MODULE 4

INFORMAL DESIGN GUIDELINES FOR RELATIONAL SCHEMAS

There are 4 informal guidelines that may be used as a measure to determine the quality of relational schema. They are

- Making sure that the semantics of the attribute is clear in the schema
- Reducing the redundant information in tuples
- Reducing the NULL values in tuples
- Disallowing the possibility of generating spurious tuples

Making sure that the semantics of the attribute is clear in the schema

Whenever we form the relational schema, there should contain some meaning among the attributes. That is, the attributes belonging to one relation must have certain real world meaning and a proper interpretation associated with them. This meaning is called semantics. Semantics relates one attribute to another

For example:

Employee

eid	ename	Join_date	esal	dno

Here all attributes define the employee relation. That is semantics of the attribute relates to the relation name

Reducing the redundant information in tuples

One goal of schema design is to minimize the storage space used by the relation. If the information is stored redundantly, it leaves to the wastage of spaces. This problem is called update anomalies. There are 3 types of update anomalies

1. Insertion Anomalies

Consider the following tables

Student

Sid	Sname	Sem	Deptno	Dname
1	Anu	1	1	Computer
2	Veena	1	1	Computer
3	Arjun	1	2	English
4	Binil	1	2	English
5	Bindhu	1	1	Computer

Problem

Whenever we insert tuples, if there are ‘n’ students in one department for each semester, then sem,deptno and dname values are repeated ‘n’ times. This situation leads to data redundancy and is called insertion anomalies

2. Deletion Anomalies

Deletion anomalies arises the situation where , if we delete the students from the table, at the time we delete the last student from the table , then the whole information about that department will also be deleted

3. Modification Anomalies

If we change one of the attribute of a table, then we must update all the student details belonging to that attribute

Reducing the NULL values in tuples

If the situation that, many of the attribute do not apply to all tuples in the relation, so that we must put NULL’s in those tuples. It leads to the wastage of storage space and may also cause some update anomalies

NULL values can be applied to a tuple at the following situations

- The attribute does not apply to a tuple
- The attribute value for tuple is unknown
- The value is known but absent

Disallowing the possibility of generating spurious tuples

Spurious tuple means the tuple which is created when two tables are joined badly. So that we should ensure that no spurious tuples are created by the join operation of any relations

Functional Dependency

A functional dependency (FD) is a constraint between two set of attributes from the database. Functional dependency is a relationship that exists when one attribute uniquely determines another attribute.

If R is a relation with set of attributes X and Y such that, a functional dependency between this attribute is represented as X ->Y. It specifies that Y is functionally dependent on X

The value of Y component of a tuple is depend on or determined by the value of X component. Alternatively, the value of X component of a tuple uniquely determines the value of Y component.

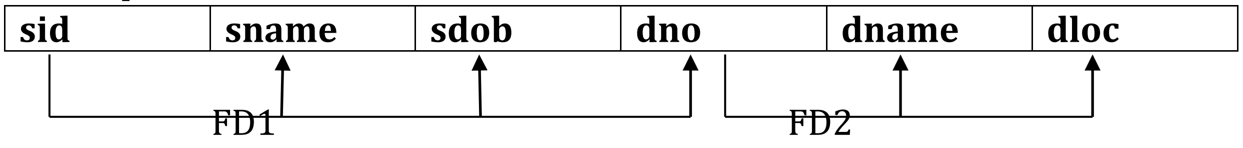
The set of attribute of X is called the left hand side of FD and Y is called the right hand side of FD

Student

sid	**sname**	**sdob**	**dno**	**dname**	**dloc**

From the above table there exist 2 functional dependencies
FD1 = **sid -> sname, sdob, dno**
FD2 = **sid -> dname, dloc**

It is represented in table as



sid	**sname**	**sdob**	**dno**	**dname**	**dloc**

Normalization

The normalization process first proposed by codd in 1972. Codd proposed 3 normal forms first, second, third then next a strong definition of third called BCNF

Normalization is a process of analyzing the given relational schema based on their functional dependencies and primary keys to achieve the desirable properties of

1. Minimizing redundancy
2. Minimizing insertion, deletion and update anomalies

Normal form: The normal form of a relation refers to the highest normal form condition that a relation meets

Normalization is done through decomposition. It guarantee 2 properties

1. **The non-additive join or lossless join property**
   This property ensures that no spurious tuples are created at the time of normalization through decomposition
2. **The dependency preservation property**
   This property ensures that after decomposing the relation, the resulting schema also contain all the functional dependencies that exists in the actual relational schema

De-normalization: it is the process of storing the higher normal form relation to a base form relation (lower normal form).

