A-Mini Project Report On

MUSIC DATABASE

carried out as part of the course DBMS

Submitted by

Suhas HS(12IT85)

S.Ashish Bharadwaj12IT63

Aneesh(12IT05)

Jeevraj Rao(12IT27)

Keerthi prasad(12IT34)

NATIONAL INSTITUITE OF TECHNOLOGY SURATHKAL

SUBMITTED TO

Mr. Ram Shastry

Synopsis

1. **TITLE OF THE PROJECT:**

   MUSIC DATABASE

2. **OBJECTIVE OF THE PROJECT:**

   This main objective of the project is to help user so that he can listen and buy songs. Based on his above two actions a user has a custom list of song recommendations that the application generates on analysis. The user is also allowed to view global trending tracks

.

3. **PROJECT CATEGORY:**

   RDBMS

4. **LANGUAGE AND SOFTWARE TOOL USED**:

   Front End: PHP

   Operating System: Windows 8, LINUX

   Back End: MYSQL

5. STRUCTURE OF THE PROJECT:

5.1. Proposed System:-

In the proposed system, the administrator can insert, delete and update songs as per the requirements. Songs are sorted can be easily maintained in this system. This project will help to recommend music which best suits the users.

5.2. Module Description

5.2.1. Administration Module:-

This is the main module in the proposed project. The administrator can read and write information about any songs such as Artist, Album description. They can also update, create and delete the songs as requirement and implementation plan.

5.2.2. Songs Module:-

This module contains the song records. The sub modules are

5.2.2.1. Artist:-

It contains details of Artist such as Name, No. Of Songs, No. Of Albums.

5.2.2.2. Album:-

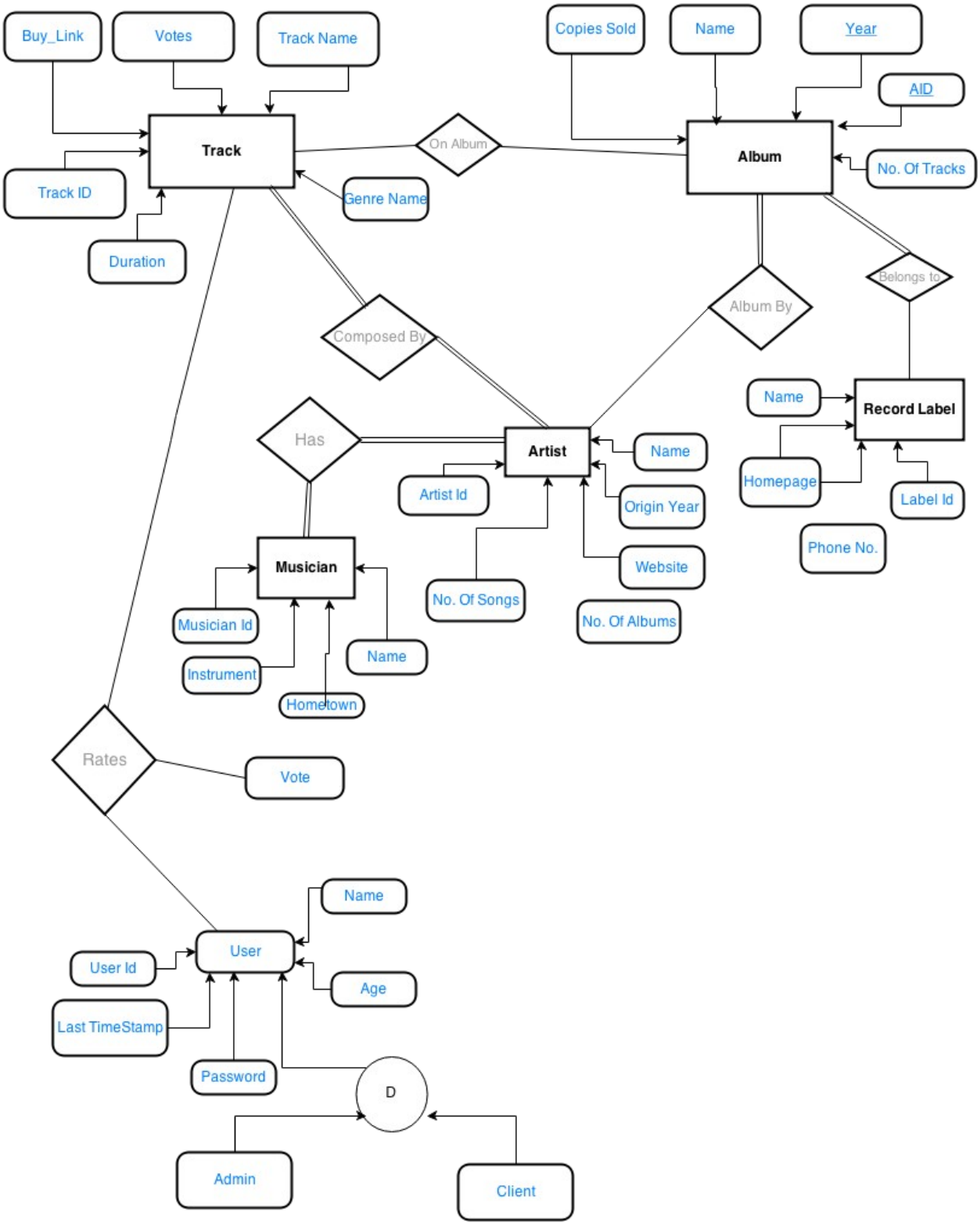
The Album module contains details of the Album.

