Structure101g SQL Flavor

Structure101g SQL Flavor

Chapter 1. Description

SQL flavor currently works with Headway Software Structure101g. This connects to multiple DB servers & databases and extracts the database metadata information from them using predominantly JDBC. But for features not supported in JDBC such as Stored Procedures & Triggers, it uses DB specific queries on metadata tables.

This flavor mainly uses the open source SchemaSpy [http://schemaspy.sourceforge.net] for DB metadata extraction, hence familiarity with it will help in correctly specifying most of the DB connectivity specific input parameters. SchemaSpy also has been enhanced to support views, stored procedures & triggers.

Chapter 2. Installation Steps

This section describe in detail, the installation steps for installing the base Headway Structure101g product and also the Tejas Software SQL Flavor on top of it.

Step 1: *Installation of the Base Structure101g Product*

1. First, install the Headyway Structure101g product from Headway Web Site [http://www.headwaysoftware.com]

2. Copy the Structure101g license that you got thru' e-mail to the Structure101g Install Directory, usually *C:\Program Files\Structure101\s101g*

3. Start Structure101g. While starting for first time, it will ask for *flavor home directory* which is by default: *C:\Documents and Settings\user\structure101g\flavors*. This file locations is referred to, in this document as $FLAVOR_INSTALLATION_HOME.

Step 2: *Installation of the SQL Flavor*

1. Install the SQL Flavor from the download site. This can be done by following the UI options:

   a. Flavors -> Install... -> Install Flavors screen

   b. Select “http://www.headwaysoftware.com/structure101/g/flavors/pre-release” in “Flavor site” field, if flavor is released.

   c. Select “http://www.headwaysoftware.com/structure101/g/flavors” in “Flavor site” field, if it is _not in pre-release. The flavor is installed in $FLAVOR_INSTALLATION_HOME as a separate version. *There can be multiple versions of the same flavor installed.*

2. Get the flavor evaluation license from Headway support.

Note: For preview flavors, evaluation license with a reasonable duration is already bundled with the flavor.

Step 3: Start using the flavor *'by pointing to your databases'* as explained in the next section. _Do not forget to provide your feedback & suggestions for improvement at the S101g SQL Flavor Forum [http://www.tejassoftware.com/forums/s101flavors]

Chapter 3. Flavor Usage Instructions

To start importing the metadata in the DBs that you wish to analyze, carry out the following steps:

Step 1: *Creating the Databases Connectivity Information*

First create the input CSV file containing the DB connectivity information. This file has the DB server & databases connectivity information. This is basically a columnar form of the JDBC connectivity string except that you do not need to worry about the variations in JDBC connectivity URL. Sample file for each supported DB types is included in:

> $FLAVOR_INSTALLATION_HOME/com.tejassoftware.sql_{ver}/examples directory.

Each DB or schema instance installed in a DB server has to have row entry. The columns to be entered are:

1. DB server fully-qualified hostname (or IP address) and port number – like mydbserver.mydomain.com:1234
2. DBType String – the string that is specified in next section for the DB server being used.
3. Database or Catalog Name – for Oracle this will be the SID name.
4. User name – user id which has at least read privilege to the specified Database.
5. Password – for the specified user name.
6. Schema name – specific schema name in the database.

For database types such as MySQL which do not support multiple schemas, this is same as Database / Catalog name.

1. Table Exclusion Pattern: regular expression pattern to exclude unwanted tables, views, triggers & stored procedures.
2. Columns Exclusion Pattern: regular expression pattern to exclude unwanted columns in tables & views.

Create an entry row like above for each database that you want to connect to.

Step 2: *Downloading the Required JDBC Drivers*

Download the JDBC drivers required for the different Database types that you want to connect to. This has to be downloaded from the DB vendor web sites.

Note: Just one naming convention that has to be followed is that the DBType string (specified in next section) has to be part of the file name*. For example, orathin-ojdbc5.jar, mysql-connector-java-5.0.4-bin.jar, etc. All these drivers are assumed to be available in a single folder called $DRIVERS_DIR.

Step 3: *Creating a New Project for DB Schema Information Import*

Create a new project for SQL flavor by selecting the UI options: `File -> New which pops up the *New Project Dialog* window.`

Step 4: *Providing the Necessary Input Parameters*

The flavor currently takes the following input parameters while importing data:

1. DB Connectivity Information CSV File: This is the full path to the CSV file prepared in **Step 1**.
2. JDBC Drivers directory which contains all the JDBC drivers jars/zip files. `Same as the $DRIVERS_DIR folder created in Step 2`.
3. Turn on cross-schema analyzer: whether cross schema dependencies have to be extracted or not.

Currently only explicit dependencies are extracted which are specified as:

{schema/DB name}.{table name}

Step 5:

Press *Finish* button to complete the DBs schemata dependency extraction. Depending on the number of servers & DBs that you are connecting to & the size of each of them, the extraction time can vary.

Step 6:

Once completed, you can explore the dependencies using the features available in the Structure101g user interface [http://www.headwaysoftware.com/products/structure101/g/]

3.1. Supported Databases & DBType String

The table below lists the databases supported. It also specifies the DBType string that is to be used for each one of them: DBType is one of:

DBType String	Database Description
db2	IBM DB2 with ‘app’ Driver
db2net	IBM DB2 with ‘net’ Driver
derby	Derby (JavaDB) Embedded Server
derbynet	Derby (JavaDB) Network Server
firebird	Firebird hsqldb HSQLDB Server
informix	Informix
maxdb	MaxDB
mssql	Microsoft SQL Server
mssql05	Microsoft SQL Server 2005
mssql-jtds	Microsoft SQL Server with jTDS Driver
mssql05-jtds	Microsoft SQL Server 2005 with jTDS Driver
mysql	MySQL ora Oracle with OCI8 Driver
orathin	Oracle with Thin Driver
pgsql	PostgreSQL
sybase	Sybase Server with JDBC3 Driver
sybase2	Sybase Server with JDBC2 Driver
udbt4	DB2 UDB Type 4 Driver

Chapter 4. Modules/Sub-modules Types & their Dependencies

This section describes the nodes and dependency types extracted from the SQL Schema & represented in the Structure101g Dependency Model. Where-ever required, the transformation from SQL statements in the Schema is explained using examples from the MySQL Sakila DB Schema [http://dev.mysql.com/doc/sakila/en/sakila.html]

4.1. Node/Module Types

Each Node/Module type represent a specific entity in the domain of analysis. For the SQL domain, we have defined a node or module type for the entities represented in table below. Each module type has its own icon which is used in the UI of the tool.

Module Type	Description
db-server	Represents the DB server in which the DBs are hosted.
sql-db	Represents the Database or schema which is hosted in the DB server.For some DB servers, this is also called the schema.
table	Represents a table in the database.
-temp-table	Represents a temporary table created in SP or trigger. The {type} could be one of global or local.
view	Represents a view constructed from a query of the tables.
package	Represents a logical grouping of SPs. Specific to only certain DBs such as Oracle.
stored-procedure	Represents a Stored Procedure(SP). A SP takes multiple parameters which are mapped to sub-modules.
function	Represents a function which has a return value.
trigger	Represents a trigger which is attached to a table or a view. There are different trigger types or variants, see note below.

