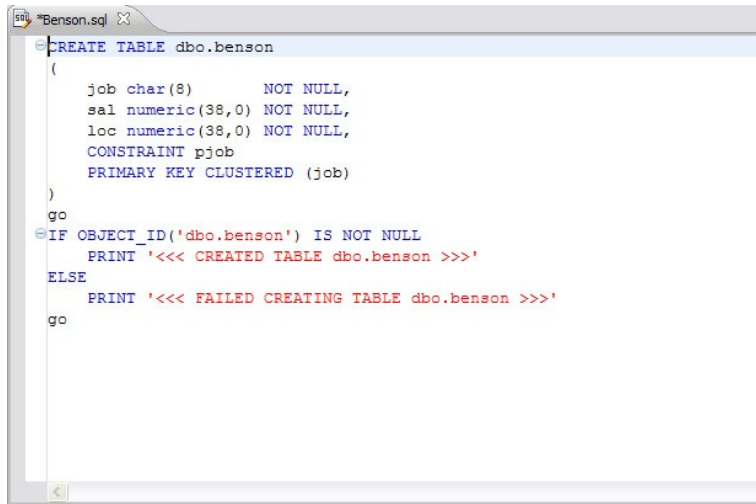


Working in SQL Editor

SQL Editor handles SQL code formats and contains context-sensitive command menus, tailored with pertinent functionality for development purposes. Other files may be opened in SQL Query Tuner, as well, but these are handled by other editors.

For example, if a text file is opened in the Workbench, SQL Query Tuner detects and opens the contents of that file in a text editor viewer with pertinent commands for that file type.

Any number of instances of SQL Editor can be active on the Workbench at the same time. Multiple instances of SQL Editor displaying different content may be active on the same Workbench. These instances will be stacked, by default, but can also be tiled side-by-side, so the content of various files can be view simultaneously for comparison or multi-tasking purposes. When an instance of SQL Editor is active, the Main Menu contains commands applicable to the file format. If a view is active, SQL Editor commands are disabled automatically, except when commands are still valid between the selected view and the file displayed in the interface.



```
SQL "Benson.sql"
CREATE TABLE dbo.benson
(
    job char(8) NOT NULL,
    sal numeric(38,0) NOT NULL,
    loc numeric(38,0) NOT NULL,
    CONSTRAINT pjob
    PRIMARY KEY CLUSTERED (job)
)
go
IF OBJECT_ID('dbo.benson') IS NOT NULL
    PRINT '<<< CREATED TABLE dbo.benson >>>'
ELSE
    PRINT '<<< FAILED CREATING TABLE dbo.benson >>>'
go
```

Among the commands SQL Editor supports via the right-click menu:

- **Revert File.** Automatically restores the working file to the original text as it appeared the last time the Save command was issued.
- **Shift Right/Shift Left.** Indents the line of code in the working file to the right or left, respectively.
- **Toggle Comments.** Hides or displays comments in the code of the working file, depending on the current hide/show state.
- **Add Block Comment/Remove Block Comment.** A block comment is used to insert a comment into SQL code that spans multiple lines and begins with a forward slash and asterisk. While block comments are typically used to insert a command that spans multiple lines, some developers find them more useful than line comments, especially if a development team is using different text editors on an individual basis. Moving code from one text editor to another often breaks line comments in the middle of a line and causes errors. Block comments can be broken without causing errors.



In addition to editing commands, some commands such as extract, drop, and execute can be accessed by right-clicking over statements in SQL code that are performed on specific tables, views, and columns. These commands will appear automatically in the appropriate menu when the code is highlighted. Full information on using these commands is found elsewhere in this documentation, based on the task each executable performs.

- **Explain Plan.** An explain plan details the steps that occur in SELECT, UPDATE, INSERT, and DELETE statements and is primarily used to determine the execution path followed by the database in its SQL execution.

