SQL Analyzer - 2.3

Target audience:

Users of the extension providing source code analysis support for **SQL** files.

Summary: This document provides basic information about the extension that provides source code analysis support for **SQL** files.

What's new 2.3.0:

- PERMANENT FIX - Duplicate external links between Cobol and SQL Tables.

What's new 2.3.1:

- FIX - IndexError: list index out of range at application level.

What's new 2.3.2:

- Permanent Fix - Table objects are missing in Teradata scan.
- Fix - Icons are missing in Enlighten for synonym on a table.
- Permanent Fix - [ORANGE][COSI][NNI] BFP is coming 0 for SQL.

What's new 2.3.3:

- Permanent Fix - SQL Analyzer: missing SQL column objects.
- Support of CREATE TYPE 2 Indexes.

What's new 2.3.4:

- Permanent Fix - SQL Objects are not getting created from SQL files.

What's new 2.3.5:

- Permanent Fix - Plugin has encountered the following error : Traceback (most recent call last).
- Permanent Fix - SQL analyzer missing objects. No stored procedures objects are created for DB2 zOS
- Permanent Fix - Warning messages when analyze ALTER TABLE .... ADD PRIMARY KEY ... statements

What's new 2.3.6:

- Fix - GUID duplicate found : SQLScriptTableColumn 'SQLScriptTableColumn?XXXX.YYYYY.ZZZZ.AAAA' UNIVERSAL_CACHE : DUPLICATE OBJECTS NAME IN SOURCE: Source File not found
- Fix - False Positive for Avoid queries using old style join convention instead of ANSI-Standard joins and Never use SQL queries with a cartesian product
- Fix - Homonyms Primary Keys on homonyms Tables, created in different files

Description

The **SQL Analyzer** (successor to the SQL Script extension) provides support for database technologies using the **ANSI SQL-92 / 99 language**. This extension uses the **Universal Analyzer framework** and is intended to analyse **DDL, DML** and **SQL exports** for a large variety **of SQL variants**:

- This extension provides source code analysis support for **DDL** and **DML*.sql files** using an **over language of the various sql variants.**
- This extension also accepts **src** and **uaxDirectory** files. Check here for more details about **sqltablesize** files.

In what situation should you install this extension?

- If you need to analyze **PostgreSQL**, **MySQL**, **MariaDB**, **SQLite**, **DB2**, **Sybase**, **Microsoft SQL Server**, **Teradata** or **Informix**
- If your application contains schemas from database vendors not supported "**out of the box**" by **CAST AIP** (see http://doc.castsoftware.com, **Supported Technologies** for more information) but, which are compliant with **ANSI SQL-92 / 99**
- When you do not have access to the online database to perform an extraction for use with CAST AIP and have instead been provided with DDL scripts

Transitioning from from the CAST AIP DB2 Analyzer to the SQL Analyzer extension

If you have been actively analyzing DB2 (z/OS or UDB) with the **DB2 Analyzer** (provided out-of-the-box in CAST AIP) you can transition to using the SQL Analyzer extension to analyze your DB2 source code. The process of transitioning is described in SQL Analyzer - 2.3.

> **Reversed links**
>
> When transitioning from the DB2 Analyzer to the SQL Analyzer, links between Tables and Indexes, Foreign Keys, Primary Keys and Unique Keys will appear to be reversed when comparing the analysis results of the DB2 Analyzer and the SQL Analyzer. This is because the representation of links in the SQL Analyzer uses a different method (which is identical for all supported RDBMS) to the DB2 Analyzer.

Vendor compatibility matrix

Official support

PostgreSQL	MySQL	MariaDB	SQLite	IBM DB2
Up to version 9.5	Up to version 5.5	Up to version 10.0	Up to version 3.x	Up to version 11.1 (UDB) and 12 (zOS)
SYBASE	Microsoft SQL Server	TERADATA	IBM Informix	
Up to version 16	Up to version 2016	Up to version 16	Up to version 12.x	

Unofficial support

ORACLE DATABASE

Unofficial support refers to CAST AIP features or official CAST AIP extensions that provide specific capabilities that have not been officially validated or tested by CAST, therefore CAST cannot guarantee the results they produce. An example of a feature in this category is the capability built into the **SQL Analyzer extension** to analyze Oracle source code: while the analysis will work, results are not guaranteed.

