack Walsh ', 91, '13 College road, Galway',
 '086654321', '01-09-2000', 'M', 'GY350', 3)


('Sue Smyth ', 90, 'Maree, Oranmore, Co. Galway',
 '087111222','25-07-1999', 'F', 'GY406', 3)


('Gerard Kelly', 112, 'Main Street, Oughterard, Co.
 Galway', '087121212', '30-12-2002', F, GY414, 1)
```

```
('Jane Casey', 111, '34 hazel park, newcastle, Galway',
'087123456', '17-05-2001', 'F', 'GY101', 1)

('Jack Walsh ', 91, '13 college road, Galway', '086654321',
'01-09-2000', 'M', 'GY350', 3)

('Sue Smyth ', 90, 'Maree, Oranmore, Co. Galway',
'087111222', '25-07-1999', 'F', 'GY406', 3)

('Gerard Kelly', 112, 'Main Street, Oughterard, Co.
 Galway', '087121212', '30-12-2002', F, GY414, 1)
```

Block 1



Block 2



How are the following supported in SEQUENTIAL file organisation (using results from example 3)?

1. Inserting a new tuple:

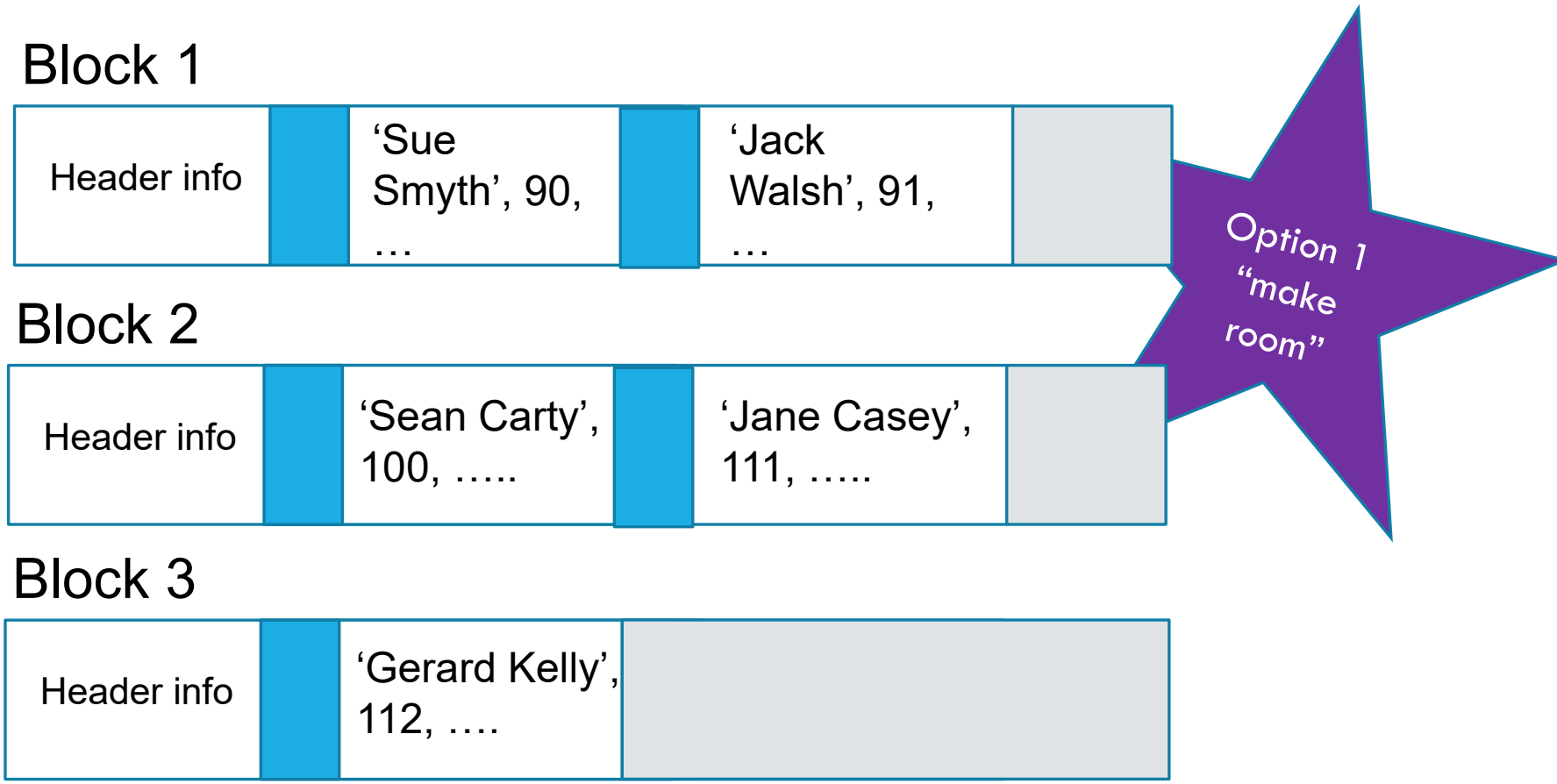
```
('Sean Carty', 100, '23 Ocean view,
Salthill, Galway', '087222333', '24-10-2002',
'M', 'GY101', 3)
```

2. Deleting an existing tuple:

```
('Jack Walsh ', 91, '13 College road,
Galway', '086654321', '01-09-2000', 'M',
'GY350', 3)
```

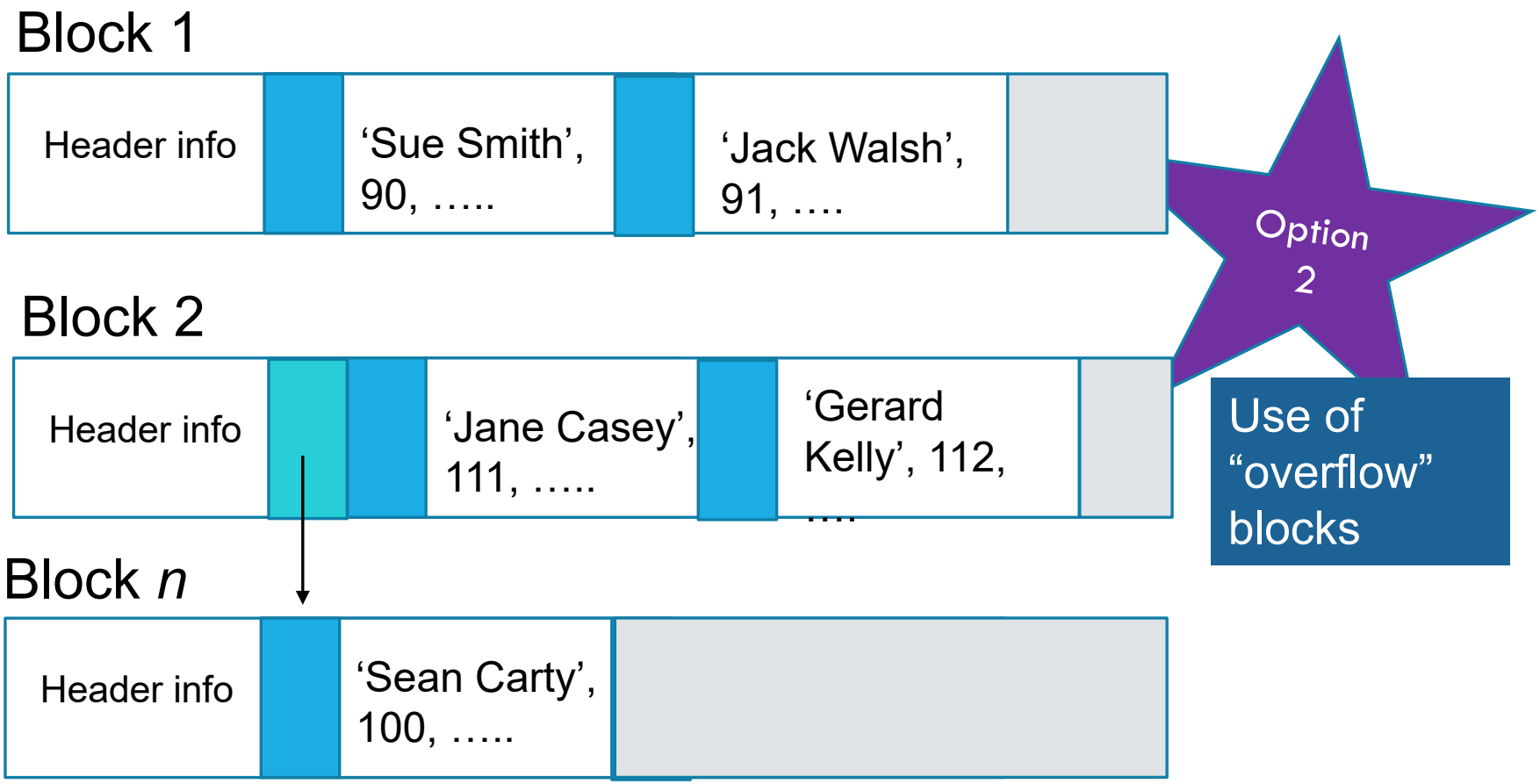
1. Inserting a new tuple:

```
('Sean Carty', 100, '23 Ocean view, Salthill, Galway',
'087222333', '24-10-2002', 'M', 'GY101', 3)
```



1. Inserting a new tuple:

```
('Sean Carty', 100, '23 Ocean view, Salthill, Galway',
'087222333', '24-10-2002', 'M', 'GY101', 3)
```

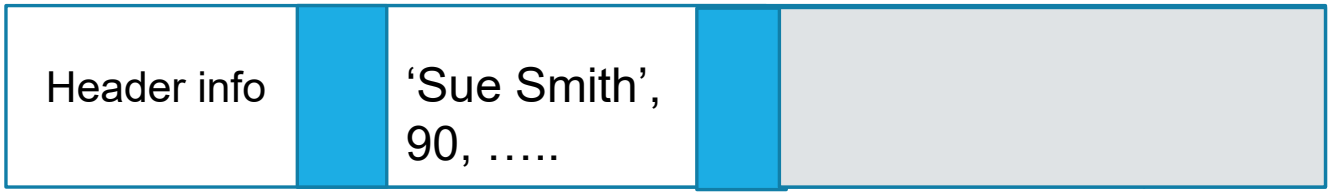


2. Deleting an existing tuple (Option 1):