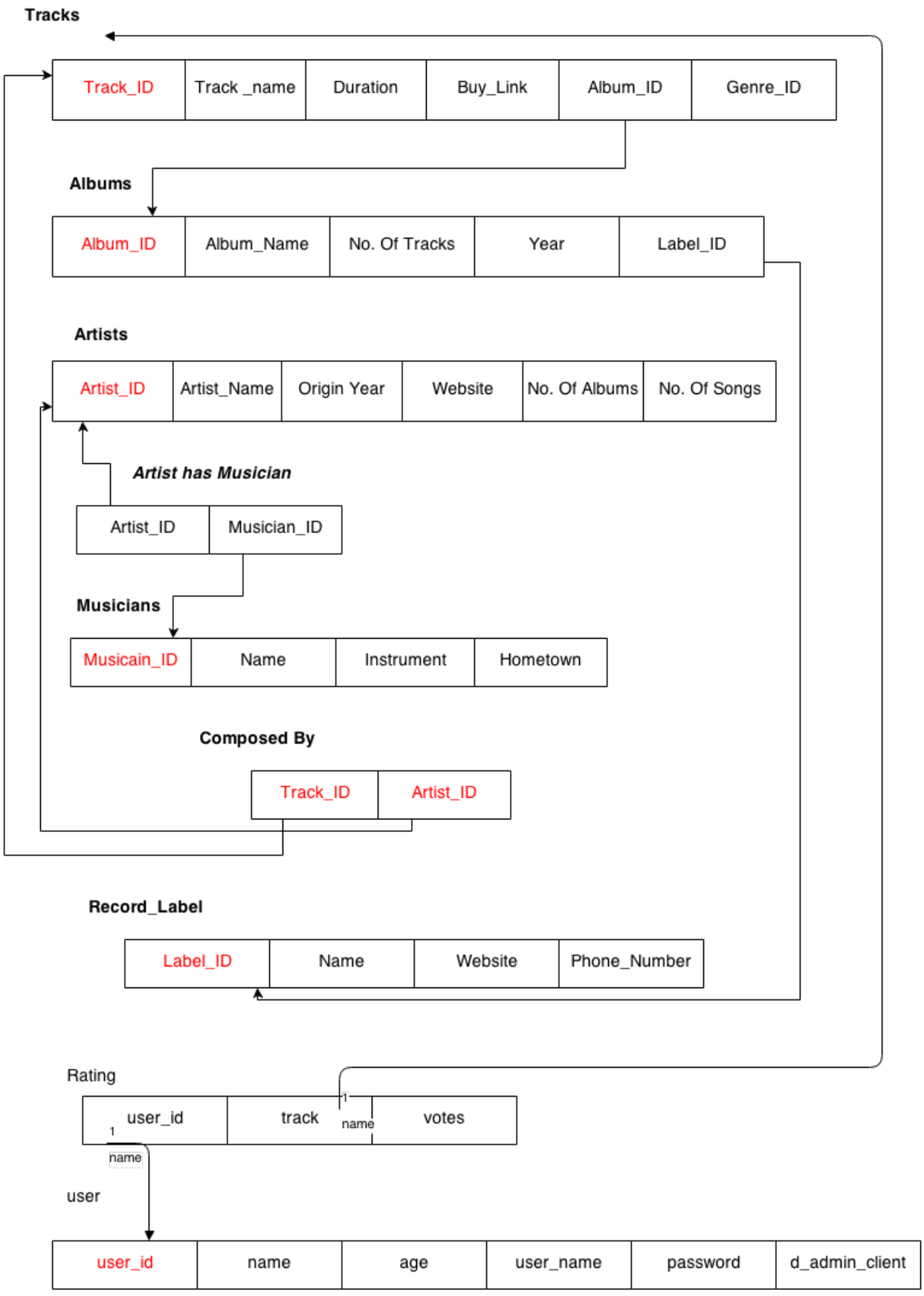
6. FUTURE SCOPE OF THE PROJECT

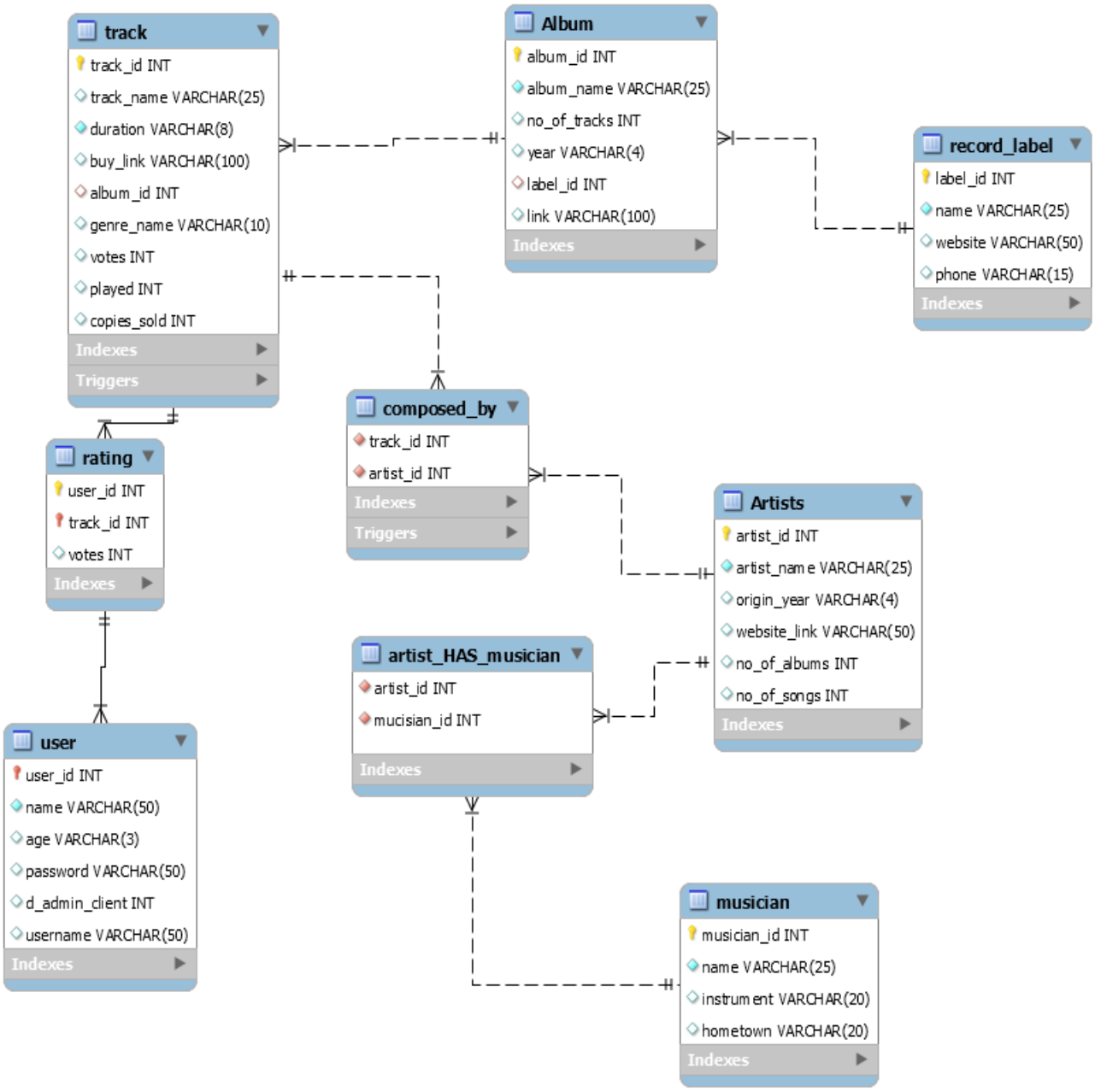
This project will help the users to find the appropriate music more quickly and efficiently. This software is developed in order to computerize the activities which take more time, if done manually.