Function Point, Quality and Sizing support

This extension provides the following support:

- **Function Points (transactions)**: a green tick indicates that OMG Function Point counting and Transaction Risk Index are supported
- **Quality and Sizing**: a green tick indicates that CAST can measure size and that a minimum set of Quality Rules exist

Function Points (transactions)	Quality and Sizing
✓	✓

CAST AIP compatibility

This extension is compatible with:

CAST AIP release	Supported
8.3.x	✓
8.2.x	✓
8.1.x	✓
8.0.x	✓
7.3.x	✗

Supported DBMS servers used for CAST AIP schemas

This extension is compatible with the following DBMS servers used to host CAST AIP schemas:

CAST AIP release	CSS	Oracle	Microsoft
All supported releases	✓	✓	✓

Prerequisites

✓	An installation of any compatible release of CAST AIP (see table above)

Download and installation instructions

Please see:

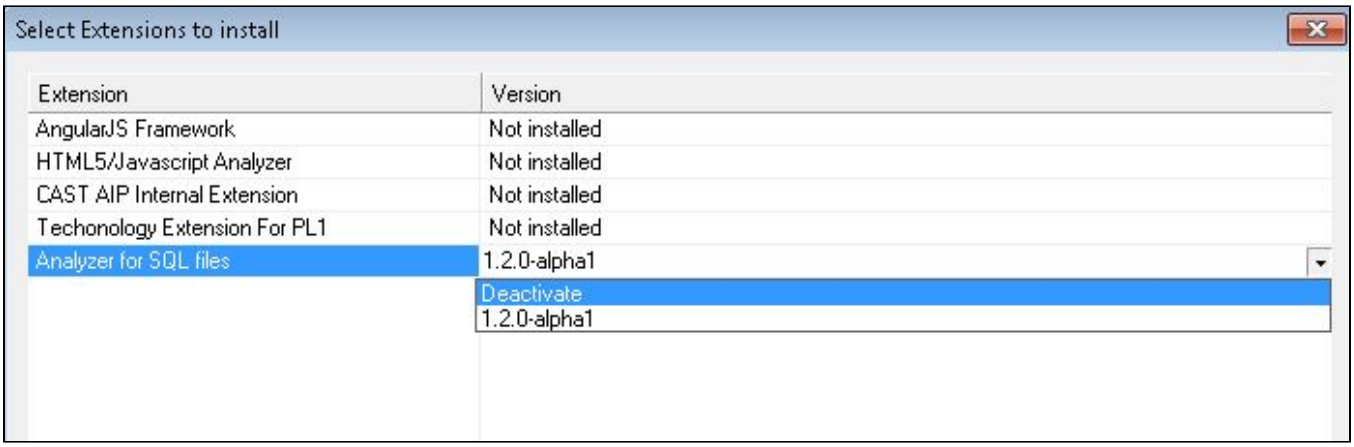
- http://doc.castsoftware.com/display/EXTEND/Download+an+extension
- http://doc.castsoftware.com/display/EXTEND/Install+an+extension

- The latest release status of this extension can be seen when downloading it from the CAST Extend server.
- Please see Known Limitations and Issues for information about an error that may occur when installing the extension if you have also already installed a very old and unsupported Universal Analyzer langage pack that conflicts with the SQL Analyzer.

Upgrade from the SQL Script extension

If you have previously used the SQL Script extension (com.castsoftware.sqlscript) on existing schemas, you should proceed as following:

- In **CAST Server Manager** use the **Manage Extensions** option on the CAST AIP schemas in which the SQL Script extension is installed
- Select **Analyzer for SQL files** and choose **deactivate** to remove the existing extension. No further actions are required.

