- There are two types of functions (built-in) in oracle.
  1. Single row / scalar functions
  2. Group functions.
- Single row / scalar functions are –
  (i) Numeric functions
  (ii) string functions
  (iii) date and time function
  (iv) conversion functions.
- Group functions are –
  (i) Aggregate functions.
- Numeric functions are –
  a) abs(x)
  b) ceil (x)
  c) floor(x)
  d) Trunc().
  e) Round()
- string functions are –
  a) lower()
  b) upper()
  c) initcap()
  d) ltrim()
  e) rtrim()
  f) trim()
  g) lpad()
  h) rpad()
  i) substr()

k) length( )

l) concat( )

- Date functions are –

    a) add-months( )

    b) months-between( )

    c) next-day( )

    d) last-day( )

    e) sysdate( )

    some other date functions are –

    f) least( )

    g) greatest( )

    h) trunc( )

    i) round( )

- Conversion functions are –

    a) to-char( )

    b) to-date( )

    c) to-number( )

- Aggregate functions are –

    a) sum( )

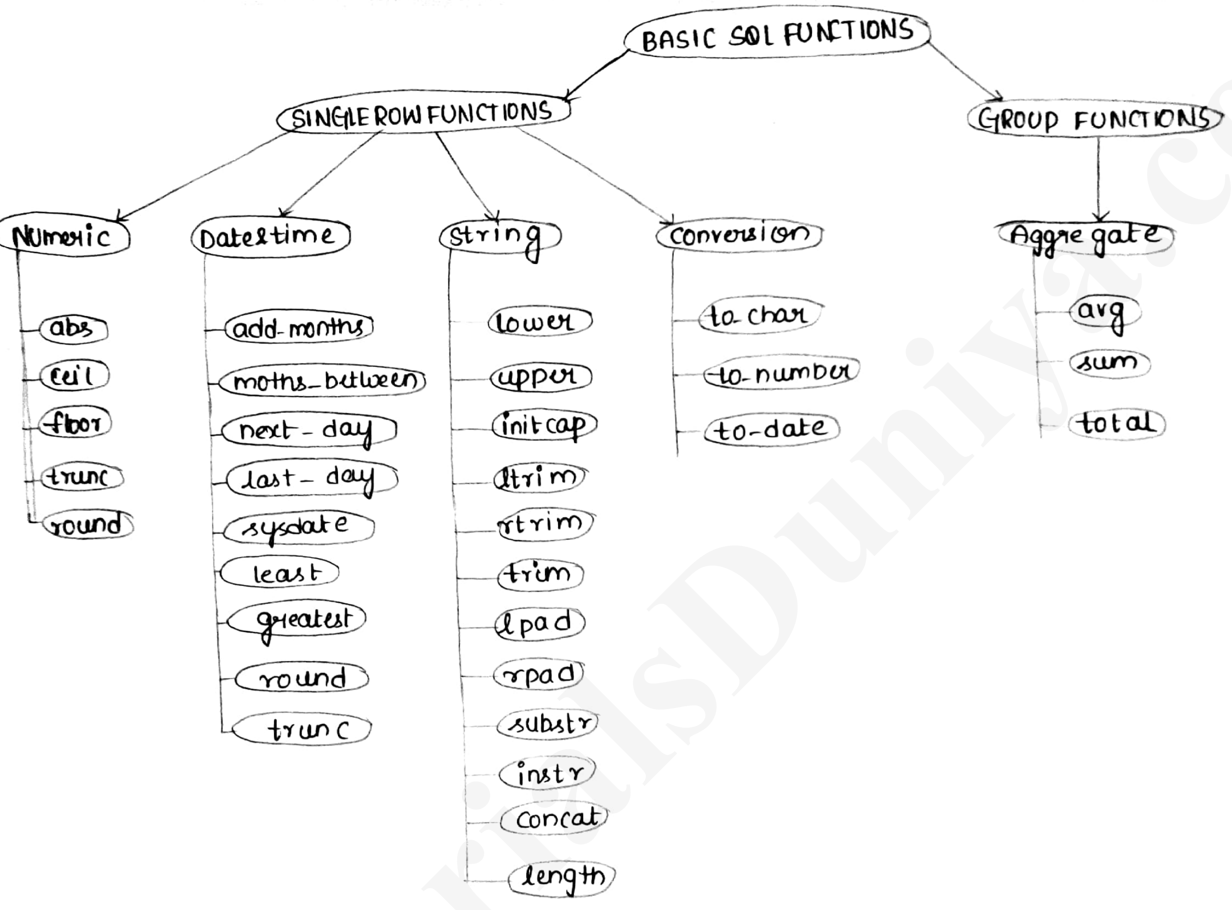
    b) avg( )

    c) total( ) etc.

