## Question 1.

Tables and indexes:

|  |  |  |  |
| --- | --- | --- | --- |
| **Table name** | **Nb of records** | **Nb of pages in heap file (approximative is fine)** | **Index (type, field/column, alternative 1/2/3, clustered / unclustered)** |
| ARTIST | 1000000 | 6396 | |  | | --- | | NORMAL,LASTNAME,alternative 2,unclustered | | NORMAL,IDARTIST,alternative 2,unclustered | |
| MOVIE1 | 449833 | 13821 | NORMAL,IDMOVIE,alternative 2,unclustered |
| MOVIE2 | 449833 | 13821 | |  | | --- | | NORMAL,IDMOVIE,alternative 2,unclustered | | NORMAL,YEAR,alternative 2,unclustered | | NORMAL,CODECOUNTRY,alternative 2,unclustered | |
| MOVIE3 | 449833 | 13821 | |  | | --- | | NORMAL,IDMOVIE,alternative 2,unclustered | | BITMAP,YEAR,alternative 2,unclustered | | BITMAP,CODECOUNTRY,alternative 2,unclustered | |
| MOVIE4 | 449833 | null | |  | | --- | | IOT-TOP,IDMOVIE,alternative1, clustered | | NORMAL,CODECOUNTRY,alternative 1, clustered | |
| MOVIERATER | 500000 | 6045 | NORMAL,EMAIL,alternative 2,unclustered |
| RATING | 629461 | 5224 | NORMAL,IDMOVIE EMAIL,alternative 2,unclustered |
| COUNTRY | 180 | 2 | NORMAL,CODE,alternative 2,unclustered |
| ROLE | 539981 | 2285 | NORMAL,IDMOVIE IDACTOR,alternative 2,unclustered |

## Question 2. Fill the following tables:

|  |  |  |  |
| --- | --- | --- | --- |
| **Attribute name** | **Cardinality (number of distinct values)** | **Minimal value** | **Maximal value** |
| MOVIE1.IDMOVIE | 449833 | 0 | 500000 |
| MOVIE1.YEAR | 20 | 1991 | 2010 |
| MOVIE1.CODECOUNTRY | 180 | aaau | aahr |

**Question 3.1 On MOVIE1 and MOVIE4 :**

SELECT TITLE FROM MOVIEi WHERE IDMOVIE=50273 ;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE** | **MODE** | **OPERATORS** | **BUFFER READS** | **DISK READS** |
| *MOVIE1* | *CHOOSE* | INDEX UNIQUE SCAN + BY INDEX ROWID | 4 | 4 |
| *MOVIE4* | *CHOOSE* | INDEX UNIQUE SCAN | 3 | 3 |

Your explanation of the SQL query (in plain English) and of the execution plan:

This query gives the title of movie 50273.

In the form above, in CHOOSE mode we use the index, since it is there. Since MOVIE1 use index of Alternative 2, it need to read the index pages and read the responding heap pages. But MOVIE4 use index of Alternative 1, it only need to read the index pages.

Question 3.2 On MOVIE2 and MOVIE4 :

SELECT COUNT(\*) FROM MOVIEi WHERE IDMOVIE BETWEEN 55273 AND 60000 ;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE** | **MODE** | **OPERATORS** | **BUFFER READS** | **DISK READS** |
| *MOVIE2* | *CHOOSE* | INDEX RANGE SCAN | 12 | 12 |
| *MOVIE4* | *CHOOSE* | INDEX RANGE SCAN | 168 | 168 |

Your explanation of the SQL query (in plain English) and of the execution plan:

This query gives the number of movies between 55273 and 60000,which is 4215.

In the form above, in CHOOSE mode we use the index, since range conditions are sensitive to clustering. MOVIE4 has more leaf blocks than MOVIE2, so index range scan in MOVIE4 need to read more pages than in MOVIE2.

Question 3.3 On MOVIE2, with RULE, CHOOSE :

SELECT TITLE FROM MOVIEi WHERE YEAR=1999 ;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE** | **MODE** | **OPERATORS** | **BUFFER READS** | **DISK READS** |
| *MOVIE2* | *RULE* | INDEX RANGE SCAN + BY INDEX ROWID | 11899 | 11247 |
| *MOVIE2* | *CHOOSE* | TABLE ACCESS  FULL | 14259 | 13822 |

Your explanation of the SQL query (in plain English) and of the execution plan:

This query gives the title of movies,which year is 1999.

In the form above, in RULE mode we use the index, since it is there.

However, in CHOOSE mode the optimizer understands that using the index would be too

expensive, thus a full scan is done instead.

Question 3.4 On MOVIE2, MOVIE3, MOVIE4 :

SELECT COUNT(\*) FROM MOVIEi WHERE YEAR=1999 ;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE** | **MODE** | **OPERATORS** | **BUFFER READS** | **DISK READS** |
| *MOVIE2* | *CHOOSE* | INDEX RANGE SCAN | 55 | 55 |
| *MOVIE3* | *CHOOSE* | BITMAP INDEX SINGLE VALUE | 8 | 8 |
| *MOVIE4* | *CHOOSE* | INDEX FAST FULL SCAN | 17743 | 17723 |

Your explanation of the SQL query (in plain English) and of the execution plan:

This query gives the number of movies, which year is 1999.