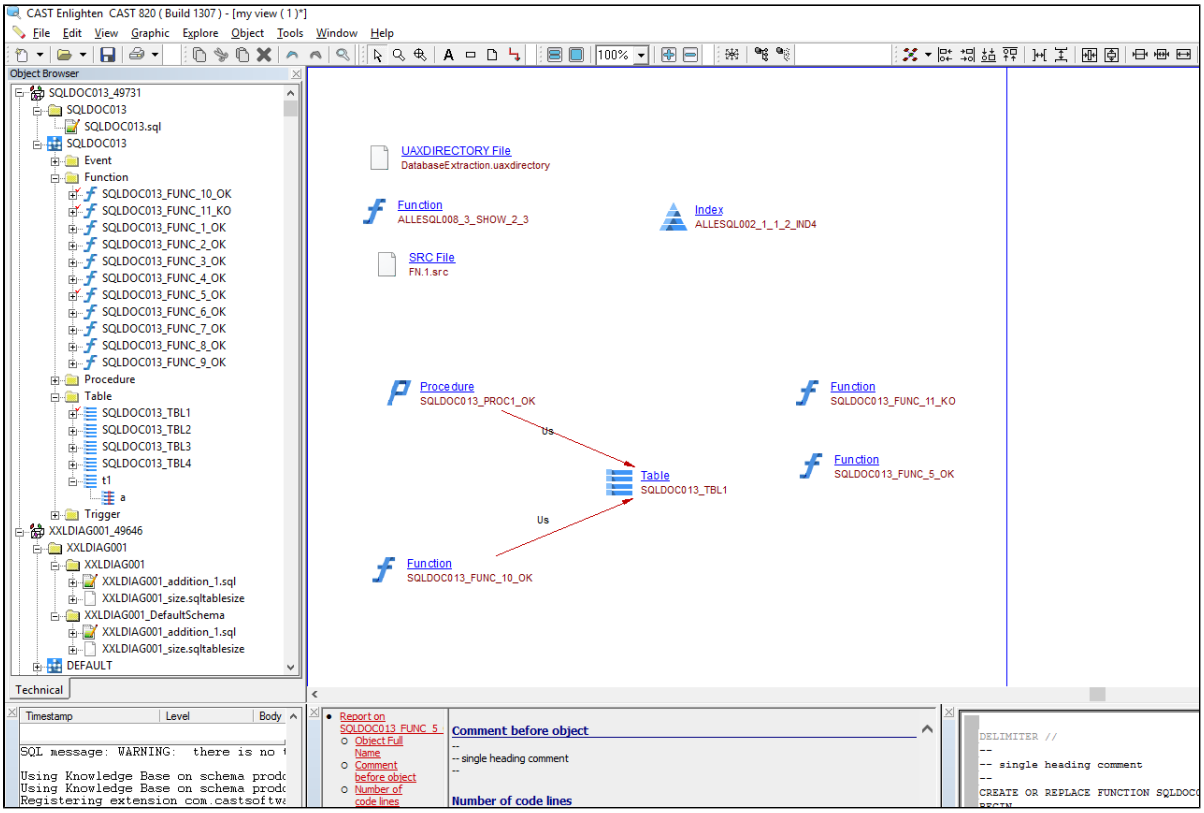


Packaging, delivering and analyzing your source code

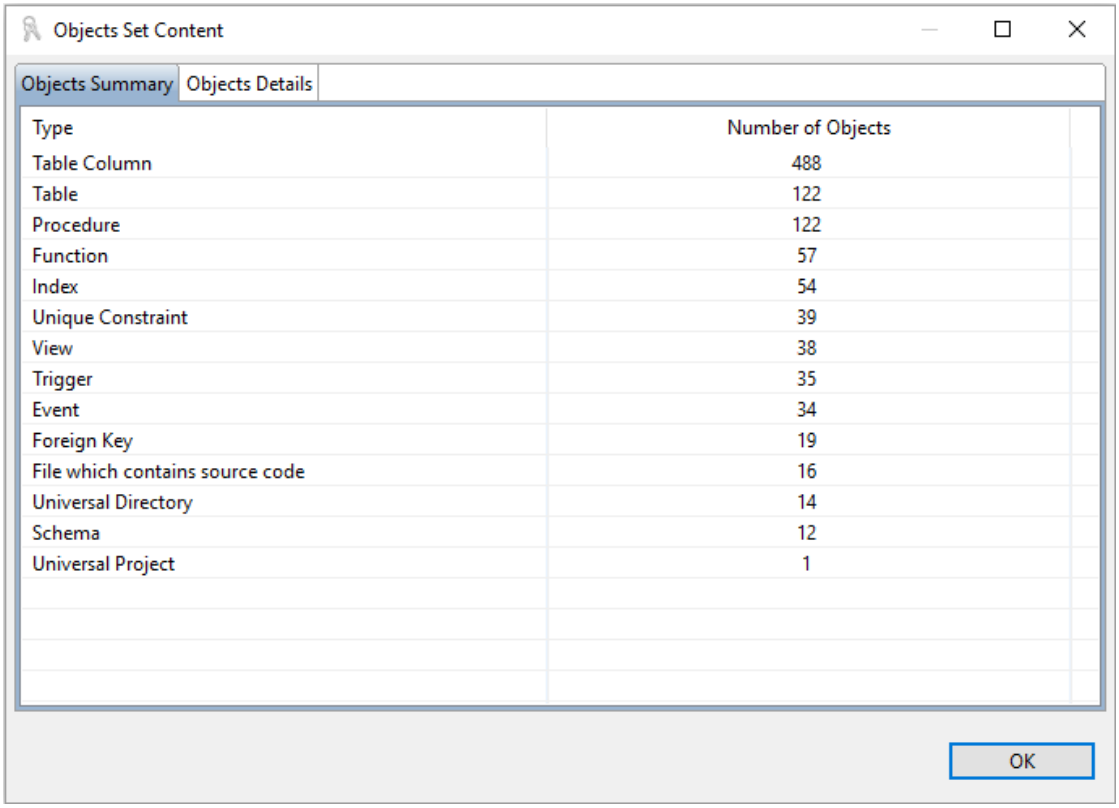
Please see: SQL Analyzer - 2.3

What results can you expect?

Once the analysis/snapshot generation has completed, you can view the results in the normal manner (for example via CAST Enlighten) - *click to enlarge*:



You can also use the CAST Management Studio option **View Analysis Unit Content** to see the objects that have been created following the analysis:



Objects

The following objects are displayed in CAST Enlighten:

Icon	Description
	Schema
	Table
	View
	Table Column
	Index
	Foreign Key
	Unique Constraint
	Procedure
	Function

	Method
	Trigger
	Package
	Type
	Event
	Synonym
	sourceFile
	DML Script File

Note that:

- Object identity is independent from the *.sql file the object comes from.
- Object identity depends on the Analysis Unit's identity. Therefore, using a new Analysis Unit or deleting and then recreating an Analysis Unit will change the object's identity and will result in added/removed objects in the subsequent analysis results.
- Typically a table will be identified by the Analysis Unit name, schema name and table name.
- When no schema can be determined, the analyzer considers that a schema named "DEFAULT" is used. But generally, identifiers are qualified in CREATE TABLE statements.

Table deletion and renaming

DROP TABLE syntax is supported for table objects **within the same file**. When creating a table through CREATE TABLE tableName (colName int, ...) followed by a DROP TABLE tableName, the table will not be recorded and thus will not be displayed in CAST Enlighten. Similarly, if a table is renamed with a **RENAME TABLE** statement (or **ALTER TABLE RENAME TO** as in SQLite and PostgreSQL), this change will be reflected in CAST Enlighten. Presently we consider *case-insensitive* names, i.e., objects named tableName, TABLEname are considered to be the same object.

