# Discuss query optimization techniques

## References:

## References:

Oracle Database 10g SQL, by Jason Price. Chapter 16 - High Performance SQL Tuning:   
<https://learning.oreilly.com/library/view/oracle-database-10g/9780072229813/chapter16.html>

Oracle Database 10g: The Complete Reference, by Kevin Loney. Chapter 17 - Creating and Managing Tables, Views, Indexes, Clusters, and Sequences – Indexes:  
<https://learning.oreilly.com/library/view/oracle-database-10g/9780072253511/ch17.html#ch17lev1sec96>

Oracle Database 10g: The Complete Reference, by Kevin Loney. Chapter 43 - The Hitchhiker’s Guide to Tuning Applications and SQL:  
<https://learning.oreilly.com/library/view/oracle-database-10g/9780072253511/ch43.html#ch43lev1sec234>

## Indexes

* An Index is a mechanism to boost performance for retrieval. “A logical pointer to a physical location”
* An index can be placed on a column or columns to help the system find the data faster.
* Every time data in the system is updated or inserted (deleted, etc), any corresponding indexes **have to be updated as well** – this is done automatically by the system.

Syntax:

CREATE [ UNIQUE | BITMAP ] INDEX index\_name ON table\_name ( {column\_name | column\_expressions} [ASC | DESC] [, …] ) [various storage attributes];

* BITMAP index – works best for columns with a small set of possible values – e.g: gender (M, F, U)
* UNIQUE index – the index prevents duplication of data – the system checks for duplicates when the index is created and each time data is added. Primary key and unique key constraints do this as well. Depending on your standards, you may choose to enforce uniqueness with indexes or constraints.
* Terms:
  + B-tree index – default and most common type of index
  + Clustered index – sorts table rows in the same physical order as the index
  + Composite index – index involving more than one column, generally start with the column that you use most often in searches because either the entire index or leading portion of the index can be used for searches and sorts in some DBMSs
  + Function-based index – an index on a computer column (e.g. UPPER(lastName))

# How, what, why to Index

### Advantage of indexing:

* Indexes speed up the retrieval of data

### Disadvantages of indexing:

* Building and maintaining an index takes time and storage space on the database server.
* Inserts, updates, and deletes on indexed columns may take longer because the index has to be updated in addition to the data itself.

### Considerations

* Frequency of queries (reads) versus frequency of updates (writes)
  + More indexes means faster retrieval but slower writes
  + More tables through normalization means less redundancy but more overhead when re-joining tables.
  + If there are more queries than data modifications, then you want to minimize the number of tables (de-normalize) and maximize the numbers of indexes.
  + The opposite is true if we have a reverse case (more data modifications than queries)
* Likelihood that queries and updates are going to be slow
  + How often the indexed columns will be used in the where clause or as part of a JOIN
  + Time vs storage space
* Whether to use special types of indexes (e.g. bitmap indexes)

### Guidelines

* Use indexes:
  + Unique indexes on primary keys (Oracle does this automatically for us).
  + Foreign key columns used in joins.
  + Columns often accessed in sorted order
  + Columns searched for ranges of values might be suitable for clustered indexes (find first row, subsequent rows will be adjacent).
* Don’t use indexes:
  + Columns that are rarely used in queries
  + Columns that can only have a few possible values (eg gender) – Although a bitmap index may be suitable in this case.
  + Small tables with few rows.
  + If you will be retrieving all rows rather than a few rows.

# Query Optimization

SQL is non-procedural – you specify what, not how. The DBMS determines how the work will be accomplished.

The DBMS has an optimizer that attempts to choose the most efficient method to execute the statement – for example, which table to access in which order, what indexes to use, etc.

* Rule-based – optimizer makes decisions based on a set of ranked guidelines – looking at which indexes are available, unique constraints, etc. This only considers the structure, not the content of the tables. (Obsolete since Oracle 10g)
* Cost-based – optimizer chooses the cheapest option (in terms of time), based on statistics kept by the system. To be effective, the statistics must be kept up to date.

There are ways to re-phrase some queries to improve the efficiency of the queries – these things are very system specific (e.g. Oracle quite different than SQL server), so check the docs. Examples include:

* There are often tools to see how a query will be run.
* Some things like the ordering of tables of the FROM clause can have an impact on how the query is executed. For example, in Oracle you often want the table with the fewest hits to be listed last in the FROM clause.

### General Guidelines / Rules of Thumb:

* If the index is on a column, that index may not be used if you use a function on the column. For example, it is better to use lastName like ‘M%’ rather than SUBSTR(lastName,1,1) = ‘M’ since the former will use an index on lastName while the latter will not.
* A LIKE expression that starts with a wildcard will not use an index. EG: lastName LIKE ‘%M%’
* Negative prerequisites (!= or NOT combined with other operators) often do no use indexes since the assumption is that most values in the column will not be equal to a value.

# Query Optimization Tips/Rules

* Use a WHERE clause to filter rows
  + If you’re only looking for a specific row, don’t ask for all rows (\*), use the WHERE clause to restrict the rows returned.
  + Example: SELECT \* FROM CUSTOMERS
    - Vs
  + SELECT \* FROM CUSTOMERS WHERE cnum IN (2001, 2002);
* Use Table joins instead of multiple queries
  + If you need information from more than one table, you should use a join rather than multiple queries.
  + Example: SELECT cname FROM Customers WHERE cnum = 2004; and then another query SELECT onum FROM Orders WHERE cnum = 2004;
    - Vs
  + SELECT onum, cname FROM CUSTOMERS c JOIN Orders o ON c.cnum = o.cnum WHERE c.cnum = 2004;
  + In general, it is faster to execute one query that is more complex than two simpler queries.
  + You should join the tables such that the table with the least number of hits is joined last. Ex, tab1 1000 rows, tab2 100 rows, tab3 10 rows, we’d want to join them in that order.
  + Also, avoid joining complex views in your queries, because doing so causes the queries for the views to be run first, followed by your actual query. If needed, rewrite the whole query (views + query) to be one query (remove the views).
* Use Fully Qualified Column References When Performing Joins
  + Always include table aliases in the prefix of each column reference. This saves the DBMS from searching the table headers to determine which table has that column.
  + Example: SELECT **o.onum, c.cname** FROM CUSTOMERS c JOIN Orders o ON c.cnum = o.cnum WHERE c.cnum = 2004;
* Use indexes on tables
  + Indexes speed up retrieval if designed properly.
* Use WHERE rather than HAVING
  + You use the WHERE clause to filter rows; you use HAVING to filter groups of rows.
  + Example: SELECT odate, avg(amt) FROM Orders GROUP BY odate HAVING odate IN (’03-OCT-00’, ’04-OCT-00’);
* Use EXISTS rather than IN for correlated subqueries
  + You use IN to check if a value is contained in a list. You use EXISTS to check for the existence of rows returned by a sub query. They are different: EXISTS just checks for the existence of rows, IN checks actual values. EXISTS typically offers better performance.
* Use EXISTS Rather than DISTINCT
  + You can suppress the display of duplicate records using DISTINCT. Whenever possible, you should use EXISTS rather than DISTINCT, because DISTINCT sorts the retrieved rows before suppressing duplicates.

## Comparing Execution Plans

* We can compare the cost of two execution plans together, the one with the lower cost should be more efficient.