```
('Jack Walsh ', 91, '13 College road, Galway',
 '086654321', '01-09-2000', 'M', 'GY350', 3)
```

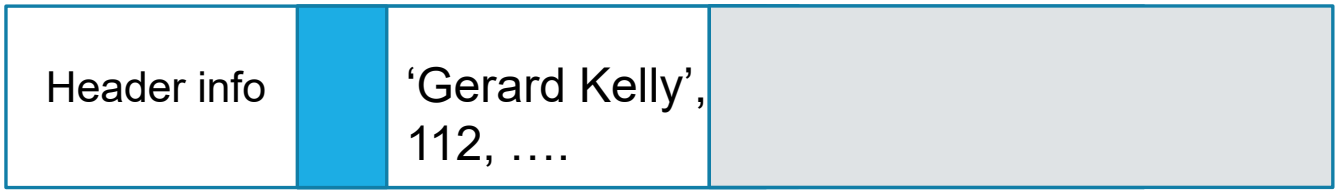
Result:

Block 1



Block 2 ….

Block 3

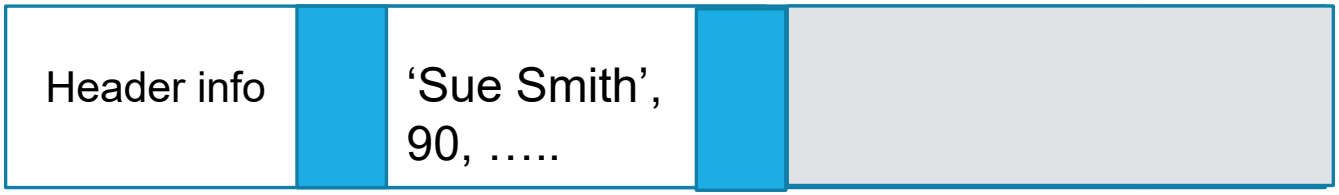


2. Deleting an existing tuple (Option 2):

```
('Jack Walsh ', 91, '13 College road, Galway',
 '086654321', '01-09-2000', 'M', 'GY350', 3)
```

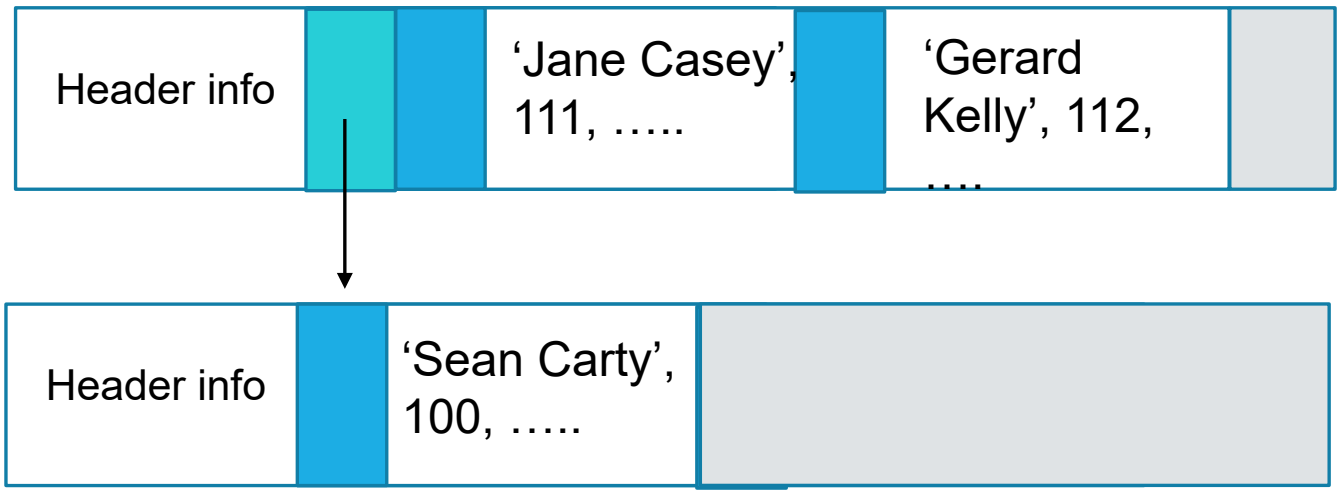
Result:

Block 1



….

Block 2



SEQUENTIAL FILE ORGANISATION

Advantages:

- Reading records in order is efficient
- Searching is efficient on key field (binary search)
- Easy to find 'next record'

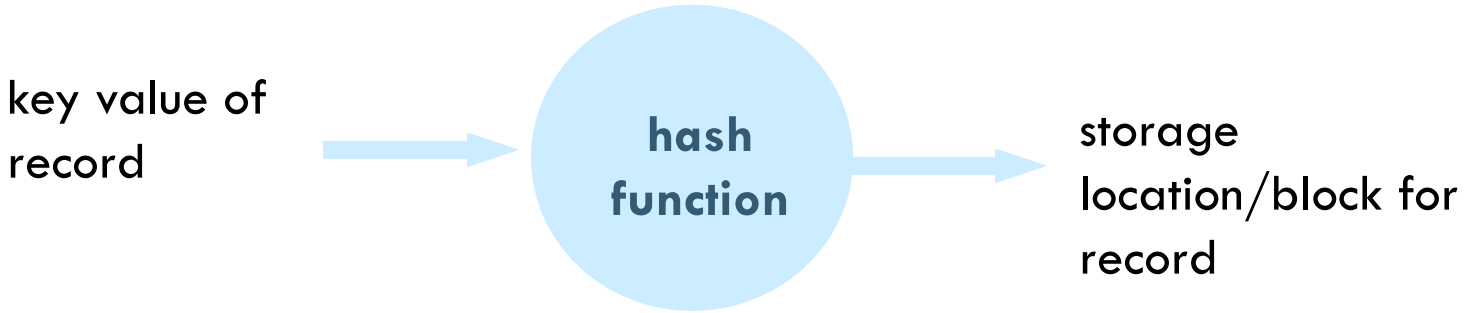
But …

- Insertion and deletion expensive as records must remain physically ordered. Pointer chains used (part of header information)
- What if searching on non-key field?

HASHING/HASHED FILE ORGANISATION

A *hash function* is computed on some attributes of each record (e.g., often key value)

The output of the hash function is the block address where the record should be placed



HASH FUNCTIONS

A common hash function is the MOD function where:

`a MOD b` or `a % b`

returns the remainder on dividing `a` by `b`, i.e. integer division.

Example:

```
20 MOD 7  = 6
100 MOD 5 = 0
```

where `b` should be a prime number – that is a number only evenly divisible by itself and 1

`http://www.onlineconversion.com/prime.htm`

EXAMPLE 4

Given the following records which should be stored in blocks based on user IDs and a hashed file organisation

The available blocks have IDs in the range 0-100 and have a blocking factor of 3

Assign the following records to blocks using user IDs:

1234

167

100

458

Example 4 steps:

1. Get prime number in the range 0-100 as close to 100 as possible - e.g., 97
2. For each key value of each record find the block number of where to place record by getting *keyvalue* mod *primenumber,* e.g., *keyvalue* mod 97

1234 MOD 97 = 70 *(97 divides in to 1234 12 times with remainder 70)*

167 MOD 97 = 70 (once)

100 MOD 97 = 3 (once)

458 MOD 97 = 70 (4 times)

Placing of the records:

Example 4 steps:

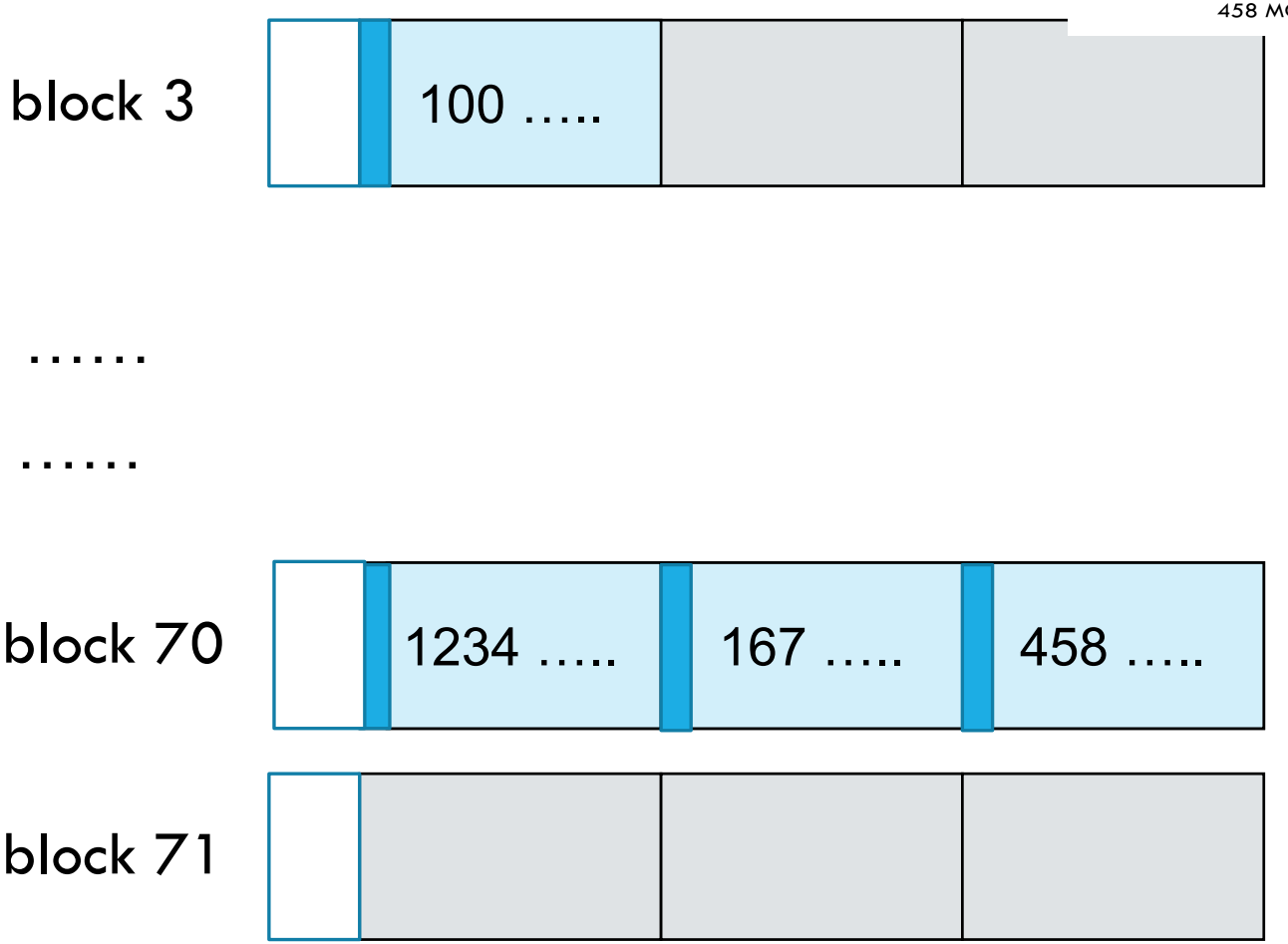
1. Get prime number in the range 0-100 as close to 100 as possible - e.g., 97
2. For each key value of each record find the block number of where to place record by getting *keyvalue* mod *primenumber*, e.g., *keyvalue* mod 97

1234 MOD 97 = 70 *(97 divides in to 1234 12 times)*

167 MOD 97 = 70 (once)

100 MOD 97 = 3 (once)

458 MOD 97 = 70 (4 times)



EXAMPLE 5: Using the student schema from example 1, and given a blocking factor of **2**, with mod function of 97, sketch the placement of the following student records using hashed file organisation:

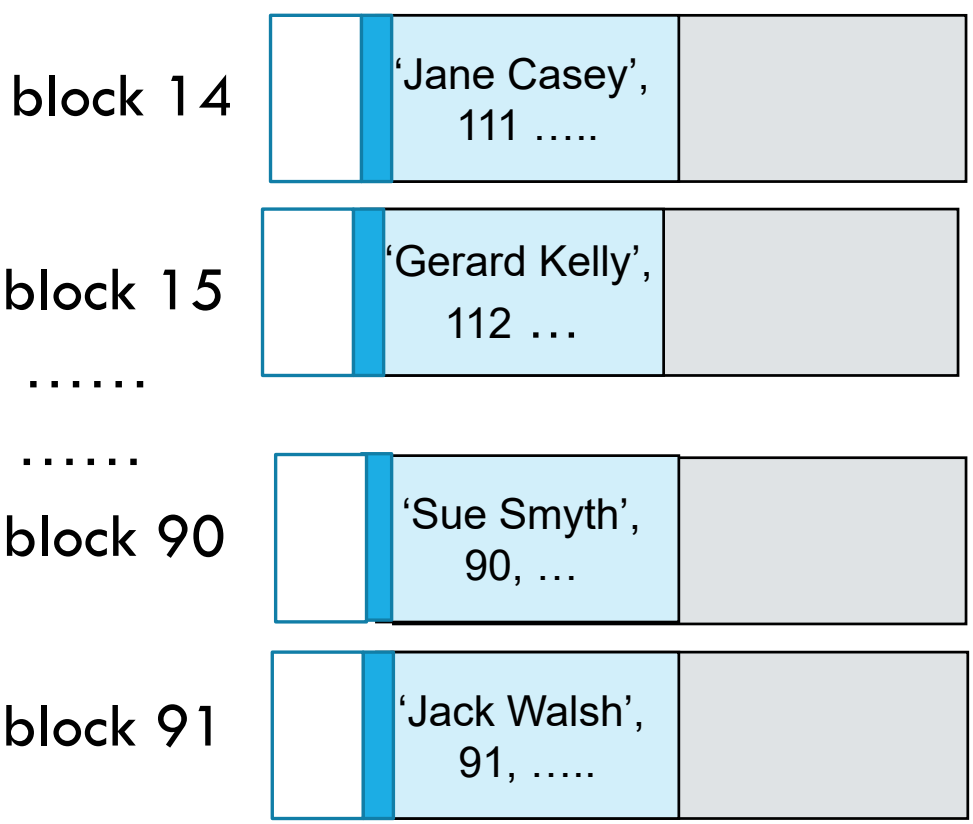
```
('Jane Casey', 111, '34 hazel park, Newcastle, Galway',
'087123456', '17-05-2001', 'F', 'GY101', 1)
```