A trigger can be defined to fire depending on the following parameters:

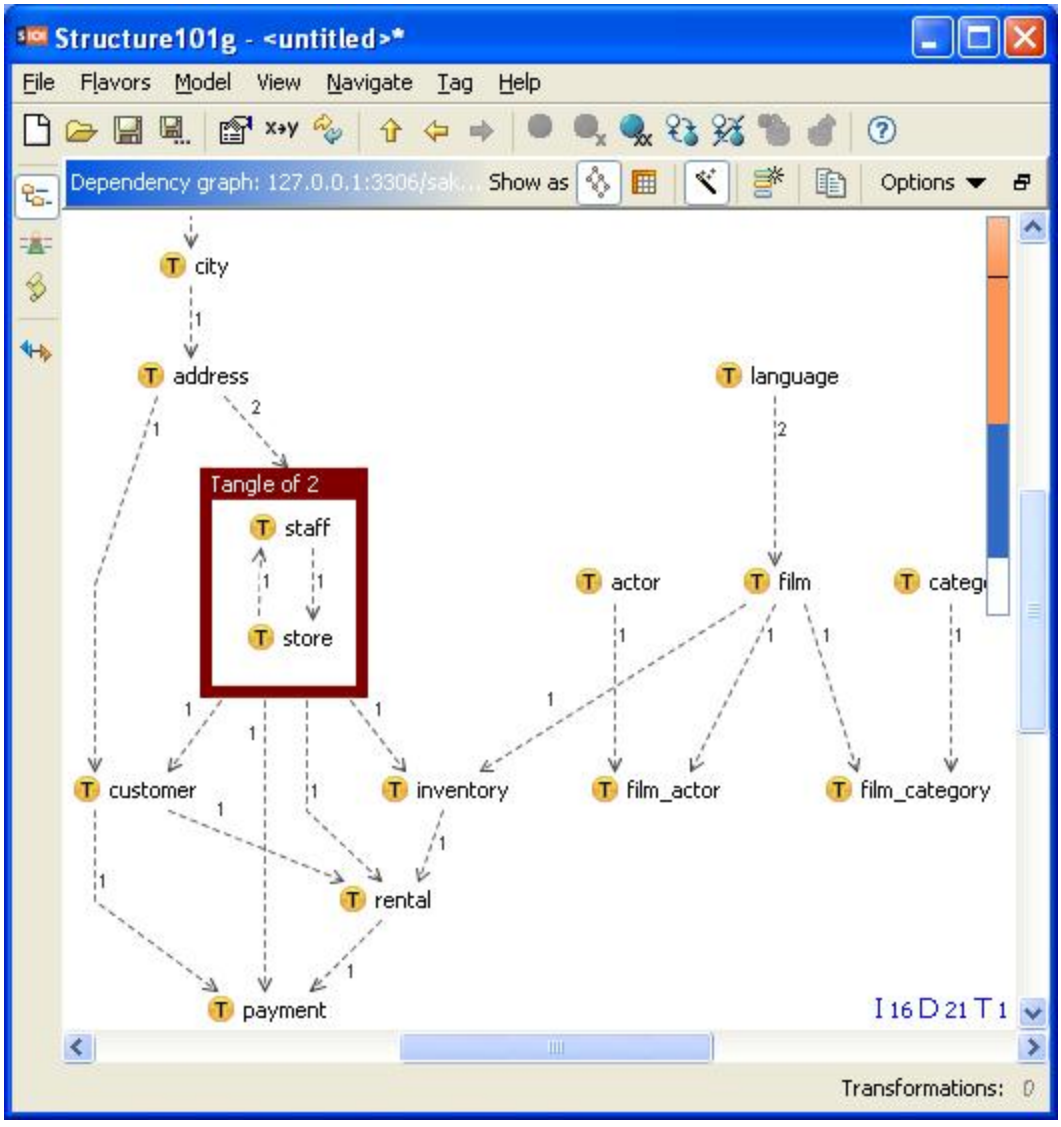
1. The timing at which it is defined to fire, can be one of: before, after or instead-of.
2. The DB operation for which it is defined, can be one of the following operations:
   a. DML Statements/Operations: Insert, Update, Delete
   b. DDL Statements/Operations: Create, Alter, Drop
   c. DB Operations: Connection, Disconnection, Log-on, Log-off
3. The scope of the trigger, can be one of statement-level or row-level.

There is one trigger type for each of combination of the three parameters with the following naming convention:

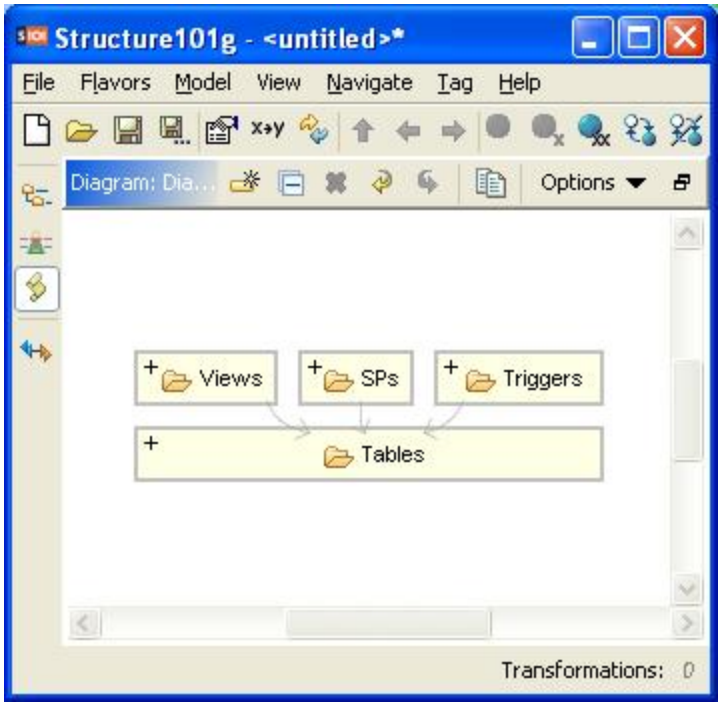
```
<timing>-<operation>-<scope>-trigger
```

We also have ‘grouping nodes’ which are represented with a folder icon in the UI. All the tables are grouped under a ‘Tables’ group. Similarly for views under Views, stored procedures & functions under SPs and triggers under Triggers. The advantage of such grouping is that it allows inter-group or intra-group exploration of the dependencies.

For examples, in the ‘Composition Perspective’, by selecting the Tables node, one can get the traditional entity-relationship diagram of the tables alone.



Similarly, by selecting the SPs node in hierarchy, the call dependency graph between the SPs & functions can be viewed. Also, as shown in the sample architectural diagram below, it becomes easier to layer the dependencies among the *four groups*.



So, the entity containment hierarchy that is visible in the 'Composition Perspective' is as follows:

```
DB server at the top -> Database/Schema -> (SPs, Tables, Triggers, Views).

Slash is used as separation character.
```

For example, the fully-qualified name of actor table in sakila DB in localhost server is:

```
localhost/sakila/Tables/actor
```

4.2. Sub-module Types

Sub-modules are nested types that are nested within modules types. For example, columns are nested or contained within a table or a view. Variables are declared within a SP. The table below shows the sub-module types that are defined. For each sub-module type, the valid parent module-type within which it can occur, is also indicated.

Sub-module Type	Description
col-foreignkey	A table column which is a foreign key to another table.
col-primarykey	A column which is a primary key of the table.
col-indexed-foreignkey	A foreign key column which is also indexed.
col-indexed-primarykey	A primary key column which is also indexed.
col-{type}	Maps to a table/view column which is of {type}.
col-unmapped	The table/view column maps to this if it is NOT one of the defined types.

in-param	SP or function input parameter.
out-param	SP or function output parameter.
inout-param	SP or function input/output parameter.
return-type	Return type from a function or SP. Since the return variable name is not declared, it is fixed as ret_var.
sql-statement	SQL statement which is called in a SP, function or trigger. The actual statement is stored in signature attribute of the node.
variable	Variable which is declared in SP, function or trigger.