Songs are sorted based on genre, artists and albums. It also gives a list of songs which are high rated to make things faster and can get information quickly. If we want any information about songs, we can access it quickly.

LOGICAL DESIGN




Tracks
Track_ID
Track _name
Duration
Buy_Link
Album_ID
Genre_ID
Albums
Album_ID
Album_Name
No. Of Tracks
Year
Label_ID
Artists
Artist_ID
Artist_Name
Origin Year
Website
No. Of Albums
No. Of Songs
Artist has Musician
Artist_ID
Musician_ID
Musicians
Musicain_ID
Name
Instrument
Hometown
Composed By
Track_ID
Artist_ID
Record_Label
Label_ID
Name
Website
Phone_Number
Rating
user_id
track
votes
1
name
1
name
user
user_id
name
age
user_name
password
d_admin_client


track
track_id INT
track_name VARCHAR(25)
duration VARCHAR(8)
buy_link VARCHAR(100)
album_id INT
genre_name VARCHAR(10)
votes INT
played INT
copies_sold INT
Indexes
Triggers
Album
album_id INT
album_name VARCHAR(25)
no_of_tracks INT
year VARCHAR(4)
label_id INT
link VARCHAR(100)
Indexes
record_label
label_id INT
name VARCHAR(25)
website VARCHAR(50)
phone VARCHAR(15)
Indexes
composed_by
track_id INT
artist_id INT
Indexes
Triggers
rating
user_id INT
track_id INT
votes INT
Indexes
Artists
artist_id INT
artist_name VARCHAR(25)
origin_year VARCHAR(4)
website_link VARCHAR(50)
no_of_albums INT
no_of_songs INT
Indexes
artist_HAS_musician
artist_id INT
mucisian_id INT
Indexes
user
user_id INT
name VARCHAR(50)
age VARCHAR(3)
password VARCHAR(50)
d_admin_client INT
username VARCHAR(50)
Indexes
musician
musician_id INT
name VARCHAR(25)
instrument VARCHAR(20)
hometown VARCHAR(20)
Indexes

Physical Design

Track Table

Total = 198 Bytes

Computation of the Blocking Factor for the table using the standard block size of 512 bytes.

Blocking factor = 512/198 = 2 Records per block.

Number of file blocks = 100000/2 = 50000 blocks

Storage Requirements:

Number of bytes wasted per block= 2 x 198=396

Percentage wastage = (512-2*198)/512 x 100 = (116/512) x 100 = 22.65%

Hence **unspanned** file type is used

Each record effectively occupies (512/2) = 256 bytes

Overhead of 58 bytes per record.

Access Method:

Binary search on this data file = log2(50000) = 15.6
Primary Indexing
Length of each index record = 11 bytes (Primary Key) + 5 bytes (Block Pointer) = 16 bytes
Index blocking factor = 512/16 = 32
Total number of entries in index file = Number of blocks = 50000

Therefore, number of blocks required 50000/32 = 1562 blocks
To perform binary search on index file log2(1562) = 10 block accesses
To search for a record we need 10+1 = 11 block accesses (to access the data file block containing record)

Secondary Indexing
Clustering index on dept_ID (after ordering)
Size of Album_ID = 11 bytes
Size of index entry = 11 + 5 = 16 bytes
Blocking Factor = 512/16 = 32 index records per block

Second level Indexing
Number of indexes = 50000/32 = 1562 blocks
Blocking factor = 512/16 = 32 records per block
Hence, number of blocks required at second level = 1
Wastage per block = 32 bytes (second level)
= 116 bytes (primary level)
Total wastage = 32 + 116 = 352 bytes

B - Trees

P x 5 + (P-1) x 11 + (P-1) x 5 < 512

21P – 16 < 512

21P < 528

P < 25.14

P = 25

Level	Number of Nodes	Number of entries	Number of pointers
Root	1	24	25
Level 1	25	600	625
Level 2	625	15000	15625

Number of blocks needed at level n = 100000/24 = 4166 blocks

Number of blocks needed at level n-1 = 4166/24 = 173 blocks

Total number of blocks = 4166 + 173 = 4339 blocks

B+ Trees

P x 5 + (P-1) x 11 <= 512

16P <= 523

P <= 32.68

P = 32

Level	Number of Nodes	Number of entries	Number of pointers
Root	1	31	32
Level 1	32	992	1024
Level 2	1024	31744	32764

Number of blocks needed at level n = 100000/31 = 3225 blocks

Number of blocks needed at level n-1 = 3225/31 = 104 blocks

Number of blocks needed at level n-2 = 104/31 = 3 blocks

Total number of blocks = 3225 + 104 + 3 = 3332 blocks

Album

Assumption- An album contains 10 tracks.

