

Database Management System
Prof. Partha Pratim Das
Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur

Lecture - 30
Indexing and Hashing/5 : Index Design

Welcome to module 30 of Database Management Systems. We have been discussing about indexing and hashing and this is a concluding module on that.

(Refer Slide Time: 00:27)

PPD

Module Recap

- Static Hashing
- Dynamic Hashing
- Comparison of Ordered Indexing and Hashing
- Bitmap Indices

SWAYAM: NPTEL-NOC MOOC's Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 6th Edition 30.2 ©Silberschatz, Korth and Sudarshan

We have in the last module discussed about different hashing techniques static and dynamic and compare that in introduce bitmap indices.

(Refer Slide Time: 00:35)

PPD

Module Objectives

- To discuss how Indexes can be created in SQL
- To deliberate on good index designs in terms of *Guidelines for Indexing*

Database System Concepts - 8th Edition 30.3 ©Silberschatz, Korth and Sudarshan

Now, in this module we would specifically look into the use cases, we will check as to how indexes can be created in SQL first. Because we will have to use it you have already known the theory of various different indices and so, on so, how do you actually tell the system to index.

And the second is the important thing as to when should you index and on what. So, we talked about a few guidelines for indexing.

(Refer Slide Time: 01:05)

PPD

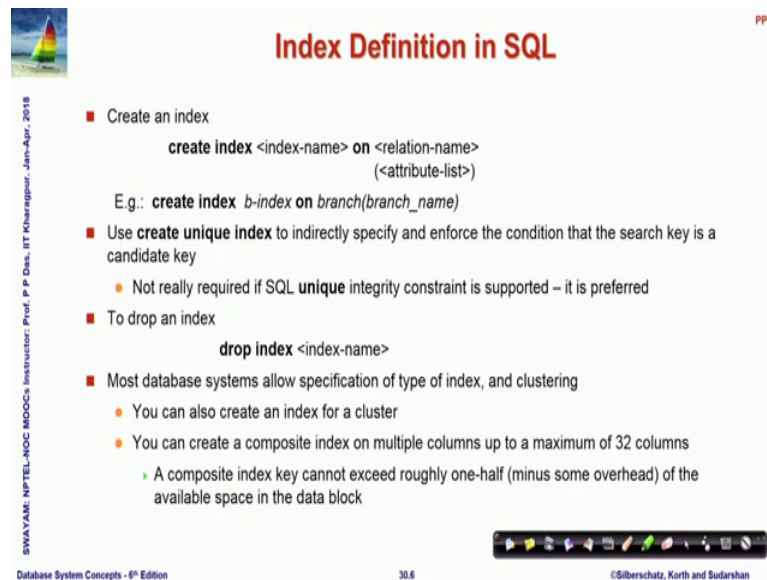
Module Outline

- Index Definition in SQL
- Guidelines for Indexing

Database System Concepts - 8th Edition 30.4 ©Silberschatz, Korth and Sudarshan

So, that is that is what we want to learn.

(Refer Slide Time: 01:09)



Index Definition in SQL

- Create an index
create index <index-name> **on** <relation-name>
(<attribute-list>)
E.g.: **create index** *b-index* **on** *branch*(*branch_name*)
- Use **create unique index** to indirectly specify and enforce the condition that the search key is a candidate key
 - Not really required if SQL **unique** integrity constraint is supported – it is preferred
- To drop an index
drop index <index-name>
- Most database systems allow specification of type of index, and clustering
 - You can also create an index for a cluster
 - You can create a composite index on multiple columns up to a maximum of 32 columns
 - A composite index key cannot exceed roughly one-half (minus some overhead) of the available space in the data block

SWAYAM: NPTEL-NOC MDOCS Instructor: Prof. P. P. Das, IIT Khargpur, Jan-April, 2018

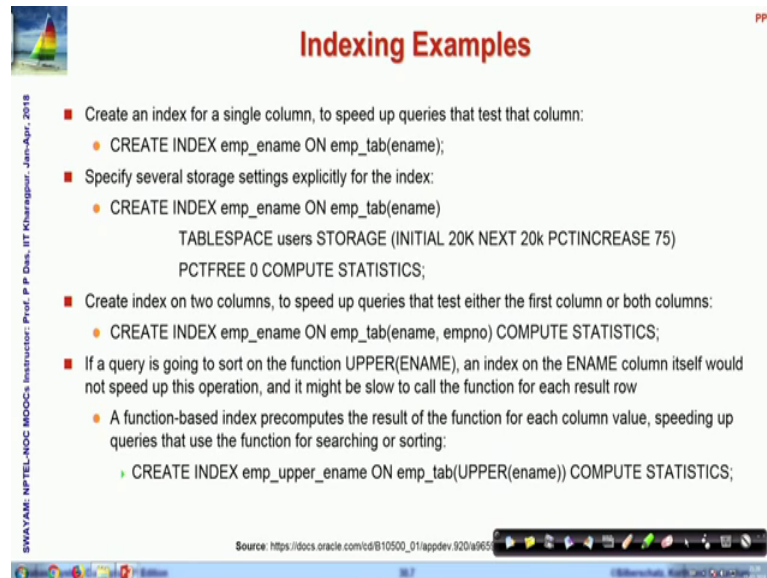
Database System Concepts - 8th Edition 30.6 ©Silberschatz, Korth and Sudarshan

So, index can be defined in SQL in very similar syntax as you create a table. So, you say create index put a name for that index and then you say on which relation are you indexing and put the list of attributes on which you are indexing. So, if branch can relation can be indexed on branch name and I may call that b index.

Now, there is a way to also express create unique index if you say create unique index and it will expect that the search key is a candidate key because I mean in the sense it all values of that will have to be distinct unique. Now, this used to be very common to do this kind of indexing earlier, but now it is more preferred that you can use unique integrity constraint in terms of the create table which we have already discussed. And that will ensure that you have that kind of a condition satisfied and you may not create unique index for that.

If you do not want an indexes actually do drop index and put the index name. So, most database system allow specification of the type of indexing and clustering that you want to do. So, you can create an index for a cluster also and you can create index for composite index for multiple columns.