First Normal Form (1NF)

1NF was designed to disallow multi-valued attributes, composite attributes and their combinations

Definition: The domain of an attribute must include only atomic(simple, indivisible) values. The value of any attribute in a tuple must be single value from the domain of that attribute

The only attribute values permitted by the 1NF are single atomic or indivisible values

Consider the example

Employee

eid	ename	esal	Eaddress
.....	.....	.....	.....

The above table contains multi-valued and composite attributes. So this relation is not in 1NF. We can normalize it in to 1NF by expanding the attributes

Employee

eid	fname	lname	esal	house	place	pin
.....	.....	.....	.....	....	....	.....

Here the above table contains only single atomic attributes. So that this relation is in 1 NF

Second Normal Form(2NF)

2NF is based on the concept of full functional dependency

A functional dependency(FD) X->Y is a full functional dependency, if removal of any attribute A from X means that the dependency does not hold any more.

Ie A £ X, (X – A) $\not\rightarrow$ Y , y does not dependent on x

A functional dependency(FD) X->Y is a partial functional dependency, if we remove any attribute A from X , then the functional dependency still holds.

Ie A £ X, (X – A) $\longrightarrow$ Y , still also y dependent on x

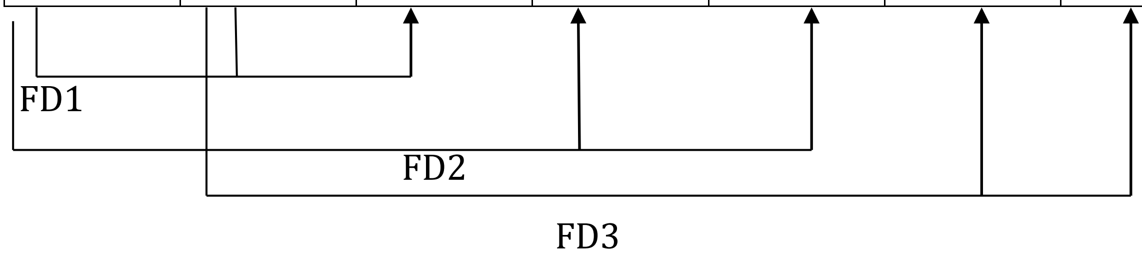
2NF definition:

It must be in 1 NF and every non-prime attribute A in R is fully functionally dependent on the primary key of R. It does not allow partial functional dependency.

Consider the example

Employee_project

eid	pno	hours	fname	Lname	pname	ploc
…….	…..	…..	….	……	…..	….



Above relation contains 3 functional dependencies

FD1= eid,pno -> hours

FD2=eid -> fname,lname

FD3= pno -> pname,ploc

If we remove eid, then FD3 exists

If we remove pno, then FD2 exists

So this relation contains partial functional dependency. So that it is not in 2NF. To normalize it into 2 NF, decompose the relation based on functional dependency. It contains 3 FD's that will become 3 new relations.

EMP-PRJT1

eid	pno	hours
….	….	….

EMP-PRJT2

eid	fname	lname
….	….	….

EMP-PRJT3

pno	pname	ploc
….	….	….