See also:

- [Understanding Automatic Error Detection](#)
- [Understanding Code Assist](#)
- [Understanding Hyperlinks](#)
- [Understanding Code Formatting](#)
- [Understanding Code Folding](#)
- [Understanding Code Quality Checks](#)
- [Understanding SQL Templates](#)

Understanding automatic error detection

SQL Editor orders and classifies SQL statements. This enables it to edit code as you work within SQL Editor and highlight errors and typographical errors in "real time". As you work, SQL Editor examines each clause in a statement and provides error reporting and other features as required.

SQL Editor identifies the following clauses and elements:

- **SELECT.** Specifies the field, constants, and expressions to display in the query results.
- **FROM.** Specifies one or more tables containing the data that the query retrieves from.

- **WHERE.** Specifies join and filter conditions that determine the rows that query returns. Join operations in a WHERE clause function in the same manner as JOIN operations in a FROM clause.
- **GROUP BY.** Specifies one or more columns used to group rows returned by the query.

Columns referenced in the SQL SELECT statement list, except for aggregate expressions, must be included in the GROUP BY clause. You cannot group by Memo, General or Blob fields.

- **HAVING.** Specifies conditions that determine the groups included in the query. If the SQL statement does not contain aggregate functions, you can use the SQL SELECT statement containing a HAVING clause without the GROUP BY clause.
- **ORDER BY.** Specifies one or more items used to sort the final query result set and the order for sorting the results.

As you develop code in SQL Editor, it automatically detects semantic errors on a line-by-line basis. Whenever an error is detected, the line is flagged by an icon located in the left-hand column of the editor.

```

CREATE TABLE dbo.benson
(
    job CHAR (8) NOT NULL,
    sal NUMERIC (38, 0) NOT NULL,
    loc NUMERIC (38, 0) NOT NULL,
    CONSTRAINT pjob PRIMARY KEY CLUSTERED (job)
)

go
IF OBJECT_ID('dbo.benson') IS NOT NULL
    PRINT '<<< CREATED TABLE dbo.benson >>>'
ELSE
    PRINT '<<< FAILED CREATING TABLE dbo.benson >>>'

go
SELECT *
FROM dbo.benson;

```

Additionally, all semantic errors detected in SQL Editor are displayed in the Problems view.

Description	Resource	Path	Location
An unexpected token "<<< FAILED CR Benson.sql"	SQL Project 1	SQL Project 1	line 14
An unexpected token "" was found. Ex: File.sql	SQL Project 1	SQL Project 1	line 6
Table benson cannot be resolved on 'da Benson.sql	SQL Project 1	SQL Project 1	line 19

Right-click the error and select **Go To** in order to find the error. SQL Query Tuner opens and navigates to the specific line of code containing the specified error.

Automatic error detection functions, such as syntax checking and semantic validation are suspended if #define or #include directives are detected in an editor window. SQL Query Tuner does not perform #define/#include substitutions on execution.

Understanding code assist

When SQL Editor has finished analyzing a partial piece of code, it displays a list of data source objects for you to select from.

SQL Editor takes the following into consideration when analyzing code for a list of possible data source objects for insertion:

- Text to be inserted
- Original text to be replaced
- Content assist request location in original text
- The database object represented by the insertion text

Generally, insertion suggestions use the following format:

```
<insertion_text> - <qualification_information>
```

Code assist is available for SELECT, UPDATE, INSERT, and DELETE statements, as well as stored procedures, and functions (built-in and user defined.)

Additionally, code suggestions can be made for DML statements nested within DDL statements. This functions in the same manner as code assist for statements that are not nested, and applies to CREATE PROCEDURE, FUNCTION, TRIGGER, TABLE, and VIEW statements.

When the code assist window is open, you can filter out singular object suggestions by pressing (Ctrl + Spacebar). This removes all objects from the assist window while retaining procedures and functions. To display objects again, press (Ctrl + Spacebar) again.


