Create an ER diagram for an Informix database

Gerd Kaluzinski (gerd.kaluzinski@de.ibm.com)  Certified IT Specialist  IBM Germany

Markus Holzbauer (holzbauer@de.ibm.com)  Advisory IT Specialist  IBM

Understanding the structure of a database is important to administrators as well as to developers. An image illustrates more details within seconds than any textual listing can deliver. This tutorial demonstrates how to use a tool that can deliver an entity-relationship graphical overview of an existing Informix® relational database and the contained objects.

Section 1. Before you start

The IBM Informix database provides a variety of administration tools, starting with the powerful onstat command that delivers detailed information about the server in ASCII format, and including GUI tools such as the OpenAdmin Tool (OAT). Developers using IBM Optim™ Development Studio already have access to information about the structures of the tables and references. With IBM Optim Development Studio, the structures can be changed or adapted. The diagram created with the scripts in this tutorial can only be displayed, but unlike Optim Development Studio, don't give you the capability to change the structure of database objects.

About this tutorial

This tutorial explains how to extract the information from the system catalog and how to use that information as the basis for graphical output that will give you insight into the database relationships. Building on these basics that show how to select and display the DDL structures, experienced Informix administrators will be able to extend the functionality of the sample script and adjust it to the needs of their own local Informix installations.
Objectives

You'll learn where to find information about database data definitions (DDL structures) and how to write select statements to extract this information from an existing Informix database. You will see how to format the selected data so you get an XML file that holds the information to create a graphical overview of the system.

Prerequisites

This tutorial is written for database administrators and application developers or programmers whose skill and experience are at a beginning to intermediate level of SQL and shell programming. You should have a general familiarity with using a Linux® or UNIX® command-line shell and a working knowledge of SQL. The knowledge of the Informix system catalog can help extend the sample scripts to meet individual needs.

System requirements

To run the examples here, you need a Linux or UNIX system with at least 10 MB of free disk space. To run it on Windows®, some minor changes are needed.

Create an ER diagram for an Informix database
query tool Informix dbaccess (provided by the server or separately installed) must be available.

The environment must be set correctly for connecting to the Informix instance. The user that runs these scripts must be able to connect to the database to be analyzed (with at least read access to the database). In addition, the user needs the ability to create files in the local working directory.

---

Section 2. Information from system catalog

System catalog

All the information needed to build the ER-Diagram can be found in the system catalog of the database. Every user who can connect to the database (connect permission) can select the DDL information from these system catalog tables using SQL.

The main tables are:

- systables
  - Name of the table
  - Owner of the table
  - Number of records
  - Number of pages used
- syscolumns
  - Names of the columns
  - Data types used
- sysconstraints
  - Columns in constraints
  - Constraint type
- sysindexes (view)
  - Columns of the primary keys and references
- sysreferences
  - Information about foreign keys
- systriggers
  - Information about triggers on tables
- syssyntable
  - Information about synonyms
- sysdepend
  - Information about table dependencies
Select tables and their columns

Let's have a look at the SQL statements that return the table name and the list of columns of that table. In addition, we select the number of rows (value at the last run of update statistics).

First, we have to create an SQL function to list the columns for a given table ID.
Listing 1. SQL function: Column list

```
CREATE FUNCTION __return_colnames__(tid INTEGER)
RETURNING LVARCHAR(30000);

DEFINE n LVARCHAR(30000);
DEFINE m LVARCHAR(30000);
LET n='';
LET m='';
FOREACH
    SELECT TRIM(colname) INTO n FROM syscolumns WHERE tabid = tid
    LET m = m || ' ' || n;
END FOREACH
RETURN m;
END FUNCTION;
```

Listing 2. Example result for `tabid 100`

```
(expression) customer_num fname lname company address1 address2 city state zipcode phone
```

Now we can write an SQL query to list the tables with columns. We use the SQL function created in Listing 1 to be able to list all columns in one output line.

