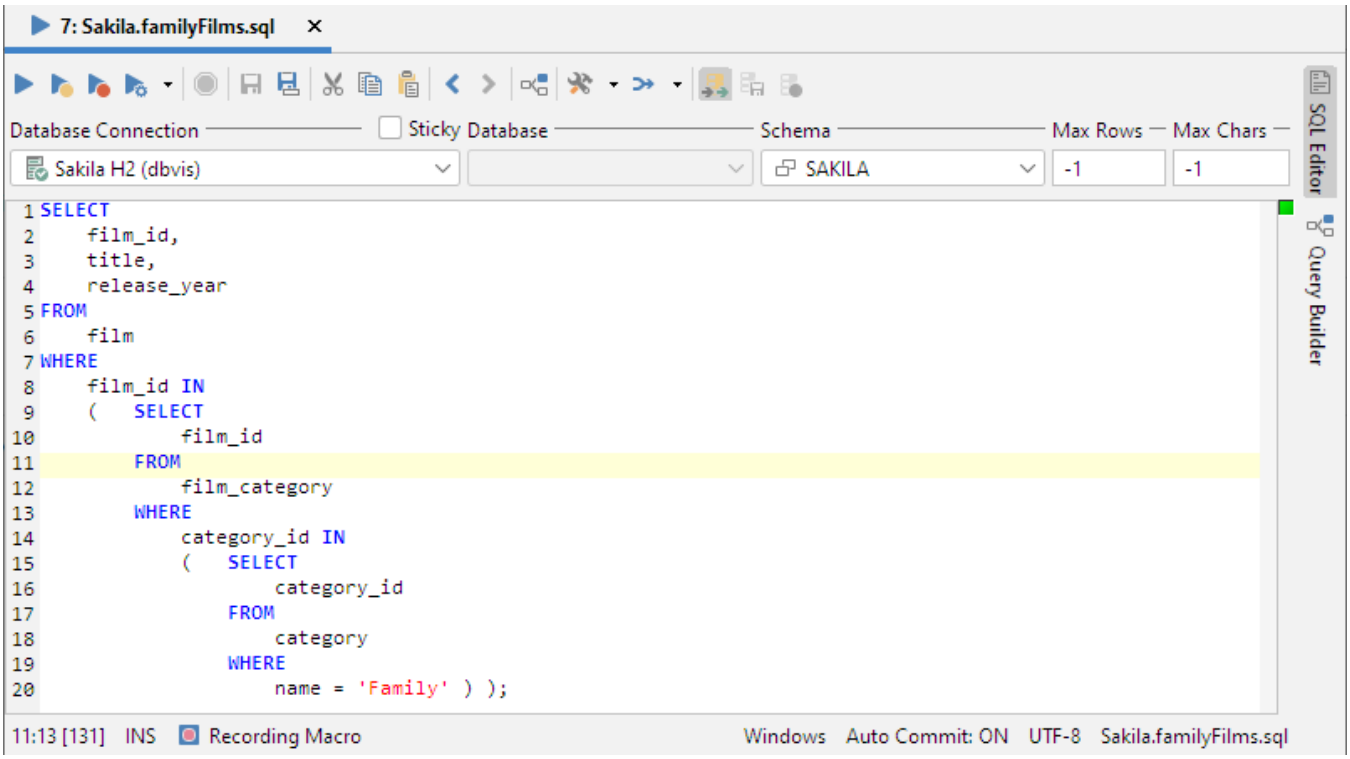
# Editing SQL Scripts

The SQL Commander contains an SQL editor, used to edit SQL scripts.



The editor area looks like this:




```
1 SELECT
2     film_id,
3     title,
4     release_year
5 FROM
6     film
7 WHERE
8     film_id IN
9     (   SELECT
10            film_id
11        FROM
12            film_category
13        WHERE
14            category_id IN
15            (   SELECT
16                    category_id
17                FROM
18                    category
19                WHERE
20                    name = 'Family' ) );
```

Above the editor is a toolbar with buttons related both to execution of scripts and to editing. The editing related buttons are covered below.

The left margin shows the line numbers.

Below the editor, you see a Status Bar with the following fields, from left to right:

1. Position:
   the current caret position in the format: `<line>:<column> [<position from top>]`
   The last figure, within square brackets, is the caret position from the top. This can be useful when you get an error message executing a script that contains this information rather than a line/column location.
2. Insert/Overwrite Mode:
   **INS** if characters you type will be inserted at the caret position or **OVR** if they will overwrite the current text at the caret position. You can toggle this mode using the **Toggle Typing Mode** keyboard shortcut, by default bound to the **Insert** key.
3. Macros:
   this field is only visible when working with macros, as described in the Recording and Playing Edit Macros section.

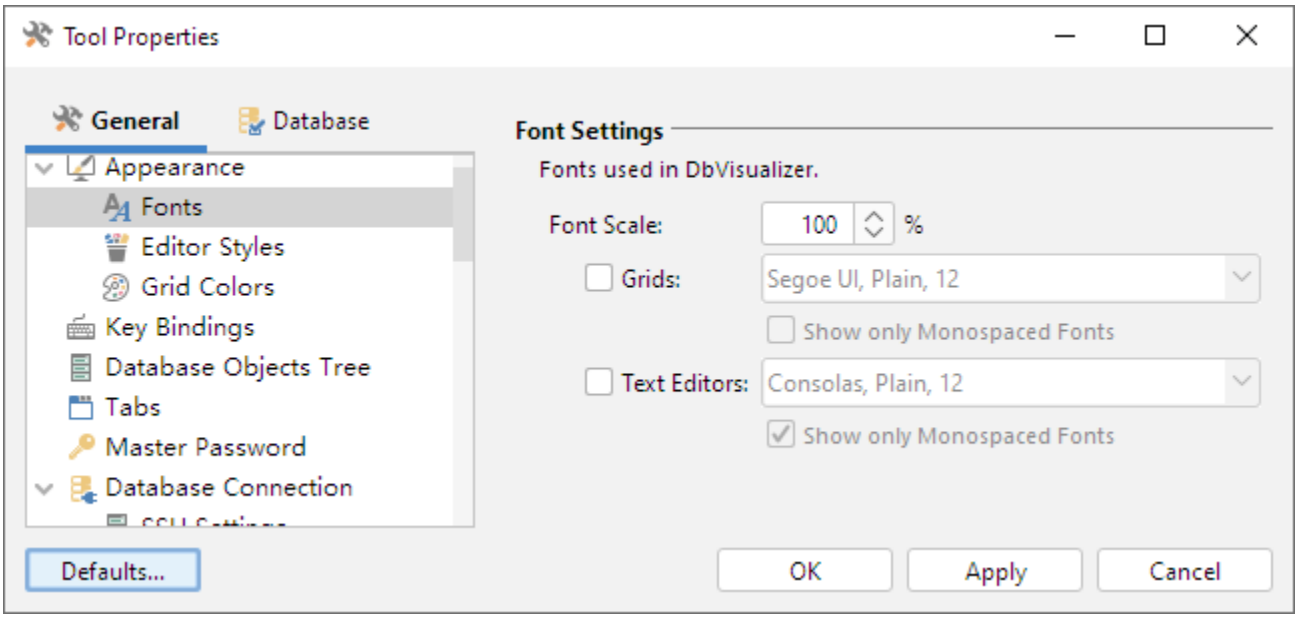
4. File Format:
   the format of the file as detected when the file was loaded (if any). Click it to select which format to use when saving the file; **Windows**, **Unix /Linux/macOS**, or **Old MacOS**.
5. Auto Commit Status:
   shows whether or not Auto Commit is enabled.
6. Character Set:
   the character encoding as detected when the file was loaded (if any). Click it to select which encoding to use when saving the file.
7. File:
   The name of the loaded file (if any). You can click on the filename to copy the file path or open the OS file chooser for the directory holding the file. If you just type into the editor without loading a file, the filename "Untitled" is shown instead. An asterisk (*****) after the filename indicates that there are unsaved edits.

The SQL Editor is like any editor you're used to when it comes to typing, scrolling etc. But it also offers additional features to help you specifically with editing SQL scripts. These are described in the following sections.

You can change how to display in the **Tools->Tool Properties** dialog, in the **General / Appearance** category. This is explained in more detailed below.