(Refer Slide Time: 02:32)



Indexing Examples

- Create an index for a single column, to speed up queries that test that column:
 - `CREATE INDEX emp_ename ON emp_tab(ename);`
- Specify several storage settings explicitly for the index:
 - `CREATE INDEX emp_ename ON emp_tab(ename)
TABLESPACE users STORAGE (INITIAL 20k NEXT 20k PCTINCREASE 75)
PCTFREE 0 COMPUTE STATISTICS;`
- Create index on two columns, to speed up queries that test either the first column or both columns:
 - `CREATE INDEX emp_ename ON emp_tab(ename, empno) COMPUTE STATISTICS;`
- If a query is going to sort on the function `UPPER(ENAME)`, an index on the `ENAME` column itself would not speed up this operation, and it might be slow to call the function for each result row
 - A function-based index precomputes the result of the function for each column value, speeding up queries that use the function for searching or sorting:
 - ▶ `CREATE INDEX emp_upper_ename ON emp_tab(UPPER(ename)) COMPUTE STATISTICS;`

Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a965

So, let us run through quickly couple of examples create an index for a single column to speed up queries the test that column. So, we are saying employee emp tab is a is a relation which has an attribute e name the employee name and we want to create an index on that if we do that then any search that is based on the employee name will become really fast.

Now, while you create the index you could also use optionally various other factors which relate to particularly the storage setting you can set what is you know what is the storage you want to keep for the index how you would like to increase increment that and so, on what is the table space. And very most interestingly you could say that compute statistics now this is something which is optional, but is very useful. For example, if you are not sure as to how your data is getting distributed in different relations and how really they are queried you would not know whether an index is good or it is inappropriate.

So, it is good to actually compute that statistics in terms of the index that you want to know that by doing this index what kind of accesses have happened? So, compute statistics will tell the database system to keep on computing this which you can subsequently refer to. You can create index on two columns also; so, here we are showing one where emp tab is indexed on employee name eme name and employee

number together. So, you saying that you create an index on both of these and compute the statistics at the same time.

Now, other ways there are index that can be created on functions. So, suppose if there is a query which going to sort based on the uppercase writing of the e name. So, if I just index the e name then that itself would not speed up the operation because while you want to sort then the e name will have to be changed into the upper case by upper and that is every time will have to do that for every record and then actually apply the sorting comparisons.

So, that will become a slow process, but you can do a function based indexing where you can specify as you can as you can see here the function based indexing where you say that you index based on upper e name. So, what will happen your actual values are in impossibly lower case or mixed case, but your index emp upper e name will get created on the in the order of the upper case of e name and will be very useful in terms of the sorting later on.

(Refer Slide Time: 05:36)

Bitmap Index in SQL

■ **create bitmap index** <index-name> on <relation-name>(<attribute-list>)

■ **Example:**

- Student (Student_ID, Name, Address, Age, Gender, Semester)
- CREATE BITMAP INDEX idx_Gender ON Student (Gender);
- CREATE BITMAP INDEX idx_Semester ON Student (Semester);

STUDENT_ID	STUDENT_NAME	ADDRESS	AGE	GENDER	SEMESTER
100	Joseph	Alarendon Township	20	M	1
101	Allen	Fraser Township	21	F	1
102	Chris	Clinton Township	20	F	2
103	Pathy	Troy	22	F	4

SEMESTER
1 1 1 0 0
2 0 0 1 0
3 0 0 0 0
4 0 0 0 1

■ **SELECT * FROM Student WHERE Gender = 'F' AND Semester =4;**

 AND 0 1 1 1 with 0 0 0 1 to get the result

Source: <https://www.tutorialcup.com/dbms/bitmap-index/>

Database System Concepts - 6th Edition 30.8 ©Silberschatz, Korth and Sudarshan

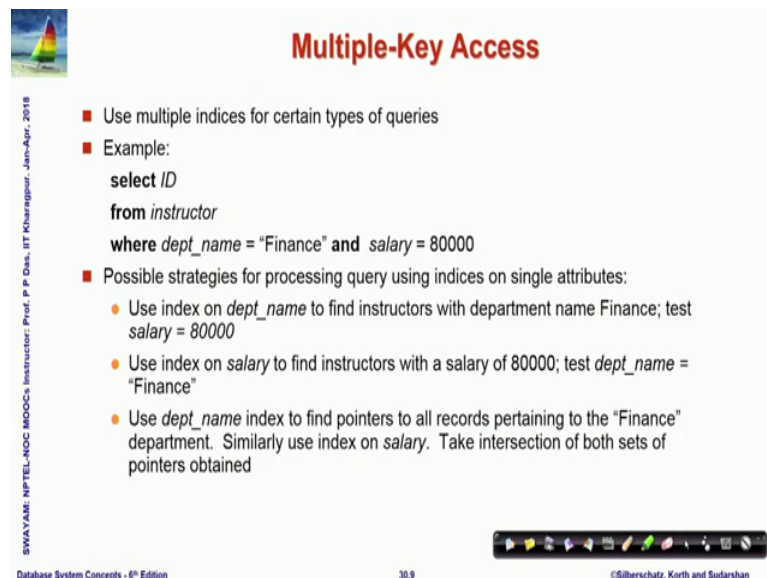
Now, you can like the normal index you can also create the bitmap index. So, you just say create bitmap index on the name and rest of the structure is similar. So, if there is a student relation which has these fields I can we can create an index on gender; we can create another index on semester these are very typical candidate for bitmap index

because gender can take 2 values here male and female semester can take 4 values 1, 2, 3, 4.

So, the bitmap are shown here and then if I want to do a select where the gender is F and semester is 4; then it is basically ending the bitmap of F which is 0 1 1 and the bitmap of semester 4 which is 0 0 1. So, if we if we add these two we will find that we have the result which is 0 0 0 1. So, which tells me that student id the fourth record of the student ID 103 is a result.

So, this is how bitmap indexing can be used in SQL.