Listing 3. SQL query: List tables with columns

```
SELECT
TRIM(a.tabname)
|| __return_colnames__(a.tabid)
|| (SELECT nrows FROM systables c WHERE a.tabid = c.tabid)
FROM systables a
WHERE tabid > 99 AND tabtype = 'T'
ORDER BY a.tabname;
```

Listing 4. Example result for tables with columns

```
(expression) "calendarpatterns cp_name cp_pattern 0.0000000000000000
(expression) calendartable c_id c_version c_refcount c_name c_calendar 11.00
0000000000000000
(expression) call_type call_code code_descr 5.0000000000000000
(expression) catalog catalog_num stock_num manu_code cat_descr cat_picture
cat_advert 74.0000000000000000
(expression) classes classid class subject 4.0000000000000000
...
```

This output shows one line for a table with all of its columns and the number of data records in the table. The next step is to analyze the referential constraints for each table. In this first example, we do the select only for the first column of the constraint. Later, if we have to get the information for all columns in a referential constraints, this statement has to be executed in a loop for part 1 until part 16 (maximum 16 columns within a foreign key).
Listing 5. Show referential constraints (only first column)

```sql
SELECT TRIM(b.tabname) || '.' || TRIM(d.colname) || ' ' || (SELECT tabname FROM systables WHERE tabid = e.ptabid) || '.' || (SELECT z.colname FROM sysconstraints w, systables x, sysindexes y, syscolumns z
WHERE w.tabid = e.ptabid AND w.constrtype = "P"
AND w.tabid = x.tabid
AND w.idlname = y.idlname
AND y.part1 = z.colno AND w.tabid = y.tabid AND w.tabid = z.tabid)
FROM sysconstraints a, systables b, sysindexes c, syscolumns d, sysreferences e
WHERE a.constrtype = "R"
AND a.idlname = c.idlname
AND c.part1 = d.colno
AND a.tabid = b.tabid
AND a.tabid = c.tabid
AND a.tabid = d.tabid
AND a.constrid = e.constrid
ORDER BY b.tabname
```

Listing 6. Show the result of the query

<table>
<thead>
<tr>
<th>expression</th>
<th>catalog.stock_num stock.stock_num</th>
</tr>
</thead>
<tbody>
<tr>
<td>expression</td>
<td>cust_calls.call_code call_type.call_code</td>
</tr>
<tr>
<td>expression</td>
<td>cust_calls.customer_num customer.customer_num</td>
</tr>
<tr>
<td>expression</td>
<td>customer_ts_data.customer_num customer.customer_num</td>
</tr>
<tr>
<td>expression</td>
<td>customer_ts_data.loc_ei_id ts_data.loc_ei_id</td>
</tr>
<tr>
<td>expression</td>
<td>geometry_columns.srid spatial_references.srid</td>
</tr>
</tbody>
</table>

Now we have all information to build the basic structures. The query results need to be enhanced with XML tags to be the base of a graphical output. In the next section, we add the XML syntax as text to these queries.

---

**Section 3. XML data and diagram**

In this section, we use the selected data to create the graphical output. To generate the picture of the ER-Diagram, we use the program dot (part of Graphviz, installed on most Linux distributions by default). The syntax for the XML as base of the ER diagram can be seen in the following listing.
Listing 7. Sample XML instructions as base of the diagram

```
digraph "Informix ER-Diagramm for Database 'test'" {
    graph [ label="\n\n\nINFORMIX ER-Diagramm for Database 'test'.\n\n Generated by ifx_schema2dot.sh\n\n\n labeljust="l" rankdir="RL" bgcolor="#f0f0f0" nodesep="0.5" ranksep="1.0" 
        fontname="Helvetica" fontsize="12" ];
    node [ fontname="Helvetica" fontsize="10" shape="plaintext" ];
    edge [ arrowsize="0.8" ];
    "table1" [label="<<TABLE BORDER="0" CELLBORDER="1" CELLSPLACING="0" BGCOLOR="#ffffff">
      <TR><TD BGCOLOR="#00649d" ALIGN="CENTER">table1</TD></TR>
      <TR><TD PORT="column1" BGCOLOR="#88ff88" ALIGN="LEFT">column1</TD></TR>
      <TR><TD PORT="column2" BGCOLOR="#82d1f5" ALIGN="LEFT">column2</TD></TR>
      <TR><TD ALIGN="CENTER" BGCOLOR="#f7f7f7">5 rows</TD></TR>
    </TABLE>> tooltip="table1"];
}
```

If we display the XML instructions from above, we get the following.

**Figure 3. Table structure**

![Table structure diagram]

In SQL, static character strings can be displayed along with the output of the query. We use this SQL feature to merge the selected values from the system tables with the static XML tags that define the structure needed for formatting.

To generate the XML syntax, we use the SQL queries from the first section and add some XML tags like "<TR>" as "fixed text." This change refers only to the selection part of the queries. These XML tags deliver the format information. To highlight the primary keys of each table, we change the background color of these columns by adding "BGCOLOR="#88ff88" ALIGN="LEFT">".