In the form above, in CHOOSE mode we use the index, since MOVIE2,MOVIE3 have index on YEAR, since we only need to count the number, finding the corresponding page in heap file is unnecessary.Each bit in the bitmap corresponds to a rowId, so MOVIE3 require the least pages to read, MOVIE2 is the second. MOVIE4 has no index on YEAR,so it full scans the table,which need to read a lot of pages.

Question 3.5 On MOVIE2, MOVIE3, with RULE, CHOOSE : SELECT TITLE FROM MOVIEi WHERE CODECOUNTRY='aaej' ;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE** | **MODE** | **OPERATORS** | **BUFFER READS** | **DISK READS** |
| *MOVIE2* | *RULE* | INDEX RANGE SCAN + BY INDEX ROWID | 2377 | 2323 |
| *MOVIE2* | *CHOOSE* | INDEX RANGE SCAN + BY INDEX ROWID | 2377 | 2271 |
| *MOVIE3* | *RULE* | TABLE ACCESS FULL | 13875 | 13822 |
| *MOVIE3* | *CHOOSE* | BITMAP INDEX SINGLE VALUE + BY INDEX ROWID | 2322 | 2317 |

Your explanation of the SQL query (in plain English) and of the execution plan:

This query gives the title of movies, which CODECOUNTRY is ‘aaej’.

In the MOVIE2, in RULE and CHOOSE mode we use the index, since it has index on CODECOUNTRY. In the MOVIE3, in RULE mode we use full scan since it is easy and cheap. In CHOOSE mode, the optimizer find it has bitmap index on CODECOUNTRY,thus a bitmap index is done instead.

# SECTION 4 – MULTI-TABLE (JOIN) QUERIES

Question 4.1 On MOVIE2 and MOVIE3, with RULE, CHOOSE :

SELECT TITLE, LASTNAME FROM MOVIEi M, ARTIST A WHERE CODECOUNTRY='aaej' AND M.IDMES=A.IDARTIST ;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE** | **MODE** | **OPERATORS** | **BUFFER READS** | **DISK READS** |
| *MOVIE2* | *CHOOSE* | INDEX RANGE SCAN + BY INDEX ROWID | 2323 | 2323 |
| *MOVIE2* | *RULE* | INDEX RANGE SCAN + BY INDEX ROWID | 2377 | 2323 |
| *MOVIE3* | *CHOOSE* | BITMAP INDEX SINGLE VALUE + BY INDEX ROWID | 2317 | 2317 |
| *MOVIE3* | *RULE* | TABLE ACCESS FULL | 13875 | 13822 |

Your explanation of the SQL query (in plain English) and of the execution plan:

This query gives the title of the movie,which have been directed by an actor and CODECOUNTRY equals ‘aaej’ , and the lastname of the corresponding director.

In the MOVIE2, in RULE and CHOOSE mode we use the index, since it has index on CODECOUNTRY. After selecting the codecountry, MOVIE2 uses HASH JOIN to join ARTIST.

In the MOVIE3, in RULE mode we use full scan since it is easy and cheap. In CHOOSE mode, the optimizer find it has bitmap index on CODECOUNTRY,thus a bitmap index is done instead.After selecting the codecountry, MOVIE3 uses HASH JOIN to join ARTIST.

Question 4.2 On MOVIE2 : SELECT lastname, title

FROM MOVIEi M, MOVIERATER MR, RATING R

WHERE R.email='aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa' AND MR.email= R.email AND M.IDMOVIE=R.IDMOVIE;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE** | **MODE** | **OPERATORS** | **BUFFER READS** | **DISK READS** |
| *MOVIE2* | *CHOOSE* | INDEX UNIQUE SCAN + BY INDEX ROWID | 3 | 3 |

Your explanation of the SQL query (in plain English) and of the execution plan:

This query gives the title of movies and lastname of the movie rater, where the moive’s movie rater’s email is 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa'. The result is ‘nfs knvi’ ‘owzd’.

In CHOOSE mode we use the index. After joining the MOVIERATER and RATING through NESTED LOOPS , we can get the IDMOVIE. Since MOVIE2 has index on IDMOVIE, we use index unique scan to read the pages when join through NESTED LOOPS.

Question 4.3 On MOVIE1 : SELECT lastname, title

FROM MOVIEi M, MOVIERATER MR, RATING R

WHERE MR.email= R.email AND M.IDMOVIE=R.IDMOVIE AND R.IDMOVIE= 367856;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE** | **MODE** | **OPERATORS** | **BUFFER READS** | **DISK READS** |
| *MOVIE1* | *CHOOSE* | INDEX UNIQUE SCAN + BY INDEX ROWID | 4 | 4 |

Your explanation of the SQL query (in plain English) and of the execution plan:

This query gives the title of movies and lastname of movie rater, where the movie id is 367856. The result is ‘nfs knvi’ ‘owzd’ and ‘pp’‘owzd’.