(Refer Slide Time: 06:50)



Multiple-Key Access

- Use multiple indices for certain types of queries
- Example:

```
select ID
from instructor
where dept_name = "Finance" and salary = 80000
```
- Possible strategies for processing query using indices on single attributes:
 - Use index on *dept_name* to find instructors with department name Finance; test *salary = 80000*
 - Use index on *salary* to find instructors with a salary of 80000; test *dept_name = "Finance"*
 - Use *dept_name* index to find pointers to all records pertaining to the "Finance" department. Similarly use index on *salary*. Take intersection of both sets of pointers obtained

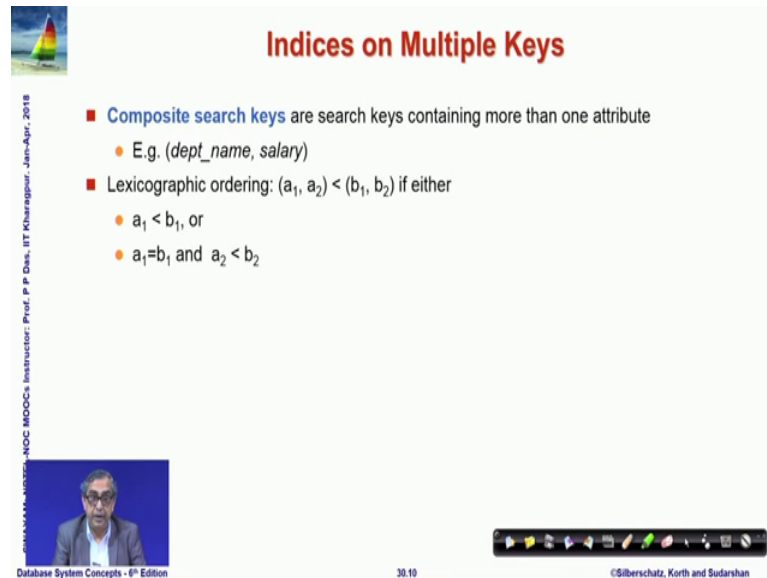
SWAYAM NPTL-NOC MOOCs Instructor: Prof. P Das, IIT Kharagpur, Jan-Apr, 2018

Database System Concepts - 9th Edition 30.9 ©Silberschatz, Korth and Sudarshan

And actually this the whole thing can be used subsequently in multiple key access for example, if you are doing a query where it is you have department name is finance and salary is 8000, then there could be several strategies for processing this query using the index values for example, if you have single index on single attributes. So, use you can use the index on department name to find instructors which have department and finance. And then test if the salaries 80000 or you can use index on salary to find instructors with salary 80000 and then test if department name is financed.

Or you can use department name index to find pointers to all records that part in to finance department. Index on salary to find all records that part in to 80000 salary and then take intersection of the both sets to get the final result.

(Refer Slide Time: 08:04)



Indices on Multiple Keys

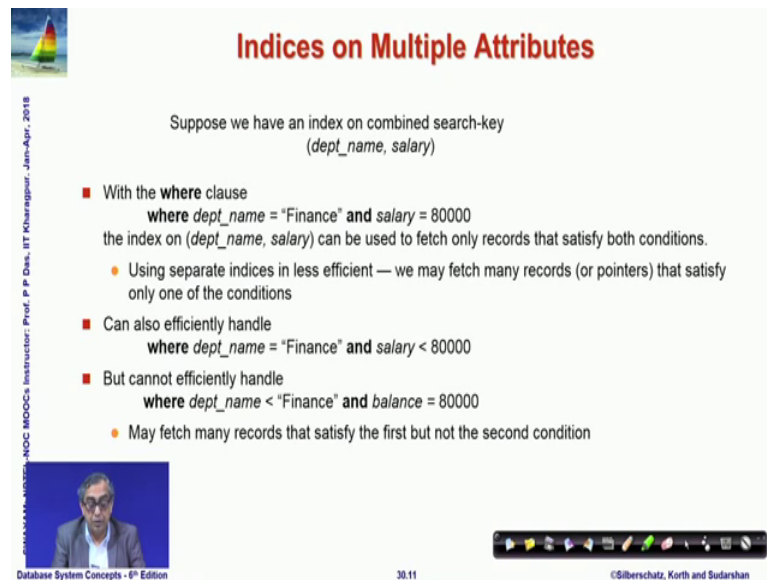
- **Composite search keys** are search keys containing more than one attribute
 - E.g. (*dept_name*, *salary*)
- **Lexicographic ordering:** $(a_1, a_2) < (b_1, b_2)$ if either
 - $a_1 < b_1$, or
 - $a_1 = b_1$ and $a_2 < b_2$

Database System Concepts - 9th Edition
©Silberschatz, Korth and Sudarshan

So, multiple key access could be achieved in terms of various single indexing single attribute indexing also; When we are doing composite search keys then naturally if there are 2 then the indexing means that you will have to define a combined lexical order. So, department name salary means that either department it is it is ordered to indexes are ordered in terms of just the department name.

Or if the department name is same then they are ordered in terms of salary this ordering in which you write the attributes in the multi composite search key is very important because you can see that for the two if the department name is same then the salary will be compared, but not the other way around.

(Refer Slide Time: 08:50)



Indices on Multiple Attributes

Suppose we have an index on combined search-key
(dept_name, salary)