Listing 8. SQL function with XML enhancements

```
CREATE FUNCTION __return_colnames__(tid INTEGER)
RETURNING LVARCHAR(30000);

DEFINE n LVARCHAR(30000);
DEFINE m LVARCHAR(30000);
LET n='';
LET m='';
FOREACH
```
All columns are tagged as row elements of a table. The columns of the primary keys are identified by the `constrype 'P'` in `sysconstraints` and are marked with a different color as background.

**Listing 9. The statement now delivers valid XML code**

```
<select ' '<tr><td port="" || trim(colname) || '</td></tr>' INTO n
FROM syscolumns WHERE tabid = tid
LET m = m || ' ' || n;
END FOREACH
RETURN m;
END FUNCTION;
```

Some format information is still missing, such as font information and the tags for the table (i.e., `<TABLE>`).

**Listing 10. SQL statement for tables, including column list with XML extensions**

```sql
SELECT ' '</td port="" || trim(a.tabname) </td>' '<tr border="0" cellpadding="1" cellspacing="0" bgcolor="#ffffff"><td bgcolor="#00649d" align="center"><font color="#ffffff">'</ || trim(a.tabname) '</font></td></tr>'
|| <table border="0" cellpadding="1" cellspacing="0" bgcolor="#ffffff">'
|| ' ' || _return_colnames__(a.tabid)
|| (select '<tr><td align="center" bgcolor="#f7f7f7">''' || c.nrows::int8 || ' rows</td></tr>'
FROM systables c WHERE a.tabid = c.tabid)
|| '</table> tooltip="''
|| trim(a.tabname)
|| ' ''
FROM systables a
WHERE tabid > 99 AND tabtype = 'T'
ORDER BY a.tabname;
```
This query delivers the complete layout for the tables to be displayed. We select only the tables with `tabid` starting with 100, where the user-defined tables begin. Below that value, the tables are internal tables. In this part, we only select real tables and restrict the selection to `tabtype 'T'`. Other tabtypes in systables represent views, synonyms or other related information.

**Listing 11. Output for table customer**

```
"customer" [label=<<TABLE BORDER="0" CELLBORDER="1" CELLSPACING="0" BGCOLOR="#ffffff">
  <TR><TD BGCOLOR="#00649d" ALIGN="CENTER"><FONT COLOR="#ffffff">customer</FONT></TD></TR>
  <TR><TD PORT="customer_num" BGCOLOR="#88ff88" ALIGN="LEFT">customer_num</TD></TR>
  <TR><TD PORT="fname" BGCOLOR="#82d1f5" ALIGN="LEFT">fname</TD></TR>
  <TR><TD PORT="lname" BGCOLOR="#82d1f5" ALIGN="LEFT">lname</TD></TR>
  <TR><TD PORT="company" BGCOLOR="#82d1f5" ALIGN="LEFT">company</TD></TR>
  <TR><TD PORT="address1" BGCOLOR="#82d1f5" ALIGN="LEFT">address1</TD></TR>
  <TR><TD PORT="address2" BGCOLOR="#82d1f5" ALIGN="LEFT">address2</TD></TR>
  <TR><TD PORT="city" BGCOLOR="#82d1f5" ALIGN="LEFT">city</TD></TR>
  <TR><TD PORT="state" BGCOLOR="#82d1f5" ALIGN="LEFT">state</TD></TR>
  <TR><TD PORT="zipcode" BGCOLOR="#82d1f5" ALIGN="LEFT">zipcode</TD></TR>
  <TR><TD PORT="phone" BGCOLOR="#82d1f5" ALIGN="LEFT">phone</TD></TR>
  <TR><TD ALIGN="CENTER" BGCOLOR="#f7f7f7">28 rows</TD></TR>
</TABLE>> tooltip="customer"];
```

Now we add a header with the information about the database and add some format and layout information. After this step, we can generate the ER diagram showing the tables and columns.

**Listing 12. Add style information**

```
$DOT_FILE="$DATABASE".dot
cat > $DOT_FILE <<EOF
digraph "Informix ER-Diagramm for Database '$DATABASE'" {
  graph [ label="INFORMIX ER-Diagramm for Database '$DATABASE'.
       Generated by ifx_schema2dot.sh\n    labeljust="l" rankdir="RL"
    bgcolor="#f0f0f0" nodeseq="0.5" ranksep="1.0"
    fontname="Helvetica" fontsize="12" ];
  node [ fontname="Helvetica" fontsize="10" shape=plaintext ];
  edge [ arrowsize="0.8" ];
EOF
```

We have created the header and can combine it with the selected table information from the scripts, where we have generated the format information for our database tables. We simply have to put all the parts together using the shell command:

```
cat t1.unl >> $DOT_FILE

cat r1.unl >> $DOT_FILE

cat s1.unl >> $DOT_FILE
```

The file `$DOT_FILE` contains the information to create a picture by using:

```
dot -T png -o $DATABASE.jpg $DATABASE.dot
```
The diagram now contains the tables, their columns, and the columns included in a primary key marked with green background. Still missing are the referential constraints in this database, so we have to add another part that draws the arrows.