Third Normal Form (3NF)

It is based on the concept of transitive dependency

A functional dependency (FD) X -> Y is a transitive functional dependency, if there exists a set of attribute Z in R , that is neither a candidate key nor a subset of any key of R and both X -> Z , Z -> Z hold

Ie, if there exists X -> Z, Z -> Y FD's then automatically X -> Y exists. It is a transitive dependency

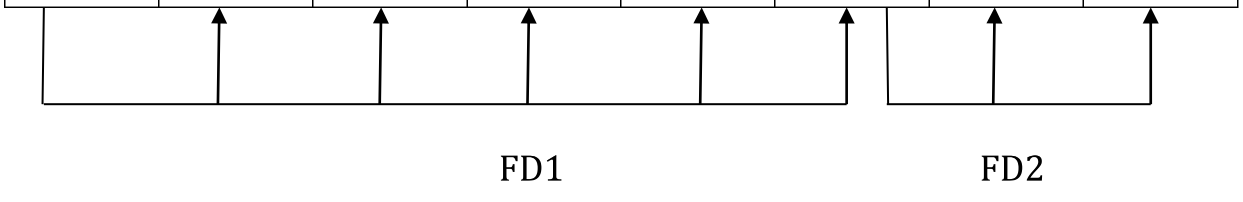
Definition:

A relational schema R is in 3NF, if it satisfies 2NF and no non-prime attribute of R is transitively dependent on the primary key

Consider the following relation

EMP-DEPT

eid	fname	lname	esal	emob	Dno	dname	dloc
….	….	….	….	….	….	….	….



This relation contains 2 functional dependencies

FD1= eid -> fname,lname,esal,emob,dno

FD2= dno -> dname,dloc

Consider the following. There is a functional dependency between

eid ->dno , dno ->dname $\Longrightarrow$ eid -> dname

also

eid ->dno , dno ->dloc $\Longrightarrow$ eid -> dloc

These are transitive dependencies. So that this relation is not in 3NF. To normalize it into 3 NF, decompose the relation based on functional dependency. The result will be

EMP-DEPT 1

eid	fname	lname	esal	emob	Dno

EMP-DEPT 2

dno	dname	dloc

It does not contain any transitive dependencies. So these relations are in 3NF

General definitions of 2NF and 3NF

Boyce Codd Normal Form(BCNF)

It was proposed as a simpler form of 3NF but found to be stricter than 3 NF. That is, **every relation in BCNF is also in 3NF. But a relation in 3 NF is not necessary in BCNF.**

Definition:

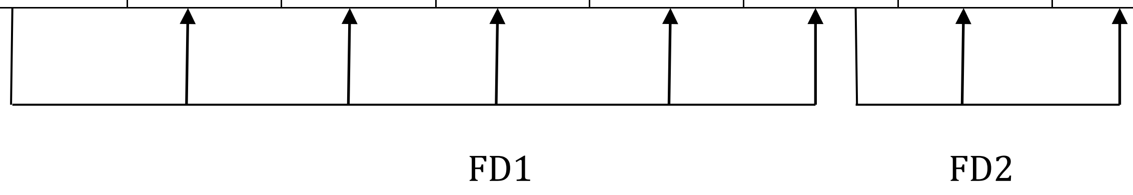
A relation schema R is in BCNF, then if for every one of its functional dependencies (FD) X->Y hold one of the following conditions

1. X -> Y is a trivial FD
2. X is a super key for schema R

Consider the following relation

EMP-DEPT

eid	fname	lname	esal	emob	Dno	dname	dloc
….	….	….	….	….	….	….	….



This relation contains 2 functional dependencies

FD1= eid -> fname,lname,esal,emob,dno

FD2= dno -> dname,dloc

Consider the following. There is a functional dependency between

eid ->dno , dno ->dname $\Longrightarrow$ eid -> dname

also

eid ->dno , dno ->dloc $\Longrightarrow$ eid -> dloc

These are transitive dependencies.

To normalize it into BCNF, decompose this relation as

EMP-DEPT 1

eid	fname	lname	esal	emob

EMP-DEPT 2

dno	dname	dloc

EMP-DEPT 3

eid	dno

Now these relations are in BCNF

Types of single level ordered indexes

Indexes are used to retrieve the data from the record efficiently. It is same as the index shown in the front of text books

Index structure is usually defined on a single field of a file, called indexing field or indexing attribute

Several types indexing available

1. Primary indexes
2. Clustering indexes
3. Secondary indexes

Primary indexes

Primary index is an ordered file whose records are of fixed length with two fields and it acts like an access structure to efficiently search and access data records in a file.