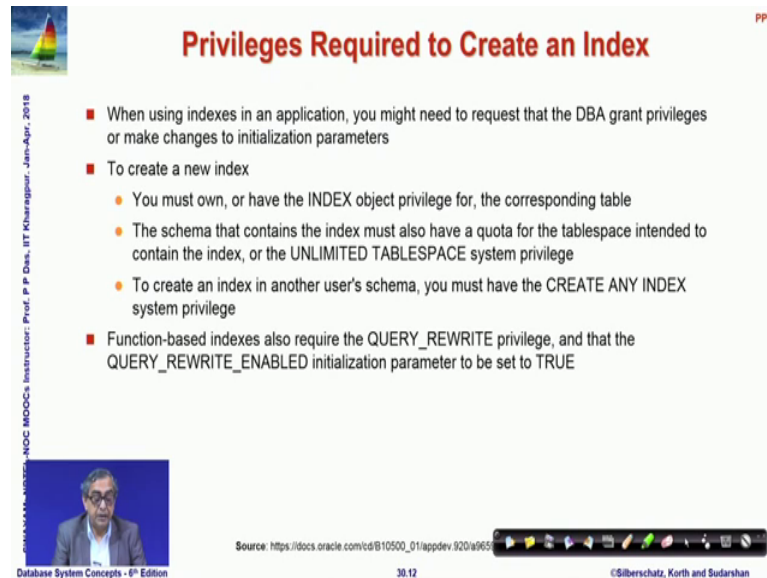
- With the **where** clause
where dept_name = "Finance" and salary = 80000
the index on (dept_name, salary) can be used to fetch only records that satisfy both conditions.
 - Using separate indices is less efficient — we may fetch many records (or pointers) that satisfy only one of the conditions
- Can also efficiently handle
where dept_name = "Finance" and salary < 80000
- But cannot efficiently handle
where dept_name < "Finance" and balance = 80000
 - May fetch many records that satisfy the first but not the second condition

Database System Concepts - 9th Edition
©Silberschatz, Korth and Sudarshan

So, when you have index on multiple attributes say again going back to that same example of department naming finance and salary being 80000. So, using separate index is less efficient though we saw how that can be done, but we can also efficiently handle if we have this indexing on department name and salary. Then we can also easily handle queries like department name is finance and salary is less than 100; it is not because as you can easily figure out because if I can find the equality then I also know what is less.

But note that we cannot efficiently handle if I say that where department name is less than finance and balance I am sorry this should be salary is 80000. The reason is that the ordering of the attributes in this composite key is department name salary. So, if there is if department name is less than is there is no way to check for the equality of salary, but if the department name equals then there is a possibility of checking on the salary. So, because of this ordering this will may fetch many records that satisfy the first one, but not the second condition.

(Refer Slide Time: 10:16)



Privileges Required to Create an Index

- When using indexes in an application, you might need to request that the DBA grant privileges or make changes to initialization parameters
- To create a new index
 - You must own, or have the INDEX object privilege for, the corresponding table
 - The schema that contains the index must also have a quota for the tablespace intended to contain the index, or the UNLIMITED TABLESPACE system privilege
 - To create an index in another user's schema, you must have the CREATE ANY INDEX system privilege
- Function-based indexes also require the QUERY_REWRITE privilege, and that the QUERY_REWRITE_ENABLED initialization parameter to be set to TRUE

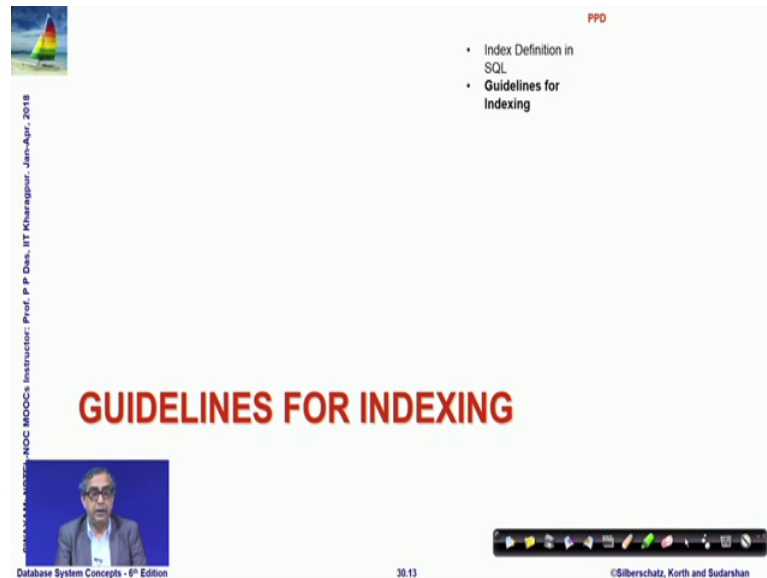
Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a965

Database System Concepts - 9th Edition ©Silberschatz, Korth and Sudarshan

Now, you should also remember that you need a special privilege to create an index because this is partly in the domain of the administrators job. So, you need the specific privileges access rights to be able to do that. So, to create a new index either you have to own or own that those set of tables on which you are creating the index and or have the index objective privilege for those tables or the schema that contains the index might also have a quota.

So, that you can because creating the index means you are will be using the temporary tablespace on a on a regular basis. And, but with this you will not be able to create index in some other user schema for that you need a global right which is the create any index kind of system privilege. So, also check if you are not being able to create an index check what is your privilege that exists function based indexes require other privileges; please check on that.

(Refer Slide Time: 11:28)



PPD

- Index Definition in SQL
- Guidelines for Indexing

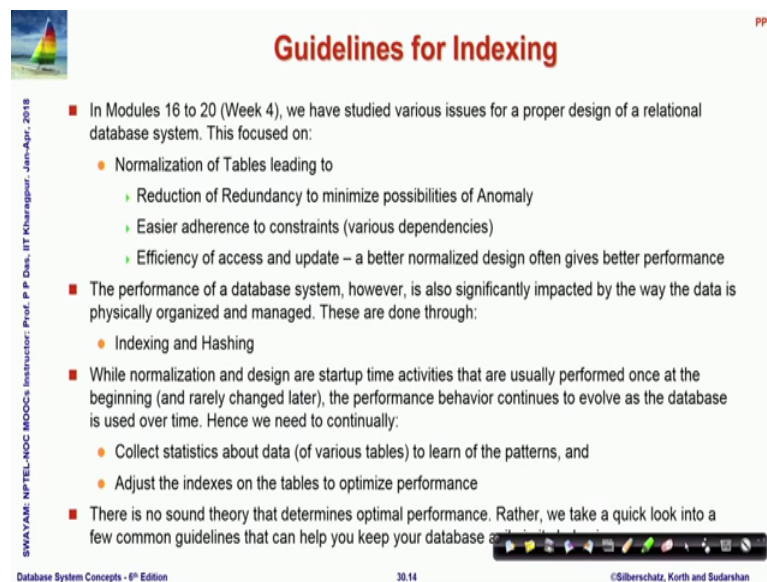
GUIDELINES FOR INDEXING

Database System Concepts - 9th Edition 30.13 ©Silberschatz, Korth and Sudarshan

SWAYAM: NPTEL-NOC MBOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Now, let us. So, we have seen how to create index how to use that in terms of the SQL application SQL query system.