4.3. Dependency or Edge Types

Dependency or edge types are defined for connections or links between two modules or sub-modules. A dependency or edge is directed. The table below shows the types & their description. Each dependency type is valid only between certain types of modules or sub-modules. This is also indicated in the table below. Some examples are:

1. A table can have a foreign key constraint to another table or itself.

2. A trigger is attached to a specific table.

3. A SP can call another one or a trigger can call a SP.

The table below shows the inter-table, trigger-table & view-table dependencies only.

Dependency Type	**Description**	**Valid Module / Sub-Module Types**
zero-to-one	Multiple but nullable foreign key of another table	From a column in one table to another column.
one-to-one	Mandatory & unique foreign key of another	From a column in one table to another column.
one-to-many	Mandatory foreign key of another table	From a column in one table to another column.
attached-to	Trigger which is attached to a table	From a trigger to a table.
selects	SQL statement which selects rows from a table	From a column in view to another table column.

To capture the dependencies introduced by functions, SPs and SQL statements in triggers, the following types are used:

Dependency Type	**Description**	**Valid Module / Sub-Module Types**
inserts	SQL statement which inserts rows in a table	From a sql-statement in SP or Trigger to a table.
selects	SQL statement which selects rows from a table	From a sql-statement in SP or Trigger to a table.

updates	SQL statement which updates columns in a table	From a sql-statement in SP or Trigger to a table.
deletes	SQL statement which deletes rows from a table	From a sql-statement in SP or Trigger to a table.
creates	SQL statement which creates a table in DB/schema	From a sql-statement in SP or Trigger to a table.
alters	SQL statement which alters a table in DB/schema	From a sql-statement in SP or Trigger to a table.
drops	SQL statement which drops a table from DB/schema	From a sql-statement in SP or Trigger to a table.
cond-inserts	SQL INSERT statement which is conditionally executed	From a sql-statement in SP or Trigger to a table.
cond-selects	SQL SELECT statement which is conditionally executed	From a sql-statement in SP or Trigger to a table.
cond-updates	SQL UPDATE statement which is conditionally executed	From a sql-statement in SP or Trigger to a table.
sets	A SET statement which sets a specific variable (in function, SP) or table column to some value.	From a trigger or SP to a variable or table column.
calls	A SP/function calling another one	From trigger, SP or function to another SP or function.
as-param-to	I/O binding for each of the params of the SP called	From variable. IO param or selected value to SP.in-param
filtered-with	This captures the dependencies in the WHERE clause in SELECT sql-statement.	From an sql-statement to a specific column in table. Input param of SP to a specific sql-statement in which param is used as filter.
join	Joins a table with selected set of columns in another table	From a sql-statement in SP or Trigger to a table.
xformed-into	Aggregation function which transforms the values retrieved in result set from temporary or normal table into a variable.	From sql-statement or variable to a variable (which could be output) or a view column.
if-then-else	IF-THEN-ELSE statement which evaluates some boolean condition.	From SP output value to variable in SP which is part of the IF condition.

Note: Since Structure101g v3.3 allows only a definition of 20-odd dependency types, we are unable to make the dependency types rich-enough by defining more sub-types. For example, we could have defined different types of joins but we are unable to do so, currently. Once this restriction is removed or limit is increased, in the base Structure101g, we should be able to provide more dependency types.

Chapter 5. SQL Statements Transformation

The previous section gave a description of the module, sub-module & dependency types. But it could have been unclear how specific SQL statements are transformed into these entities and their dependency types. This section gives some examples of how specific SQL statements are transformed into instantiation of these types. All the examples use the sample Sakila Schema available here [http://dev.mysql.com/doc/sakila/en/sakila.html]

Since most of the examples below are from SPs, the leading string the fully-qualified name is dropped and only the parts from the SP name is shown (for sake of brevity).

5.1. SELECT with One Variable

The following statement is part of the SP: inventory_held_by_customer.

```
DECLARE v_customer_id INT;
    SELECT customer_id INTO v_customer_id
    FROM rental
    WHERE return_date IS NULL  AND inventory_id = p_inventory_id;
```

p_inventory_id is an INPUT parameter which is extracted from JDBC metadata information. The above statement would create the following node & edge instances:

1. **variable:** inventory_held_by_customer/v_customer_id
2. **in-param:** inventory_held_by_customer/p_inventory_id
3. **sql-statement:** inventory_held_by_customer/stmt1
4. **selects:** from stmt1 to rental.customer_id
5. **filtered-with:** from stmt1 to rental.return_date
6. **filtered_with:** from inventory_held_by_customer/p_inventory_id to stmt1
7. **xformed-into:** from stmt1 to v_customer_id

In the last sets association, the selected value in stmt1 is set into the v_customer_id variable.

5.2. SELECT with JOIN & Boolean Return Value

The following statement is part of the SP: inventory_in_stock.

```
DECLARE v_out     INT;
    SELECT COUNT(rental_id) INTO v_out
    FROM inventory LEFT JOIN rental USING(inventory_id)
    WHERE inventory.inventory_id = p_inventory_id
    AND rental.return_date IS NULL;

    IF v_out > 0 THEN
```

```
  RETURN FALSE;
ELSE
  RETURN TRUE;
END IF;
```

The above statement would create the following node & edge instances:

1. **variable:** inventory_in_stock/v_out
2. **in-param:** inventory_in_stock/p_inventory_id
3. **sql-statement:** inventory_in_stock/stmt2
4. **selects:** from stmt2 to inventory
5. **joins:** from stmt2 to rental.inventory_id
6. **filtered-with:** from p_inventory_id to stmt2
7. **filtered_with:** from inventory_in_stock/stmt2 to rental.return_date
8. **xformed-into:** from stmt2 into v_out
9. **if-then-else:** from v_out to ret_var

In the last statement, depending on some condition on v_out variable, a boolean value is returned into ret_var. ret_var is a fixed name of the return variable.

5.3. TEMPORARY Table with INSERT from SELECT

The following statement is part of the SP: rewards_point

```
CREATE TEMPORARY TABLE tmpCustomer
        (customer_id SMALLINT UNSIGNED NOT NULL PRIMARY KEY);

    INSERT INTO tmpCustomer (customer_id)
    SELECT p.customer_id
    FROM payment AS p
    WHERE DATE(p.payment_date) BETWEEN last_month_start AND last_month_end
    GROUP BY customer_id
    HAVING SUM(p.amount) > min_dollar_amount_purchased
    AND COUNT(customer_id) > min_monthly_purchases;

    SELECT COUNT(*) FROM tmpCustomer INTO count_rewardees;

    DROP TABLE tmpCustomer;
```

The above statement would create the following node & edge instances:

1. **in-param:** rewards_point/min_dollar_amount_purchased

2. **in-param:** rewards_point/min_monthly_purchases
3. **out-param:** rewards_point/count_rewardees
4. **local-temp-table:** rewards_point/tmpCustomer
5. **col-smallint:** rewards_point/tmpCustomer/customer_id
6. **sql-statement:** rewards_point/stmt1 (for CREATE TEMPORARY TABLE ...)
7. **sql-statement:** rewards_point/stmt2 (for INSERT INTO ... SELECT ...)
8. **sql-statement:** rewards_point/stmt3 (for SELECT *COUNT* FROM ...)
9. **sql-statement:** rewards_point/stmt4 (for DROP TABLE ...)
10. **create:** from stmt1 to tmpCustomer
11. **selects:** from stmt1 to payment.customer_id
12. **filtered-with:** from stmt2 to payment.payment_date
13. **filtered-with:** from min_dollar_amount_purchased to stmt2
14. **filtered-with:** from min_monthly_purchase to stmts2
15. **inserts:** from stmt2 to tmpCustomer.customer_id
16. **selects:** from stmt3 to tmpCustomer
17. **xformed-into:** from stmt3 to count_rewardees – xform function is: COUNT
18. **drops:** from stmt4 to tmpCustomer

5.4. Multiple SELECT with Aggregated Return Value

This examples is from SP: get_customer_balance but only one statement (out of 3 SELECT statement) is shown.

```
DECLARE v_rentfees DECIMAL(5,2);
  DECLARE v_overfees INTEGER;
  DECLARE v_payments DECIMAL(5,2);

  SELECT IFNULL(SUM(film.rental_rate),0) INTO v_rentfees
    FROM film, inventory, rental
    WHERE film.film_id = inventory.film_id
      AND inventory.inventory_id = rental.inventory_id
      AND rental.rental_date <= p_effective_date
      AND rental.customer_id = p_customer_id;

  RETURN v_rentfees + v_overfees - v_payments;
```

The above statement would create the following node & edge instances:

1. **variable:** get_customer_balance/v_rentfees
2. **variable:** get_customer_balance/v_overfees
3. **variable:** get_customer_balance/v_payments
4. **in-param:** p_effective_date
5. **in-param:** p_customer_id
6. **return-type:** ret_var – signature is DECIMAL
7. **sql-statement:** get_customer_balance/stmt1
8. **selects:** from stmt1 to film.rental_rate
9. **filtered-with:** from stmt1 to inventory.film_id
10. **filtered-with:** from stmt1 to rental.inventory_id
11. **filtered-with:** from p_effective_date to stmt1
12. **filtered-with:** from p_customer_id to stmt1
13. **xformed-into:** from stmt1 to v_rentfees
14. **xformed-into:** from v_overfees to ret_var
15. **xformed-into:** from v_rentfees to ret_var
16. **xformed-into:** from V-payments to ret_var

5.5. SP Calls Another SP from WHERE Clause

This example is from SP: film_in_stock

```
SELECT inventory_id
     FROM inventory
     WHERE film_id = p_film_id
     AND store_id = p_store_id
     AND inventory_in_stock(inventory_id);

     SELECT FOUND_ROWS() INTO p_film_count;
```

The above statement would create the following node & edge instances:

1. **in-param:** p_film_id
2. **in-param:** p_store_id
3. **out-param:** p_film_count
4. **sql-statement:** film_in_stock/stmt1

5. **selects:** from stmt1 to inventory.inventory_id
6. **calls:** from stmt1 to inventory_in_stock
7. **as-param-to:** from inventory.inventory_id to inventory_in_stock.p_inventory_id
8. **filtered-with:** from p_film_id to stmt1
9. **filtered-with:** from p_store_id to stmt1
10. **filtered-with:** from stmt1 to inventory_in_stock.ret_var
11. **xformed-into:** from stmt1 to p_film_count

5.6. Conditional Update in Trigger

This example if for Sakila schema trigger: upd_film attached to film table

```
BEGIN
    IF (old.title != new.title) or (old.description != new.description) THEN
    UPDATE film_text
        SET title=new.title, description=new.description, film_id=new.film_id
        WHERE film_id=old.film_id;
    END IF;
    END;
```

The above statement in the trigger results in the creation of the following node & edge type instances:

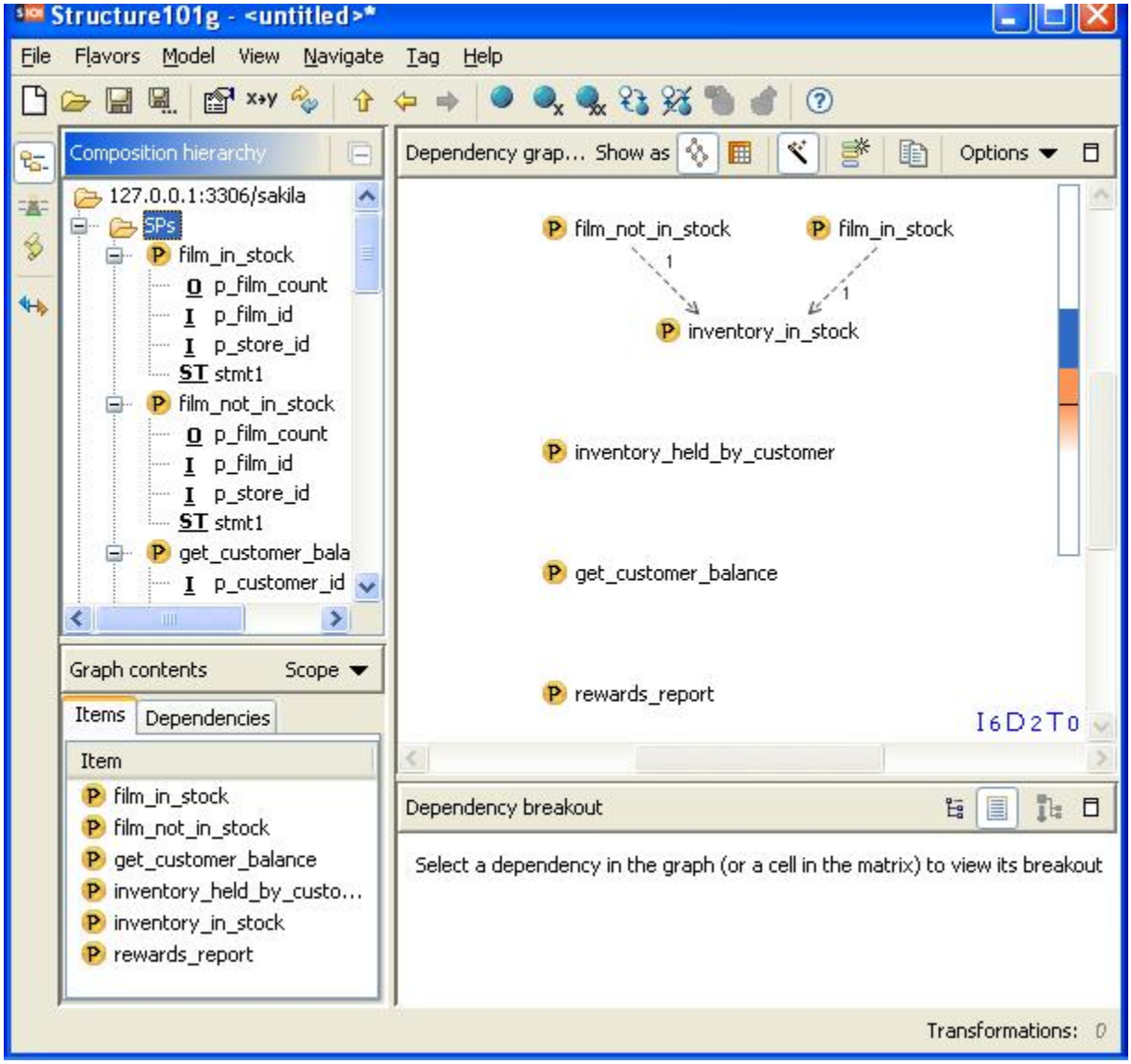
1. **attached-to:** from triggers/upd_film to film
2. **cond-updates:** from triggers/upd_film to film_text

Chapter 6. Exploration of Dependencies in UI

In this section we give examples & screen shots of Structure101g UI for exploring the dependencies extracted from SQL schema. The MySQL Sakila DB Schema is taken as example for illustration.

