SQL Best Practices

* Do not use SELECT \* in your queries.
* Always use table aliases when your SQL statement involves more than one source.
* Use the more readable ANSI-Standard Join clauses instead of the old style joins.
* Do not use column numbers in the ORDER BY clause.
* Always use a column list in your INSERT statements.
* Don’t ever use double quotes in your T-SQL code.
* Do not prefix your stored procedure names with “sp\_”.
* Always use a SQL formatter to format your sql like
* Optimize Your Queries For the Query Cache\
* When the same query is executed multiple times, the result is fetched from the cache, which is quite fast.

Why is SELECT \* considered harmful

* Not all the fields are indexed, forcing a full table scan - less efficient
* What you save to send SELECT \* over the wire risks a full table scan
* Returning more data than is needed
* Returning trailing columns using variable length data type can result in search overhead

Use the more readable ANSI-Standard Join clauses instead of the old style joins

* With ANSI joins, the WHERE clause is used only for filtering data. Whereas with older style joins, the WHERE clause handles both the join condition and filtering data. Furthermore ANSI join syntax supports the full outer join. The first of the following two queries shows the old style join, while the second one shows the new ANSI join syntax:
* *-- old style join*   
  SELECT a.Au\_id,   
         t.Title   
  FROM   TITLES t,   
         AUTHORS a,   
         TITLEAUTHOR ta   
  WHERE  a.Au\_id = ta.Au\_id   
         AND ta.Title\_id = t.Title\_id   
         AND t.Title LIKE ‘%Computer%’
* *--ANSI join syntax*   
  SELECT a.Au\_id,   
         t.Title   
  FROM   AUTHORS a   
         INNER JOIN TITLEAUTHOR ta   
           ON a.Au\_id = ta.Au\_id   
         INNER JOIN TITLES t   
           ON ta.Title\_id = t.Title\_id   
  WHERE  t.Title LIKE ‘%Computer%’
* Do not use column numbers in the ORDER BY clause

Always use column names in an order by clause. Avoid positional references. Consider the following example in which the second query is more readable than the first one:

SELECT OrderID, OrderDate

FROM Orders

ORDER BY 2

SELECT OrderID, OrderDate

FROM Orders

ORDER BY OrderDate

* Always use a column list in your INSERT statements
* Always specify the target columns when executing an insert command. This helps in avoiding problems when the table structure changes (like adding or dropping a column). Consider the following table:

CREATE TABLE EUROPEANCOUNTRIES

(

Countryid INT PRIMARY KEY,

Countryname VARCHAR(25)

)

* Here’s an INSERT statement without a column list , that works perfectly:

INSERT INTO EuropeanCountries

VALUES (1, ‘Ireland’)

Now, let’s add a new column to this table:

ALTER TABLE EuropeanCountries

ADD EuroSupport bit

* Now run the above INSERT statement. You get the following error from SQL Server: Server: Msg 213, Level 16, State 4, Line 1 Insert Error: Column name or number of supplied values does not match table definition. This problem can be avoided by writing an INSERT statement with a column list as shown below:

INSERT INTO EuropeanCountries

(CountryID, CountryName)

VALUES (1, ‘England’)