The index file has two fields. **The first field specifies the primary key of the first record in each block and the second field specifies the pointer to that record. Data is stored in blocks. Index file contains one index entry for each block in the file**

We can represent it as

< k(i) , p(i) >

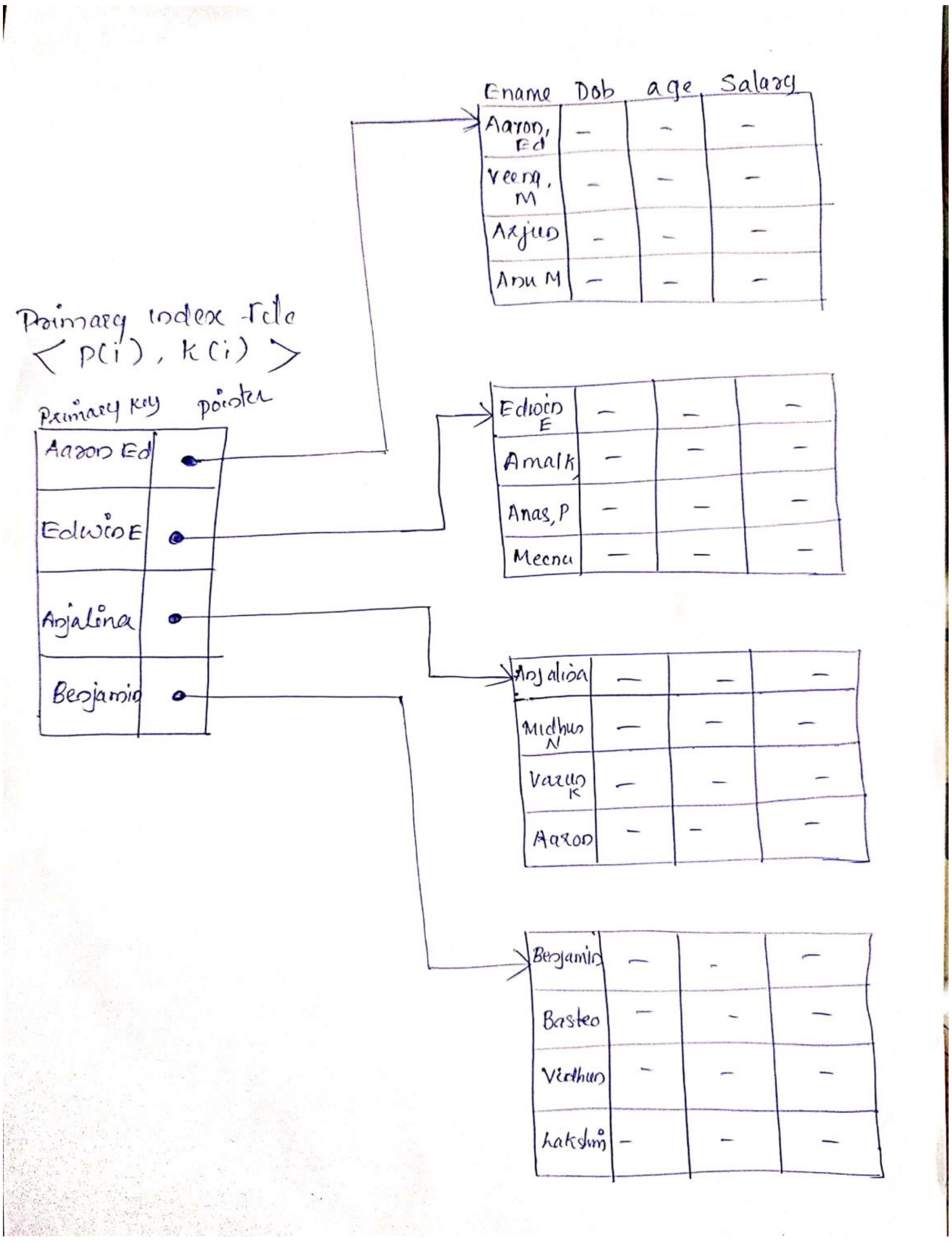
K(i) – primary key of i^{th} record

P(i) – pointer to i^{th} record

The total number of entries in the index file is equal to the number of disk block in the data file.