## Font Settings

In the **Appearance/Fonts** category, you can select the font for **Text Editors** to control the font in the SQL Editor (**Monospaced Fonts** are usually a good choice for code editors).

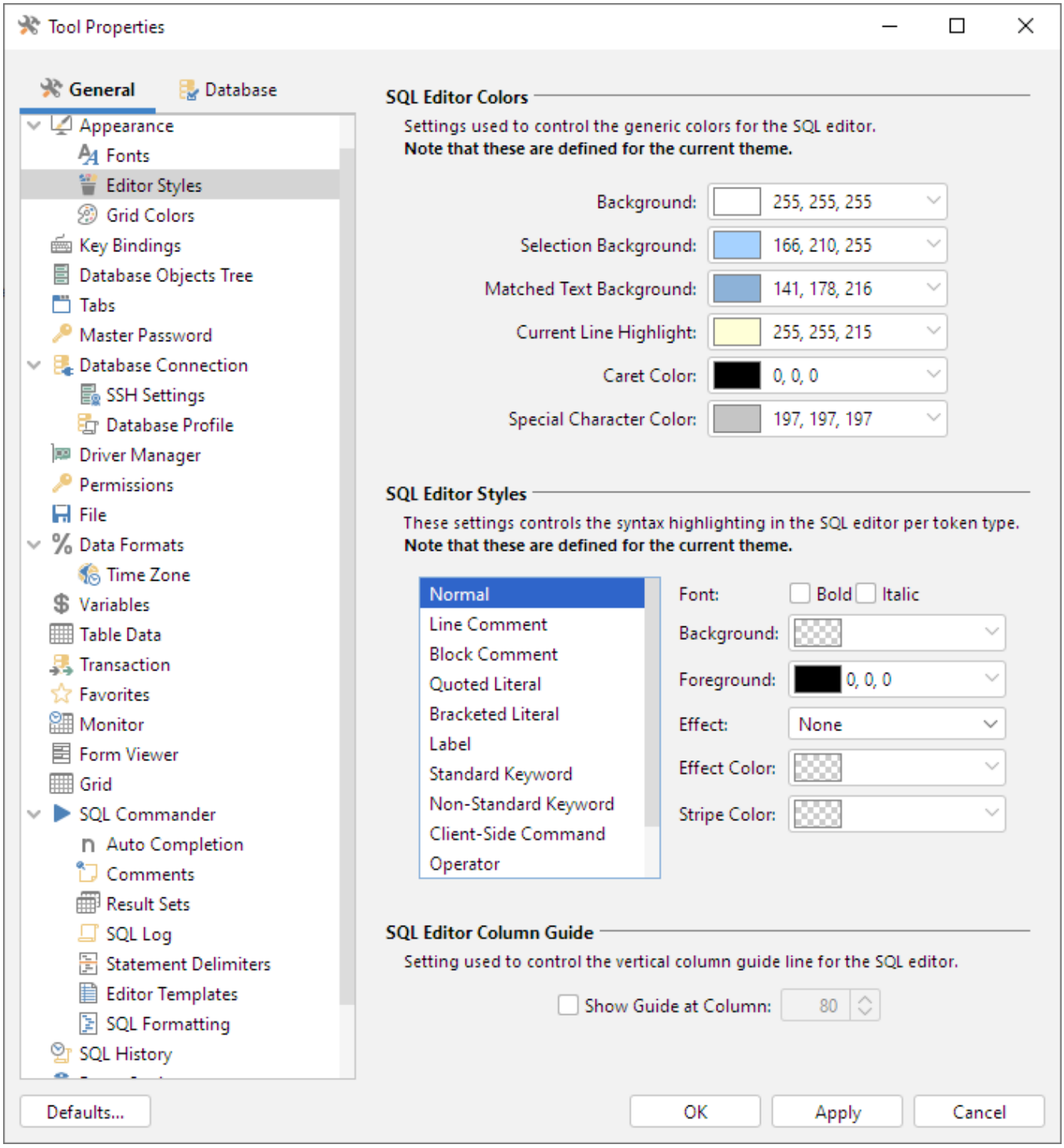



## Editor Styles

An SQL script consists of keywords, operators, object identifiers, quoted text, etc. It may also contain comments. To make it easier to see at a glance what is what, the SQL Editor displays words using different font styles depending on their classification. For instance, keywords are displayed with a bold blue font, while quoted text is displayed with a regular type red font.

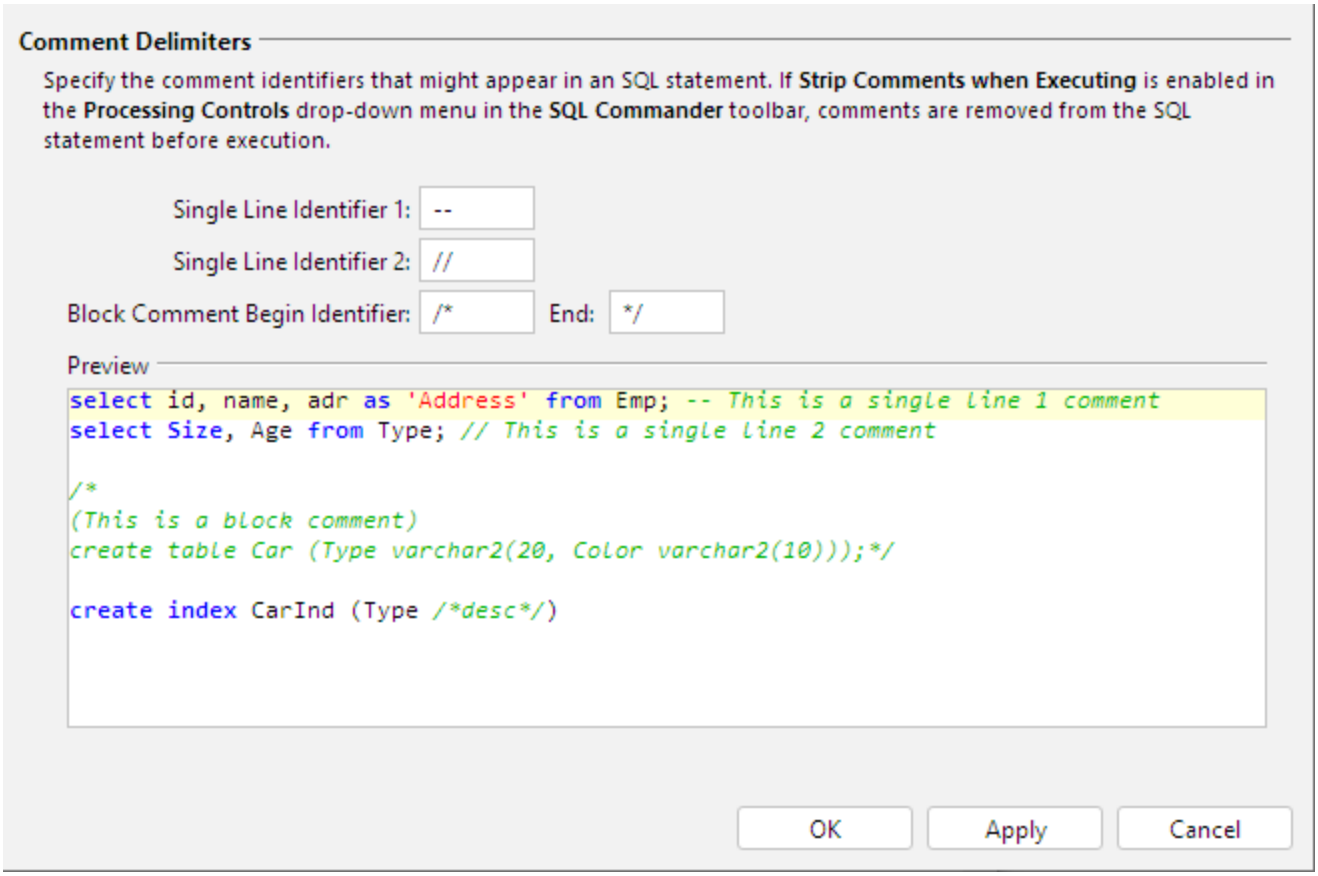
In the **Appearance/Editor Styles** category you can select colors for the different kinds of words, as well as the editor selection background color, the current line highlight color and the editor background color, and more.