(Refer Slide Time: 11:43)



Guidelines for Indexing

- In Modules 16 to 20 (Week 4), we have studied various issues for a proper design of a relational database system. This focused on:
 - Normalization of Tables leading to
 - Reduction of Redundancy to minimize possibilities of Anomaly
 - Easier adherence to constraints (various dependencies)
 - Efficiency of access and update – a better normalized design often gives better performance
- The performance of a database system, however, is also significantly impacted by the way the data is physically organized and managed. These are done through:
 - Indexing and Hashing
- While normalization and design are startup time activities that are usually performed once at the beginning (and rarely changed later), the performance behavior continues to evolve as the database is used over time. Hence we need to continually:
 - Collect statistics about data (of various tables) to learn of the patterns, and
 - Adjust the indexes on the tables to optimize performance
- There is no sound theory that determines optimal performance. Rather, we take a quick look into a few common guidelines that can help you keep your database as

Database System Concepts - 9th Edition 30.14 ©Silberschatz, Korth and Sudarshan

SWAYAM: NPTEL-NOC MBOCs Instructor: Prof. P. P. Das, IIT Kharagpur, Jan-Apr, 2018

Now, we will quickly take a look into why how should we index and where. So, if you recall in the modules 16 to 20 in the week four we have studied various issues of a proper design of relational database system, we focused on normalization of tables that can reduce redundancy and minimize anomaly how can we easily adhere to various

constraints how to improve the efficiency of access and update a better normalized design often gives better performance.

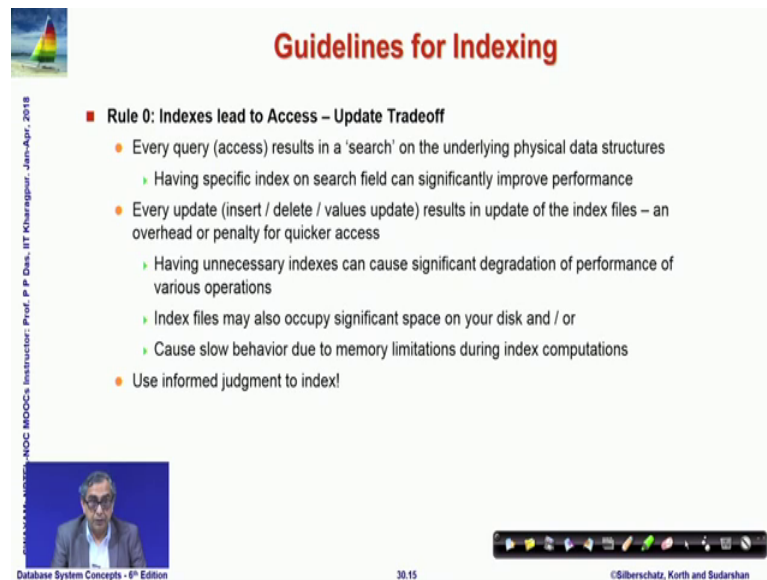
For example, we optimizing the minimizing the requirement for computing join and all those. So, those advantages we have saw, but the actual performance of a database system is significantly impacted by the way the physical data is organized and managed which does not come across in terms of the logical design that we have seen. So, these are what are being achieved in terms of indexing and hashing. So, this is where we we need to understand the actual boundary to the physical organization and that is what we have been trying to do.

So, if you think back while you are normalizing at the design level. So, those are the startup time activities; so, usually we will design and normalize and you know make the create table and do all that at the beginning of a database system. And it is really it will really be changed later because it will have severe implications, but the performance behavior will continue to evolve will continue to change because the design does not tell you exactly what the statistics of that data would be what the behavior of the data would be. So, it will evolve as data base is used over time.

Hence you will need to continuously collect statistics about the data of various tables to learn of the patterns as to which table is getting heavier which where what are the attributes on which more accesses are happening, where what kind of queries you are getting and you have to adjust the indexes on the tables to optimize the performance.

So, that is the whole requirement all about unfortunately unlike the functional dependency or multivalued dependency theories that we studied in the design space; there is no sound theory that determines optimal performance. So, all that have is more and expertise that you develop through experience. So, what I will take you through are a set of few common guidelines about how to keep your database agile while you are you go through the life cycle of different data coming in and going out.

(Refer Slide Time: 14:23)



Guidelines for Indexing

■ **Rule 0: Indexes lead to Access – Update Tradeoff**

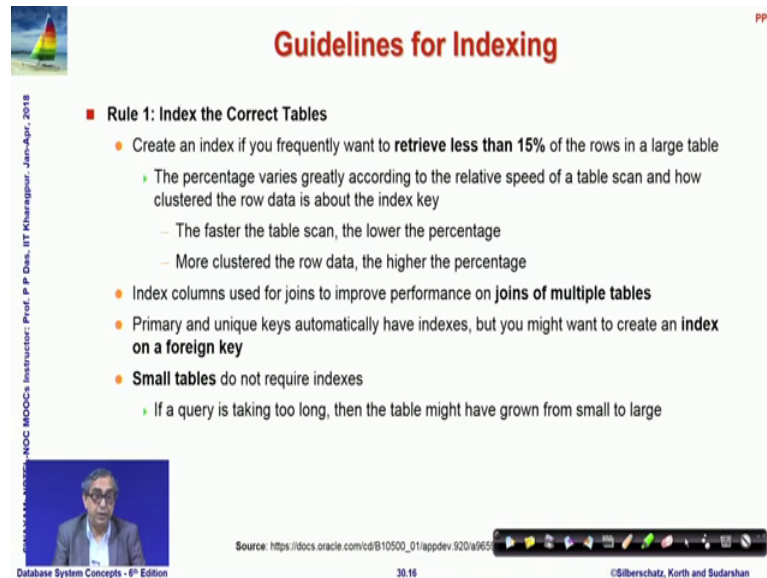
- Every query (access) results in a 'search' on the underlying physical data structures
 - ▶ Having specific index on search field can significantly improve performance
- Every update (insert / delete / values update) results in update of the index files – an overhead or penalty for quicker access
 - ▶ Having unnecessary indexes can cause significant degradation of performance of various operations
 - ▶ Index files may also occupy significant space on your disk and / or
 - ▶ Cause slow behavior due to memory limitations during index computations
- Use informed judgment to index!