```
('Jack Walsh ', 91, '13 College road, Galway',
'086654321', '01-09-2000', 'M', 'GY350', 3)
```

```
('Sue Smyth ', 90, 'Maree, Oranmore, Co. Galway',
'087111222', '25-07-1999', 'F', 'GY406', 3)
```

```
('Gerard Kelly', 112, 'Main Street, Oughterard, Co.
 Galway', '087121212', '30-12-2002', F, GY414, 1)
```

```
('Jane Casey', 111, '34 hazel park, Newcastle, Galway', '087123456', '17-05-
 2001', 'F', 'GY101', 1)

('Jack Walsh ', 91, '13 College road, Galway', '086654321', '01-09-2000', 'M',
 'GY350', 3)

('Sue Smyth ', 90, 'Maree, Oranmore, Co. Galway', '087111222', '25-07-1999', 'F',
 'GY406', 3)

('Gerard Kelly', 112, 'Main Street, Oughterard, Co. Galway', '087121212', '30-12-
 2002', F, GY414, 1)
```




```
111 mod 97 = 14
```

```
91 mod 97 = 91
```

```
90 mod 97 = 90
```

```
112 mod 97 = 15
```

How are the following supported in HASHED file organisation (using results from example 5)?

1. Inserting a new tuple:

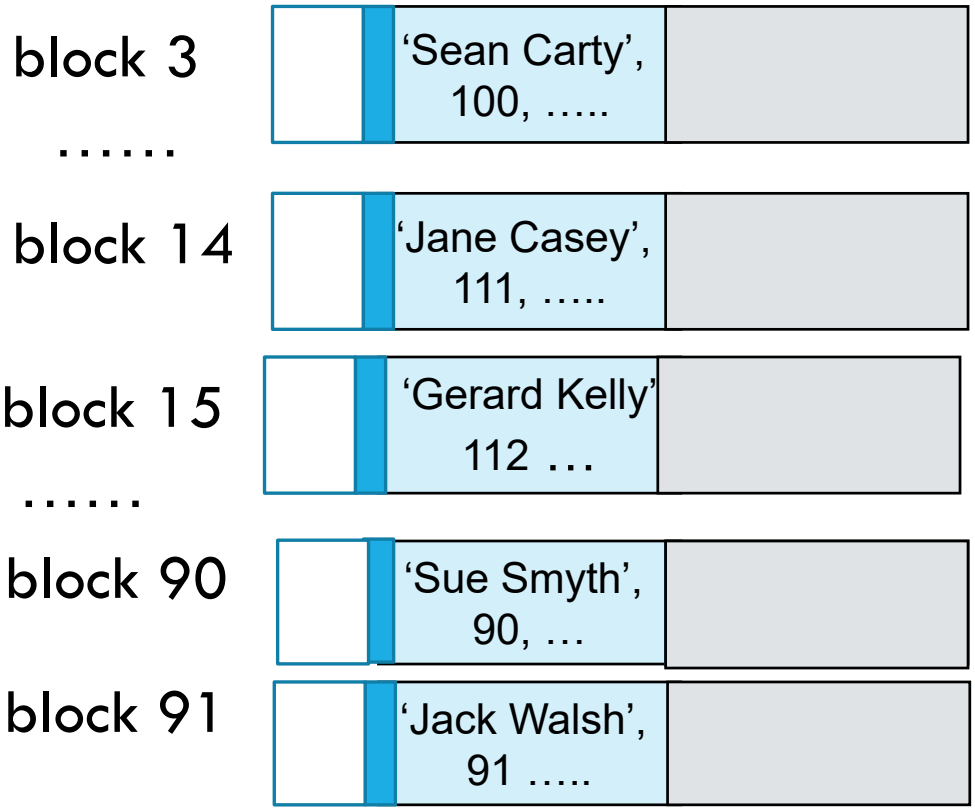
```
('Sean Carty', 100, '23 Ocean view, Salthill, Galway', '087222333', '24-10-2002', 'M', 'GY101', 3)
```

2. Deleting an existing tuple:

```
('Jack Walsh ', 91, '13 College road, Galway', '086654321', '01-09-2000', 'M', 'GY350', 3)
```

1. Inserting a new tuple:

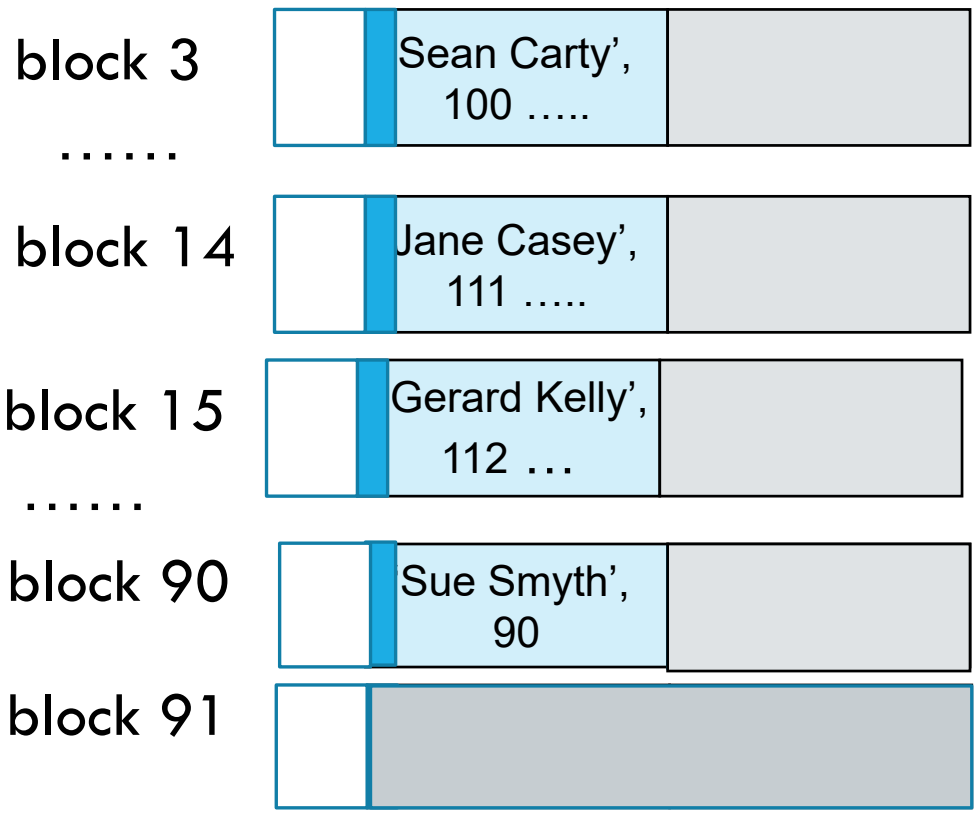
```
('Sean Carty', 100, '23 Ocean view, Salthill,
Galway', '087222333', '24-10-2002', 'M',
'GY101', 3)
```




```
100 mod 97 = 3
```

2. Deleting an existing tuple:

```
('Jack Walsh ', 91, '13 College road, Galway', '086654321',
 '01-09-2000', 'M', 'GY350', 3)
```