- <u>Dual table</u>:–

This is a single row and single column dummy table provided by DB. This is used to perform mathematical calculations without using table.

```
Select * from dual;
DUMMY
------
X
```


BASIC SQL FUNCTIONS
SINGLE ROW FUNCTIONS
GROUP FUNCTIONS
Numeric
Date&time
String
Conversion
Aggregate
abs
ceil
floor
trunc
round
add-months
moths_between
next-day
last-day
sysdate
least
greatest
round
trunc
lower
upper
initcap
ltrim
rtrim
trim
lpad
rpad
substr
instr
concat
length
to-char
to-number
to-date
avg
sum
total

Explain in detail about cluster and Multilevel indexes.

Clustering indexes :-

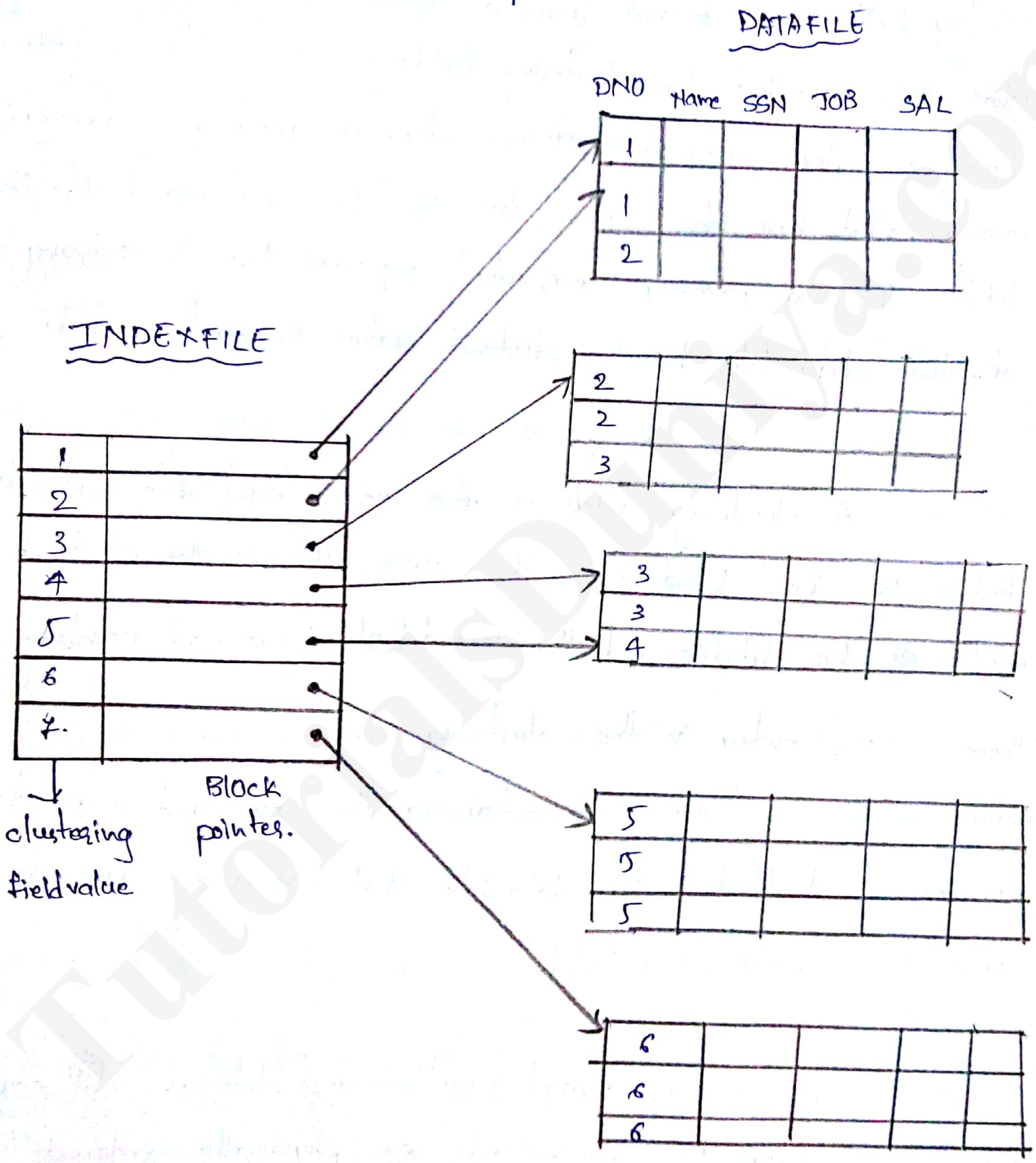
If records of a file are physically ordered on a nonkey field - which doesnot have a distinct value for each record: that field is called the clustering field. We can create a different type of index, called a clustering index, to speed up retrieval of records that have the same value for the clustering field. This differs from a primary index, which requires that the ordering field of the data file have a distinct value for each record.