In CHOOSE mode we use the index. Since MOVIE1 has index on IDMOVIE, we first select IDMOVIE=367856 by index, then using NESTED LOOPS JOIN to join other tables.

Question 4.4 On MOVIE3 :

SELECT LASTNAME, COUNT(\*) FROM MOVIEi M, ARTIST A

WHERE YEAR=1999 AND M.IDMES=A.IDARTIST GROUP BY A.LASTNAME;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE** | **MODE** | **OPERATORS** | **BUFFER READS** | **DISK READS** |
| *MOVIE3* | *CHOOSE* | BITMAP INDEX SINGLE VALUE + BY INDEX ROWID | 11200 | 11200 |

Your explanation of the SQL query (in plain English) and of the execution plan:

This query gives lastname of the director of movies that were released in 1999 and were directed by an actor, and the number of films directed by actors and released in 1999 according to lastname.

In CHOOSE mode we use the bitmap index. Since MOVIE3 has bitmap index on YEAR, we first select YEAR=1999 by index, then using HASH JOIN to join other tables.

Question 4.5 On MOVIE3 : SELECT LASTNAME, COUNT(\*)

FROM ARTIST

WHERE IDARTIST IN (SELECT DISTINCT IDMES

FROM MOVIEi WHERE YEAR=1999)

GROUP BY LASTNAME;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE** | **MODE** | **OPERATORS** | **BUFFER READS** | **DISK READS** |
| MOVIE3 | CHOOSE | BITMAP INDEX SINGLE VALUE + BY INDEX ROWID | 11200 | 11200 |

Your explanation of the SQL query (in plain English) and of the execution plan:

This query gives lastname of the director of movies that were released in 1999 and were directed by an actor, and the number of actors who directed the movie and released in 1999 according to lastname.

In CHOOSE mode we use the bitmap index. Since MOVIE3 has bitmap index on YEAR, we first select YEAR=1999 by index, then using HASH JOIN to join other tables.

Question 4.6 On Movie3 :

SELECT COUNT(\*)

FROM MOVIEi M, ROLE R, ARTIST A

WHERE M.IDMOVIE=R.IDMOVIE AND R.IDACTOR=A.IDARTIST AND M.IDMES=A.IDARTIST AND CODECOUNTRY='aaej' AND YEAR=1999;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE** | **MODE** | **OPERATORS** | **BUFFER READS** | **DISK READS** |
| MOVIE3 | CHOOSE | BITMAP INDEX SINGLE VALUE + BY INDEX ROWID | 167 | 166 |

Your explanation of the SQL query (in plain English) and of the execution plan:

This query gives the number of movies which was released in 1999 with CODECOUNTRY ‘aaej’ and have been directed by one of their actors.

In CHOOSE mode we use the bitmap index. Since MOVIE3 has bitmap index on YEAR and CODECOUNTRY,we first select YEAR=1999 and CODECOUNTRY=’aaej’ by index, then using NESTED LOOPS JOIN to join other tables.

Question 4.7 On MOVIE2 and MOVIE4 :

SELECT TITLE

FROM MOVIEi M, RATING R WHERE M.IDMOVIE = R.IDMOVIE GROUP BY TITLE

HAVING AVG(RATE) > 15;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE** | **MODE** | **OPERATORS** | **BUFFER READS** | **DISK READS** |
| *MOVIE2* | *CHOOSE* | TABLE ACCESS FULL | 13824 | 13822 |
| *MOVIE4* | *CHOOSE* | INDEX FAST FULL SCAN | 17743 | 17723 |

Your explanation of the SQL query (in plain English) and of the execution plan:

This query gives title of movies, which average rate of same titles is higher than 15.

In CHOOSE mode,we use the FULL SCAN in MOVIE2 and MOVIE4, the optimizer understands that there is no filtering before and using the index would be too expensive, thus a full scan is done instead.

Question 4.8 On MOVIE2 and MOVIE4 :

SELECT TITLE

FROM MOVIEi M, (SELECT IDMOVIE, AVG(RATE) as AVG\_RATE

FROM RATING

GROUP BY IDMOVIE) N

WHERE M.IDMOVIE = N.IDMOVIE AND AVG\_RATE > 15;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **TABLE** | **MODE** | **OPERATORS** | **BUFFER READS** | **DISK READS** |
| *MOVIE2* | *CHOOSE* | TABLE ACCESS FULL | 14977 | 13822 |
| *MOVIE4* | *CHOOSE* | INDEX FAST FULL SCAN | 18874 | 17723 |

Your explanation of the SQL query (in plain English) and of the execution plan:

This query gives title of movies,which average rate of same IDMOVIE is higher than 15.

In CHOOSE mode,we use the FULL SCAN in MOVIE2 and MOVIE4, the optimizer understands that using the index would be too expensive, thus a full scan is done instead.