**Listing 13. SQL query to get the foreign keys**

```sql
UNLOAD TO 'r$X.unl' DELIMITER ';
SELECT 
    TRIM(b.tabname) 
    || ':"' || d.colname || ':"w -> "' 
    || (SELECT tabname FROM systables WHERE tabid = e.ptabid) 
    || ':"'
    || (SELECT z.colname FROM sysconstraints w, systables x, sysindexes y, syscolumns z 
        WHERE w.tabid = e.ptabid AND w.constrtype = "P" AND w.tabid = x.tabid  
        AND w.idxname = y.idxname 
        AND y.part$X = z.colno AND w.tabid = y.tabid AND w.tabid = z.tabid) 
    || ':"e [arrowhead=none dir=back arrowtail=crowodot]' 
FROM sysconstraints a, systables b, sysindexes c, syscolumns d, sysreferences e 
WHERE a.constrtype = "R" 
AND a.idxname = c.idxname
```

Create an ER diagram for an Informix database
AND c.part$X = d.colno
AND a.tabid = b.tabid
AND a.tabid = c.tabid
AND a.tabid = d.tabid
AND a.constrid = e.constrid

The result of this statement is a row for every connection among columns of different tables. The query has to be run for all the 16 possible elements of the foreign key. The output looks like the listing below.

Listing 14. List of the foreign keys

```
"catalog":"stock_num":w ->
"stock","stock_num":e [arrowhead=none dir=back arrowtail=crowodot];
"cust_calls","call_code":e [arrowhead=none dir=back arrowtail=crowodot];
"call_type","call_code":e [arrowhead=none dir=back arrowtail=crowodot];
"customer","customer_num":e [arrowhead=none dir=back arrowtail=crowodot];
"customer_ts_data","customer_num":w ->
"customer","customer_num":e [arrowhead=none dir=back arrowtail=crowodot];
"customer_ts_data","loc_esi_id":w ->
"ts_data","loc_esi_id":e [arrowhead=none dir=back arrowtail=crowodot];
"geometry_columns","srid":w ->
"spatial_references","srid":e [arrowhead=none dir=back arrowtail=crowodot];
"items","stock_num":w ->
"stock","stock_num":e [arrowhead=none dir=back arrowtail=crowodot];
"items","order_num":w ->
"orders","order_num":e [arrowhead=none dir=back arrowtail=crowodot];
"orders","customer_num":w ->
"customer","customer_num":e [arrowhead=none dir=back arrowtail=crowodot];
"stock","manu_code":w ->
"manufact","manu_code":e [arrowhead=none dir=back arrowtail=crowodot];
"catalog","manu_code":w ->
"stock","manu_code":e [arrowhead=none dir=back arrowtail=crowodot];
"customer_ts_data","measure_unit":w ->
"ts_data","measure_unit":e [arrowhead=none dir=back arrowtail=crowodot];
"items","manu_code":w ->
"stock","manu_code":e [arrowhead=none dir=back arrowtail=crowodot];
"customer","measure_unit":w ->
"customer_ts_data","direction":w ->
"ts_data","direction":e [arrowhead=none dir=back arrowtail=crowodot];
```

Now we can put the information about the foreign keys at the end of the table instructions, and we get the complete diagram of the tables and relations among the tables. At this point, the main part of the construction of the ER-Diagram is done. Synonyms, views, triggers, table owners, and many other things could be of interest for database analysis. In the following section, you will see how this additional information can be added to our newly created diagram.
Figure 5. Database tables and constraints

(View a larger version of Figure 5.)
Section 4. More information and other database objects

There is a lot of more information that can be read from the system tables. There are also more things that belong to a database than tables and constraints. The decision of what to display within the diagram is difficult. It may not be a good idea to show all possible information and database objects within one diagram. At the other side, there might be information that is missing for a specific view.

At this point, you should decide what to include and what to eliminate from the scripts provided. Less can be more, so a restriction to show only a subset of features and tables might be helpful.

Data types

The additional information of the data types used in the table can be very useful. This information can be selected from the table syscolumns, but the values have to be decoded to a human readable form. The additional information of the data types needs more space, so this should only be an optional feature.