The following table displays a list of all possible object suggestions, and the format in which SQL Editor inserts the suggestions into a statement:

Object and stored procedure suggestions

Object Suggestion	Syntax/Example
Table	(TABLE) [catalog].[schema] EMPLOYEE - (TABLE)HR
Alias Table	(TABLE ALIAS) [catalog].[schema]tableName EMPLOYEE - (TABLE ALIAS)HRJOBS
Column	datatype - (Column) [catalog].[schema].tableName JOB_TITLE:varchar(20) - (Column)HRJOBS
Alias Column	datatype - (COLUMN ALIAS) [catalog].[schema].tableName.columnName JOB_TITLE:int-(COLUMN ALIAS)HR.JOBS.JOB_ID
Schema	(SCHEMA) [catalog] dbo-(SCHEMA)NorthWind
Catalog	(CATALOG)
Call	Call HR.ADD_JOB_HISTORY

Function suggestions

Function Suggestion	Syntax/Example
Built-in	SELECT A FROM HR.DEPARTMENTS WHERE HR.DEPARTMENTS AVG
User-Defined	SELECT + FROM HR.CLIENTS WHERE HR.F_PERSONAL

 Function suggestions are only available for Oracle.

SQL Editor detects incomplete or erroneous code, processes the code fragments, and then attempts to apply the appropriate logic to populate the code.

As code is typed into SQL Editor, the application 'reads' the language and returns suggestions based on full or partial syntax input.

Depending on the exact nature of the code, the automatic object suggestion feature behaves differently; this enables SQL Editor to provide reasonable and 'intelligent' suggestions on coding.

Additionally, semantic validations can be made for DML statements nestled within DDL statements. This functions in the same manner as validation for top-level statements, and applies to CREATE PROCEDURE, FUNCTION, TRIGGER, TABLE, and VIEW statements.

The following chart displays the possible statement fragments that SQL Editor will attempt to suggest/populate with objects:

Statement Fragment Elements	Object Suggestion Behavior
SELECT	A list of tables, when selected automatically, prompts the user to select a column.
UPDATE and DELETE	A list of tables appears in the FROM and/or WHERE clause.
INSERT	A list of tables and views appears in the INSERT INTO and OPEN BRACKET clause prior to values. A list of columns based on the table or view name appears in the OPEN BRACKET or VALUES clause.

In addition to DML statements, SQL Editor also suggests objects based on specific fragmented syntax per line of code:


Statement Syntax	Object Suggestion Behavior
A partial DML statement (for example SEL ... indicates a fragment of the SELECT clause)	<p>The keyword is completed automatically, assuming SQL Editor can match it. Otherwise, a list of suggested keywords is displayed.</p> <p>If the preceding character is a period, and the word prior is a table or view, a list of columns appears.</p> <p>If the word being typed is a part of a table name (denoted by a schema in front of it) the table name is autocompleted.</p> <p>If the word being typed has a part of a column name (denoted by a table in front of it) the column name is autocompleted.</p>
Without typing anything.	A list of keywords appears.
A period is typed.	<p>If the word prior to the period is a name of a table or view, a list of columns is displayed.</p> <p>If the word prior to the period is a schema name, a list of table names is displayed.</p> <p>If the word prior to the period is either a table name or a schema name, then both a list of columns and a list of table names is displayed</p>

To activate code suggestions:

By default, code suggestions are automatically offered if you stop typing in SQL Editor for one second. You can turn off the automated suggestion feature on the Code Assist preferences page.

If automated code suggestion is disabled, you can still access the suggestion window using the following method:

1. Click the line that you want SQL Editor to suggest an object for.
2. Press (**CTRL + Spacebar**) on your keyboard. SQL Editor 'reads' the line and presents a list of tables, views or columns as appropriate based on statement elements.

 On a per platform basis, auto-suggestion behavior may vary.

To modify object suggestion parameters, including setting it from automatic to manual, see [Specify Code Assist Preferences](#).

You can speed up the performance of the code assist functionality by enabling data source indexing either when you connect to the data source, see [Working with Data Sources](#) or on the Preferences page, see [Specify Data Source Indexing Preferences](#).