6.1. Intra-Group Dependencies

By intra-group dependencies, we mean those between tables only or between views only or between SPs only. There might not be any links between two triggers. The following screen shot shows the intra-group links for Views only for the Sakila schema. This can be viewed by selecting the `Views` group node for the spceific DB/schema.



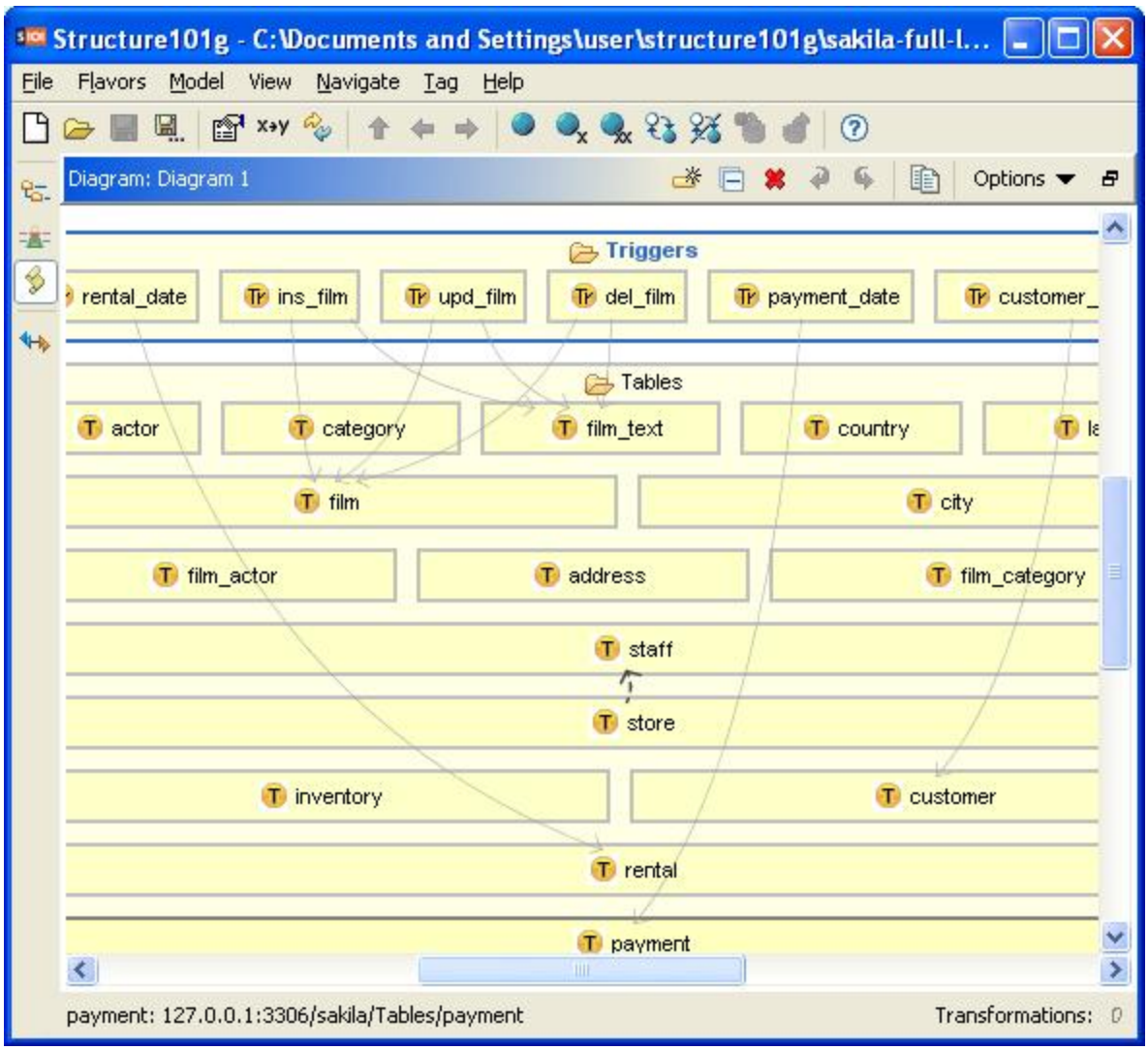
6.2. Triggers to Tables Inter-Group Dependencies

By inter-group dependencies, we mean those between triggers and tables or between views and tables, etc. There is no explicit way to explore these dependencies but can be created as a new diagram in the architecture perspective.

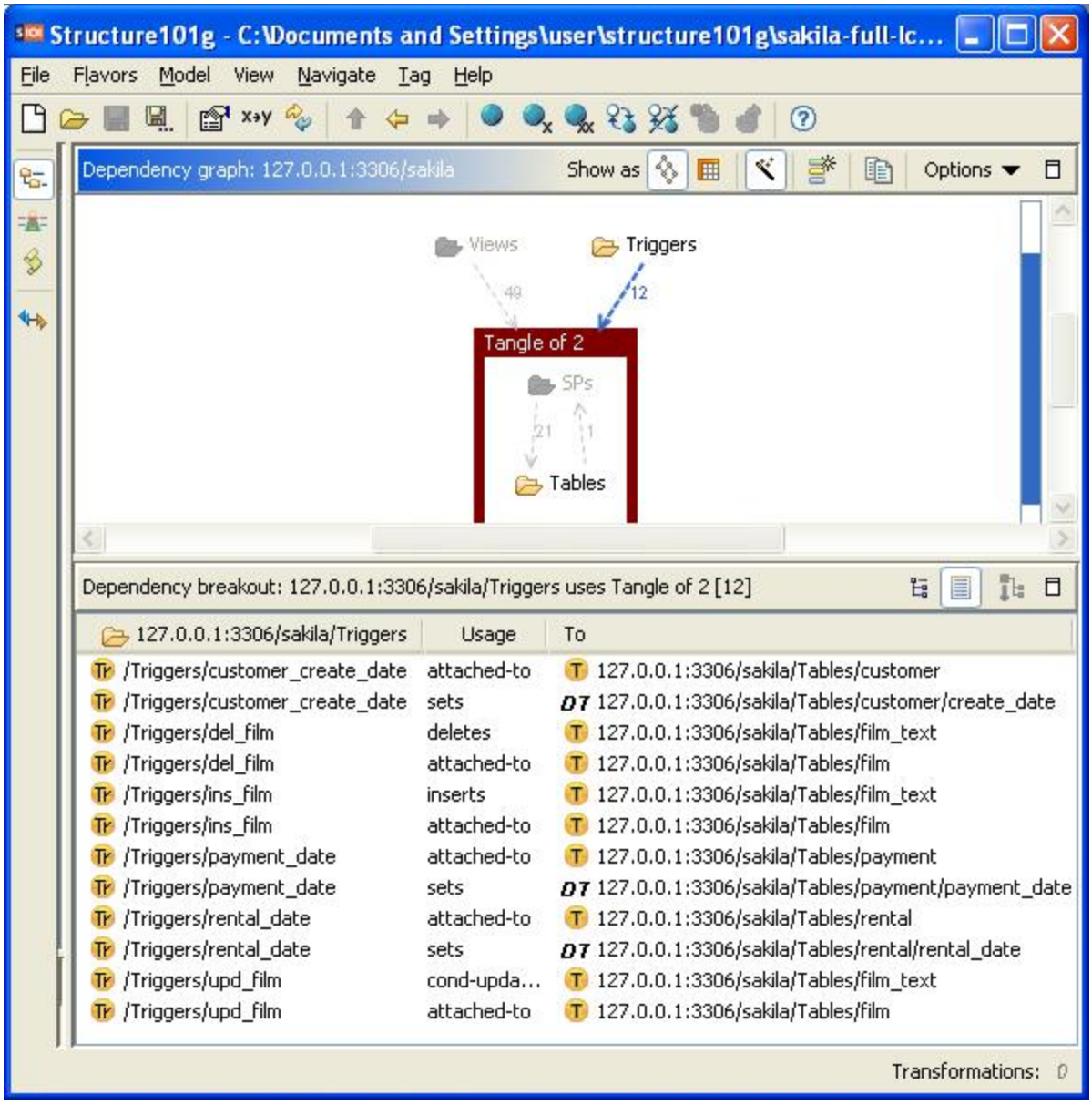
Here are the steps to create this architecture diagram:

1. Select the `Composition Perspective` and select the DB/Schema in the composition hierarchy.
2. Click on the `Create new architecture diagram based on displayed dependency graph` icon.
3. Select Views block & delete it
4. Select SPs block & delete it
5. Select the Views block
6. Click the drop-down `Options` at top-right corner in diagram pane & select `show dependencies on selected`. This will hide the dependencies between tables & show only from the triggers to the tables.

The following screen shot shows the Triggers to Tables dependencies for Sakila Schema.



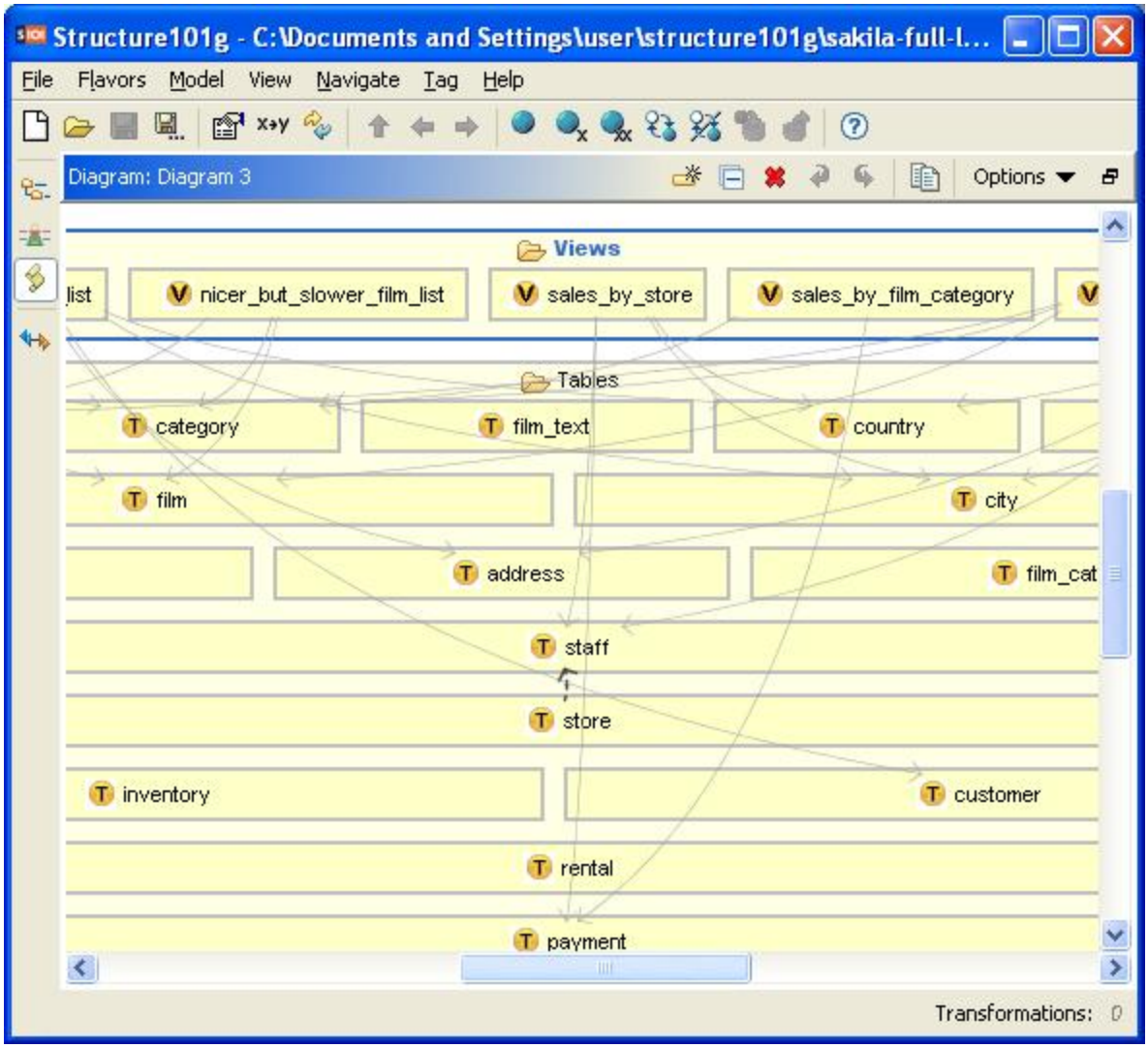
If one wants to view the list of inter-group dependencies from triggers to tables, the `Composition perspective` can be used as shown below for Sakila schema:



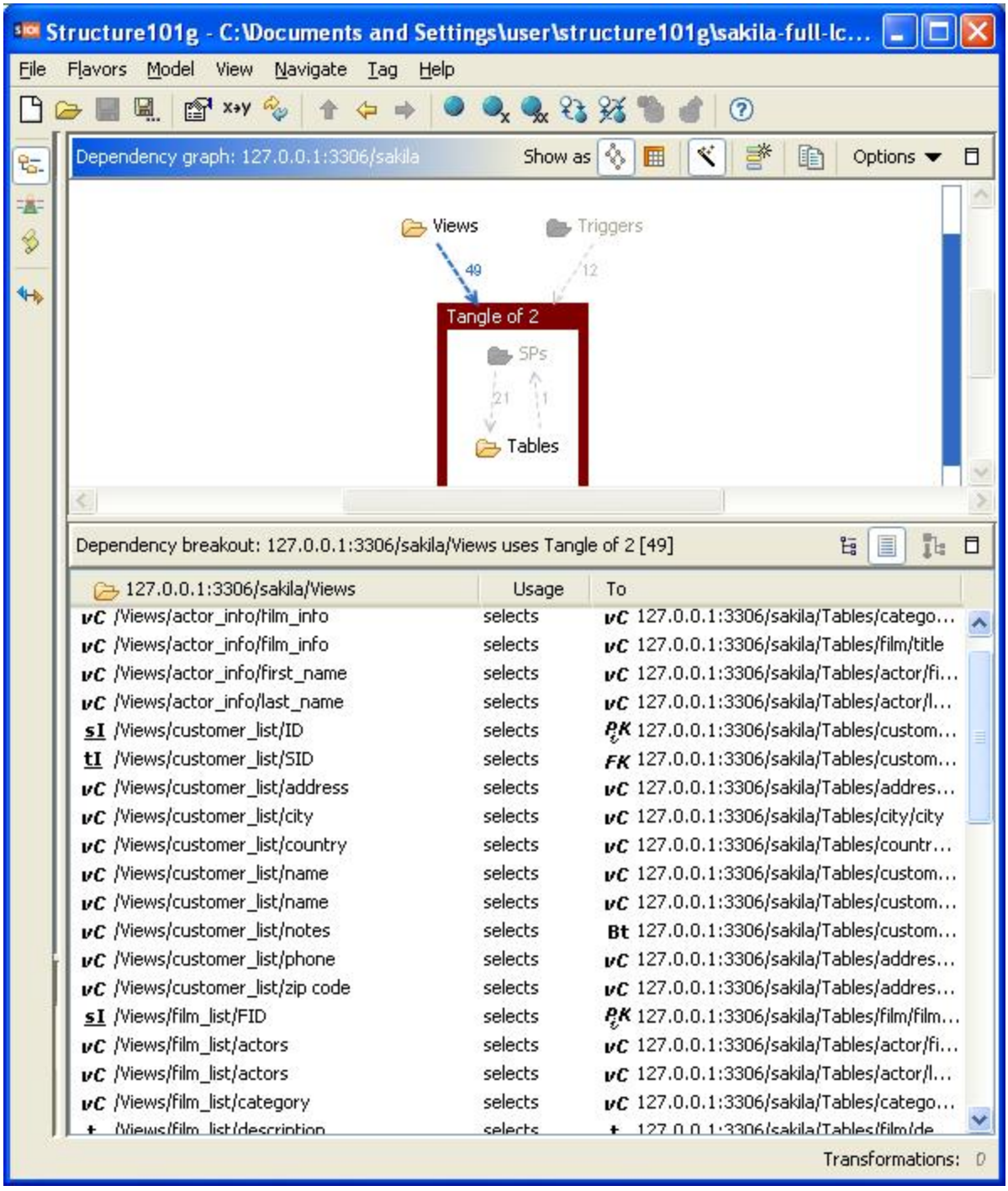
6.3. Views to Tables Inter-Group Dependencies

Similar steps can be followed as above for getting the views to tables dependencies.

The following screen shot shows the dependencies from Views to Tables for Sakila Schema.

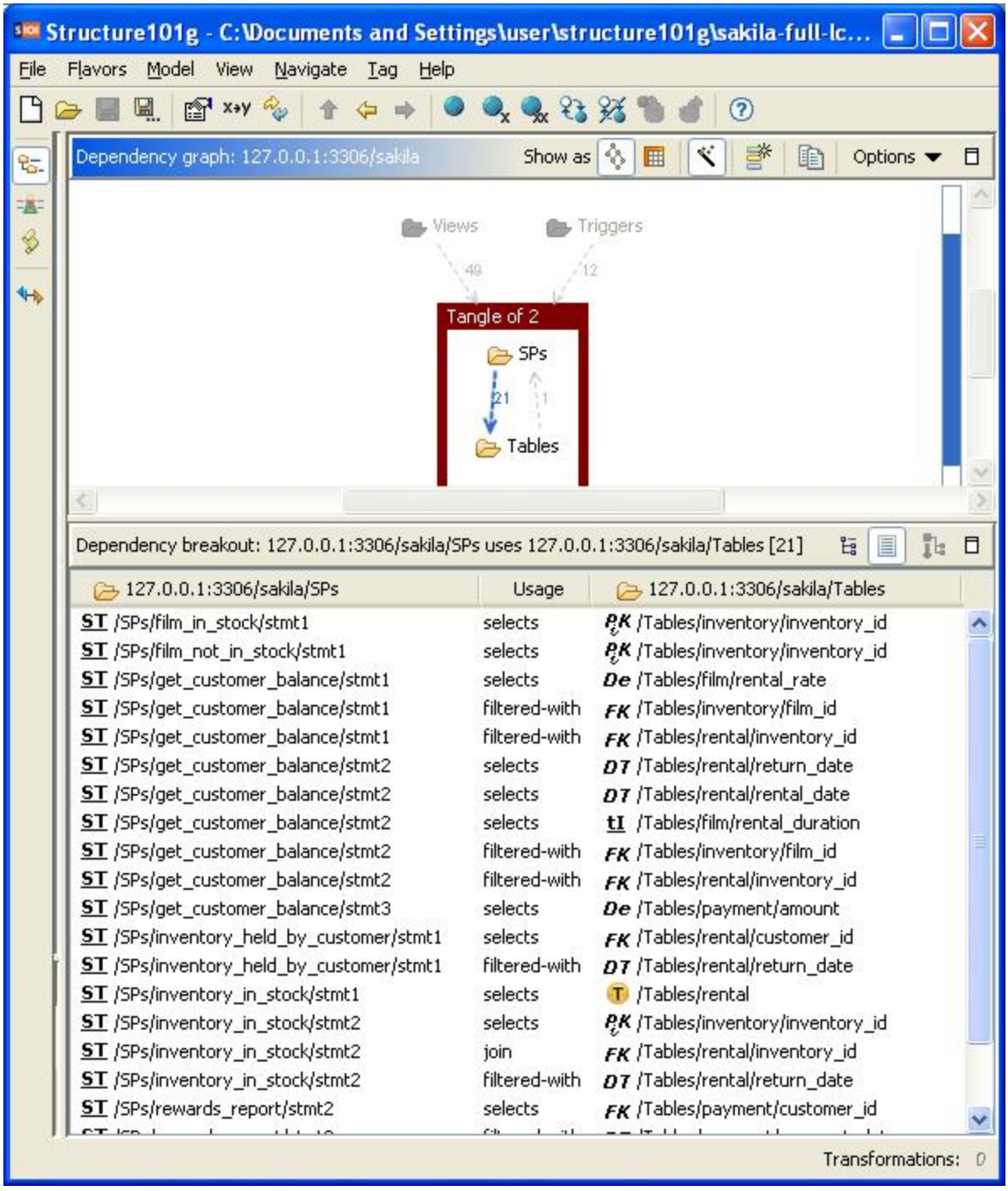


A listing of these links from Views to Tables can be got by selecting the aggregated edge in `Composition Perspective` which appears as below:



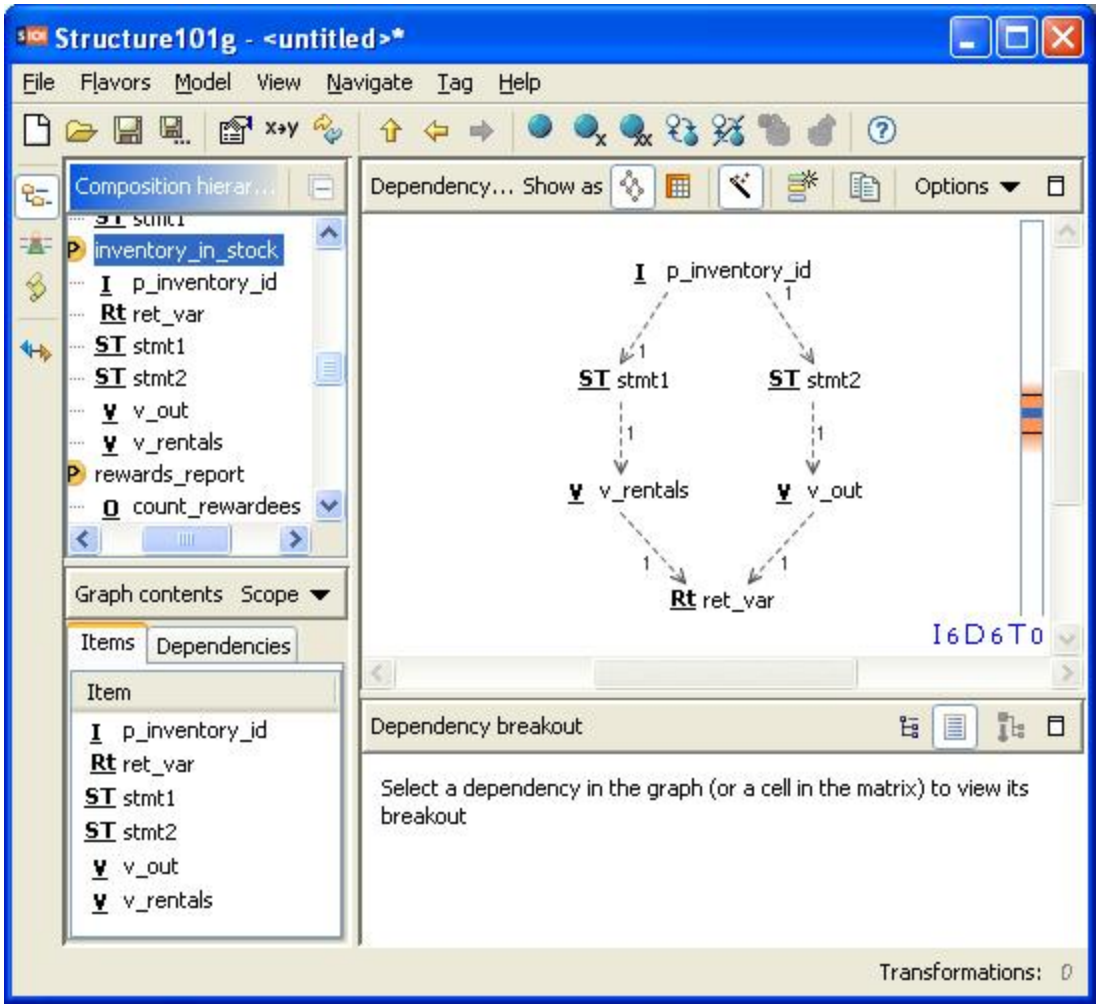
6.4. SPs to Tables Inter-Group Dependencies

Similar steps can be used for getting this group of links. The `Composition Perspective` can be used for getting a list of those dependencies as the screen shot below shows:

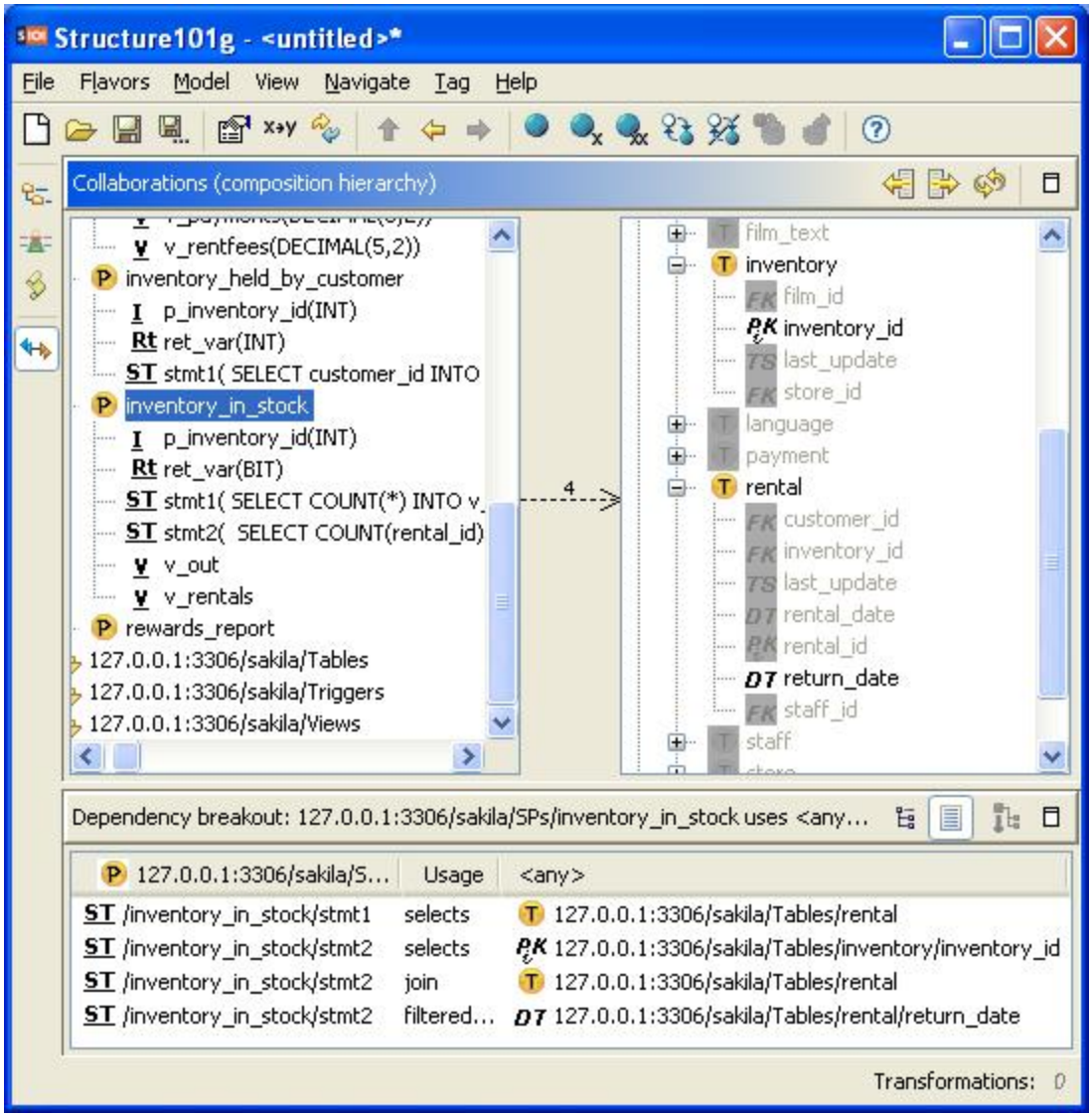


6.5. Drilling Down on a Stored Procedure

For each stored procedure, one can drill down into the details of sub-modules contained in it by selecting the SP in `Composition Perspective` hierarchy view or in `Collaboration Perspective`. The inventory_in_stock SP in the `Composition perspective` shows up as:



This only shows the internal dependencies of a SP and does not show the outside dependencies – meaning on tables, triggers, etc. - for the SP. To get that you need to switch to the Collaboration Perspective using: `Select inventory_in_stock -> Right-Click -> Select 'Go to suppliers of ....'` The following screen shot shows such external dependencies for the inventory_in_stock SP. The SQL statement also shows up as a tooltip.



Similarly one can go back to the `Composition Perspective` from `Collaboration Perspective` by doing: `Select inventory_in_stock -> Right-Click -> Select 'Go to composition of ...'`

Chapter 7. Known Issues

None for the implemented features.

Appendix A. Glossary

COUNT	*