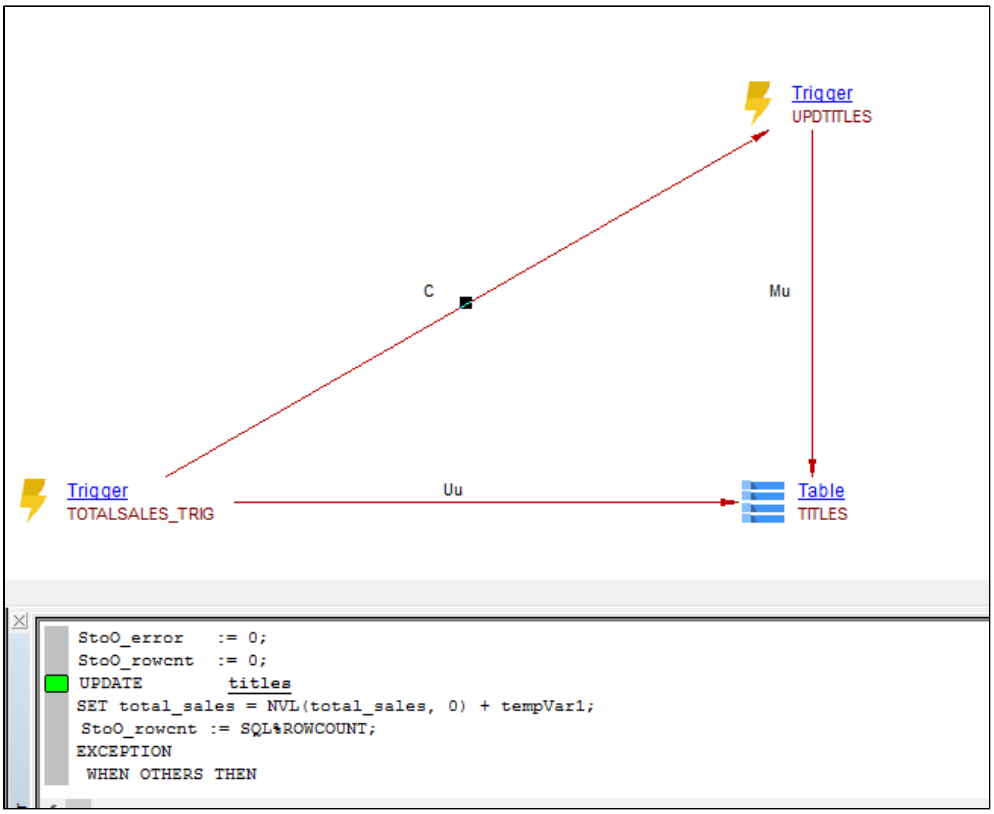
Links

Links are created for transaction and function point needs:

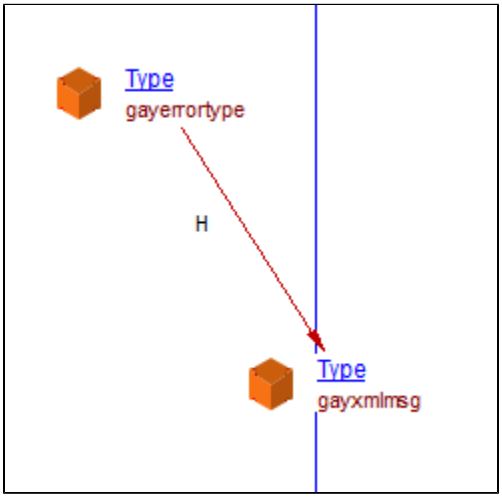
DDL

You can expect the following links on the DDL side within the same sql file:

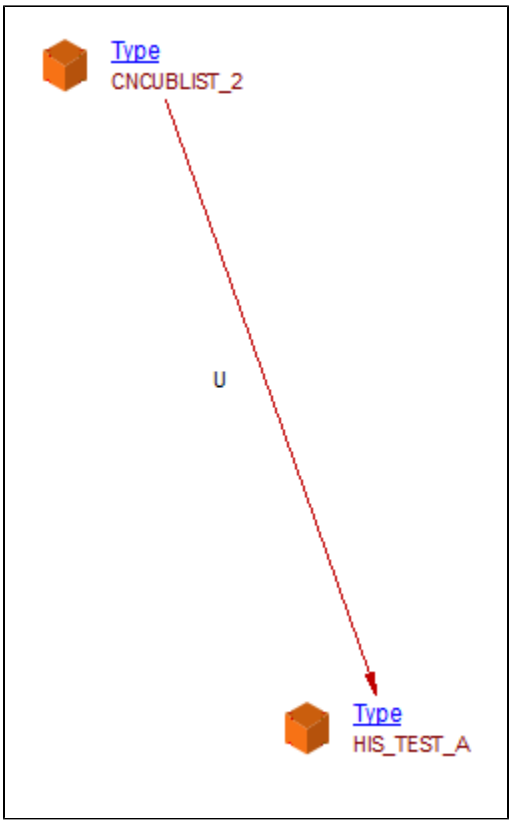
- useSelect, useInsert, useUpdate, useDelete Links from **Procedure** / **Function** / **Event** to **Table / View**
- callLink from **Procedure** / **Function / Event** to **Procedure** / **Function**
- useSelect from **View to Table / View** used in the query of the view
- callLink from **View** to **Function**
- relyonLink from **Index** to the **Table**
- relyonLink from **Index** to the **Column** implied in the index
- relyonLink from **Synonym** to **Table** / **View** / **Function** / **Procedure** / **Package** aliased by Synonym
- referLink from **Table / Table Column** to a **Table** / **Table Column** referenced in a **Foreign Key**
- callLink to the correct **Trigger** where the tables is accessed in insert/update/delete
    - example a **Trigger** declared as BEFORE INSERT on a table, any insert to that table will call the trigger...



- inheritLink from sub **Type** to super **Type**:
  - example : CREATE OR REPLACE type gayerrortype under gayxmlmsg ....



- useLink for PL/SQL tables, example : CREATE OR REPLACE TYPE "CNCUBLIST_2" is table of "HIS_TEST_A" ;



DML

You can expect the following links on the DML side :

- Links from **SQL Script** to **Table** provided as dependencies
- Links from client code to **Table** provided as dependencies

Quality Rules

You can find a full list of rules delivered with this extension here:

https://technologies.castsoftware.com/?rlH=extensions/com.castsoftware.sqlanalyzer/2.3.3.json

Name	Client Side Support
Never use SQL queries with a cartesian product (1101000)	COBOL, PB, VB, .NET, JAVA, C/C++/OBJC, IOS, PYTHON
Never use SQL queries with a cartesian product on XXL Tables (1101002)	COBOL, PB, VB, .NET, JAVA, C/C++/OBJC, IOS, PYTHON
Avoid non-indexed SQL queries (1101004)	COBOL, PB, VB, .NET, JAVA, C/C++/OBJC, IOS, PYTHON
Avoid non-indexed XXL SQL queries (1101006)	COBOL, PB, VB, .NET, JAVA, C/C++/OBJC, IOS, PYTHON

Avoid non-SARGable queries (1101008)	COBOL PB VB .NET JAVA C/C++/OBJC IOS PYTHON
Avoid NATURAL JOIN queries (1101010)	COBOL PB VB .NET JAVA C/C++/OBJC IOS PYTHON
Specify column names instead of column numbers in ORDER BY clauses (1101012)	COBOL PB VB .NET JAVA C/C++/OBJC IOS PYTHON
Avoid queries using old style join convention instead of ANSI-Standard joins (1101014)	COBOL PB VB .NET JAVA C/C++/OBJC IOS PYTHON
Always define column names when inserting values (1101026)	COBOL PB VB .NET JAVA C/C++/OBJC IOS PYTHON
Use MINUS or EXCEPT operator instead of NOT EXISTS and NOT IN subqueries (1101028)	COBOL PB VB .NET JAVA C/C++/OBJC IOS PYTHON
Avoid exists and not exists independent clauses (1101032)	COBOL PB VB .NET JAVA C/C++/OBJC IOS PYTHON
DISTINCT should not be used in SQL SELECT statements (1101034)	COBOL PB VB .NET JAVA C/C++/OBJC IOS PYTHON
Use ANSI standard operators in SQL WHERE clauses (EMBEDDED SQL) (1101036)	COBOL PB VB .NET JAVA C/C++/OBJC IOS PYTHON
Replace OR conditions testing equality on the same identifier in SQL WHERE clauses by an IN test condition (1101038)	COBOL PB VB .NET JAVA C/C++/OBJC IOS PYTHON

Special note about XXL/XXS support

See SQL Analyzer - 2.3 for more information.