Understanding hyperlinks

SQL Editor supports hyperlinks that are activated when a user hovers their mouse over a word and presses the CTRL key. If a hyperlink can be created, it becomes underlined and changes color. When the hyperlink is selected, the creation script for the hyperlink object is opened in a new editor.

Hyperlinks can be used to link to tables, columns, packages, and other reference objects in development code. Additionally, hovering over a hyperlink on a procedure or function of a call statement will open it. You can also use the hyperlink feature on function calls in DML statements.

Clicking a hyperlink performs an action. The text editor provides a default hyperlink capability. It allows a user to click on a URL (for example, <https://www.idera.com>) and database object links.

Hyperlink options (look and feel) can be modified via the Hyperlinking subnode in the Editors > Text Editors node of the Preferences panel.

 Hyperlink functionality relies on certain objects being captured in the Object Index. If the index is turned off, or has been restricted in what information it captures, users will be unable to link them (as they are non-existent within the Index.) To specify object index types, see [Specify Data Source Indexing Preferences](#).

Understanding code formatting

Code formatting provides automatic code formatting in SQL Editor while you are developing code.

To access the code formatter, select the open editor you want to format and select Ctrl+Shift+F. The code is formatted automatically based on formatting parameters specified in the Code Formatter subnode of the SQL Editor node in the Preferences panel.

You can also format an entire group of files from Project Explorer. To do so, select the directory or file and execute the Format command via the shortcut menu. The files will be formatted automatically based on your formatting preferences. See [Specify Code Formatter Preferences](#) for more information.

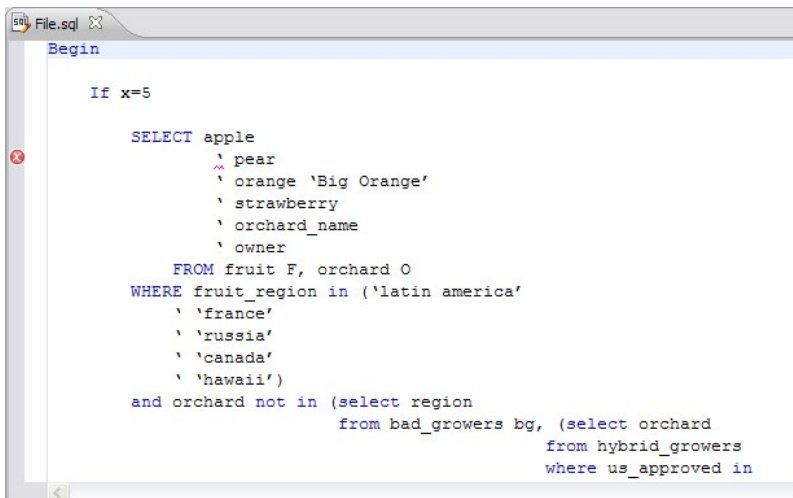
The following examples display a list of code formatting parameters and the resultant output in SQL Editor, based on the same set of SQL statements.

Custom code formatting example 1

The following chart indicates a list of custom code formatting parameters and their corresponding values. The chart is followed by the actual syntax as it would appear in SQL Editor, based on the formatting parameter values.

Compare the parameters and formatted code in Example 2 with this example for a concept of how custom formatting works.

Custom Code Formatting Parameter	Value (if applicable)
Stack commas separated by lists?	Yes
Stack Lists with ___ or more items.	3
Indent Size?	2
Preceding commas?	Yes
Spaces after comma?	1
Trailing commas?	–
Spaces before comma?	–
Right align FROM and WHERE clauses with SELECT statement?	Yes
Align initial values for FROM and WHERE clauses with SELECT list?	Yes
Place SQL keywords on their own line?	No
Indent size?	–
Indent batch blocks?	Yes
Number of new lines to insert	1
Indent Size	5
Right Margin?	80
Stacked parentheses when they contain multiple items?	No
Stacked parentheses when the list contains ___ or more items.	–
Indent size?	5
New line after first parentheses?	No
Indent content of conditional and looping constructs?	Yes
Number of new lines to insert?	1
Indent size?	5



```
Begin
If x=5
    SELECT apple
        pear
        orange 'Big Orange'
        strawberry
        orchard_name
        owner
    FROM fruit F, orchard O
    WHERE fruit_region in ('latin america'
        'france'
        'russia'
        'canada'
        'hawaii')
    and orchard not in (select region
                        from bad_growers bg, (select orchard
                                              from hybrid_growers
                                              where us_approved in
```