The **SQL Editor Column Guide** is an optional visual guide that appears as a thin vertical line at specified column (this works best if you use monospaced fonts).



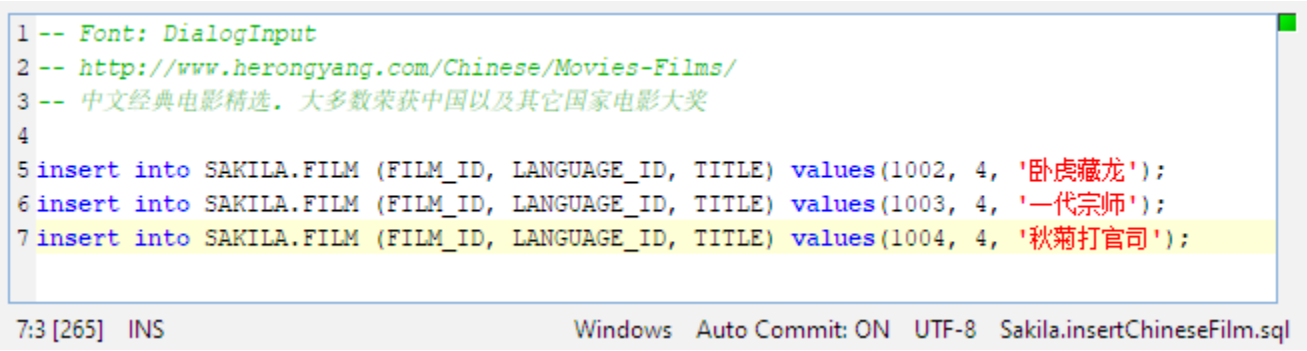

## Comments

The editor uses the Tool Properties settings from the **SQL Commander/Comments** category under the General tab to detect comments.




## Charsets and Fonts

You can also change the SQL Editor font family, which is useful and necessary in order to display characters for languages like Chinese, Japanese, etc., in **Tool Properties** in the **Appearance/Fonts** category to set the font for the SQL Editor (see Internationalization and Localization (i18N and L10N) for more information).



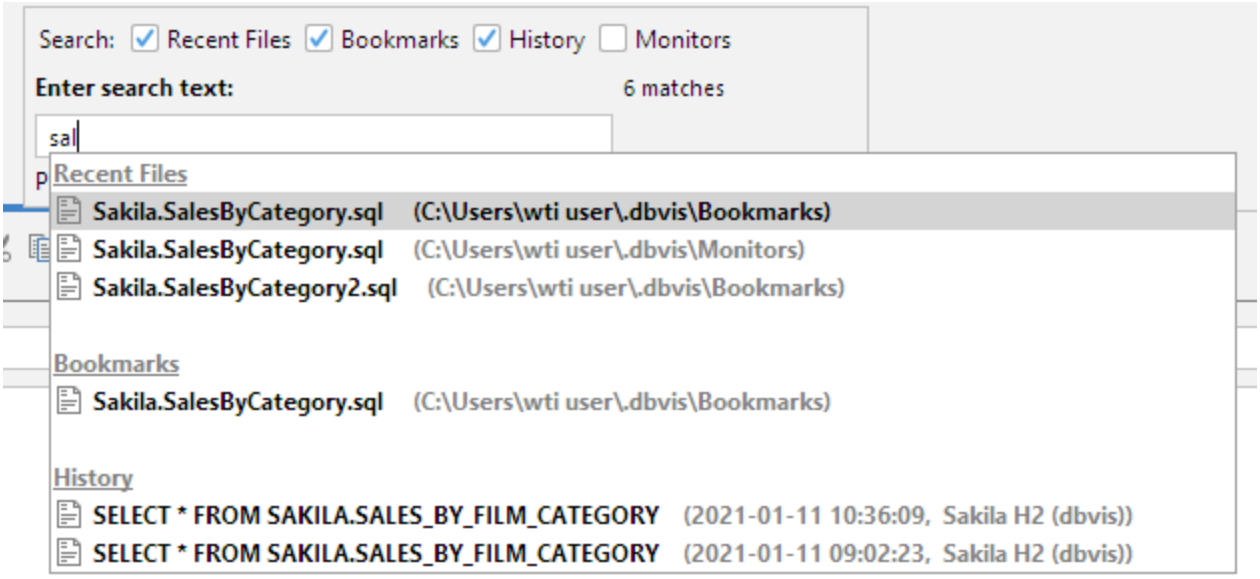

## Loading and Saving Scripts

The SQL editor supports loading statements from a file and saving the content of the editor to a file. Use the standard file operations, **Open**, **Save** and **Save As** in the **File** main menu or the main toolbar to accomplish this. Loading a file loads it into a new **SQL Commander** tab or activates the tab that already holds it.

The name of the loaded file is listed in the status bar of the editor, with the full file path shown in the window title. The editor tracks any modifications and indicates changes with an asterisk (*) after the filename. When you close the SQL Commander tab or exit DbVisualizer, you are asked what to do if there are any pending edits that need to be saved.

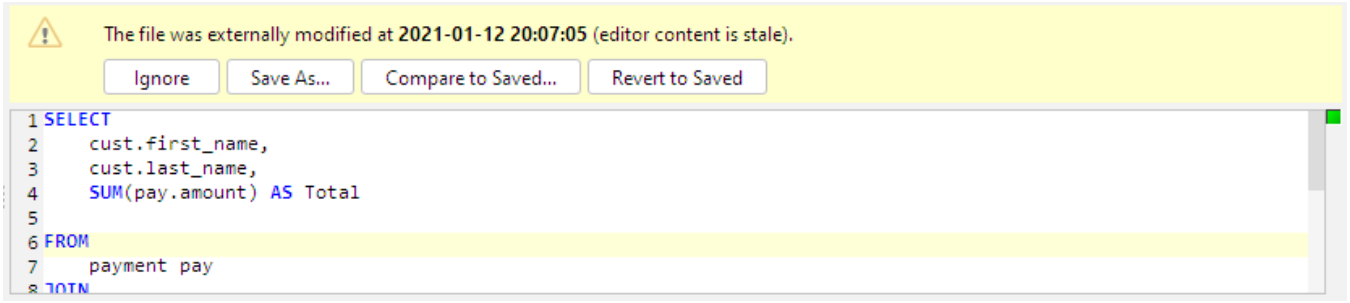
The **File->Open Recent** submenu lists the recently loaded files. How many recent files to keep track of can be specified in the Tool Properties dialog, in the **SQL Commander** category under then General tab.

You can also use the **Quick File Open** feature to open recent files as well as Bookmarks and History entries. By default, it is bound to the **Ctrl+Alt+O** key combination, and is also available via a main toolbar button as well as in the main **File->Quick File Open** menu.

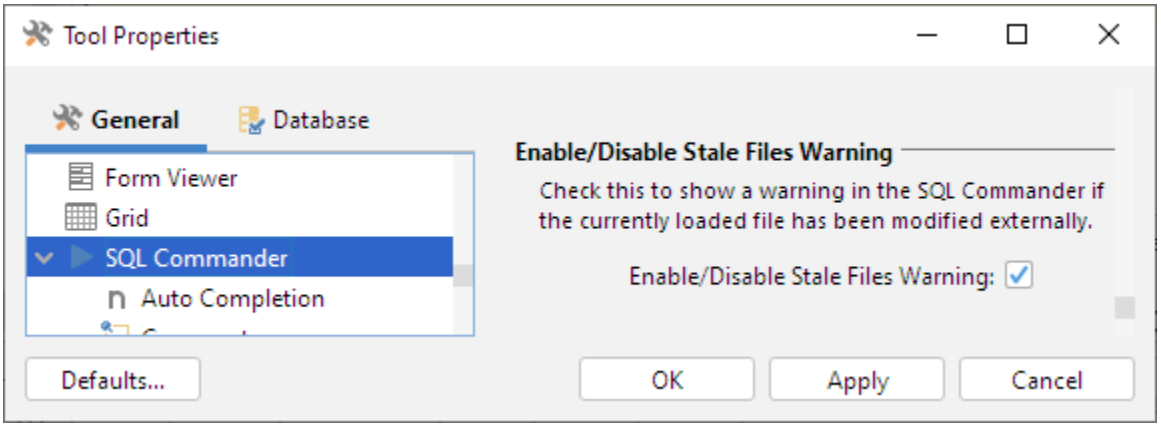



## Stale Files Warning

**SQL Commander** monitors open files to detect external changes, for example if an open file is modified by an external editor, reloaded from a network connection or repository, etc. If a file is externally modified, you will see a warning ribbon at the top of the editor area, presenting options to handle the situation:




You can turn off this control in **Tool Properties/SQL Commander:**

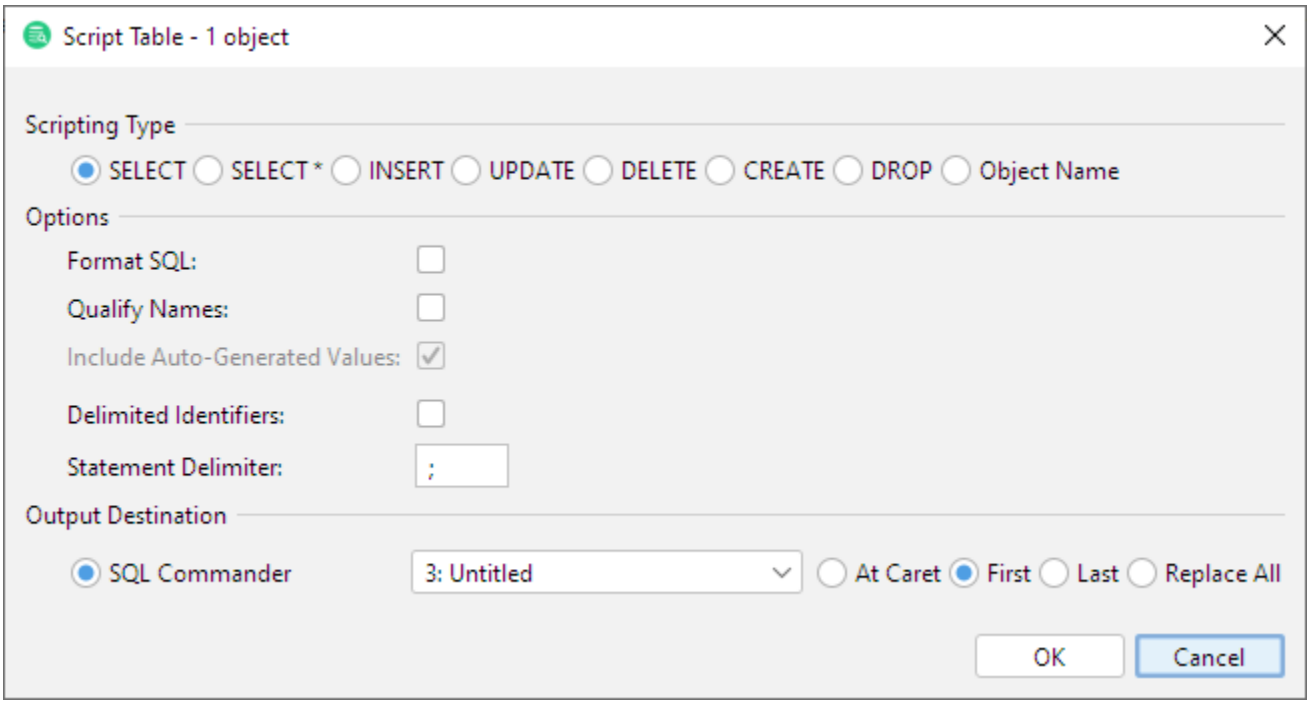



## Drag and Drop a File

You can also select a file in the platform's file browser and drop it somewhere in the DbVisualizer window. If you drop it in an editor, the file content is inserted at the caret position in the editor. If you instead drop it in the toolbar area, the file is opened in a new **SQL Commander** tab.

## Drag and Drop Database Objects

If you want to include an object shown in the database objects tree, you can select the node and drop it in the editor where you want it inserted. The **Script Object** dialog is shown where you can select exactly what you want to insert in the editor.




First of all, you can select to insert an SQL statement based on the dropped object, e.g. a SELECT statement or a CREATE statement. You can also choose to just insert the object name. The choices available depends on the type of object you drop.

In the **Options** area, you can opt to format the SQL before it is inserted and use qualifiers and quoted identifiers, and even change which statement delimiter to use.

The **Output Destination** is set to the SQL Commander tab you dropped the object on by default, but you can change your mind and pick another destination. If you stick with an **SQL Commander** as the destination, you can tell where in the editor to insert the text.

You can also open this dialog from the **Databases** tab, from the object's right-click menu.

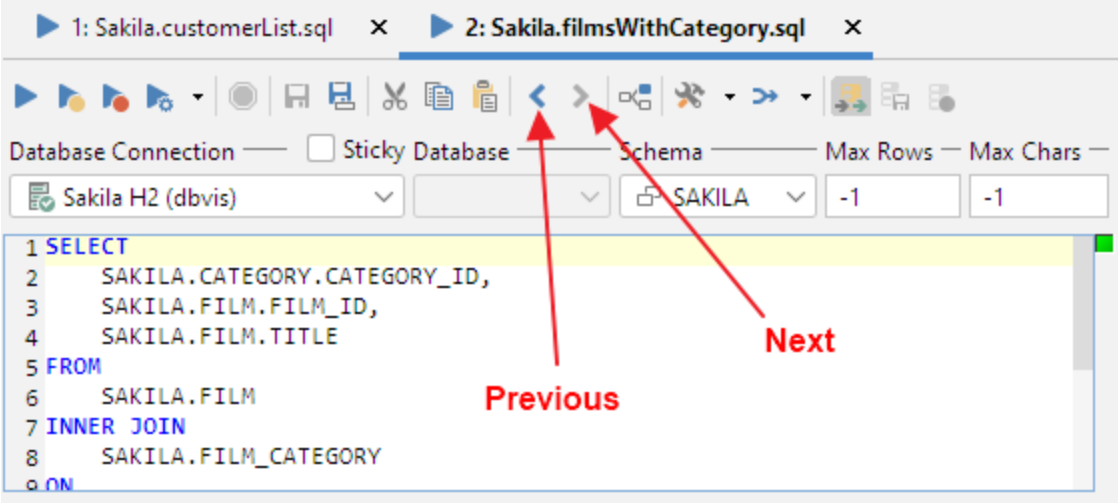
If you just want to insert the object names in the editor, hold down the **Ctrl** key (or the **Alt** key on macOS) while dragging and dropping. This behavior can be reversed in **Tool Properties**, in the **SQL Commander** category, so that dropping without pressing a key inserts the names and pressing the key launches the dialog.

## Loading and Saving Bookmarks and Monitors

Bookmarks and Monitors are also files, but with special meaning. See the Managing Frequently Used SQL for how to create and edit them in the SQL Editor.

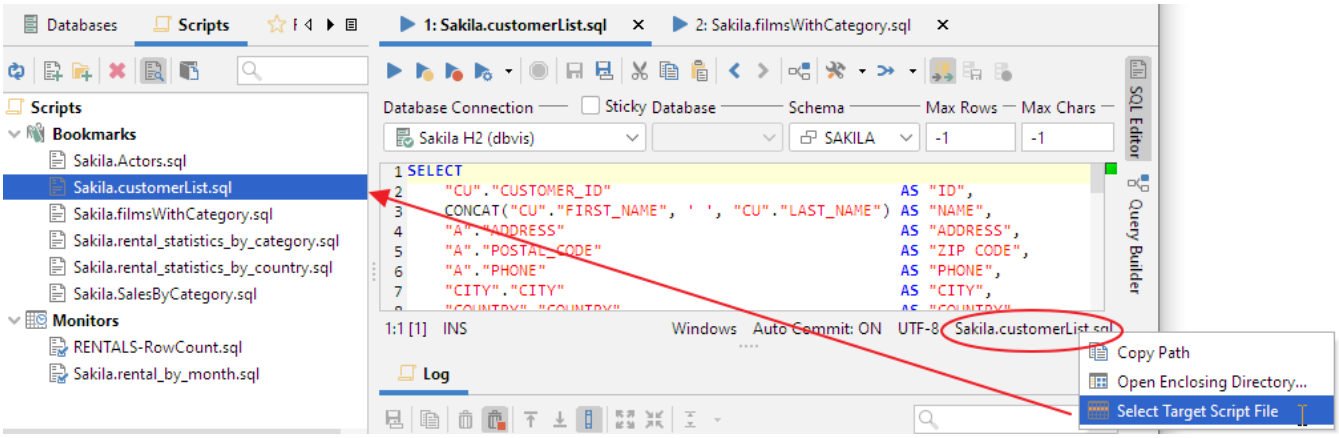
## Navigating Between History Entries

When you execute a script, DbVisualizer saves it as a history entry, see the Re-Executing SQL Statements section for details. You can use the **Previous** a nd **Next** buttons in the editor toolbar to navigate between (load) these entries.