Special notes about Quality Rules on client side

Some Quality Rules are calculated on SQL queries on the client-side with some limitations:

- Quality Rules will be enabled on client-side code only if the server-side code has been analyzed with SQL Analyzer extension.
- For Java client-side code, SQL statements used in parameters of methods including a SQL parametrization rule are analyzed.

Example of call to a parametrized method

```
class Foo
{
   final static String TABLE_NAME = "Person";

        void method()
        {
            String query = "select * from " + this.TABLE_NAME;
            java.sql.Statement.execute(query );
        }
}
```

- But 'queries' visible in the DLM (that need reviewing) are not analyzed:

Example ofa query visible in the DLM

```
class Foo
{
        // not passed to an execute something
        private final static String text = "select name from Person";
}
```

- Queries to Hibernate or JPA objects or any other ORM are not analyzed
- COBOL EXEC SQL queries are analysed
- SQL queries founded in Python code

Special note about redundant Quality Rules

Please see SQL Analyzer - 2.3.

Errors and warning

See here the full list of analysis **errors** and **warnings**: SQL Analyzer - 2.3.

Known limitations and issues

Installation

If you encounter the following **error** in CAST Server Manager while installing the SQL Analyzer extension, please perform the workaround described here and then attempt the installation again. This error may occur if you have installed a **very old and unsupported custom** Universal Analyzer language pack that used the same metamodel type names as used in the official SQL Analyzer extension.

```
SQL Analyzer is incompatible with the schema metamodel. It is generally due to an extension that has changed
it's ids
```

Analysis

- All name resolution is considered as **case insensitive**: this may produce erroneous links on a case insensitive platform 'playing with case' :
    - 2 different tables with the same case insensitive name will be both called
- **Procedure** resolution handles **overriding** when the number of parameters are matched or number and optionals are matched. Otherwise, when calling an overridden procedure, all overrides will be called. Below are some examples here is a single call Link, between the 2nd func1 and func2:

Match number of parameters

```
CREATE FUNCTION func1() RETURNS integer AS
begin
    DELETE FROM table1 WHERE ID in (SELECT ID FROM table2);
end;

CREATE FUNCTION func1(mode integer) RETURNS integer AS
begin
    DELETE FROM table1 WHERE ID in (SELECT ID FROM table2);
end;

CREATE FUNCTION func2(mode integer) RETURNS integer AS
begin
    select func1 (mode);
end;
```

Match number of parameters and how many are optionals

```
CREATE FUNCTION func1(i_mode integer) RETURNS integer AS
begin
    DELETE FROM table1 WHERE ID in (SELECT ID FROM table2);
end;

CREATE FUNCTION func1(mode integer := 1) RETURNS integer AS
begin
    DELETE FROM table1 WHERE ID in (SELECT ID FROM table2);
end;

CREATE FUNCTION func2(mode integer) RETURNS integer AS
begin
    select func1 ();
end;
```

- **Dynamic SQL** statements are resolved when:
  - the sql statement is readable, even for sliced statements. For example:

```
TABLE1, TABLE2, TABLE3 and TABLE4 are visibles and useSelect link were be added
```

```
CREATE PROCEDURE test
AS
L_QryStr varchar2(4000);
begin
L_QryStr := 'select S.COL1, P.COL2, P.COL3, P.COL4, P.COL5, ' ||
' P.COL6, B.COL7, R.COL8, B.COL9, B.COL10' ||
' from TABLE1 S, TABLE2 P, TABLE3 B, ' ||
' ( select distinct R.COL1, R.COL2 ' ||
' from TABLE4 R ' ||
' where R.COL3 = 99999 ';
EXECUTE IMMEDIATE L_QryStr;
end;
/
```

test_table is visible and useDelete link will be added

```
CREATE PROCEDURE test
AS
begin
EXECUTE IMMEDIATE 'truncate table test_table';
end;
/
```

test_table is visible and useDelete link will be added

```
CREATE PROCEDURE test
AS
L_QryStr varchar2(4000);
begin
        L_QryStr := 'truncate table ';
        L_QryStr := L_QryStr || ' test_table ';
        EXECUTE IMMEDIATE L_QryStr;
end;
/
```