Database System Concepts - 9th Edition
©Silberschatz, Korth and Sudarshan

So, the first rule I say rule 0 is the indexes lead to access update tradeoff we have already seen this at every query results in a search in the underlying physical data structure as we have understood. Having specific index on search certainly can improve performance, but as we have already noted every update with the be it insert, delete or update of values will result in update of the index files.

So, it is an overhead or penalty for quicker access that we are paying. So, having unnecessary indexes can cause significant degradation of performance. Index files will also occupy significant space on your disk and it may actually cause to slow down your behavior due to memory limitation during index computation. So, rule 0 indexes lead to this trade off always be watchful about that use judgment to index.

(Refer Slide Time: 15:26)



Guidelines for Indexing

Rule 1: Index the Correct Tables

- Create an index if you frequently want to **retrieve less than 15%** of the rows in a large table
 - ▶ The percentage varies greatly according to the relative speed of a table scan and how clustered the row data is about the index key
 - The faster the table scan, the lower the percentage
 - More clustered the row data, the higher the percentage
- Index columns used for joins to improve performance on **joins of multiple tables**
- Primary and unique keys automatically have indexes, but you might want to create an **index on a foreign key**
- **Small tables** do not require indexes
 - ▶ If a query is taking too long, then the table might have grown from small to large

Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a965

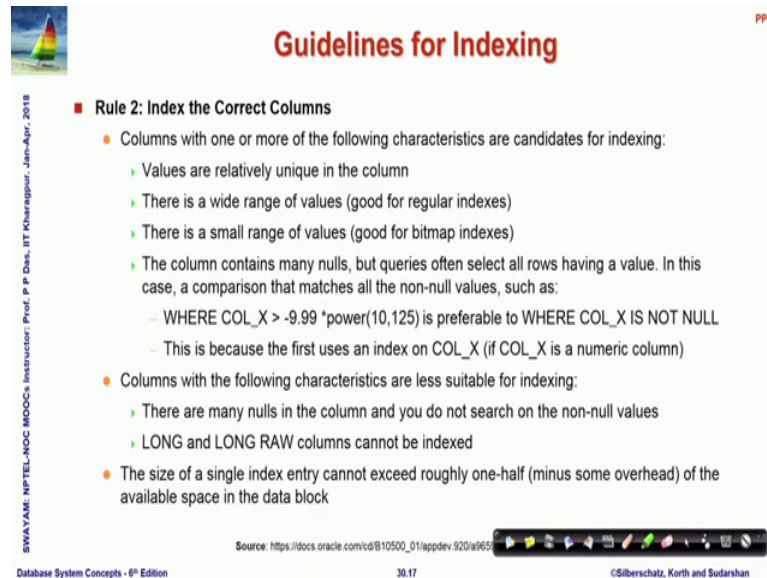
Database System Concepts - 9th Edition 30.16 ©Silberschatz, Korth and Sudarshan

Rule 1 index the correct tables decide which tables are best candidates for index to creating an index if you frequently want to retrieve say less than 15 percent of the rows in a large table. Now first 15 percent is a ballpark number this can vary greatly according to the relative speed of the table scan ah.

Fast of the table scan you can use a lower percentage of cut off more cluster the row data you can use a higher percentage for cut up. Index tables index columns used for joining multiple tables if you have situations or multiple tables are used in joins on a on a moderately regular basis then the columns are used in the join in these tables; these tables should be indexed based on those.

The primary and unique keys automatically have indexes, but you might want to you have an index on the foreign key. So, consider that small tables do not require index if a query is requiring unnecessarily long time or unexpectedly long time; it is time to check if the table has become really big compared to small and it might be term to index that.

(Refer Slide Time: 16:45)



Guidelines for Indexing

Rule 2: Index the Correct Columns

- Columns with one or more of the following characteristics are candidates for indexing:
 - Values are relatively unique in the column
 - There is a wide range of values (good for regular indexes)
 - There is a small range of values (good for bitmap indexes)
 - The column contains many nulls, but queries often select all rows having a value. In this case, a comparison that matches all the non-null values, such as:
 - WHERE COL_X > -9.99 *power(10,125) is preferable to WHERE COL_X IS NOT NULL
 - This is because the first uses an index on COL_X (if COL_X is a numeric column)
- Columns with the following characteristics are less suitable for indexing:
 - There are many nulls in the column and you do not search on the non-null values
 - LONG and LONG RAW columns cannot be indexed
- The size of a single index entry cannot exceed roughly one-half (minus some overhead) of the available space in the data block

Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a965

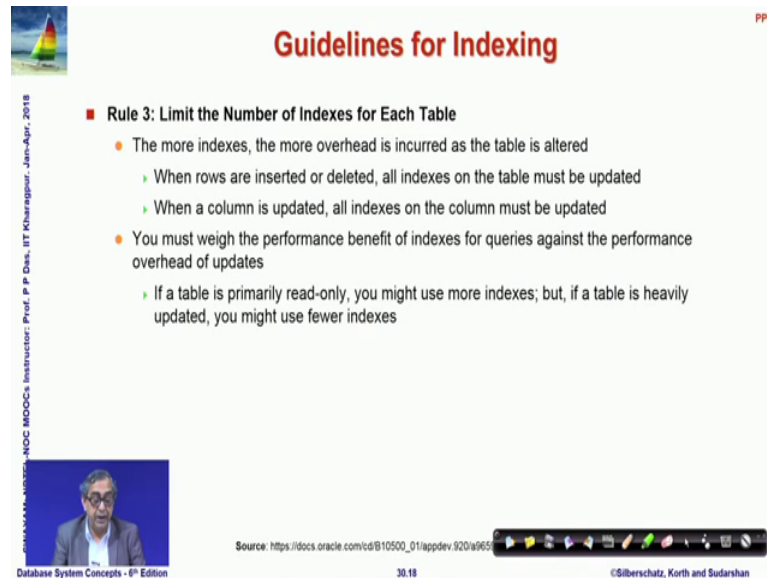
Database System Concepts - 8th Edition 30.17 ©Silberschatz, Korth and Sudarshan

So, rule 1 index the correct tables and certainly related to that is index the correct columns. The columns with some of the characteristics I have just noted down are good candidates for indexing values are relatively unique in the column, then indexing will give you a good benefit. There is a wide range of values where you can your regular indexes will work well.