Custom code formatting example 2

The following chart indicates a list of custom code formatting parameters and corresponding values. The chart is followed by the actual syntax as it would appear in SQL Editor based on the formatting parameter values. Compare the parameters and formatted code in Example 1 with this example for a concept of how custom formatting works.

Custom Code Formatting Parameter	Value (if applicable)
Stack commas separated by lists?	Yes
Stack Lists with __ or more items.	2
Indent Size?	0
Preceding commas?	--
Spaces after comma?	Yes
Trailing commas?	Yes
Spaces before comma?	2
Right align FROM and WHERE clauses with SELECT statement?	No
Align initial values for FROM and WHERE clauses with SELECT list?	--
Place SQL keywords on their own line?	Yes
Indent size?	4
Indent batch blocks?	No
Number of new lines to insert	1
Indent Size	5
Right Margin?	80
Stacked parentheses when they contain multiple items?	Yes
Stacked parentheses when the list contains __ or more items.	2
Indent size?	2
New line after first parentheses?	Yes
Indent content of conditional and looping constructs?	--
Number of new lines to insert?	1
Indent size?	5

```
B'l=ile.sql

Begin

If x=S

    SELECT

        apple ,
        pear ,
        orange Big Orange' ,
        strawberry ,
        orchard name ,
        owner

    FROM

        fruit F ,
        orchard 0

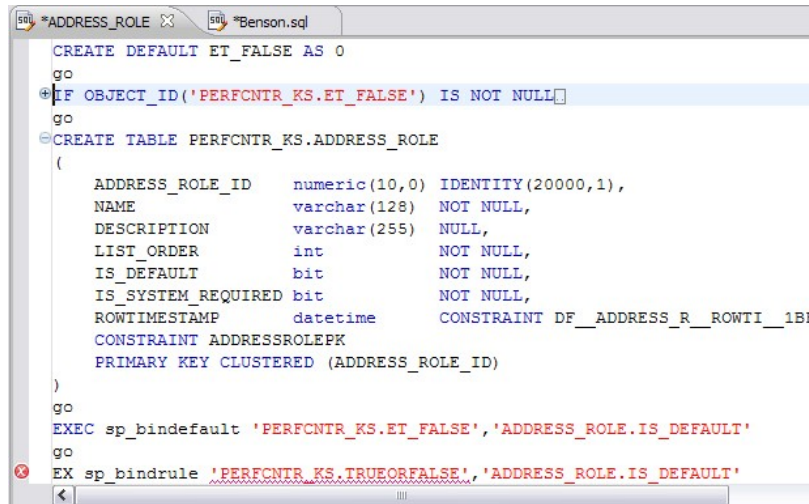
    WHERE

        fruit_region in

            'latin america' ,
            'france' ,
            'russia' ,
            'canada' ,
```

Understanding code folding

SQL Editor features code folding that automatically sorts code into an outline-like structure within the editor window for easy navigation and clarity while developing code.




The editor window automatically inserts collapsible nodes in the appropriate lines of code for organizational purposes. This enables you to expand and collapse statements, as needed, while developing code in particularly large or complicated files.

Understanding code quality checks

Code quality markers provide annotations that prevent and fix common mistakes in the code. These notes appear in a window on any line of code where the editor detects an error, and are activated by clicking the light bulb icon in the margin or by pressing Ctrl + I.

