SQL Optimization

Data and Code

1) Prefer smaller tables. Use normal forms. Put infrequent columns in their own table. Put columns that will be null at the end.
2) Select the columns you need and don't use asterisk.
3) Use where to limit rows
4) Avoid implicit conversion in where clauses (compare the same types)
5) Intersect usually outperforms Inner Join. Intersect performs its own Distinct.
6) Minus generally performs better than join. Minus performs its own Distinct.
7) Avoid correlated subquery if possible. Try Inner Join.
8) If inner query is small as compared to outer, then use In Condition. If inner query is large, then use the Exists condition.
9) Generally, multi-column IN condition performs slightly better than ANDs/ORs.
10) Use with clause if you have a lot of repeated subqueries.
11) Use append hint to possibly reduce Insert times.
12) Use ON clause (INNER JOIN on A.respondent_ID =B.Respondent_ID) for join criteria and the Where clause for subsetting criteria.
13) In from clause, tables with fewer rows (being subsetted) first.
14) Avoid Cartesian Products, if possible.

Index

1) Gathering statistics on your index is a vital step.
2) B-Tree is a general purpose index used with many different types.
3) Can place an index on one, two or more columns. Look at what is referenced in Where clause for guidance on which to pick and what to index.
4) Index automatically created for primary key columns, which prevents data duplication and helps with joins involving those columns.
5) Bitmap Index is used often in data warehouses where the data changes infrequently. Or if there are a lot of AD-HOC queries (WHERE clause with ANDs/ORs). Or Degree of cardinality is low or moderately low as compared to number of rows in table.
6) Use function-based index when a function needs to be called on a column. Otherwise it will not register.
7) Bitmap join index joins a table in advance, so it takes less time at runtime.

Partition

1) Breaks apart large table into several smaller tables.
2) Based on one or more columns.
3) Range partitioning partitions a table by range of values (Survey date, etc)
4) List partitioning partitions a table by specific list of values (gender, etc..)

5) Composite partitions are partitions of partitions (Range-List, List-List, etc).
6) Can create an index within each individual partition. Or one index across all partitions. Or create an index with its own partition.
7) List Partitioning creates partitions using discrete values of a single column.