Total=162 Bytes.

Computation of the Blocking Factor for the table using the standard block size of 512 bytes.

Blocking factor = 512/162 = 3 Records per block.

Number of file blocks = 10000/3 = 3334 blocks

Storage Requirements:

Percentage wastage = (512-162 x 3)/512*100 = 5.078%

Hence **unspanned** file type is used

Each record effectively occupies 170.67 bytes of storage

Overhead of 8.67 bytes per record.

B - Trees

P x 5 + (P-1) x 11 + (P-1) x 5 < 512

21P – 16 < 512

21P < 528

P < 25.14

P = 25

Level	Number of Nodes	Number of entries	Number of pointers
Root	1	24	25

Level 1	25	600	625

Number of blocks needed at level n = 10000/24 = 416 blocks

Number of blocks needed at level n-1 = 416/24 = 17 blocks

Total number of blocks = 416+17 = 433 blocks

B+ Trees

Internal Nodes

P x 5 + (P-1) x 11 <= 512

16P <= 523

P <= 32.68

P = 32

Leaf Nodes

P x 5 + P x 11 + 5 < 512

P < 31.68

P =31

Level	Number of Nodes	Number of entries	Number of pointers
Root	1	31	32
Level 1	32	992	1024
Level 2	1024	31744	32768

Number of blocks needed at level n = 10000/31 = 322 blocks

Number of blocks needed at level n-1 = 322/31 = 10 blocks

Total number of blocks = 322 + 10 = 332 blocks

Record Label

Assumption-A record label has 20 albums.

Total = 101 Bytes

Computation of the Blocking Factor for the table using the standard block size of 512 bytes.

Blocking factor = 512/101 = 5 Records per block.

Number of file blocks = 500/ 5= 100 blocks

Storage Requirements:

Unspanned:

Percentage wastage = (512-101x 5)/512*100 = 1.36%

Hence **unspanned** file type is used.

Each record effectively occupies 102.4 bytes of storage

Overhead of 1.4 bytes per record.

Access Method:

Unspanned: = 7 block access

If we use primary indexing

a. Length of each entry in index file = 11 + 5 = 16 bytes
b. Index blocking factor = 512/16 = 32 record/block

Total number of entries in index file = number of blocks = 100

Therefore, number of blocks needed for index file = 100/32 = 3 blocks

To perform binary search $\log_2 3$ = 2 block accesses

Multilevel Indexes
First level

c. Number of Indexes = 100
d. Number of blocks needed = 100/32 = 3 blocks

Second level

a. Number if blocks needed = 3/32 = 1 block

Total blocks= 3 + 1 = 4 blocks

wastage = (4 x 512) – (3 x 16 + 100 x 16)

= 400

B- Trees

P x 5 + (P-1) x 11 + (P-1) x 5 < 512

21P – 16 < 512

21P < 528

P < 25.14

P = 25

Level	Number of Nodes	Number of entries	Number of pointers
Root	1	24	25

Number of blocks needed at level n = 500/24= 20 blocks

Number of blocks = 20 blocks

B+ Trees

Internal Nodes

P x 5 + (P-1) x 11 <= 512

16P <= 523

P <= 32.68

P = 32

Leaf Nodes

P x 5 + P x 11 + 5 < 512

P < 31.68

P =31

Level	Number of Nodes	Number of entries	Number of pointers
Root	1	31	32

Number of blocks needed at level n = 500/31= 16 blocks

Number of blocks = 16 blocks

User

Total = 175 Bytes

Computation of the Blocking Factor for the table using the standard block size of 512 bytes.

Blocking factor = 512/175 = 2 Records per block.

Number of file blocks = 10000/2 = 5000 blocks

Hence **unspanned** file type is used

Percentage wastage = (512-175 x 2)/512*100 = 31.64%

Hence **unspanned** file type is used

Each record effectively occupies 256 bytes of storage

Overhead of 81 bytes per record.