- table name is valued via a variable which could be resolved. For example:

```
CREATE PROCEDURE test
AS
emp_rec  emp%ROWTYPE;
sql_stmt VARCHAR2(200);
begin
    sql_stmt := 'SELECT * FROM || emp_rec.T1 ||  WHERE job = 1';
    EXECUTE IMMEDIATE sql_stmt;
end;
/
```

emp table is visible and useSelect link will be added

```
CREATE OR REPLACE package body test_package as

 type T_1 is table of varchar2(22);
 emp_rec  emp%ROWTYPE;

 PROCEDURE test
is
sql_stmt VARCHAR2(2000);
emp_tab VARCHAR2(200);
begin
    emp_tab := emp_rec;
    sql_stmt := 'SELECT * FROM || emp_tab ||  WHERE job = :j';
    EXECUTE IMMEDIATE sql_stmt;
end test;

end;
/
```

- OPEN-FOR-USING : emp table and test procedure are linked by a use SELECT link

```
create table emp (col1 int);
/
CREATE OR REPLACE type body test_body as

 type t_emp is table of t_table index by varchar2(22);
 emp_rec  emp%ROWTYPE;

 member PROCEDURE test(o_cursor OUT my_cursor)
IS
BEGIN
   OPEN    o_cursor FOR
   'SELECT  rule_id,
           expression_id,
           parent_expression_id,
           operator
   FROM    emp
   ORDER   BY rule_id, expression_id';
END test;

end;
/
```

```
create table emp (col1 int);
/
CREATE OR REPLACE type body test_body as

 type t_emp is table of t_table index by varchar2(22);
 emp_rec  emp%ROWTYPE;

 member PROCEDURE test(o_cursor OUT my_cursor)
IS
L_SQL varchar(20000) := '';
BEGIN
    L_SQL := 'SELECT  rule_id,
           expression_id,
           parent_expression_id,
           operator
   FROM    emp
   ORDER   BY rule_id, expression_id';

   OPEN o_cursor FOR L_SQL;
END test;

end;
/
```

- **ALTER TABLE ... ADD ...** syntax is **supported**. All other syntaxes, such as ALTER TABLE ... DELETE .. or ALTER TABLE ... DROP ... or ALTER TABLE ... MODIFY ... etc. are **not supported**.
- **Moving a table** from one database/scheme to another is **not supported** through **RENAME TABLE** *schema**1***.tableName1 **TO** *schema**2***. tableName2.
- **Sequences** are not taken in to account and that is not a limitation but a choice because they have no effect on transactions nor Quality Rules
- **Oracle synonyms on packages** are not taken in to account.
- For the QR 7156 Avoid Too Many Copy Pasted Artifacts, total values are displayed but no detail values.
- For the QR 1101012 Specify column names instead of column numbers in ORDER BY clauses, the case when a function that returns a number or a numeric variable is used in order by is not reported to violate the rule.

CAST Health Dashboard (ex. Application Analytics Dashboard)

Starting with release 2.1 of the SQL Analyzer extension, the name used to represent the technology has changed from **SQL Analyzer** to **SQL**: if you have already transfered snapshots that contain SQL Analyzer analysis results in to a Measurement Service schema, you'll see **SQL Analyzer** in the **list of technologies** instead of **SQL**.

If you would like to change the technology name for existing snapshots, you can change it using the following SQL query run against the Measurement Service schema:

Change technology name in AAD

```
update DSS_OBJECT_TYPES
set OBJECT_TYPE_NAME = 'SQL'
where OBJECT_TYPE_ID = 1101000
```

What's new 2.3.5:

- Permanent Fix - Plugin has encountered the following error : Traceback (most recent call last).
- Permanent Fix - SQL analyzer missing objects. No stored procedures objects are created for DB2 zOS
- Permanent Fix - Warning messages when analyze ALTER TABLE .... ADD PRIMARY KEY ... statements