For example, if a statement reads select * from SCOTT.EMP, SCOTT.DEPT, when you click the light bulb icon or press Ctrl + I, a window appears beneath the line of code that suggests Add join criteria. When you click on a proposed fix, the statement is automatically updated to reflect your change.

The following common errors are detected by the code quality check function in the editor:

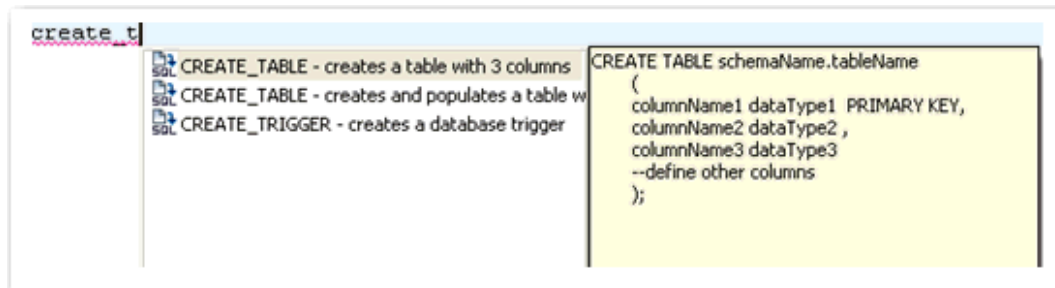
Code Quality Check Type	Definition
Statement is missing valid JOIN criteria	<p>If a SELECT statement contains missing join criteria, when it is executed, it can produce a Cartesian product between the rows in the referenced tables. This can be problematic because the statement will return a large number of rows without returning the proper results.</p> <p>The code quality check detects missing join criteria between tables in a statement and suggests join conditions based on existing foreign keys, indexes, and column name/type compatibility.</p> <p>Example:</p> <p>The following statement is missing a valid JOIN criteria:</p> <pre>SELECT*FROM employeee,customer,c,sales_orders WHERE e.employee_id = c.salesperson_id</pre> <p>The code quality check fixes the above statement by adding an AND clause:</p> <pre>SELECT*FROM employeee,customer,c,sales_orders WHERE e.employee_id = c.salesperson_id AND s.customer_id = c.customer_id</pre> <div> This code quality check is valid for Oracle join conditions.</div>
Invalid or missing outer join operator	<p>When an invalid outer join operator exists in a SELECT statement, (or the outer join operator is missing altogether), the statement can return incorrect results.</p> <p>The code quality check detects invalid or missing join operators in the code and suggests fixes with regards to using the proper join operators.</p> <p>Example:</p> <p>The following statement is missing an outer join operator:</p>