A clustering index is also an ordered file with two fields, the first field is of the same type as the clustering field. of the datafile. and the second field is a block pointer.

There is one entry in the clustering index for each distinct value of the clustering field, containing the value and a pointer to the first block in the data file that has a record with that value for its clustering field.

Notice that record insertion and deletion still cause problems, because the data records are physically ordered. To alleviate the problem of each insertion, it is common to reserve a whole block (or a cluster of contiguous blocks) for each value of the clustering field. all records with that value are placed in the block. This makes insertion and deletion relatively straight forward.

A clustering index is another example of a nondense index, because it has an entry for every distinct value of the indexing field rather than for every record in the file.

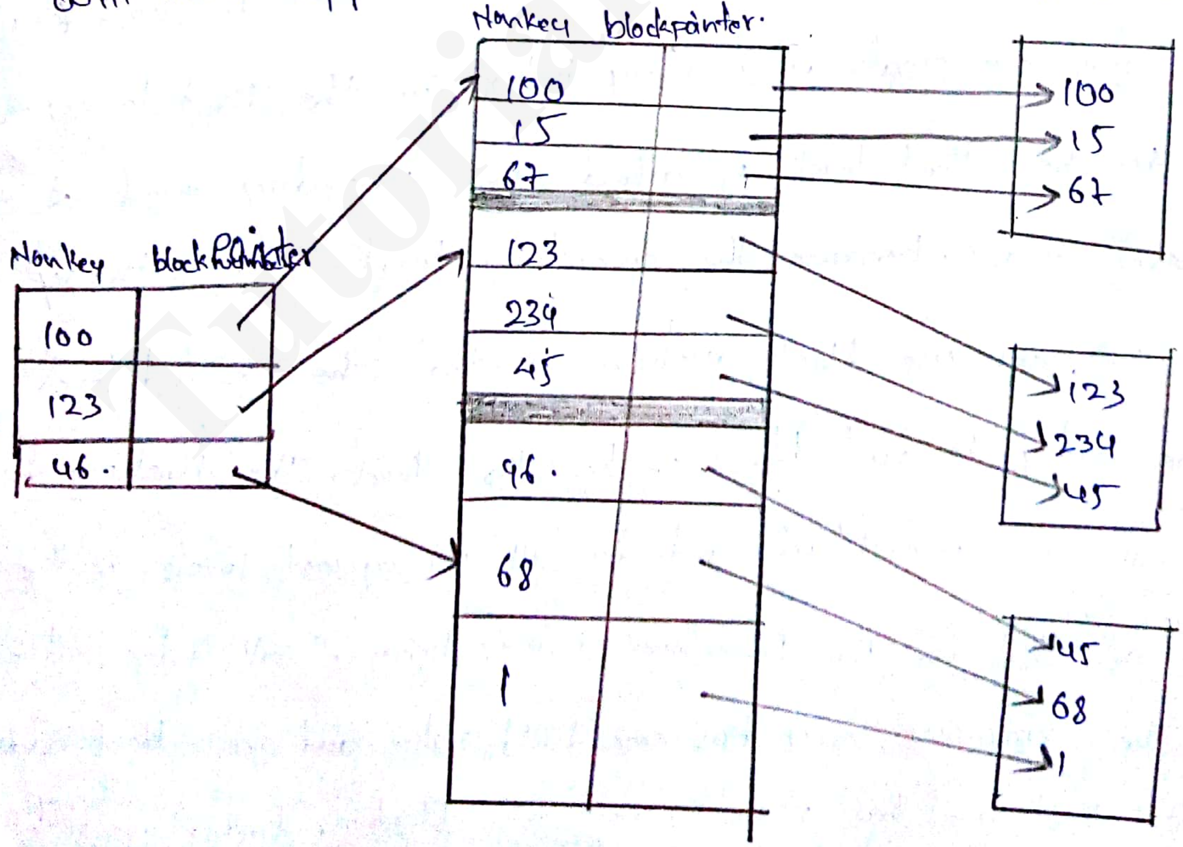


Multilevel indexes:-

The idea behind a multilevel index is to reduce the part of the index that we continue to search. by bfri, the blocking factor for the index, which is larger than 2. Hence, the search space is reduced much faster. The value bfri, is called the fan-out of the multilevel index, and we will refer to it by the symbol fo. Searching a multilevel index requires approximately ($\log_{fo} bi$) block. access which is a smaller number than for binary search if the fanout is larger than 2.

A multilevel index considers the index file, which we will now refer to as the first (or base) level of a multilevel index, as an ordered file with a distinct value for each K(i). Hence we can create a primary index for the first level; this index to the first level is called the secondary level of the multilevel index. Because the secondary level is a primary index, we can use block anchors so that the second level has one entry for each block of the first level. The blocking factor bfri of the second level - and for all subsequent levels - is the same as that for the first-level index, because all index entries are the same size; each has one field value and one block address. if the fist level has r1 entries and the blocking factor - which is also the fan-out - for the index is bfri = fo; then the first level needs $(r1/fo)$ blocks, which is therefore the number of entries r2 needed at the second level of the index.

We can repeat this process for the second level. The third level, which is a primary index for the second level. has an entry for each second-level, so the number of third-level entries is $r3 = (r2/fo)$. we can repeat this process untill all the entries of some index level t fit in a single block. this block at the t th level is called the top index level. Each level reduces the number of entries at the previous level by a factor of fo - the index fan-out - so we can use the formula $1 \leq (r1/((fo)^t))$ to calculate t. Hence a multilevel index with r1 first-level entries will have approximately t levels. where $t = (\log_{fo}(r1))$.