Access Method:

Binary search on this data file = $\log_2(5000)$ = 12.2 (approx.)
Primary Indexing
Number of index records =50
Index blocking factor = 512/16 = 32
Total number of entries in index file = Number of blocks = 5000

Therefore, number of blocks required 5000/32 = 156 blocks
To perform binary search on index file $\log_2(156) = 7.2$ block accesses

Artists

Total = 112 Bytes

Computation of the Blocking Factor for the table using the standard block size of 512 bytes.

Blocking factor = 512/112 = 4 Records per block.

Storage Requirements

Number of file blocks = 10000/4=2500 blocks

Hence **unspanned** file type is used.

Percentage wastage = (512-112 x 4)/512*100 = 12.5%

Hence **unspanned** file type is used

Each record effectively occupies 128 bytes of storage

Overhead of 16 bytes per record.

Access Method:

i) Binary search on this data file = log2(2500) = 11.2(approx.)
ii) Primary Indexing
Length of each index record = 11 bytes (Primary Key) + 5 bytes (Block Pointer) = 16 bytes
Index blocking factor = 512/16 = 32
Total number of entries in index file = Number of blocks = 2500
Therefore, number of blocks required 2500/32 = 78 blocks

To perform binary search on index file $\log_2(78) = 6$ block accesses

Musician

Total = 76 Bytes

Computation of the Blocking Factor for the table using the standard block size of 512 bytes.

Blocking factor = 512/76 = 6 Records per block.

Storage Requirements

Number of file blocks = 100000/6=16667 blocks

Hence **unspanned** file type is used.

Percentage wastage = (512-76 x 6)/512*100 = 10.93%

Hence **unspanned** file type is used.

Each record effectively occupies 128 bytes of storage

Overhead of 16 bytes per record.

Rating

Total = 33 Bytes

Computation of the Blocking Factor for the table using the standard block size of 512 bytes.

Blocking factor = 512/33 = 15 Records per block.

Storage Requirements

Number of file blocks = 100000/15=6667 blocks

Hence **unspanned** file type is used.

Percentage wastage = (512-33 x 15)/512*100 = 3.32%

Hence **unspanned** file type is used

Each record effectively occupies 34.13 bytes of storage

Overhead of 1.13 bytes per record.

Calculating disk acces time for 1 block:

Size of a block: 512 bytes/sector

Seek time: 12 ms

Disk rotation speed: 5400 rpm

Data Transfer speed: 4 mbps

Overhead = 1ms

Queuing delay: 0 s

12 + 1/5400 (1/rpm) + 512/4000000 (1/b/s) +1 (ms)

12 + (1/90(1/rps) * 1000) + 512/4000 (1/b/ms) +1 (ms)

12 + 11 + 0.1 + 1

24.1 ms

Considering average rotational delay, we get

12 + (1/(90 * 2)(1/rps) * 1000) + 512/4000 (1/b/ms) +1 (ms)

12 + 5.5 + 0.1 + 1

18.6 ms

1. Consider the following Search query:

We have movies and ratings for each movies:

Select *from tracks order by desc;

So we can treat this as no of block accesses for above query;

Size of 1 movie entry=198 bytes.

Blocking factor = floor(512/198)=2 =>2 records per block.

So no. of blocks required=ceiling(100000/2)=50000 blocks.

Case 1:

No ordering based on any field.

Linear search will take O(n/2) block accesses,

So approximately 50000/2 =25000 block accesses.

So total time taken for query = 25000 * 18.6 = 465000ms

Case 2:

Ordering based on votes,

So binary search would require ceiling(log(50000) to the base 2)+1=16+1=17 block accesses.

So total time taken for query = 17 * 18.6 = 316.2ms

Case 3:

Index based on track_id;,

Size of user_id= 11 bytes, block ptr size = 5 bytes.

So size of index entry=11+5=16 bytes.

So no. Of index entries per block=floor(512/16)=32 entries.