	<p>SELECT * FROM employee e, customer c WHERE e.employee_id=c.salesperson_id(+)ANDc.state='CA'</p> <p>The code quality check fixes the above statement by providing the missing outer join operator to the statement:</p> <p>SELECT * FROM employee e,customer c WHERE e.employee_id = c.salesperson_id(+) AND c.state(+) = 'CA'</p>
Transitivity issues	<p>The performance of statements can sometimes be improved by adding join criteria, even if a join is fully defined. If this alternate join criterion is missing in a statement, it can restrict the selection of an index in Oracle's optimizer and cause performance problems.</p> <p>The code quality check detects possible join conditions by analyzing the existing conditions in a statement and calculating the missing, alternative join criteria.</p> <p>Example:</p> <p>The following statement contains a transitivity issue with an index problem:</p> <p>SELECT * FROM item i, product p, price pr WHERE i.product_id = p.product_id AND p.product_id = pr.product_id</p> <p>The code quality check fixes the above statement with a transitivity issue by adding the missing join condition:</p> <p>SELECT * FROM item i, product p, price pr WHERE i.product_id = p.product_id AND p.product_id = pr.product_id AND i.product_id = pr.product_id\</p>
Nested query in WHERE clause	<p>It is considered bad format to place sub-queries in the WHERE clause of a statement, and such clauses can typically be corrected by moving the sub- query to the FROM clause instead, which preserves the meaning of the statement while providing more efficient code.</p> <p>The code quality check fixes the placement of sub-queries in a statement, which can affect performance. It detects the possibility of moving sub- queries from the FROM clause of the statement.</p> <p>Example:</p> <p>The following statement contains a sub-query that contains an incorrect placement of a WHERE statement:</p> <p>SELECT*FROM employee WHERE employee_id=(SELECT MAX(salary) FROM employee)</p> <p>The code quality check fixes the above statement by correcting the sub- query issue:</p> <p>SELECT employee.* FROM employee (SELECT DISTINCT MAX(salary) col1 FROM employee) t1 WHERE employee_id = t1.col1</p>
Wrong place for conditions in a HAVING clause	<p>When utilizing the HAVING clause in a statement:</p> <p>It is recommended to include as few conditions as possible while utilizing the HAVING clause in a statement. SQL Query Tuner detects all conditions in a given HAVING statement and suggests equivalent expressions that can benefit from existing indexes.</p> <p>Example:</p> <p>The following statement contains a HAVING clause that is in the wrong place:</p> <p>SELECT col_a, SUM(col_b) FROM table_a GROUP BY col_a HAVING col_a > 100</p> <p>The code check fixes the above statement by replacing the HAVING clause with equivalent expressions:</p> <p>SELECT col_a, SUM(col_b) FROM table_a WHERE col_a > 100 GROUP BY col_a</p>
Index suppressed by a function or an arithmetic operator	<p>In a SELECT statement, if an arithmetic operator is used on an indexed column in the WHERE clause, the operator can suppress the index and result in a FULL TABLE SCAN that can hinder performance.</p> <p>The code quality check detects these conditions and suggests equivalent expressions that benefit from existing indexes.</p> <p>Example:</p> <p>The following statement includes an indexed column as part of an arithmetic operator:</p> <p>SELECT * FROM employee WHERE 1 = employee_id - 5</p> <p>The code quality check fixes the above statement by reconstructing the WHERE clause:</p> <p>SELECT * FROM employee WHERE 6 = employee_id</p>

<p>Mismatched or incompatible column types</p>	<p>When the data types of join or parameter declaration columns are mismatched, the optimizer is limited in its ability to consider all indexes. This can cause a query to be less efficient as the system might select the wrong index or perform a table scan, which affects performance.</p> <p>The code quality check flags mismatched or incompatible column types and warns that it is not valid code.</p> <p>Example:</p> <p>Consider the following statement if Table A contains the column col int and Table B contains the column col 2 varchar(3):</p> <pre>SELECT * FROM a, b WHERE a.col = b.col;</pre> <p>In the above scenario, the code quality check flags the 'a.col = b.col' part of the statement and warns that it is not valid code.</p>
<p>Null column comparison</p>	<p>When comparing a column with NULL, the !=NULL condition may return a result that is different from the intended command, because col=NULL will always return a result of false. Instead, the NULL/IS NOT NULL operators should be used in its place.</p> <p>The code quality check flags occurrences of the !=NULL condition and replaces them with the IS NULL operator.</p> <p>Example:</p> <p>The following statement includes an incorrect col = NULL expression:</p> <pre>SELECT * FROM employee WHERE manager_id = NULL</pre> <p>The code quality check replaces the incorrect expression with an IS NULL clause:</p> <pre>SELECT * FROM employee WHERE manager_id IS NULL</pre>

Understanding SQL templates

SQL Query Tuner provides code templates for DML and DDL statements that can be applied to the Editor via the (Ctrl + Spacebar) command. When you choose a template from the menu that appears, SQL Editor automatically inserts a block of code with placeholder symbols that you can modify to customize the code for your own purposes.



Code templates are available for DML, ALTER, DROP, CREATE, and platform specific commands.

A comprehensive set of DDL/DML templates are available, with statement alternatives varying by DBMS and specific DBMS versions. You can modify and create new templates via the SQL Templates panel on the Preferences dialog. See for more information on how to create and alter SQL code templates.