## Navigating to Script location

By selecting the script filename in the right bottom corner it is possible to copy its path or locate it in the file system or in the **Scripts** tab.



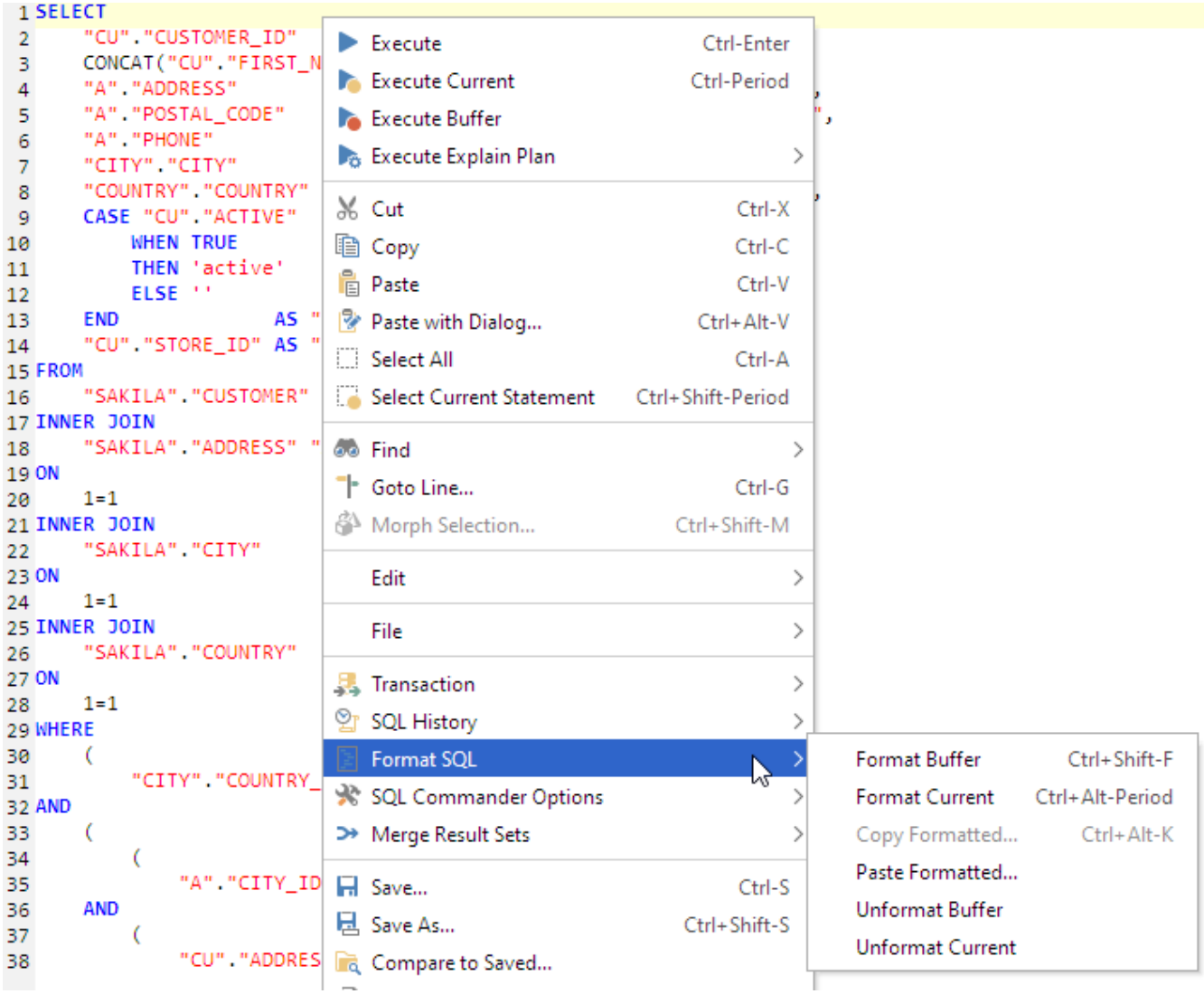

## Confirming Overwriting Unsaved Changes

By default, you have to confirm overwriting unsaved changes in an editor, e.g. when navigating between history entries, and when closing an SQL Commander tab with unsaved edits. You can disable these confirmation popups in the Tool Properties dialog, under the **SQL Commander** category under the General tab.

## SQL Formatting

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.
The **SQL Commander** main menu on the (or right-click in the editor) and its **Format SQL** sub menu contains operations for formatting SQL statements.




- **Format Buffer** and **Format Current** formats the complete editor content or the current SQL (at cursor position) respectively.
- **Copy Formatted** and **Paste Formatted** are powerful tools for copying SQL statements between programs written in languages like Java, C#, PHP, VB, etc. and the SQL Editor. Both operations display a dialog where you can adjust some of the formatting options, most importantly the **Target SQL** option and the **SQL is Between** option. **Target SQL** can be set to a number of common programming language formats.
- **Unformat Buffer** and **Unformat Current** produces compact statements by removing unnecessary whitespace.

**Example:**

To copy an SQL statement and paste it as Java code for adding it to a Java StringBuffer:

1. Select the statement. Example:
   `SELECT * FROM SAKILA.STAFF)`
2. Choose **SQL->Format SQL->Copy Formatted**,
3. Set **Target SQL** to Java StringBuffer,
4. Click **Format** to place the formatted statement on the system clipboard,
5. Paste it into your Java code. Example:

```
StringBuffer sql = new StringBuffer();
sql.append("SELECT ");
sql.append(" * ");
sql.append("FROM ");
sql.append(" SAKILA.STAFF");
```

To copy a statement wrapped in code from a program:

1. Select the code containing an SQL statement in your program,
2. Copy it to the system clipboard,
3. Choose **SQL->Format SQL->Paste Formatted**,
4. Check **SQL is Between** and enter the character enclosing the SQL statement in the code,
5. Click **Format** to extract the SQL statement and paste the formatted SQL in the editor.

## Settings

All formatting is done according to the settings defined in the **Tool Properties** dialog, in the **SQL Commander/SQL Formatting** category under the General tab.

There are many things you can configure; use the  default example or your own SQL to check the effect of the settings. After making some changes, press **Apply** and format again to see the result.