```
91 mod 97 = 91
```

QUESTION: Is 97 a good choice for this problem … with 20000 records?

No! Will use blocks from 0 to 96 (97 blocks)

With a blocking factor of 2, at most can fit 97x2 = 194 records

Need a much larger prime number and more blocks

Prime number close to 10000, e.g., 10009, but not much room for growth

Prime number close to 20000, e.g., 19751, would be much better

(http://www.onlineconversion.com/prime.htm)

Criteria for choosing hash function

- Easy and quick to compute (as mod function is)
- Should uniformly distribute hash addresses across the available space … Picking a prime number for the mod function helps with this … but cannot guarantee it
- Anticipate file growth (insertions and deletions) so only a fraction of each block is initially filled, thus leaving room to insert new records

COLLISIONS

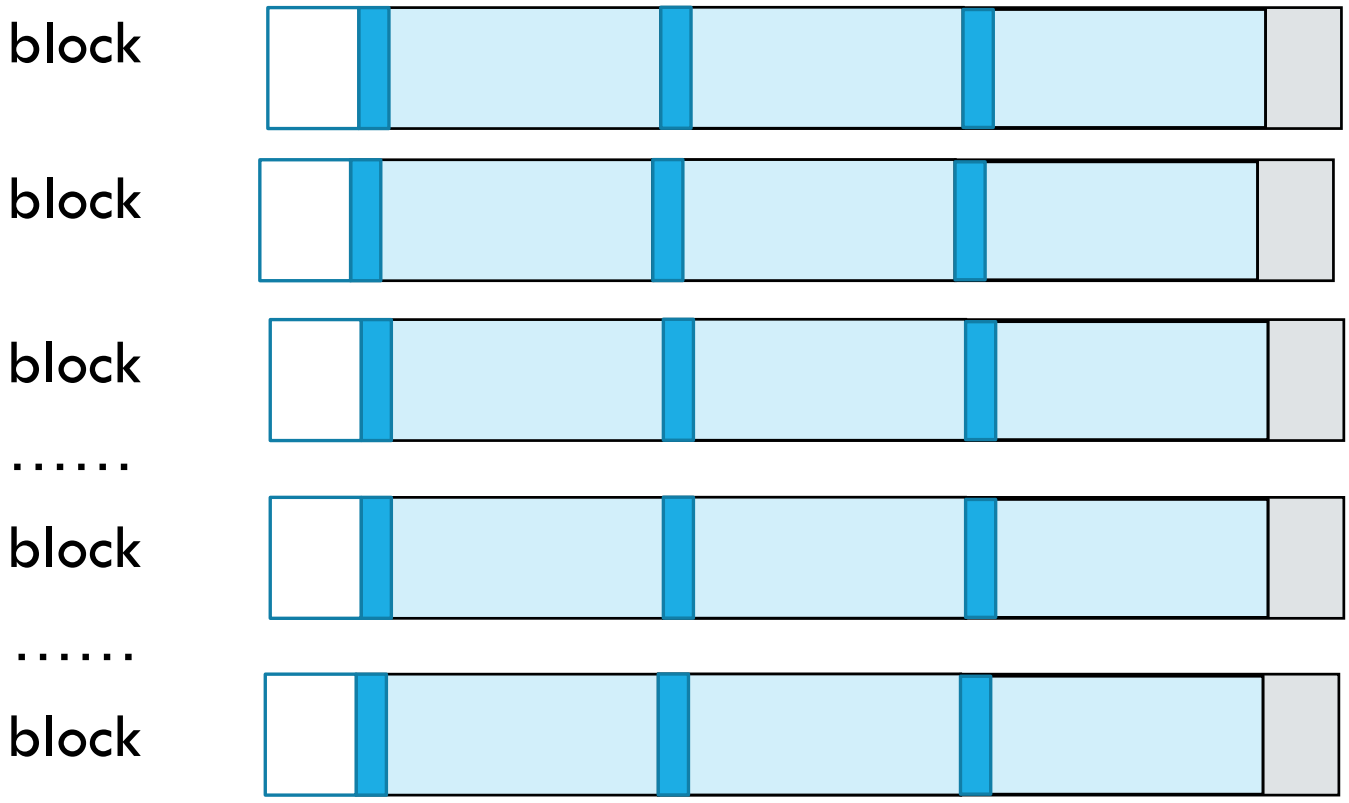
- However, at any stage, two or more key field values can hash to the same location … if there is no room to place record this is called a **collision**
- If a collision occurs, and there is no space in block for new record, then must find a new location … this is called **collision resolution**
- One approach to collision resolution is **Linear Probing**
  - Hash function returns block location i for record
  - If there is no room in block *i* check block *i+1*, *i+2* etc. until a block with room is found
  - Can degrade to a linear scan if load factor is high

EXAMPLE 6:

Given the following key field values of five records, show how the associated records are assigned to blocks using a *hashed file organisation* with the mod function (mod 7) where a blocking factor of 3 is being used and with linear probing collision resolution.

Key values of records: 24, 73, 20, 9, 10, 31

Placing of the records 24, 73, 20, 9, 10, 31 using mod 7 and a blocking factor of 3 and linear probing collision resolution



Calculating blocks IDs:

24 mod 7 =

73 mod 7 =

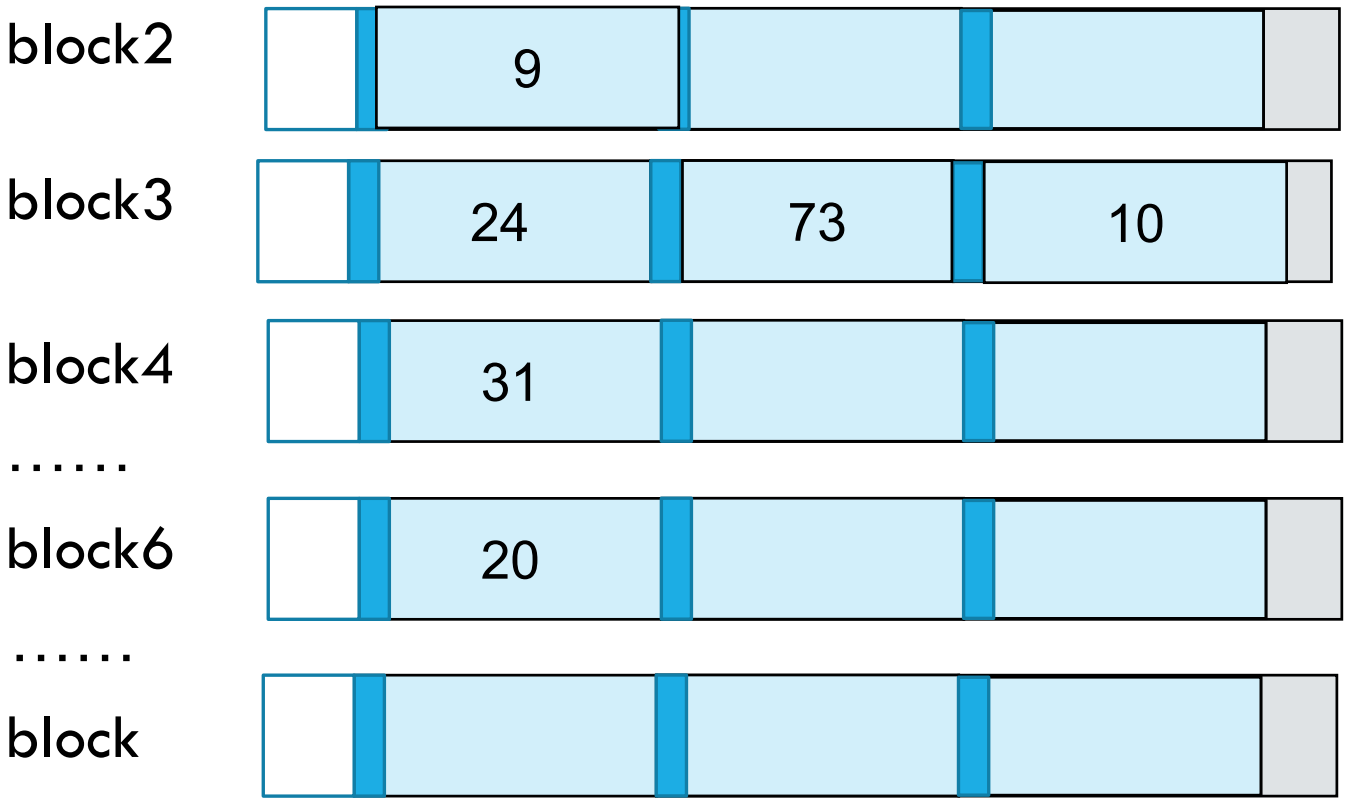
20 mod 7 =

9 mod 7 =

10 mod 7 =

31 mod 7 =

Placing of the records 24, 73, 20, 9, 10, 31 using mod 7 and a blocking factor of 3 and linear probing collision resolution



Calculating blocks IDs:

```
24 mod 7 = 3
73 mod 7 = 3
20 mod 7 = 6
9 mod 7 = 2
10 mod 7 = 3
31 mod 7 = 3
```

DATABASE INDEXES

Indexing speeds-up operations that are not efficiently supported by the basic file organisation.

Consists of ***index entries***

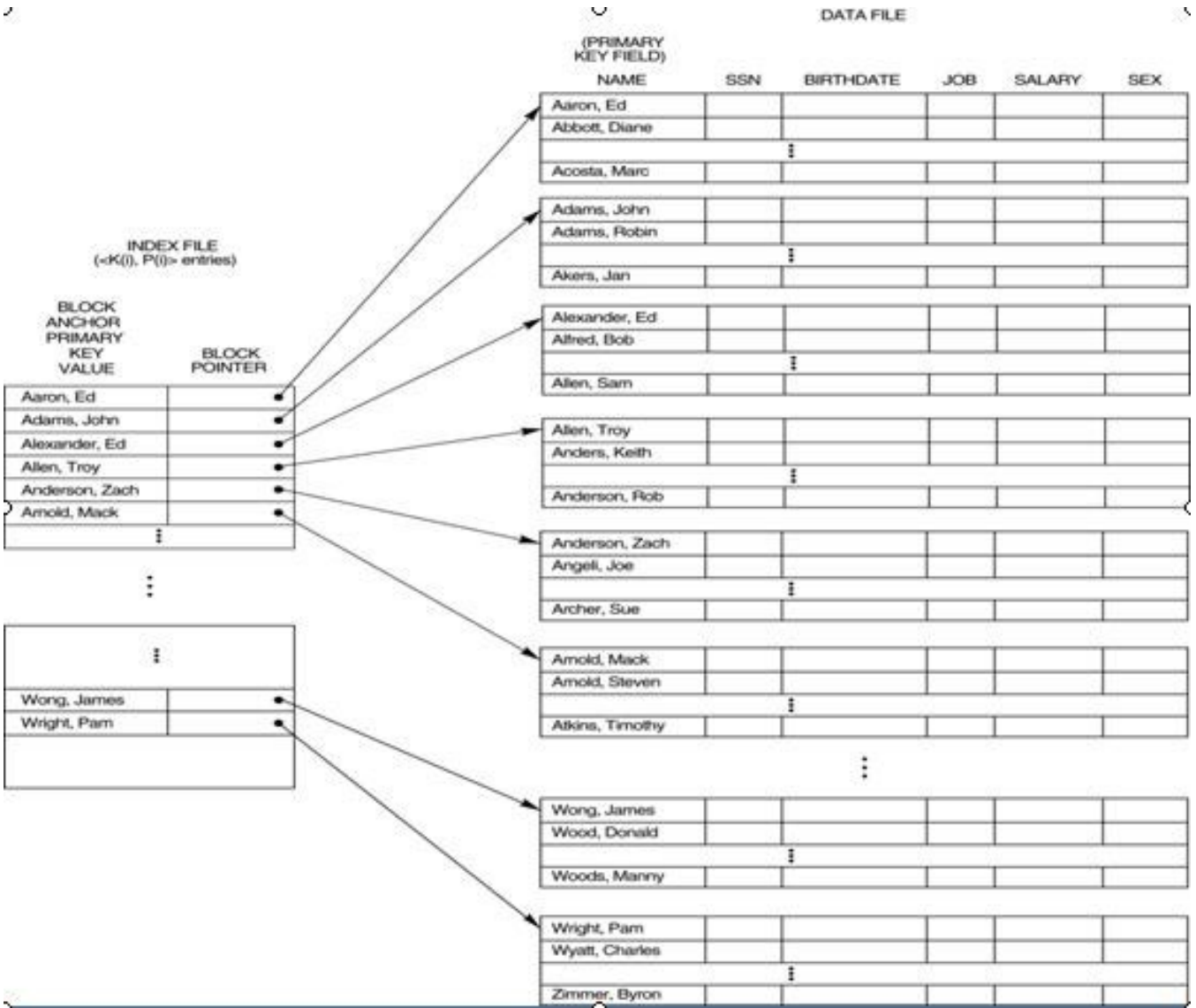
Each index entry consists of:

- index key
- record or block pointer

The index entries are placed on disk, either in sequential sorted order (ordered indexes) or hashed order.

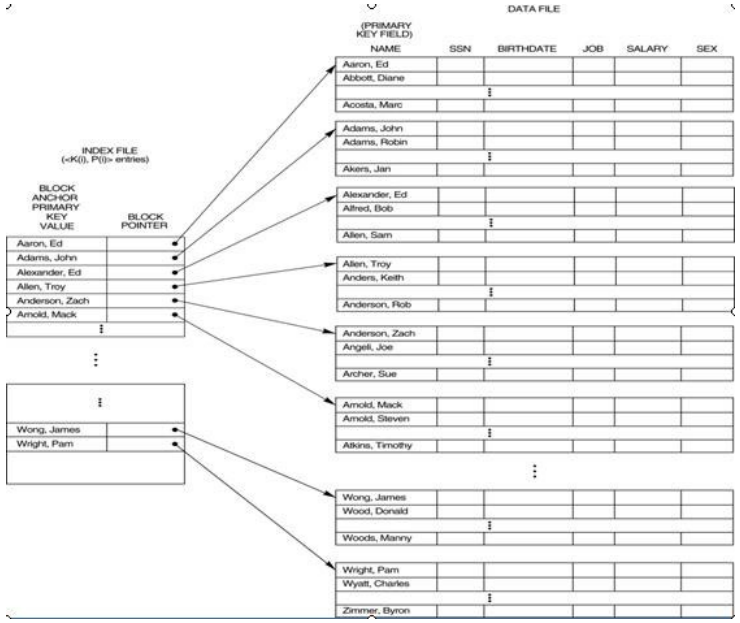
A complete index may be able to reside in main memory

Example of index file organisation of staff schema on name



To access a record using indexing key:

1. Retrieve index file
2. Search through it for required field (based on index key value)
3. Answer query or return to secondary storage for the block which contains the required record.



Dense vs sparse indexes

An index is dense if it contains an entry for every record in the file

A dense index may be created for any index key
A sparse/non-dense index contains an entry for each block rather than an entry for every record in the file and can only be used if the records are assigned to blocks in sorted (sequential) order based on the index key

- A sparse index is called a **primary index**

More on primary indexes

- The total number of entries in the index is the same as the number of **blocks** in the ordered file
- The first record in each block is called the **anchor record** of the block

Advantages:

- Fewer index entries than records so index file is smaller

Disadvantages:

- Insertions and deletions a problem - may have to change anchor record
- Searching may take longer

EXAMPLE 7: Indexed file Organisation

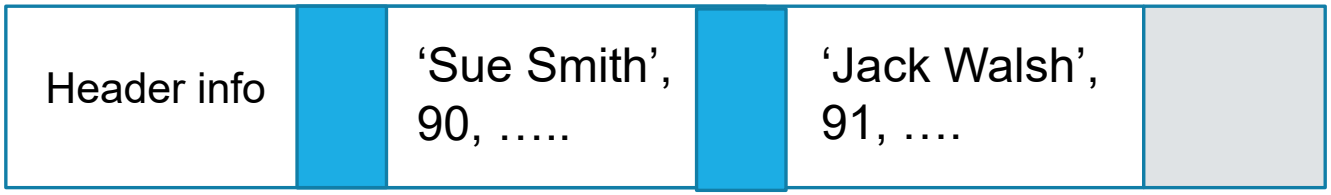
Given the student schema from Example 1, with primary key `studentID`. With the aid of a diagram, illustrate how a **dense** indexing file organisation might operate (with blocking factor of 2 and sequential file organisation)

e.g. for the examples:

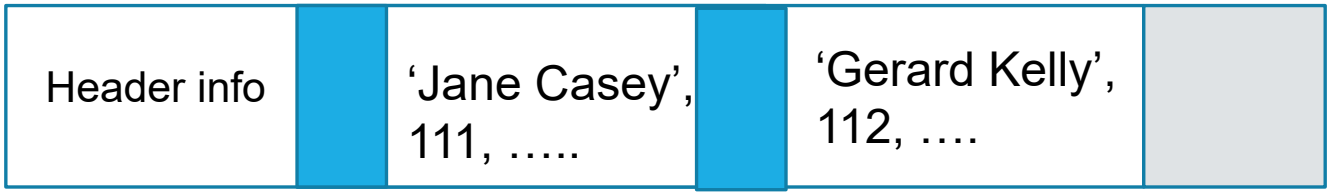
```
('Jane Casey', 111, '34 hazel park, Newcastle, Galway',
'087123456', '17-05-2001', 'F', 'GY101', 1)


('Jack Walsh ', 91, '13 College road, Galway',
'086654321', '01-09-2000', 'M', 'GY350', 3)


('Sue Smyth ', 90, 'Maree, Oranmore, Co. Galway',
'087111222', '25-07-1999', 'F', 'GY406', 3)


('Gerard Kelly', 112, 'Main Street, Oughterard, Co.
Galway', '087121212', '30-12-2002', F, GY414, 1)
```

DENSE …. Index entry for each record

Block 1



Block 2



Index file

90	Block 1
91	Block 1
111	Block 2
112	Block 2

How are the following supported in dense indexed sequential file organisation (using example 7)?

1. Inserting a new tuple:

```
('Sean Carty', 100, '23 Ocean view,
Salthill, Galway', '087222333', '24-10-
2002', 'M', 'GY101', 3)
```

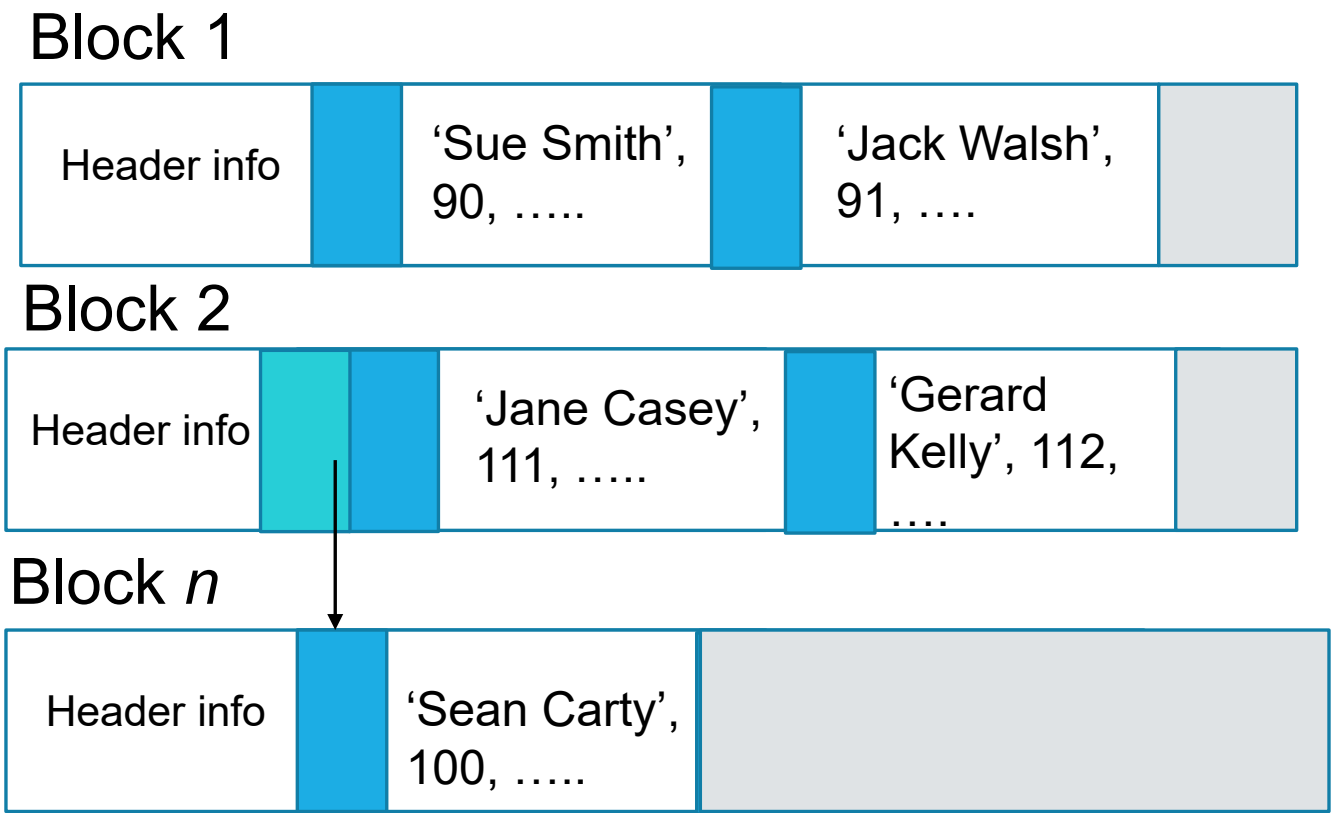
2. Deleting an existing tuple:

```
('Jack Walsh ', 91, '13 College road,
Galway', '086654321', '01-09-2000', 'M',
'GY350', 3)
```

Inserting a tuple … two updates needed

Index file

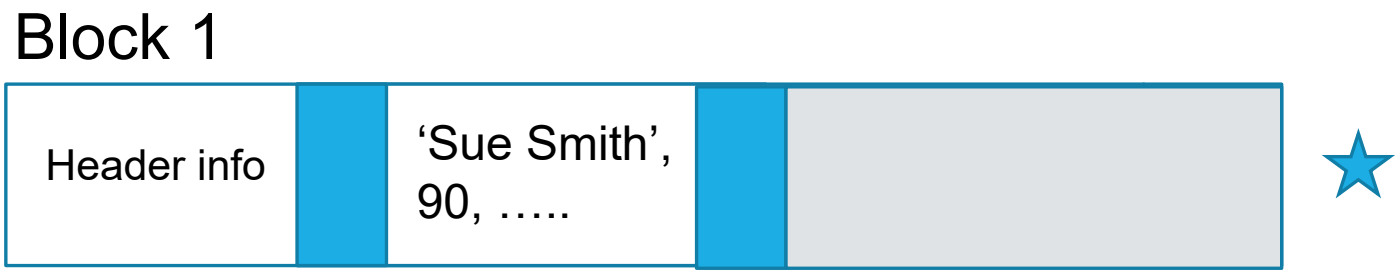
90	Block1
91	Block 1
100	Block n
111	Block 2
112	Block 2

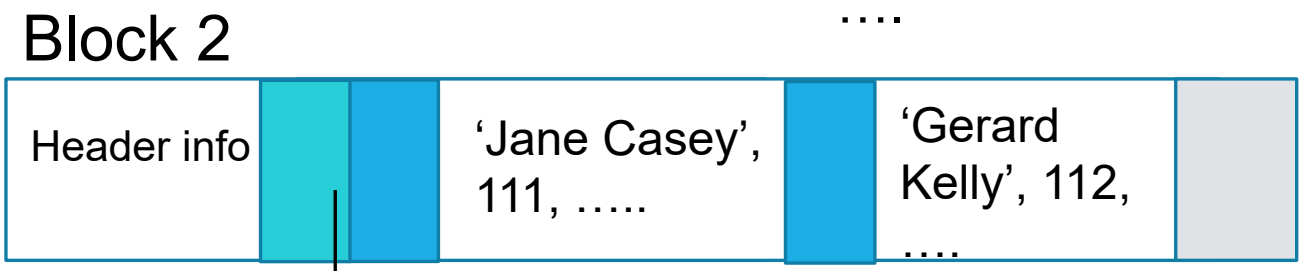


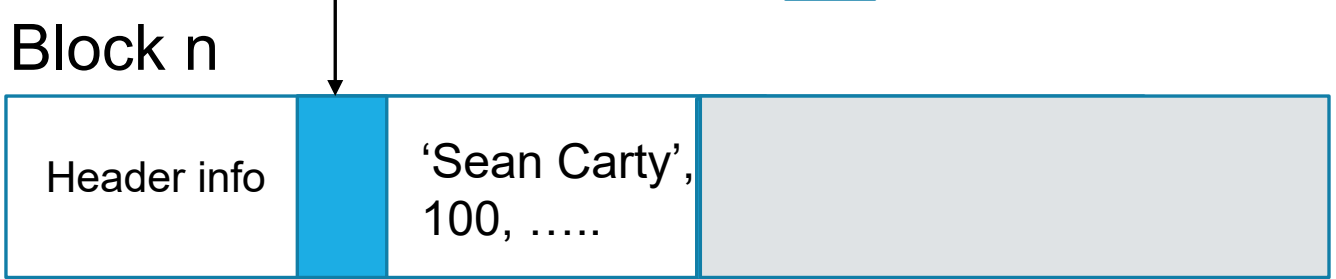
Deleting a tuple … two deletions needed

Index file

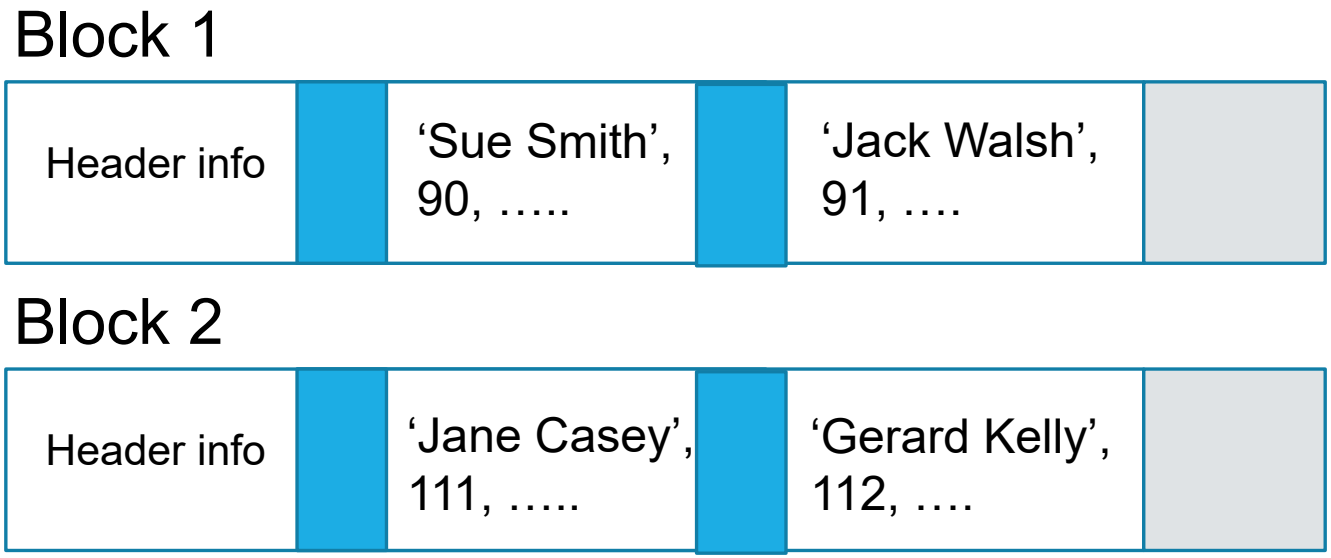
90	Block1
100	Block n
111	Block 2
112	Block 2







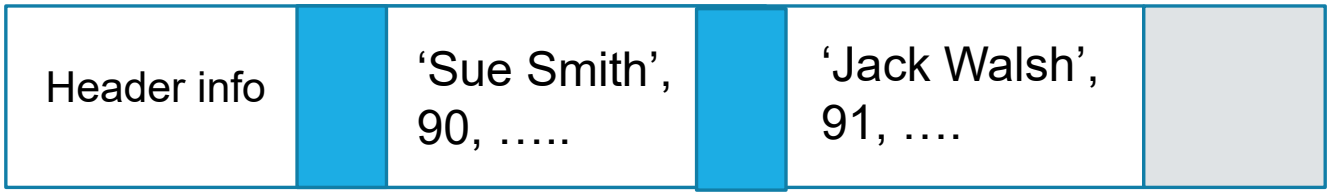
Example 8: Using the illustrated example from example 7, show how the organization of data looks for **non-dense** indexing (sequential organization)



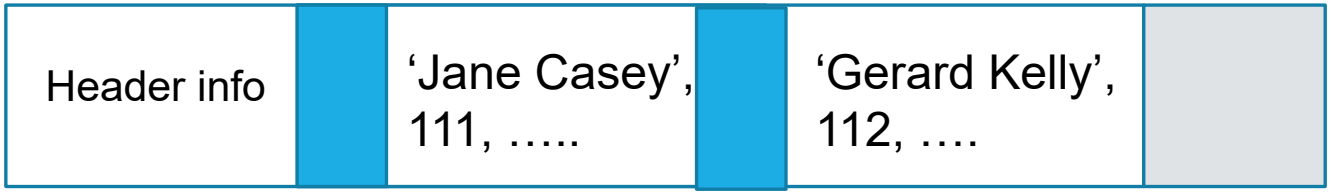
Sparse/Non-dense ….

Index entry for each **block**

Block 1



Block 2



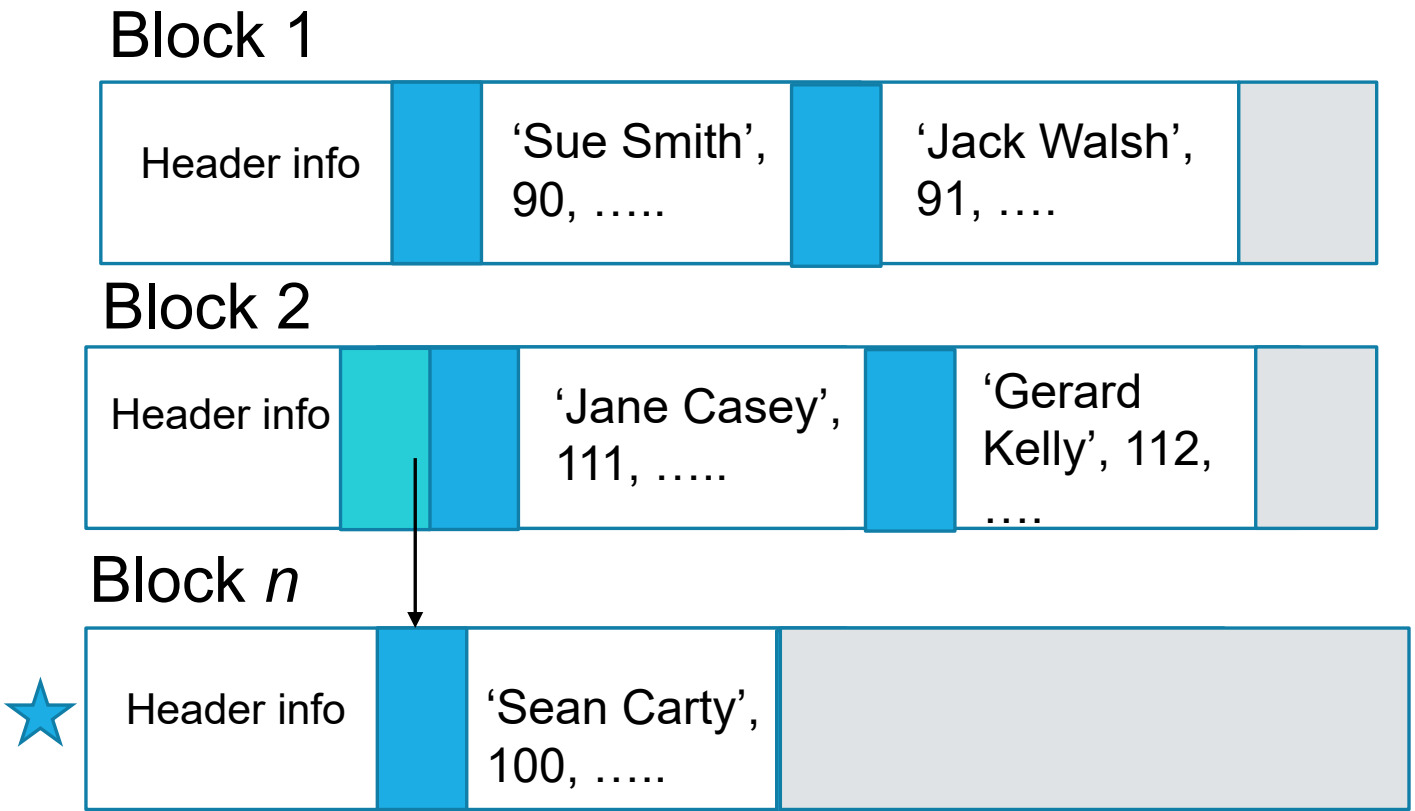
Index file

90	Block 1
111	Block 2

Inserting a tuple with sparse indexing

Index file

90	Block1
100	Block n
111	Block 2

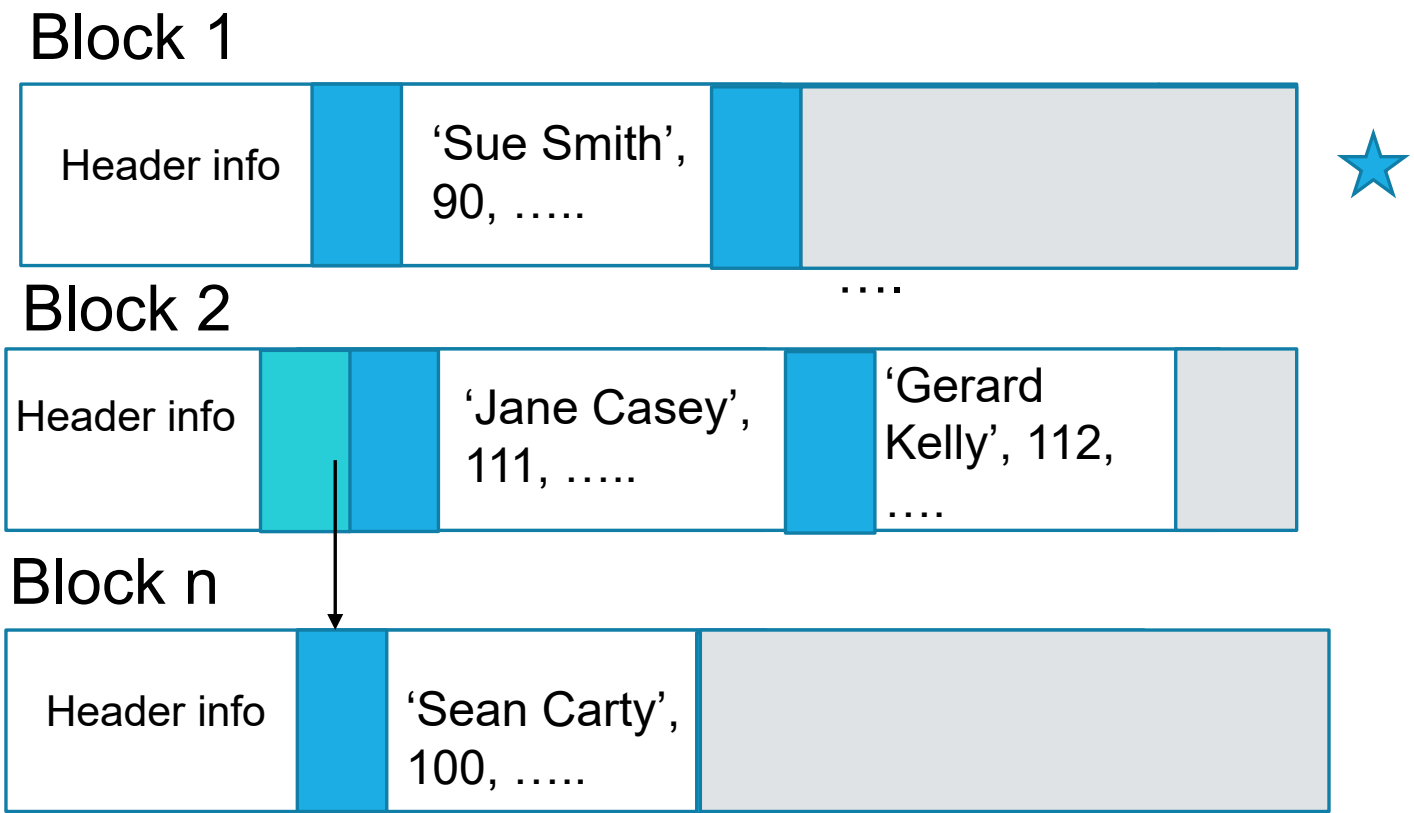


Deleting a tuple … with sparse indexing

Index file

90	Block1
100	Block n
111	Block 2

No change in index



CLUSTERED AND SECONDARY INDEXES

Records that are logically related are physically stored close together on the disk (i.e., in the same blocks or consecutive blocks)

Records are physically ordered on a non-key field that does not have a distinct value for each record

Clustering index consists of:

- clustering field value
- block pointer to first block that has a record with that value for clustering field

Advantages/disadvantages of clustering:

Quick access on clustering field but have to search all blocks in querying on non-clustering fields

Example 9:

Consider a file holding the employee information from the Company schema where each record contains a positive integer indicating the department where an employee works. Show how a clustering index on department number (DNO) might operate on such data – with blocking factor of 3

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

Option 1:
Fill all blocks

Index value (dno)	*block value*
1	b1
4	b1
5	b2

Index file

b1

James E Borg, …. , 888665555, …..	…, 1
Alicia Z Zelaya, …. 999887777, ……	…, 4
Jennifer S Wallace, …. 987654321, ….	…, 4

b2

Ahmad V Jabbar, ……, 987987987, …..	…., 4
John B Smith, …… , 123456789, ….	….., 5
Franklin T Wong, ……., 333445555, ….	……, 5

b3

Ramesh K Narayan, …. 666884444, ….	……., 5
Joyce A English, …….., 453453453, ……	….., 5

bN

Option 2:

Leave 'space' for other records with that clustering field value

Index value (dno)	*block value*
1	b1
4	b2
5	b3

Index file

b1

James E Borg, …. , 888665555, …..	…, 1

b2

Alicia Z Zelaya, …. 999887777, ……	…, 4
Jennifer S Wallace, …. 987654321, ….	…, 4
Ahmad V Jabbar, ……, 987987987, …..	…., 4

b3