Total no. Of index entries = total no. Of blocks in data file = 50000

So index blocking factor(no. Of blocks required to store index entries,bfri=

ceiling(50000/32)=1562 entries

So now search will take ceiling(log(1562) to base 2)+1=11+1=12 block accesses

So total time taken for query = 12 * 18.6 = 223.2ms

QUERY ANALYSIS

Query1

Select t.track_name,a.album_name from (select track_name,album_id from track where track_id in (select track_id from composed_by where artist_id in (select artist_id from Artists where artist_name='tool'))) as t inner join Album a on t.album_id=a.album_id order by t.track_name

Timing (as measured at client side):
Execution time: 0:00:0.00000000

Timing (as measured by the server):
Execution time: 0:00:0.00087541
Table lock wait time: 0:00:0.00100100

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 6
Rows examined: 36

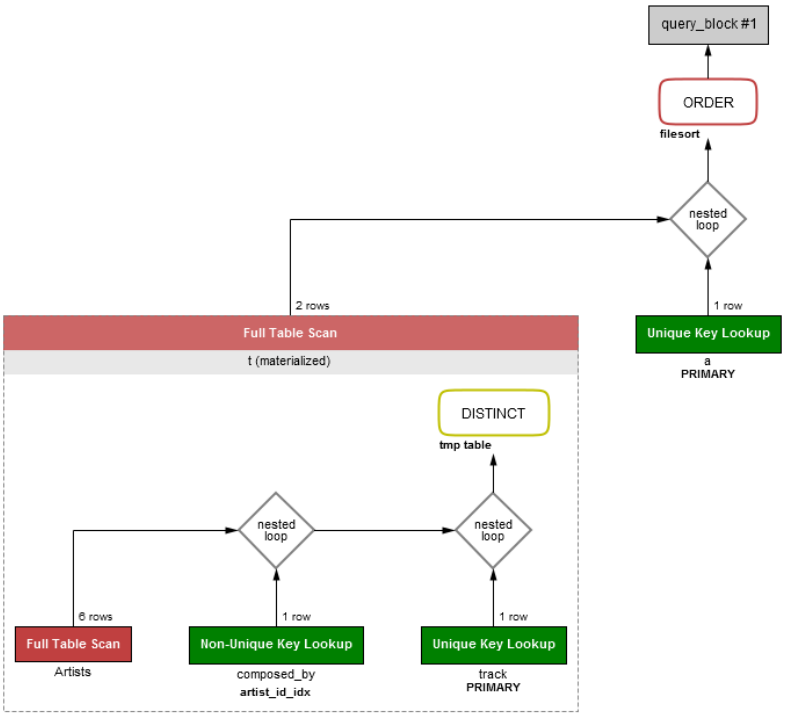
Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 2

Joins per Type:
Full table scans (Select_scan): 2
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 6
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 1

Index Usage:
No Index used

Other Info:
Event Id: 38
Thread Id: 25



QUERY 2

Select artist_name, website_link from Artists where artist_id in(

select artist_id from composed_by where track_id in(

select track_id from track where genre_name in(

select genre_name from track where track_id in(

```
select track_id from rating where user_id = 1 and votes =
(select max(votes) from rating where user_id=1)
))
AND track_id not in (
select track_id from rating where user_id = 1 and votes =
(select max(votes) from rating where user_id=1)
)));
```

!

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.01600000

Timing (as measured by the server):
Execution time: 0:00:0.00115595
Table lock wait time: 0:00:0.00100100

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 1
Rows examined: 92

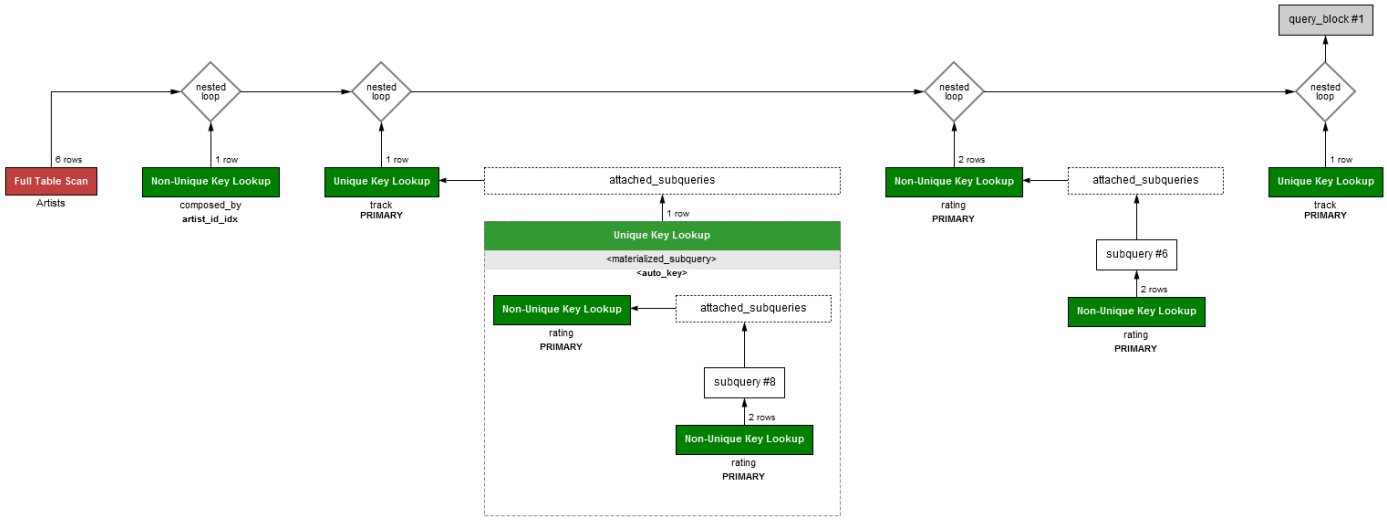
Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 1