The first record in each block of data file is called the anchor record or block anchor

Index is classified as dense or sparse. A dense index has index entry for every record in the data file. Sparse index has index entry for some of the records in the data file



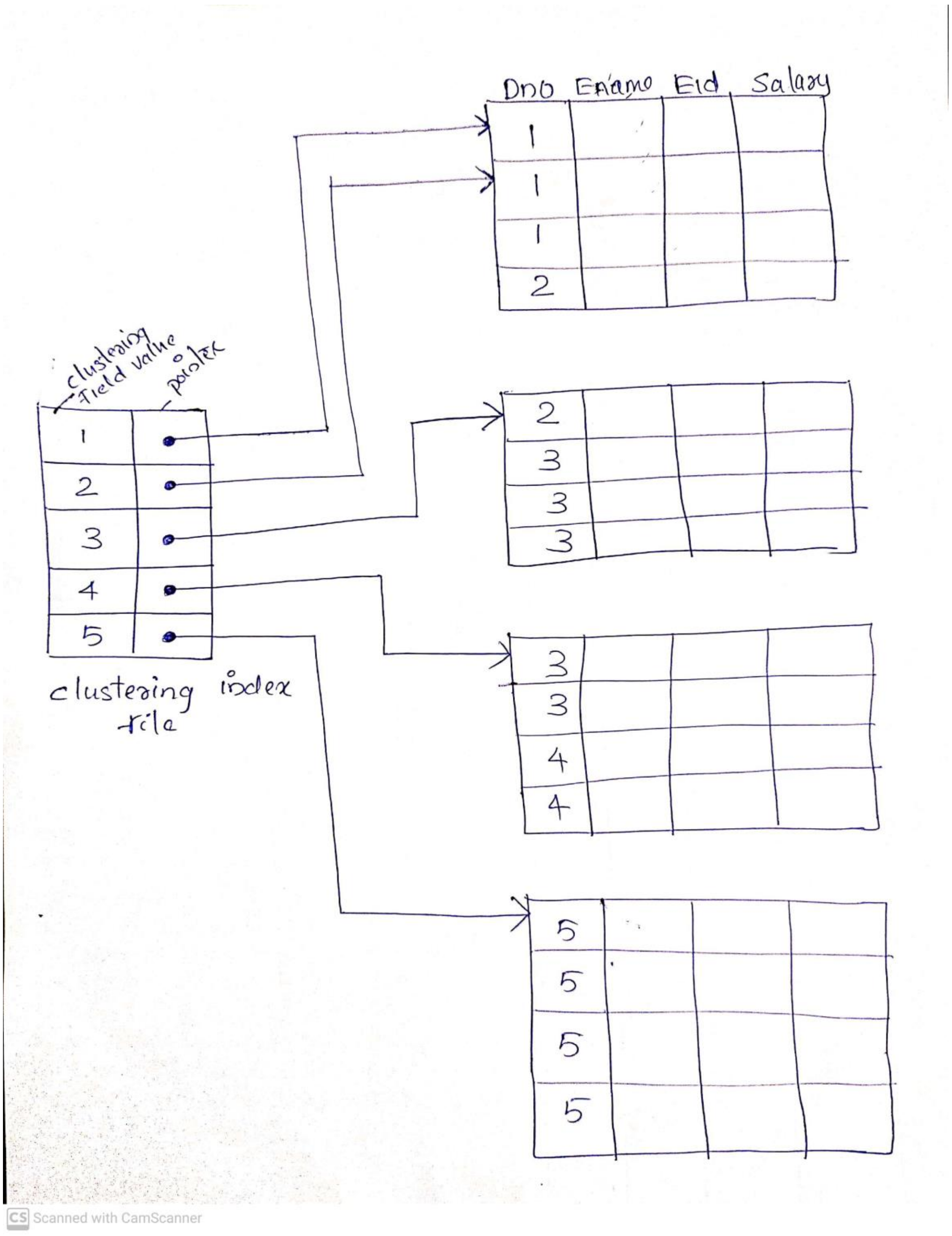
Primary index is an example of sparse index