John B Smith, …… , 123456789, ….	….., 5
Franklin T Wong, ……., 333445555, ….	……, 5
Ramesh K Narayan, …. 666884444, ….	…., 5

b4

Joyce A English, ……., 453453453, ……	….., 5

Option 3:

Use a *Secondary Index and sequential file organisation*

A secondary index is an index whose index (clustering) value specifies an order different to the underlying sequential order of the file.

Any attribute can be chosen as the clustering index value.

Any number of secondary indexes can be built with different clustering index values.

b1

John B Smith,… ,123456789, ….	…., 5
Franklin T Wong, ., 333445555, …	…., 5
Joyce A English, ., 453453453, .	…., 5

b2

Ramesh K Narayan, ….666884444,	…., 5
James E Borg, …. , 888665555,	…, 1
Jennifer S Wallace, …. 987654321,	….., 4

b3

Ahmad V Jabbar, ……, 987987987,	…., 4
Alicia Z Zelaya, …. 999887777, ……	…, 4

Option 3: *Secondary Index*

Index value (dno) | *block value*

1	A
4	B
5	C

Clustering Index file

Secondary Indexes

A

b2		

B

b2	b3	

C

b1	b2	

b1

John B Smith,… ,123456789, …. …., 5
Franklin T Wong, ., 333445555, … …., 5
Joyce A English, ., 453453453, . …., 5

b2

Ramesh K Narayan, ….666884444, …., 5
James E Borg, …. , 888665555, …, 1
Jennifer S Wallace, …. 987654321, ….., 4

b3

Ahmad V Jabbar, ……, 987987987, …., 4
Alicia Z Zelaya, …. 999887777, …… …, 4

SECONDARY INDEXES

- Does not impact the actual storage of records (which blocks they reside in – which can be sequential)
- Can define multiple secondary indexes as well as a primary index

For example:

Clustering Index

1	A
4	B
5	C

Primary index

123456789	b1
666884444	b2
987987987	b3

Secondary Indexes

A

b2		

B

b2	b3	

C

b1	b2	

b1

John B Smith,… ,123456789, …. …., 5
Franklin T Wong, ., 333445555, … …., 5
Joyce A English, ., 453453453, . …., 5

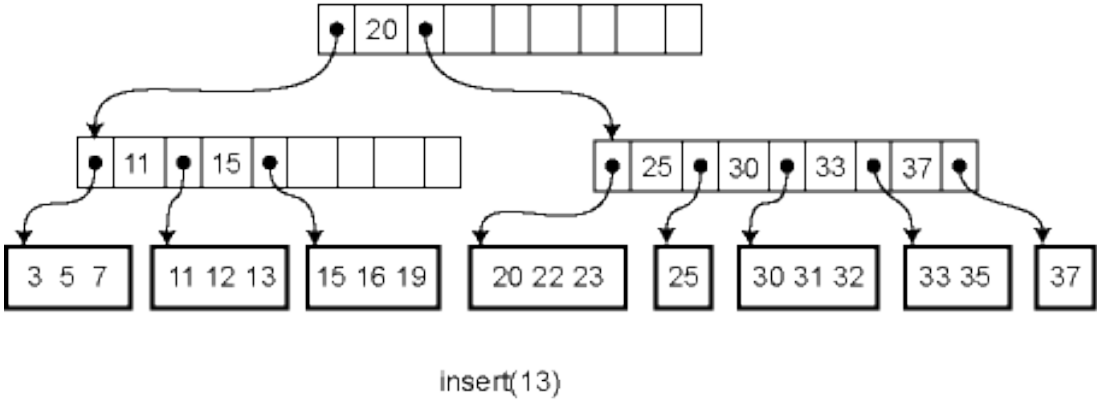
b2

Ramesh K Narayan, ….666884444, …., 5
James E Borg, …. , 888665555, …, 1
Jennifer S Wallace, …. 987654321, ….., 4

b3

Ahmad V Jabbar, ……, 987987987, …., 4
Alicia Z Zelaya, …. 999887777, …… …, 4

B-TREES



Most commercial systems use an indexing structure called B-trees, and specifically B+ trees.

B-trees allow as many levels of indexes as is appropriate for the file being indexed

B-trees manage the space in blocks so that every block is between half-used and completely full

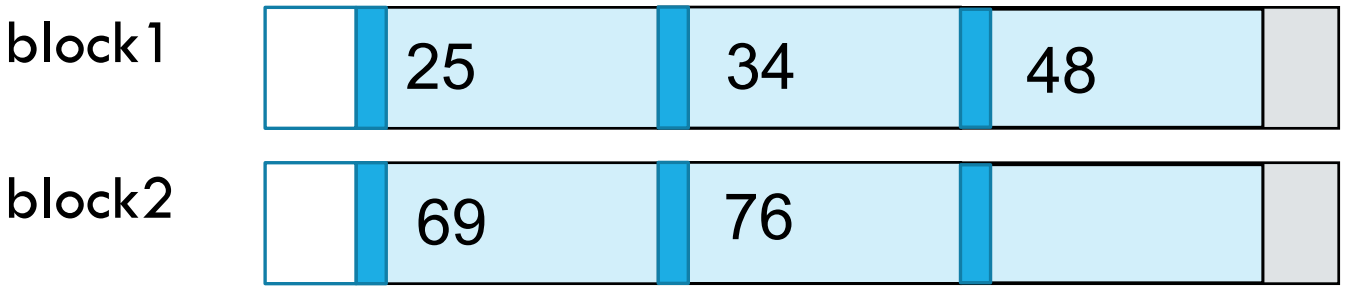
B-trees consist of sequences of pointers arranged in a tree data structure

CLASS WORK ….
WINTER 2017 QUESTION ON FILE ORGANISATIONS

(b)	**Given an unspanned memory organisation, fixed record length, a blocking factor of 3, and five records with the following primary keys:** **25, 34, 48, 69, 76**
(i)	With the aid of examples, outline the main advantages and disadvantages of placing the given records in blocks under a sequential file organisation. (5)