Figure 6. Data types of the columns

<table>
<thead>
<tr>
<th>customer_num</th>
<th>SERIAL NOT NULL</th>
</tr>
</thead>
<tbody>
<tr>
<td>fname</td>
<td>CHAR (15)</td>
</tr>
<tr>
<td>lname</td>
<td>CHAR (15)</td>
</tr>
<tr>
<td>company</td>
<td>CHAR (20)</td>
</tr>
<tr>
<td>address1</td>
<td>CHAR (20)</td>
</tr>
<tr>
<td>address2</td>
<td>CHAR (20)</td>
</tr>
<tr>
<td>city</td>
<td>CHAR (15)</td>
</tr>
<tr>
<td>state</td>
<td>CHAR (2)</td>
</tr>
<tr>
<td>zipcode</td>
<td>CHAR (5)</td>
</tr>
<tr>
<td>phone</td>
<td>CHAR (18)</td>
</tr>
</tbody>
</table>

Triggers

The information if there are triggers for insert, update, delete, or select can be displayed as one additional row in the listing of a table, so this information can be added without influence of the appearance.
Synonyms

Alias names for tables, the synonyms may be helpful to find the tables, used in statements. This information can be selected using the tables syssyntable and systables. These synonyms don’t have their own columns, and are only alternate names of tables.

Figure 7. Synonyms referencing a table

Views

Views depend on one or more tables, so we can show this relation between a view and the tables, where it is built from. The list of columns makes no sense because a view can contain a lot of functionality, such as SUM, AVG, SUBSTR(), so there is no option to show the relationship between the columns of a view and the columns of the tables.

Figure 8. Views referencing one or more tables
Index columns

Performance analysis often needs to know what columns are at the first position of an index. In the sample script, these columns are marked with an HTML tag for a different color.

Table owner

Sometimes objects of a database have different owners. To manage permissions, it is important to know who is the owner of the objects. So we added the value of systables.owner to the table name to show who has created the object.

Subset of tables

After listing the features that can be displayed, we provide a way to reduce the number of tables displayed. In most cases, there is a set of tables, included in a statement, to be analyzed; you do not need the several hundreds of tables contained in the database. Most times, it is useful to select only a subset of tables that are of interest. We added this option in our script to reduce the diagram to a defined list of table names. In this sample script, we included the option to select only one table, using the `-t tablename` option, or to select a list of tables using `-f file_with_tablenames`.

Summary

A relational database provides everything needed to get the information about the structures of the objects and references within the database.

Using this information and adding some information for the display of the objects, every database can be displayed in a graphical mode.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
<th>Download method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample shell scripts</td>
<td>ifx_er_diagram.zip</td>
<td>10KB</td>
<td>HTTP</td>
</tr>
<tr>
<td>Sample xml-file and stores</td>
<td>ifx_er_data.zip</td>
<td>50KB</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

*Information about download methods*
Resources

Learn

- See "Using the Informix System Catalogs" to learn how to exploit the variety of information in the system catalog.
- Hints for SQL scripts are available in the Informix administrators blog.
- Learn more about Information Management at the developerWorks Information Management zone. Find technical documentation, how-to articles, education, downloads, product information, and more.
- Stay current with developerWorks technical events and webcasts.
- Follow developerWorks on Twitter.

Get products and technologies

- Build your next development project with IBM trial software, available for download directly from developerWorks.
- Now you can use DB2 for free. Download DB2 Express-C, a no-charge version of DB2 Express Edition for the community that offers the same core data features as DB2 Express Edition and provides a solid base to build and deploy applications.

Discuss

- Participate in the discussion forum for this content.
- Check out the developerWorks blogs and get involved in the developerWorks community.
About the authors

Gerd Kaluzinski

Gerd Kaluzinski is a Certified IT Specialist who works with Informix since 1992. He works as consultant at customer sites in IBM Softwaregroup Services and as trainer for IBM education. His specialties are Informix replication (HDR/RSS/ER/GRID), Datablades (for example, TimeSeries, Excalibur) and Stored Procedures. He is the editor of the monthly "German IBM Informix Newsletter."

Markus Holzbauer

Markus Holzbauer is advisory IT specialist and lead system engineer for Informix at GTS Service Delivery/Server Systems Operations, Germany. He has worked with Informix since Version 4 as consultant, supporter, DBA, systems administrator, and other roles. Time permitting, he writes SQL scripts published for the Informix administrators blog (https://www.ibm.com/developerworks/mydeveloperworks/blogs/informix_admins_blog/?lang=en_us).

© Copyright IBM Corporation 2012  
Trademarks  
(www.ibm.com/developerworks/ibm/trademarks/)