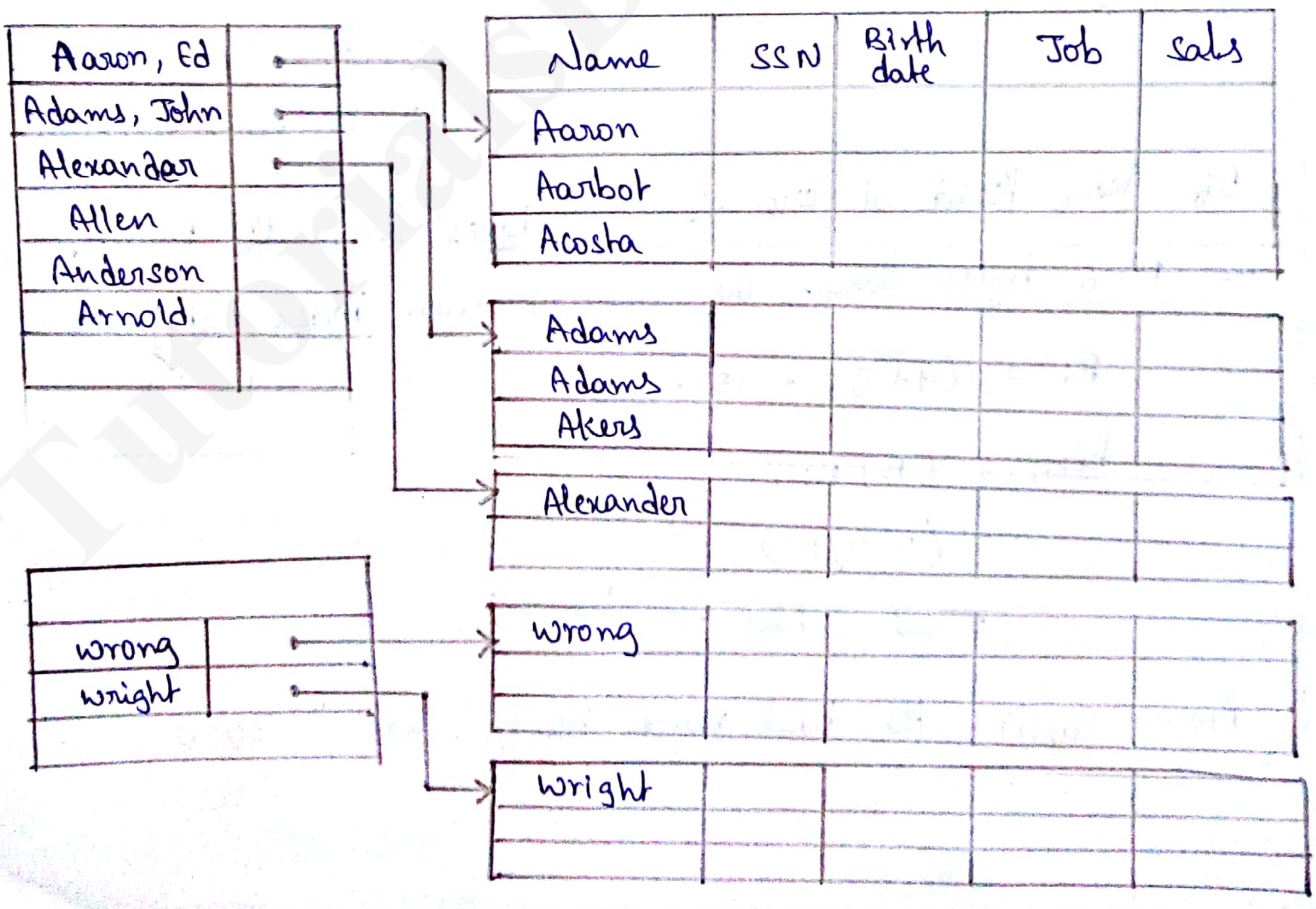


By considering an Example, show how to reduce access with primary index.

- Primary index :- It is an ordered file whose records are of fixed length with 2 fields. The first field is of the same datatype as ordering key field called primary key of datafile. Second field is a pointer to disk block. we will refer these 2 field values of index Entry i as <k(i),P(i)>

To create a primary index on the ordered file is shown in fig. we use NAME field as primary key.

< k(1) = (Aaron, Ed), P(1) = address of block1 >
< k(2) = (Adams, John), P(2) = address of block2 >
< k(3) = (Alexander, Ed), P(3) = address of block3 >



Ex:- Illustrate the saving in block accesses that is attainable when a primary index is used to search for a record.

→ Suppose that we have an ordered file that $r = 30,000$ records stored on a disk with blocksize $B = 1024$ bytes.

File records are of fixed size & are with record length $R = 100$ bytes.

The blocking factor for file would be $bfr = (B/R)$

$= (1024/100)$

$= 10$ records per block.

→ No. of blocks needed for the file is $b = (r/bfr)$

$= (30,000/10)$

$= 3000$ blocks $\longrightarrow$ ①

→ Binary search the data file would be $\log_2 b$.

$= \log_2 3000$

$= 12$ block accesses $\longrightarrow$ ②

If key field of file is $V = 9$ bytes long the block pointer is $P = 6$ bytes long. The size of each index entry is

$R_i = (9+6) = 15$ bytes.

$bfr_i = (B/R_i)$

$= (1024/15)$

$= 68$ Entries per block

From Eq ①, the total no. of blocks are "3000"

The no. of index blocks is $b_i = (r_i / bfr_i)$

$= (3000/68)$

$= 45$ blocks $\longrightarrow \textcircled{3}$

$(\log_2 b_i) = (\log_2 45)$

$= 6$ block access $\longrightarrow \textcircled{4}$

we need one additional block to data file for total of $6+1=7$ block access.

→ From eq ①, we have 3000 blocks, it access 12 blocks.

→ But in eq ③, we access the 45 blocks and time access only 6 block. So, it reduces access time. A major problem with a primary index. as with any ordered file is insertion & deletion of records.