Clustering index

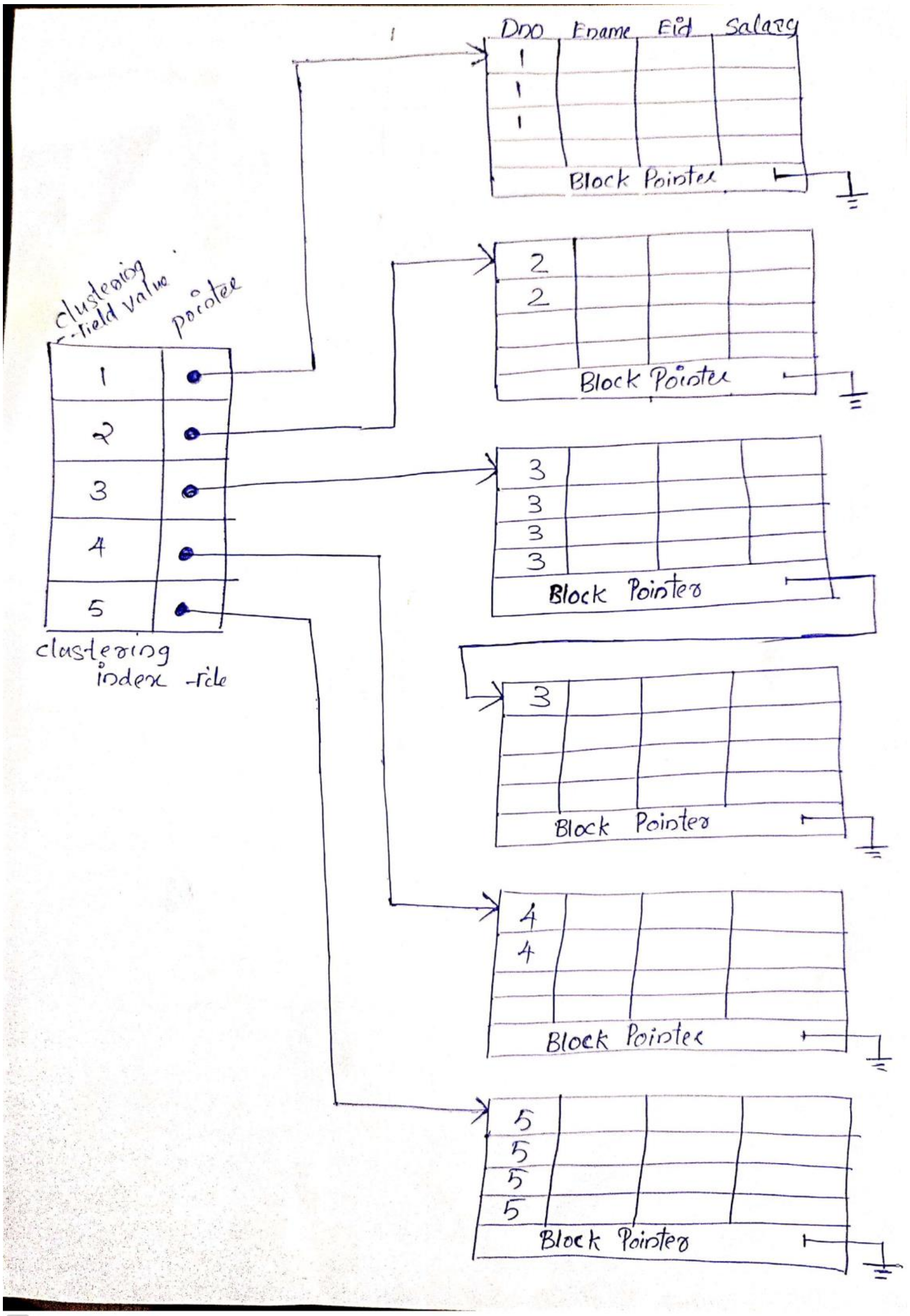
In clustering index, the record are ordered on a non-key field. That is ,it does not have distinct value for each record in the file. Then that non-key filed is called clustering field and the data file in which contains the clustering field is called clustering file.

Clustering index also have 2 fields. The first field is the value of clustering field and the second field is the pointer to that field.

Pointer is always point to the first record in the data block which the record contains.