Joins per Type:
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 0

Index Usage:
No Index used

Other Info:
Event Id: 46
Thread Id: 25



QUERY 3

Select artist_name, website_link, count(artist_id) from artists

where artist_id in (

select artist_id from composed_by where track_id in(

select track_id from track where album_id in(

select album_id from album where label_id in(

select label_id from record_label where name = 'studio a'

))))

group by artist_id;

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.00000000

Timing (as measured by the server):
Execution time: 0:00:0.00079672
Table lock wait time: 0:00:0.00000000

Errors:
Had Errors: NO
Warnings: 0

Rows Processed:
Rows affected: 0
Rows sent to client: 0
Rows examined: 6

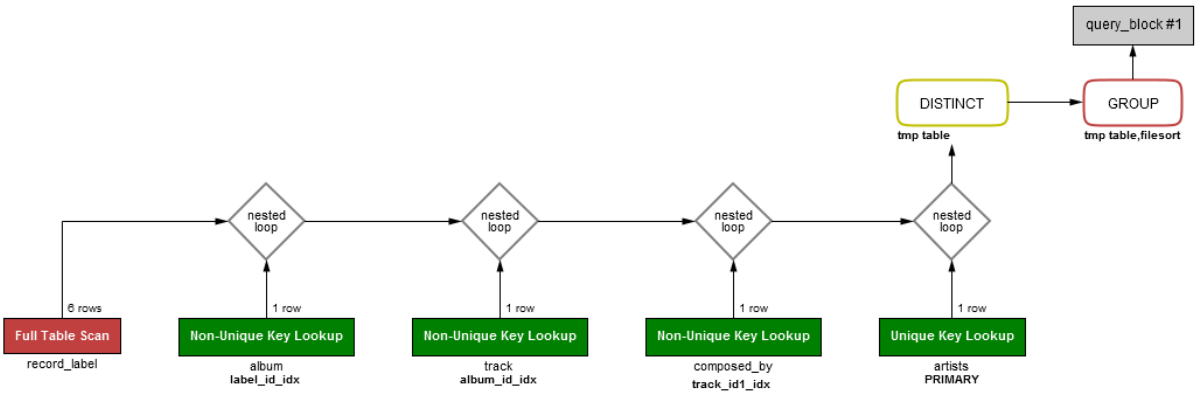
Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 2

Joins per Type:
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 0
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 1

Index Usage:
No Index used

Other Info:
Event Id: 48
Thread Id: 25



QUERY 4

Select avg(t.duration), a.album_name from Album a

inner join track t

on(t.album_id = a.album_id)

where a.album_name = 'lateralus'

group by t.album_id;

Query Statistics

Timing (as measured at client side):
Execution time: 0:00:0.00000000

Timing (as measured by the server):
Execution time: 0:00:0.00070734
Table lock wait time: 0:00:0.00000000

Errors:
Had Errors: NO
Warnings: 3

Rows Processed:
Rows affected: 0
Rows sent to client: 1
Rows examined: 14

Temporary Tables:
Temporary disk tables created: 0
Temporary tables created: 1

Joins per Type:
Full table scans (Select_scan): 1
Joins using table scans (Select_full_join): 0
Joins using range search (Select_full_range_join): 0
Joins with range checks (Select_range_check): 0
Joins using range (Select_range): 0

Sorting:
Sorted rows (Sort_rows): 1
Sort merge passes (Sort_merge_passes): 0
Sorts with ranges (Sort_range): 0
Sorts with table scans (Sort_scan): 1

Index Usage:
No Index used

Other Info:
Event Id: 50
Thread Id: 25