There is a small range of values where bitmap indexes will give you good results. So, use those in column contains many nulls, but queries often select all rows having a value. So, there are column have lot of null values, but whenever you do a query then you actually take out rows which have values. So, in those cases you can you can actually if you involve certain I mean if you write the SQL query by keeping the condition in a way. So, that the index can be used that is for example, you could put a condition such that only non null values will be matched.

Compared to that if you have just taken (Refer Time: 17:54) at non null check your first query would run faster; if you have an index on the COL X because the query would be able to work on that index. So, these are things that you should do in terms of the column. And if a column has the kind of characteristic that there are many nulls in the column and you typically do not search non null values; then it is good it it is better not to index those columns long and long row columns cannot be indexed anyway. So, this remember the rule 2 index the correct columns.

(Refer Slide Time: 18:29)



Guidelines for Indexing

Rule 3: Limit the Number of Indexes for Each Table

- The more indexes, the more overhead is incurred as the table is altered
 - When rows are inserted or deleted, all indexes on the table must be updated
 - When a column is updated, all indexes on the column must be updated
- You must weigh the performance benefit of indexes for queries against the performance overhead of updates
 - If a table is primarily read-only, you might use more indexes; but, if a table is heavily updated, you might use fewer indexes

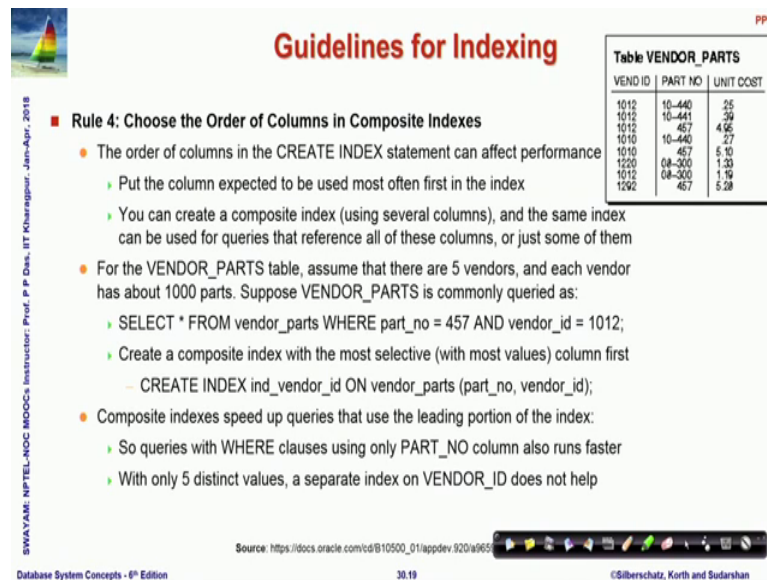
Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a965

Database System Concepts - 9th Edition 30.18 ©Silberschatz, Korth and Sudarshan

Then the rule 3 limit the number of indexes for each table the more the index more over it we have already seen this as a part of rule 2 rule 0 as well. When rows are inserted or deleted indexes of a table must be updated when columns are updated all the indexes on the column must be updated. So, there is a lot of cost; so, half as limited number of indices as will start with purposed.

So, you must regularly weigh the benefit of having the indexed for queries against the performance overhead of the updates. For example, if a table is primarily read only you might use more indexes because the overhead will be less, but if a table is heavily updated you might use fewer number of indices.

(Refer Slide Time: 19:14)



Guidelines for Indexing

Table VENDOR_PARTS

VENDOR ID	PART NO	UNIT COST
1012	10-440	25
1012	10-441	35
1012	457	4.56
1010	10-440	27
1010	457	5.10
1200	08-200	1.33
1012	08-200	1.19
1202	457	5.28

Rule 4: Choose the Order of Columns in Composite Indexes

- The order of columns in the CREATE INDEX statement can affect performance
 - Put the column expected to be used most often first in the index
 - You can create a composite index (using several columns), and the same index can be used for queries that reference all of these columns, or just some of them
- For the VENDOR_PARTS table, assume that there are 5 vendors, and each vendor has about 1000 parts. Suppose VENDOR_PARTS is commonly queried as:
 - SELECT * FROM vendor_parts WHERE part_no = 457 AND vendor_id = 1012;
 - Create a composite index with the most selective (with most values) column first
 - CREATE INDEX ind_vendor_id ON vendor_parts (part_no, vendor_id);
- Composite indexes speed up queries that use the leading portion of the index:
 - So queries with WHERE clauses using only PART_NO column also runs faster
 - With only 5 distinct values, a separate index on VENDOR_ID does not help

Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a965

Database System Concepts - 8th Edition 30.19 ©Silberschatz, Korth and Sudarshan

Rule 4 choose the order of columns in the composite index. So, you have already seen couple of slides back I talk to you to the impact of what is the impact of ordering in other columns in terms of composite index. So, the order of columns in the create index statement can affect performance. So, the column that you expected to be used most often put that as the first index.

Because it is also possible that you are actually not doing a query which takes the whole of the composite search key, but a part of it, but if you have a composite search key index you will still benefit if the query is using the attributes from the first part of the index. So, here I am showing some example say there is a vendors part table and let us say there are 5 vendors and let us say. So, there is vendor id part number and unit cost forget about unit cost for this consideration.

So, it is primarily part number and vendor id. So, let us say there are 5 vendors and each vendor has about 1000 parts. So, and let us say that it is queried like this that the part number in such and such and vendor id is such and such you get you select all all that matches.