**Example:**

```
-- Basic SELECT example, with Sub-SELECT and JOIN
SELECT e.LAST_NAME AS "Last Name", e.FIRST_NAME AS "First Name", d.DEPARTMENT_NAME AS "Department", e.SALARY AS
"Salary", e.SALARY + e.SALARY * e.COMMISSION_PCT, e.COMMISSION_PCT * 100 || '%', ROUND(e.SALARY / ( SELECT MAX
(SALARY) FROM HR.EMPLOYEES), 2) * 100 AS "Percentage of Max" FROM HR.EMPLOYEES e INNER JOIN HR.DEPARTMENTS d ON
( e.DEPARTMENT_ID = d.DEPARTMENT_ID) WHERE d.DEPARTMENT_ID IN (10, 20, 90, 210) AND e.SALARY > 3000;
-- CASE example
SELECT FIRST_NAME, LAST_NAME, SALARY, CASE WHEN SALARY > 10000 THEN 'High' WHEN SALARY BETWEEN 5000 AND 999
THEN 'Midlevel' ELSE 'Low' END AS "Income Level", CASE DEPARTMENT_ID WHEN 40 THEN 'Administration' WHEN 20 THEN
'Sales Related' ELSE 'Other' END AS "Special Departments" FROM EMPLOYEES;
-- JOIN example, with GROUP BY, HAVING and ORDER BY
SELECT COUNT(d.DEPARTMENT_NAME) AS "Departments per Location", c.COUNTRY_NAME, l.STATE_PROVINCE FROM
DEPARTMENTS d INNER JOIN LOCATIONS l ON d.LOCATION_ID = l.LOCATION_ID INNER JOIN COUNTRIES c USING (COUNTRY_ID)
GROUP BY c.COUNTRY_NAME, l.STATE_PROVINCE HAVING COUNT(d.DEPARTMENT_NAME) > 1 ORDER BY 2, 3, 1;
-- UPDATE example
UPDATE EMPLOYEES SET COMMISSION_PCT = 10 WHERE COMMISSION_PCT = 0 AND SALARY < 5000;
-- INSERT example
INSERT INTO EMPLOYEES ( FIRST_NAME, LAST_NAME ) VALUES ( 'Roger', 'Bjarevall' );
-- DELETE example
DELETE FROM EMPLOYEES WHERE HIRE_DATE < to_timestamp('1900-01-10', 'RR-MM-DD');
-- CREATE TABLE example
CREATE TABLE DEPARTMENTS ( DEPARTMENT_ID NUMBER(4) NOT NULL, DEPARTMENT_NAME VARCHAR2(30) NOT NULL, MANAGER_ID
NUMBER(6), LOCATION_ID NUMBER(4), CONSTRAINT DEPT_ID_PK PRIMARY KEY (DEPARTMENT_ID), CONSTRAINT DEPT_LOC_FK
FOREIGN KEY (LOCATION_ID) REFERENCES "LOCATIONS" ("LOCATION_ID"), CONSTRAINT DEPT_MGR_FK FOREIGN KEY
(MANAGER_ID) REFERENCES "EMPLOYEES" ("EMPLOYEE_ID"), CONSTRAINT DEPT_NAME_NN CHECK ("DEPARTMENT_NAME" IS NOT
NULL) );
```

**Formatted with default settings:**

```
-- Basic SELECT example, with Sub-SELECT and JOIN
SELECT
    e.LAST_NAME           AS "Last Name",
    e.FIRST_NAME          AS "First Name",
    d.DEPARTMENT_NAME     AS "Department",
    e.SALARY              AS "Salary",
    e.SALARY + e.SALARY * e.COMMISSION_PCT,
    e.COMMISSION_PCT * 100 || '%',
    ROUND(e.SALARY /
    (   SELECT
            MAX(SALARY)
        FROM
            HR.EMPLOYEES), 2) * 100 AS "Percentage of Max"
FROM
    HR.EMPLOYEES e
INNER JOIN
    HR.DEPARTMENTS d
ON
    (
        e.DEPARTMENT_ID = d.DEPARTMENT_ID)
WHERE
    d.DEPARTMENT_ID IN (10,
                        20,
                        90,
                        210)
AND e.SALARY > 3000;

-- CASE example
SELECT
```

```
    FIRST_NAME,
    LAST_NAME,
    SALARY,
    CASE
        WHEN SALARY > 10000
        THEN 'High'
        WHEN SALARY BETWEEN 5000 AND 999
        THEN 'Midlevel'
        ELSE 'Low'
    END AS "Income Level",
    CASE DEPARTMENT_ID
        WHEN 40
        THEN 'Administration'
        WHEN 20
        THEN 'Sales Related'
        ELSE 'Other'
    END AS "Special Departments"
FROM
    EMPLOYEES;

-- JOIN example, with GROUP BY, HAVING and ORDER BY
SELECT
    COUNT(d.DEPARTMENT_NAME) AS "Departments per Location",
    c.COUNTRY_NAME,
    l.STATE_PROVINCE
FROM
    DEPARTMENTS d
INNER JOIN
    LOCATIONS l
ON
    d.LOCATION_ID = l.LOCATION_ID
INNER JOIN
    COUNTRIES c
USING
    (COUNTRY_ID)
GROUP BY
    c.COUNTRY_NAME,
    l.STATE_PROVINCE
HAVING
    COUNT(d.DEPARTMENT_NAME) > 1
ORDER BY
    2,
    3,
    1;

-- UPDATE example
UPDATE
    EMPLOYEES
SET
    COMMISSION_PCT = 10
WHERE
    COMMISSION_PCT = 0
AND SALARY < 5000;

-- INSERT example
INSERT INTO
    EMPLOYEES
    (
        FIRST_NAME,
        LAST_NAME
    )
    VALUES
    (
        'Roger',
        'Bjarevall'
    );

-- DELETE example
DELETE
FROM
    EMPLOYEES
```

```
WHERE
    HIRE_DATE < to_timestamp('1900-01-10', 'RR-MM-DD');

-- CREATE TABLE example
CREATE TABLE
    DEPARTMENTS
    (
        DEPARTMENT_ID   NUMBER(4) NOT NULL,
        DEPARTMENT_NAME VARCHAR2(30) NOT NULL,
        MANAGER_ID      NUMBER(6),
        LOCATION_ID     NUMBER(4),
        CONSTRAINT DEPT_ID_PK PRIMARY KEY (DEPARTMENT_ID),
        CONSTRAINT DEPT_LOC_FK FOREIGN KEY (LOCATION_ID) REFERENCES "LOCATIONS" ("LOCATION_ID"),
        CONSTRAINT DEPT_MGR_FK FOREIGN KEY (MANAGER_ID) REFERENCES "EMPLOYEES" ("EMPLOYEE_ID"),
        CONSTRAINT DEPT_NAME_NN CHECK ("DEPARTMENT_NAME" IS NOT NULL)
    );
```

**Unformatted to compact form:**

```
/*-- Basic SELECT example, with Sub-SELECT and JOIN*/ SELECT e.LAST_NAME AS "Last Name", e.FIRST_NAME AS "First
Name", d.DEPARTMENT_NAME AS "Department", e.SALARY AS "Salary", e.SALARY + e.SALARY * e.COMMISSION_PCT, e.
COMMISSION_PCT * 100 || '%', ROUND(e.SALARY / ( SELECT MAX(SALARY) FROM HR.EMPLOYEES), 2) * 100 AS "Percentage
of Max" FROM HR.EMPLOYEES e INNER JOIN HR.DEPARTMENTS d ON ( e.DEPARTMENT_ID = d.DEPARTMENT_ID) WHERE d.
DEPARTMENT_ID IN (10, 20, 90, 210) AND e.SALARY > 3000;
/*-- CASE example*/ SELECT FIRST_NAME, LAST_NAME, SALARY, CASE WHEN SALARY > 10000 THEN 'High' WHEN SALARY
BETWEEN 5000 AND 999 THEN 'Midlevel' ELSE 'Low' END AS "Income Level", CASE DEPARTMENT_ID WHEN 40 THEN
'Administration' WHEN 20 THEN 'Sales Related' ELSE 'Other' END AS "Special Departments" FROM EMPLOYEES;
/*-- JOIN example, with GROUP BY, HAVING and ORDER BY*/ SELECT COUNT(d.DEPARTMENT_NAME) AS "Departments per
Location", c.COUNTRY_NAME, l.STATE_PROVINCE FROM DEPARTMENTS d INNER JOIN LOCATIONS l ON d.LOCATION_ID = l.
LOCATION_ID INNER JOIN COUNTRIES c USING (COUNTRY_ID) GROUP BY c.COUNTRY_NAME, l.STATE_PROVINCE HAVING COUNT(d.
DEPARTMENT_NAME) > 1 ORDER BY 2, 3, 1;
/*-- UPDATE example*/ UPDATE EMPLOYEES SET COMMISSION_PCT = 10 WHERE COMMISSION_PCT = 0 AND SALARY < 5000;
/*-- INSERT example*/ INSERT INTO EMPLOYEES ( FIRST_NAME, LAST_NAME ) VALUES ( 'Roger', 'Bjarevall' );
/*-- DELETE example*/ DELETE FROM EMPLOYEES WHERE HIRE_DATE < to_timestamp('1900-01-10', 'RR-MM-DD');
/*-- CREATE TABLE example*/ CREATE TABLE DEPARTMENTS ( DEPARTMENT_ID NUMBER(4) NOT NULL, DEPARTMENT_NAME
VARCHAR2(30) NOT NULL, MANAGER_ID NUMBER(6), LOCATION_ID NUMBER(4), CONSTRAINT DEPT_ID_PK PRIMARY KEY
(DEPARTMENT_ID), CONSTRAINT DEPT_LOC_FK FOREIGN KEY (LOCATION_ID) REFERENCES "LOCATIONS" ("LOCATION_ID"),
CONSTRAINT DEPT_MGR_FK FOREIGN KEY (MANAGER_ID) REFERENCES "EMPLOYEES" ("EMPLOYEE_ID"), CONSTRAINT DEPT_NAME_NN
CHECK ("DEPARTMENT_NAME" IS NOT NULL) );
```

