Using Indexed Views

Source	http://www.microsoft.com/technet/prodtechnol/sql/2005/impprfiv.mspx

How the Query Optimizer Uses Indexed Views

The SQL Server query optimizer automatically determines when an indexed view can be used for a given query execution. The view does not need to be referenced directly in the query for the optimizer to use it in the query execution plan. Therefore, existing applications may take advantage of the indexed views without any changes to the application itself; only the indexed views have to be created.

Optimizer Considerations

The query optimizer considers several conditions to determine if an indexed view can cover the entire query or a portion of it. These conditions correspond to a single FROM clause in the query and consist of the following:

- **The tables in the query FROM clause must be a superset of the tables in the indexed view FROM clause.**
- **The join conditions in the query must be a superset of the join conditions in the view.**
- **The aggregate columns in the query must be derivable from a subset of the aggregate columns in the view.**
- **All expressions in the query select list must be derivable from the view select list or from the tables not included in the view definition.**
- **One predicate subsumes another if it matches a superset of the rows matched by the other. For example, "T.a=10" subsumes "T.a=10 and T.b=20." Any predicate subsumes itself. The part of the predicate of the view that restricts values of one table must subsume the part of the predicate of the query that restricts the same table. Furthermore, it must do so in a way that SQL Server can verify.**
- **All columns in the query search condition predicates that belong to tables in the view definition must appear in one or more of the following in the view definition:**
    - **A GROUP BY list.**
    - **The view select list if there is no GROUP BY.**
    - **The same or equivalent predicate in the view definition.**

Cases (1) and (2) allow SQL Server to apply a query predicate to rows from the view to further restrict the rows of the view. Number (3) is a special case where no filtering is needed on the column, so the column needn't appear in the view.

If the query contains more than one FROM clause (subqueries, derived tables, UNION), the optimizer may select several indexed views to process the query, and apply them to different FROM clauses.2

Example queries demonstrating these conditions are presented at the end of this document. Allowing the query optimizer to determine which indexes, if any, to use in the query execution plan is the recommended best practice.

Using the NOEXPAND view hint

When SQL Server processes queries that refer to views by name, the definitions of the views normally are expanded until they refer only to base tables. This process is called *view expansion*. It's a form of macro expansion.

The NOEXPAND view hint forces the query optimizer to treat the view like an ordinary table with a clustered index. It prevents view expansion. The NOEXPAND hint can only be applied if the indexed view is referenced directly in the FROM clause. For example,

```
SELECT Column1, Column2, ... FROM Table1, View1 WITH (NOEXPAND)
WHERE ...
```

Use NOEXPAND if you want to be sure to have SQL Server process a query by reading the view itself instead of reading data from the base tables. If for some reason SQL Server chooses a query plan that processes the query against base tables when you'd prefer that it use the view, consider using NOEXPAND. You must use NOEXPAND in all versions of SQL Server other than Developer and Enterprise editions to have SQL Server process a query against an indexed view directly. You can see a graphical representation of the plan SQL Server chooses for a statement using the SQL Server Management Studio tool Display Estimated Execution Plan feature. Alternatively, you can see different non-graphical representations using SHOWPLAN_ALL, SHOWPLAN_TEXT, or SHOWPLAN_XML. See SQL Sever books online for a discussion of the different versions of SHOWPLAN.

Using the EXPAND VIEWS query hint

When processing a query that refers to a view by name, SQL Server always expands the views, unless you add the NOEXPAND hint to the view reference. It attempts to match indexed views to the expanded query, unless you specify the EXPAND VIEWS query hint in an OPTION clause at the end of the query. For example, suppose there is an indexed view View1 in the database. In the following query, View1 is expanded based on its logical definition (its CREATE VIEW statement), and then the EXPAND VIEWS option prevents the indexed view for View1 from being used in the plan to solve the query.

```
SELECT Column1, Column2, ... FROM Table1, View1 WHERE ...
OPTION (EXPAND VIEWS)
```

Use EXPAND VIEWS if you want to be sure to have SQL Server process a query by accessing data directly from the base tables referenced by the query, instead of possibly accessing indexed views. EXPAND views may in some cases help eliminate lock contention that could be experienced with an indexed view. Both NOEXPAND and EXPAND VIEWS can help you evaluate performance with and without use of indexed views when you test your application.

***~~~ End of Article ~~~***