(ii)	With the aid of a diagram, and using sequential file organisation, differentiate between a dense and non-dense indexing of the given five records. (5)
(iii)	With the aid of an example, describe what is meant by secondary indexing. (5)
(iv)	(i) With the aid of a diagram, show where the given five records would be placed in blocks under a hashed file organisation. The mod function (mod 7) should be used in addition to linear probing. (5)

Given an unspanned memory organisation, fixed record length, a blocking factor of 3, and five records with the following primary keys:
25, 34, 48, 69, 76

(i) With the aid of examples, outline the main advantages and disadvantages of placing the given records in blocks under a sequential file organisation. (5)

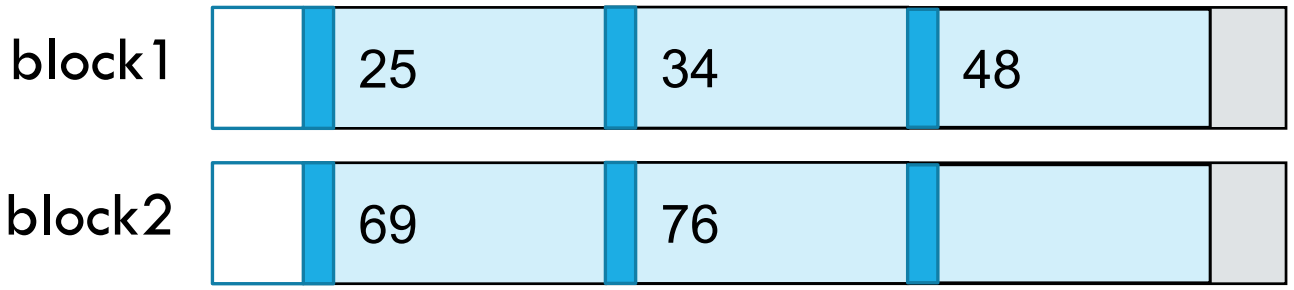


Advantages … reading on key field value (in order)
Disadvantages … maintaining sorted order when adding records

(ii)	With the aid of a diagram, and using sequential file organisation, differentiate between a dense and non-dense indexing of the given five records. (5)

Dense

25	block1
34	block1
48	block1
69	block2
76	block2



Non Dense (Primary Index)

25	block1
69	block2

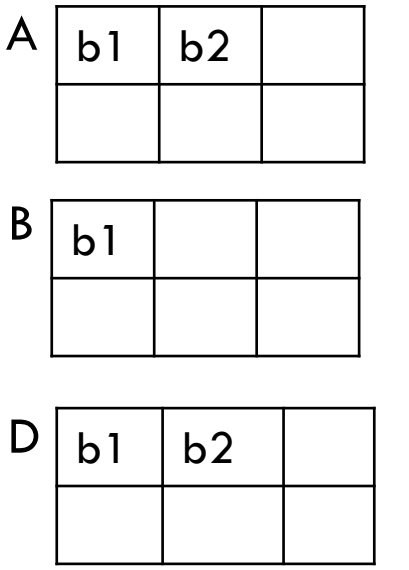
With dense indexing we should have an entry for every record; 5 records implies 5 index entries
With non-dense indexing we should have an entry for every block associated with the file; 2 blocks implies 2 index entries. The key value of the first record in each block is used as the index value.

(iii)	With the aid of an example, describe what is meant by secondary indexing. (5)

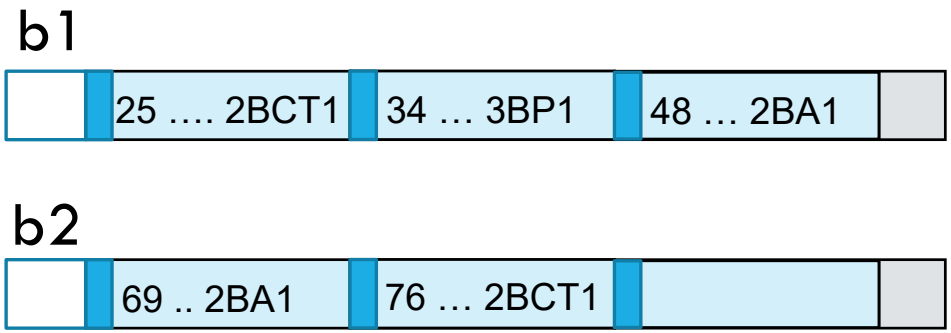
Example: Assuming the primary keys are student IDs (e.g., 25, 34, 48) and we also store the course code for each student (e.g., 2BA, 3BP, etc.) as well as other student information (not shown). Records are assigned to blocks based on the primary key, with a blocking factor of 3.
Course code can be used as a clustering index and the actual references to the blocks holding the student records are stored in a secondary index.

Clustered Index

2BCT1	A
3BP1	B
3BLE1	C
2BA1	D
2BDS1	E

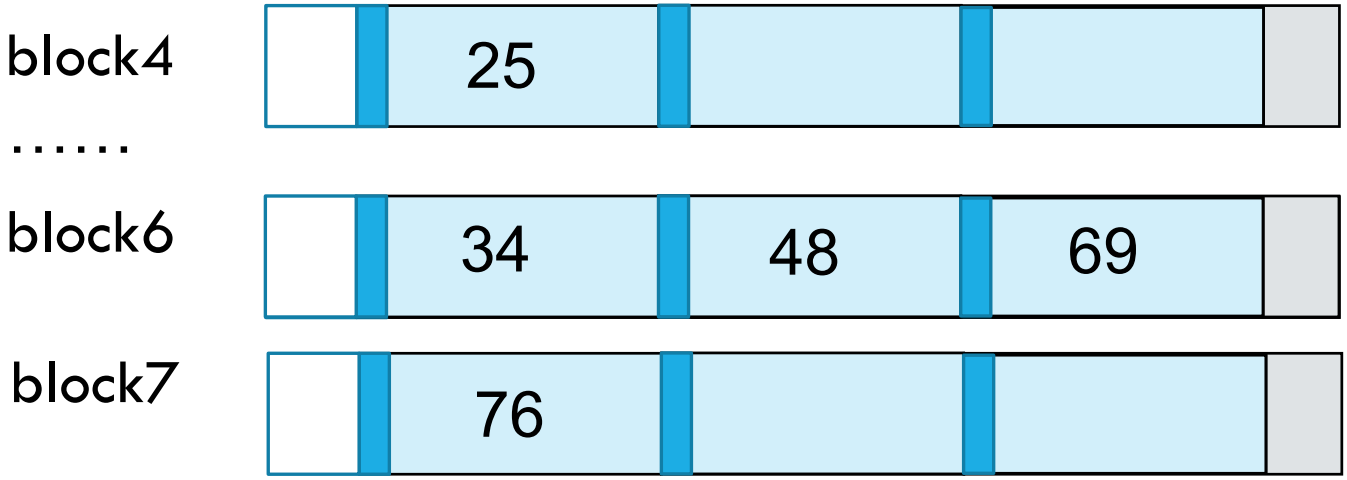


Secondary Indexes



(iv) (i) With the aid of a diagram, show where the given five records would be placed in blocks under a hashed file organisation. The mod function (mod 7) should be used in addition to linear probing. (5)

25, 34, 48, 69, 76



Calculating blocks IDs:

```
25 mod 7 = 4
34 mod 7 = 6
48 mod 7 = 6
69 mod 7 = 6
76 mod 7 = 6
```

SUMMARY: IMPORTANT TO KNOW

- Blocking factor
- Basic 3 organisations: Heaped, Sequential and Hashed (with collision resolution)
- Indexed – Dense and non-dense
- Clustered Index and secondary indexes (not B+ trees)