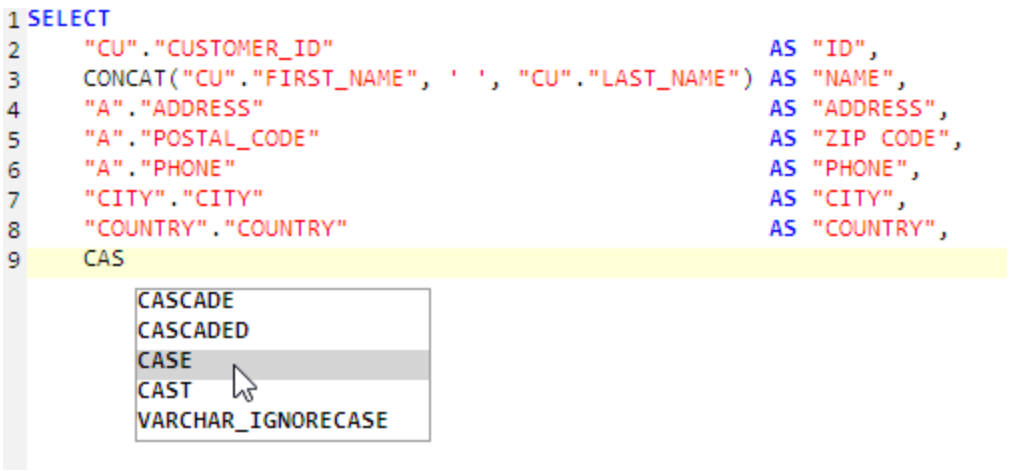
## Auto Completion

Only in DbVisualizer Pro

This feature is only available in the DbVisualizer Pro edition.

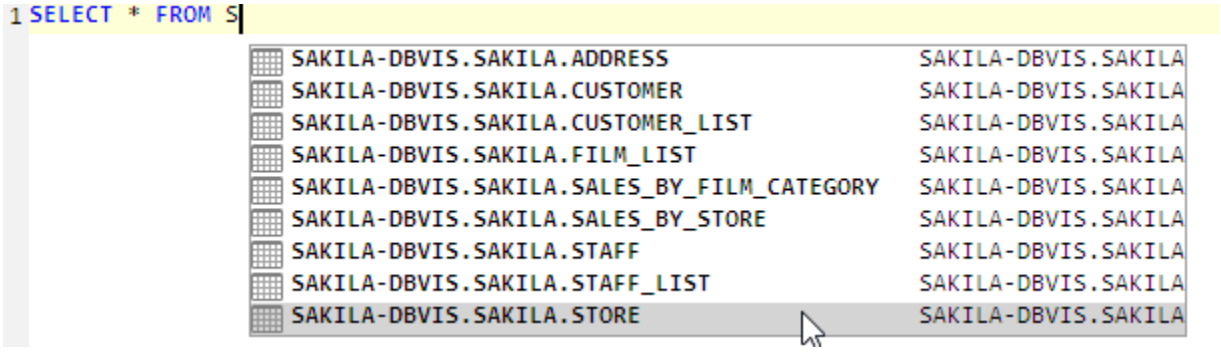
Auto completion is a convenient feature used to assist you when editing SQL statements and DbVisualizer commands. By default, you activate auto completion with the key binding **Ctrl-SPACE**, but you can also configure it to activate as you type (in the Tool Properties dialog, in the **SQL Editor/Auto Completion** category under the General tab).

With the caret in any place in a statement where you can type something other than a table name or a column name, and at least one character just before the caret, activating auto completion displays a list of keywords that starts with the letters you have typed so far. As you continue to type, the list narrows.

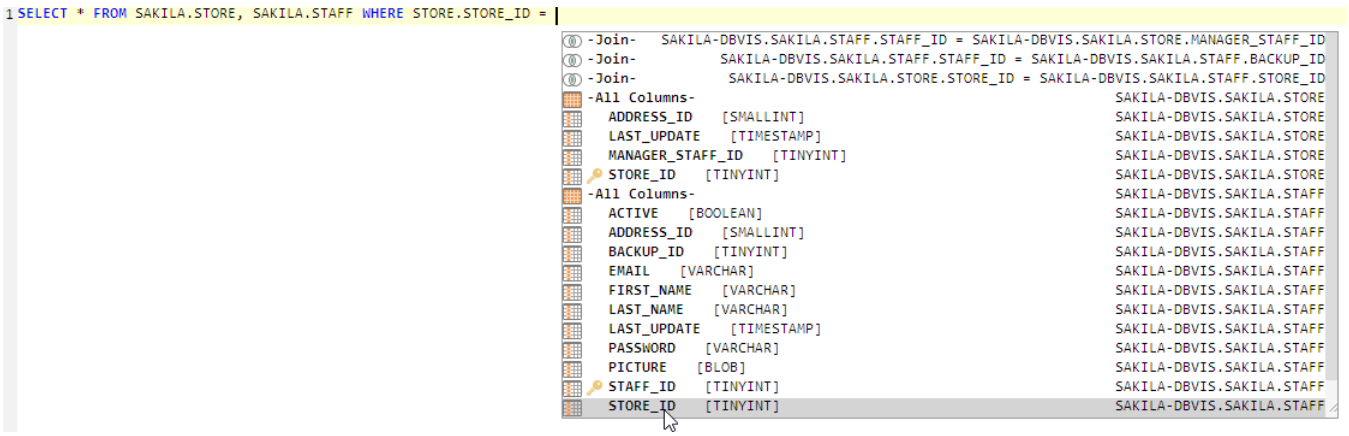



The list of keywords is database specific, selected based on the database type for the connection currently selected in the **Database Connection** list above the editor.

With the caret placed where a table or view name may be typed in a supported SQL statement type, the auto completion list shows a list of tables and views from the currently selected database connection, assuming you are actually connected to the database. The following figure shows the completion pop up with table names that contain the letter S.



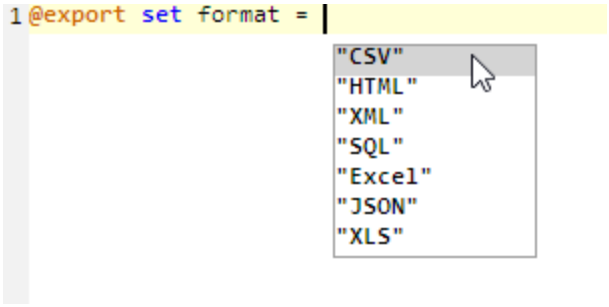

A completion pop-up showing column names is shown when the caret is placed where a column name may be typed.



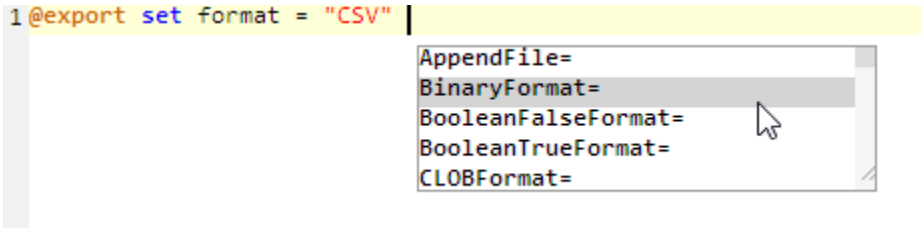

DbVisualizer provides auto completion for table and columns names for the following DML commands:

- SELECT
- INSERT
- UPDATE
- DELETE

Auto completion for DbVisualizer commands is very similar. Activating it after a partial command name lists all matching commands. If you activate it after a complete command name, you get a list of all valid parameters for the command. After a parameter name, you can select from a list of valid values.




For the `@export set` command, the parameter list is adapted for the specified output format after you have entered the `Format` parameter setting, for instance only showing parameters that are valid for the CSV format.