Now, if you create a composite index then it should be on part table number vendor id not the other way around. Because if you if you do that then queries where only part number is used will also run faster, but the vendor id here is not a very good candidate for indexing as a as a first attribute because it has only five possible values very small number of

value. So, indexing really does not help here it cannot discriminate after indexing there will be lot of clusters still found.

(Refer Slide Time: 21:17)

Guidelines for Indexing

■ Rule 5: Gather Statistics to Make Index Usage More Accurate

- The database can use indexes more effectively when it has statistical information about the tables involved in the queries
 - ▶ Gather statistics when the indexes are created by including the keywords COMPUTE STATISTICS in the CREATE INDEX statement
 - ▶ As data is updated and the distribution of values changes, periodically refresh the statistics by calling procedures like (in Oracle):
 - DBMS_STATS.GATHER_TABLE_STATISTICS and
 - DBMS_STATS.GATHER_SCHEMA_STATISTICS

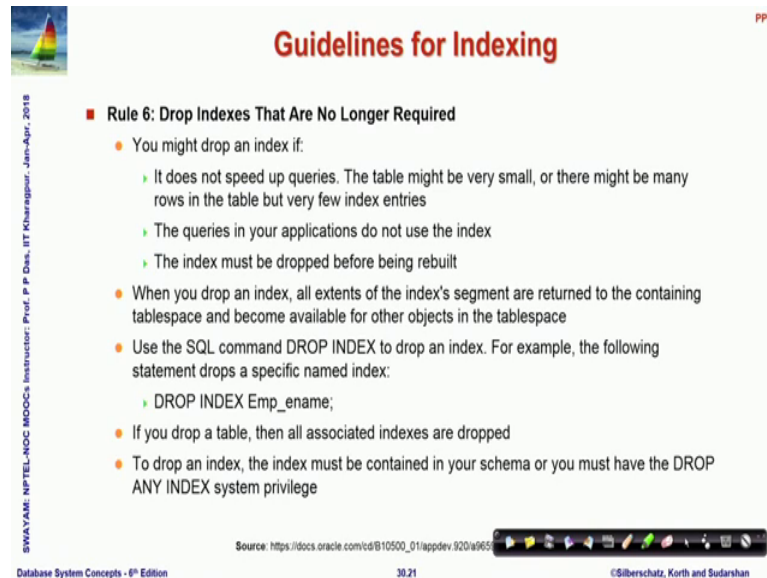
Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a965

Database System Concepts - 6th Edition 30.20 ©Silberschatz, Korth and Sudarshan

Rule number 5 gather statistics to make index usage more accurate that is a that is a very very important factor the database can use indexes more effectively if the statistical information is available. So, gather statistics learn from that we have already discussed how to gather statistics from the create index statement.

And then there are functions these are function names in oracle in your system you might want to check up what these functions are called. So, by that you can find out statistics about the tables and the schema that you have and use that information to subsequently optimize the index.

(Refer Slide Time: 21:58)



Guidelines for Indexing

Rule 6: Drop Indexes That Are No Longer Required

- You might drop an index if:
 - It does not speed up queries. The table might be very small, or there might be many rows in the table but very few index entries
 - The queries in your applications do not use the index
 - The index must be dropped before being rebuilt
- When you drop an index, all extents of the index's segment are returned to the containing tablespace and become available for other objects in the tablespace
- Use the SQL command `DROP INDEX` to drop an index. For example, the following statement drops a specific named index:
 - `DROP INDEX Emp_ename;`
- If you drop a table, then all associated indexes are dropped
- To drop an index, the index must be contained in your schema or you must have the `DROP ANY INDEX` system privilege

Source: https://docs.oracle.com/cd/B10500_01/appdev/920/a965

Database System Concepts - 8th Edition 30.21 ©Silberschatz, Korth and Sudarshan

The last rule 6 is drop index that are no longer required. So, if an index might be dropped because for several reasons for example, it is it simply does not speed up the queries. So, table might have become too small there will be many rows in the table, but very few index increase right we have seen these are not the ideal.

So, it may not have been the case earlier and now it may be the case. So, in that case that index should be drop because it is not helping you the queries in your application do not use the index the query you have certain indexes and the queries are done on other attributes or other composite attributes. So, it is not the indexes of no value and; obviously, index must be dropped before being rebuilt if you are rebuilding if you are creating new index in a new way then. So, make a judgment and drop indexes which are no longer required as an when you observe that in drop indexes and improve the performance.

When you drop an index all extents of the indexes segment are return to the tablespace. So, this is basically the space management and SQL command for this you already know now please keep in mind that if you drop a table then all associated indexes are automatically dropped because; obviously, if the data is not there then how about their index. So, to drop an index you need the drop any index system privilege we talked about privileges earlier too.

(Refer Slide Time: 23:26)

Module Summary

- Learnt to create Indexes in SQL
- Introduced a few rules for good index

Database System Concepts - 8th Edition 30.22 ©Silberschatz, Korth and Sudarshan

So, this summarizes our discussions on indexing and hashing. So, here in this particular module you have learned to create index in SQL and introduce few rules for good index. Overall in this week in all the 5 modules we have learnt about how to speed up query processing, how to speed up the execution of access insert delete queries in your database through the lifetime. And we have looked at various different indexing schemes, we have looked at hashing and made comparisons.

So, one take back that you can certainly have is the most important indexing scheme or indexing structure is the B plus tree which can be used for data files as well as for index files and several like SQL server uses the B plus tree only. Now, hashing options we have looked at and we have seen that it has a varied acceptability it is a powerful technique, but not all systems use that equally strongly.

And we have then made understood that indexing a database or tables on different attributes is a very delicate responsibility which has to be done with a lot of judgment. And for that statistics must be rightly collected and good judgment in terms of the distribution of the data, access of the data nature of queries all these need to be considered carefully so, that you can really get good performance from the design that you have.

So, on top of the knowledge of good design that you acquired through the all the theory of normalization and you know redundancy removal; your good judgment in terms of

good appropriate index design will take you a long way in terms of making a very good database system engineer.