In clustering index, there exists certain problems as we insert new data. The modified form of clustering index is shown below



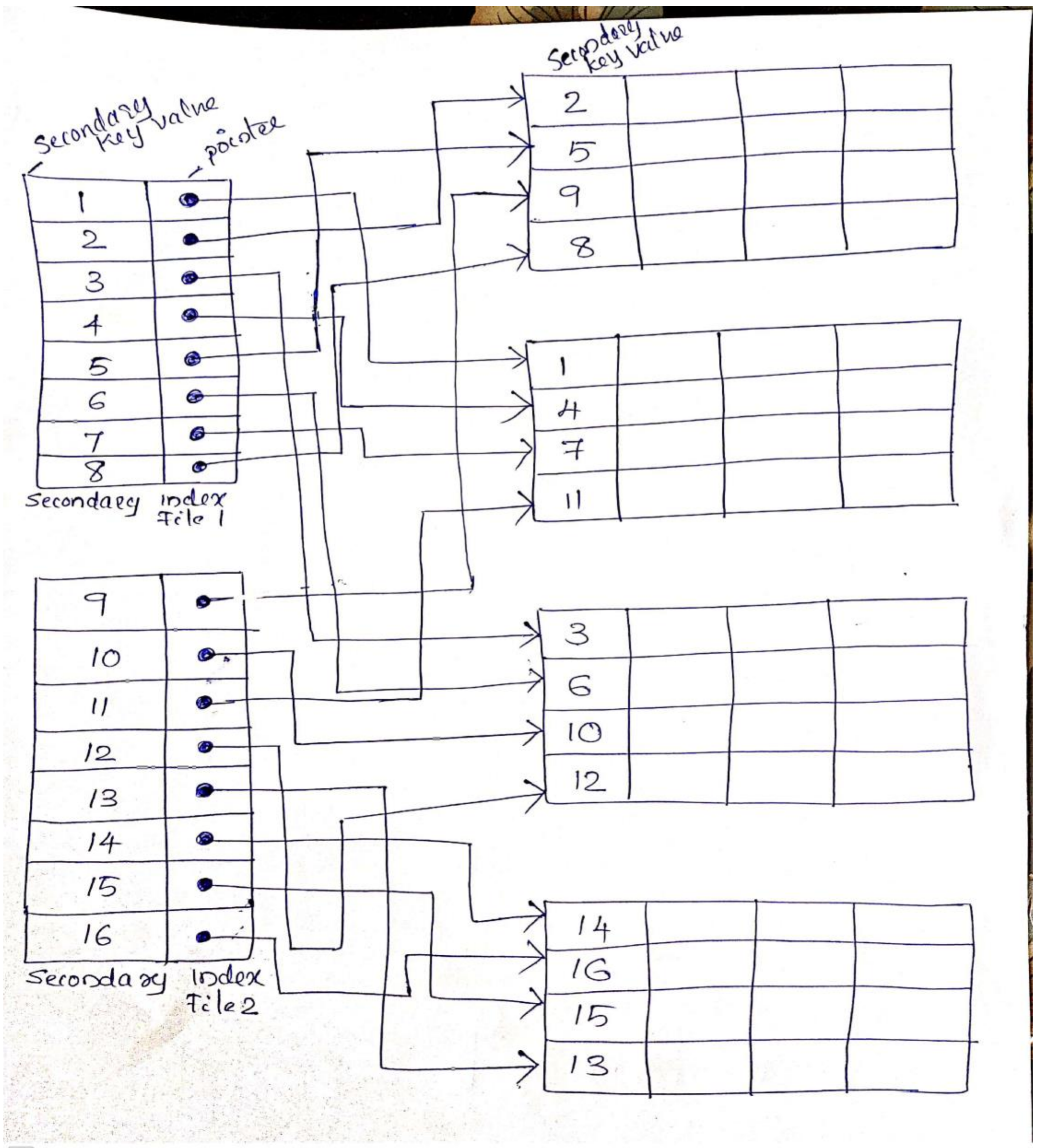
Clustering index is an example of sparse index

Secondary index

Secondary index provides secondary means of accessing the data file if it has some primary access exists.secondary index is based on secondary key or alternative key of the record which has unique value in every record

Secondary index file contains two fields. The first field is the seondary key value and the second foeld is the pointer to that field.

Secondary index has pointer to all records in the file. It is an example of dense index



Scanned with CamScanner