To display the completion pop-up, use the key binding **Ctrl-SPACE** (by default). You select an entry in the pop-up menu with a mouse double-click, the **ENTER** key, or the **TAB** key. To cancel the pop-up, press the **ESC** key.

If there are several SQL statements in the editor, make sure to separate them using the statement delimiter character (the default is ";").

In order for the column name completion pop-up to appear, you must first make sure there are table names in the statement.

All table names that have been listed in the completion pop-up are cached by DbVisualizer to make sure subsequent displays of the pop-up is performed quickly without asking the database. The cache is cleared only when doing a **Refresh** in the database objects tree or reconnecting the database connection.

The **Database** and **Schema** lists above the editor are used to limit the list of tables in the auto complete pop-up to those in the selected database and/or schema. To include all tables, select the blank entries in these lists. The default selections for the lists can be set as connection properties, in the **SQL Commander** category.

It is possible to fine-tune how auto completion works in the connection properties.

- Enable or disable the use of identifier qualifiers (i.e. qualifying table names with the schema name) in the **[Database Type]/Qualifiers** category,
- Enable or disable the use of delimited identifiers (e.g. quotes around a table name) in the **[Database Type]/Delimited Identifiers** category.

Sorting, when to show the popup, upper/lower case transformation, etc. can be configured in the **Tool Properties** dialog, in the **SQL Editor/Auto Completion** category under the **General** tab.

## Recording and Playing Edit Macros

If you repeatedly need to run a sequence of edit operation, you can record them as a macro and play it as many times as needed during an editing session. The editor status bar indicates when a recording is in progress and when a macro is available to play.

As an example, suppose you have some plain text that you need to convert into INSERT statements:

```
12345 123456
89012 890123
45678 456789
```

Place the caret at the beginning of the first line and start the macro recording, using the right-click menu or the corresponding key binding, and then type text and use key bindings to perform the following operations:

1. Type `insert into mytable values('`
2. **Insertion Point to Next Word**
3. Type `',`
4. **Insertion Point to Next Word**
5. Type `'`
6. **Insertion Point to Next Word**
7. Type `');`
8. **Insertion Point Down**
9. **Insertion Point to Beginning of Line**

Then stop the recording. You now have a macro for converting a single line to an INSERT  statement. To convert the remaining lines, just use Play Macro for each line. The result will look like this:

```
insert into mytable values('12345', '123456');
insert into mytable values('89012', '890123');
insert into mytable values('45678', '456789');
```

The Find operation, by default mapped to the **Find** key and **Ctrl-F** key stroke, can not be recorded. You must instead use **Find Selection**, **Find with Dialog**, **Find Next** and **Find Previous**. Mouse gestures are also not recorded, only key strokes and menu selections.

## Folding Selected Text

If you work with a large script, it can sometimes be helpful to hide parts of it. You can do so using the Code Folding feature.

Select the text you want to hide and then choose **Folding Operations->Toggle Fold Selection** in the right-click menu. The selected text is then replaced (visually only) with a folding marker.

Here's an unfolded script with the column expression selected:

```
SELECT
    COUNTRY                                                 AS "Country",
    SUM(PAYMENT.AMOUNT)                                     AS "Total Paid",
    ROUND(AVG(PAYMENT.AMOUNT), 2)                           AS "Avg Paid",
    COUNT(DISTINCT RENTAL.RENTAL_ID)                        AS "Rentals",
    COUNT(DISTINCT FILM.FILM_ID)                            AS "Films",
    COUNT(DISTINCT PAYMENT.CUSTOMER_ID)                     AS "Customers",
    SUM(DATEDIFF(DAY, RENTAL.RENTAL_DATE, RENTAL.RETURN_DATE))  AS "Total Duration",
    AVG(DATEDIFF(DAY, RENTAL.RENTAL_DATE, RENTAL.RETURN_DATE))  AS "Avg Duration"
    FROM PAYMENT
        JOIN CUSTOMER ON PAYMENT.CUSTOMER_ID = CUSTOMER.CUSTOMER_ID
        JOIN ADDRESS ON CUSTOMER.ADDRESS_ID = ADDRESS.ADDRESS_ID
        JOIN CITY ON ADDRESS.CITY_ID = CITY.CITY_ID
        JOIN country ON CITY.COUNTRY_ID = country.COUNTRY_ID
        JOIN RENTAL ON PAYMENT.RENTAL_ID = RENTAL.RENTAL_ID
        JOIN INVENTORY ON RENTAL.INVENTORY_ID = INVENTORY.INVENTORY_ID
        JOIN FILM ON INVENTORY.FILM_ID = FILM.FILM_ID

        GROUP BY COUNTRY ORDER BY COUNTRY;
```

And here is the same script with the selection folded:

```
SELECT
...
    FROM PAYMENT
        JOIN CUSTOMER ON PAYMENT.CUSTOMER_ID = CUSTOMER.CUSTOMER_ID
        JOIN ADDRESS ON CUSTOMER.ADDRESS_ID = ADDRESS.ADDRESS_ID
        JOIN CITY ON ADDRESS.CITY_ID = CITY.CITY_ID
        JOIN country ON CITY.COUNTRY_ID = country.COUNTRY_ID
        JOIN RENTAL ON PAYMENT.RENTAL_ID = RENTAL.RENTAL_ID
        JOIN INVENTORY ON RENTAL.INVENTORY_ID = INVENTORY.INVENTORY_ID
        JOIN FILM ON INVENTORY.FILM_ID = FILM.FILM_ID

        GROUP BY COUNTRY ORDER BY COUNTRY;
```

You can fold more than one part of a script using the same procedure.

To unfold just one part, select the folding marker (be careful to select all of it) and then choose **Toggle Fold Selection** from the menu again. To unfold all folded parts, use **Expand All Foldings**.

## Selecting a Rectangular Area

In some cases, it is handy to be able to select a rectangular area in the middle of a script. Say, for instance, that you need to copy just the first part of a few lines and paste it at the beginning of some other lines.

To do this in the SQL editor, click the mouse where you want to start the selection and then press the **Alt** key (by default) while you extend the selection by dragging the mouse. If you prefer to use the **Ctrl** key as the modifier, you can change the default in Tool Properties in the SQL Commander category under the General tab.

```
SELECT
    SAKILA.CATEGORY.CATEGORY_ID,
    SAKILA.FILM.FILM_ID,
    SAKILA.FILM.TITLE
FROM
    SAKILA.FILM
INNER JOIN
    SAKILA.FILM_CATEGORY
ON
    (
        SAKILA.FILM.FILM_ID = SAKILA.FILM_CATEGORY.FILM_ID)
INNER JOIN
    SAKILA.CATEGORY
ON
    (
        SAKILA.FILM_CATEGORY.CATEGORY_ID = SAKILA.CATEGORY.CATEGORY_ID);
```

## Highlighting Matches

Instead of searching for occurrences of a text string and navigating to each occurrence, it is sometimes useful to get all occurrences highlighted. To do this, select a text string that is at least three characters long and contains at least one letter or digit. You can use the **Tool Properties** dialog to enable or disable this feature (in the **General / SQL Commander** category) or change the colors used (in the **General / Appearance / Editor Styles** category).

```
SELECT
    SAKILA.CATEGORY.CATEGORY_ID,
    SAKILA.FILM.FILM_ID,
    SAKILA.FILM.TITLE
FROM
    SAKILA.FILM
INNER JOIN
    SAKILA.FILM_CATEGORY
ON
    (
        SAKILA.FILM.FILM_ID = SAKILA.FILM_CATEGORY.FILM_ID)
INNER JOIN
    SAKILA.CATEGORY
ON
    (
        SAKILA.FILM_CATEGORY.CATEGORY_ID = SAKILA.CATEGORY.CATEGORY_ID);
```

## Tab Key Treatment

Pressing the TAB key in the editor inserts eight (8) space characters by default. If you instead want a TAB character to be inserted, or want to insert another number of space characters, you can specify this in the **Tool Properties** dialog, in the **General / SQL Commander** category under the General tab.

## Key Bindings

The editor shortcuts, or key bindings, can be redefined in the **Tool Properties** dialog, in the **Key Bindings** category under the **General** tab (see Changing Keyboard Shortcuts). Expand the **Editor Commands** node to manage all editor actions and the **Main Menu/Edit** node to manage the key bindings for the edit operations in the right-click editor menu and the main window **Edit** menu.