Table of Contents

Introduction

This document is the accompanying material to the paper:

Eessaar, E. "On Query-based Search of Possible Design Flaws of SQL Databases". SCSS 12.

In this document, we present SQL statements that one can use to detect the occurrences of the SQL database design antipatterns, which have been described by Karwin [1]. Each such antipattern describes a particular type of database design flaw. Therefore, the results of the queries in case of a database point to the possible design flaws of the database.

The statements use the views of the Information Schema and hence the data from the system catalog of database. Some of the queries also read data from other schemas. The results of the queries will be ordered by schema name and table name. We have created and tested the statements based on the PostgreSQL ™ 9.2 database management system (DBMS). The statements take into account the specifics of the DBMS. Firstly, we have to take the specifics into account in the search conditions of queries. For instance, we have to bear in mind that PostgreSQL™ sometimes rewrites Boolean expressions of check constraints and provides some system-defined types that are not specified in the SQL standard. In addition, the proposed queries use some implementation-specific functions, operators, and modules. Moreover, we use the procedural language PL/pgSQL to write table functions that return a table with zero or more rows and one or more columns.

1 Queries that are used to detect design flaws

For each pattern, we present its name, short informal description of the detection approaches, and SQL statements that implement the approaches.

We do not claim that the proposed detection approaches and their accompanying queries are the only possible approaches to detect the occurrences of the antipatterns by using queries.

Pattern: Format Comma-Separated Lists

Find all the columns of base tables with the type VARCHAR or TEXT and for each found column *c* try to determine, based on the actual values in the column, whether *c* contains lists of values. One may try to do it by finding out, whether *c* contains values that themselves contain separation characters like “,” or “;”. If *c* is defined in terms of a domain, the base type of which is VARCHAR or TEXT, then the function analyses the column as well. The search conditions of the dynamically generated SELECT statements contain regular expressions.

```
CREATE OR REPLACE FUNCTION f_check_format_comma_separated_list()
RETURNS TABLE (table_schema VARCHAR(128), table_name VARCHAR(128), column_name
VARCHAR(128)) AS $$
DECLARE
      sql_stmt TEXT;
      cnt BIGINT;
      varchar_columns RECORD;
BEGIN
      RAISE NOTICE 'Detecting possible occurrences of the antipattern "Format
Comma-Separated Lists"';
      FOR varchar_columns IN SELECT c.table_schema, c.table_name, c.column_name
FROM INFORMATION_SCHEMA.columns AS c INNER JOIN INFORMATION_SCHEMA.tables AS t USING
(table_schema, table_name) WHERE c.data_type IN ('character varying', 'text') AND
t.table_type='BASE TABLE' ORDER BY c.table_schema, c.table_name LOOP
            table_schema:= varchar_columns.table_schema;
            table_name:= varchar_columns.table_name;
            column_name:= varchar_columns.column_name;
            sql_stmt:='SELECT Count(' || quote_ident(column_name) || ') AS c FROM
' || quote_ident(table_schema) ||'.' || quote_ident(table_name) || ' WHERE '||
quote_ident(column_name) || '~''(.+)([,;]{1}.+)+''';
            EXECUTE sql_stmt INTO cnt;
            IF cnt>0 THEN
                  RETURN NEXT;
            END IF;
      END LOOP;
      RAISE NOTICE 'Detection completed';
      RETURN;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER
SET search_path = information_schema, pg_temp;
```

```
SELECT * FROM f_check_format_comma_separated_list();
```

Pattern: Always Depend on One's Parent

Find the foreign key constraints where the referencing table and the referenced table are the same.

```
SELECT DISTINCT rc.constraint_schema AS table_schema, fk_table.table_name,
rc.constraint_name FROM (INFORMATION_SCHEMA.referential_constraints AS rc INNER
JOIN INFORMATION_SCHEMA.key_column_usage AS fk_table
ON (rc.constraint_schema=fk_table.constraint_schema
AND rc.constraint_name=fk_table.constraint_name)) INNER JOIN
INFORMATION_SCHEMA.constraint_table_usage AS pk_table ON
(rc.unique_constraint_schema=pk_table.constraint_schema
AND rc.unique_constraint_name=pk_table.constraint_name)
WHERE fk_table.table_schema=pk_table.table_schema AND
fk_table.table_name=pk_table.table_name
ORDER BY rc.constraint_schema, fk_table.table_name;
```

Pattern: One Size Fits All

Find base tables, the primary key of which is a simple key that consist of a column with the name *id* (the name is case insensitive) and with an exact numeric type: NUMERIC, DECIMAL, SMALLINT, INTEGER, or BIGINT. The query also detects base tables where the *id* column is defined in terms of a domain.

```
SELECT tc.constraint_schema AS table_schema, tc.table_name
FROM INFORMATION_SCHEMA.table_constraints AS tc
WHERE tc.constraint_type='PRIMARY KEY' AND
(SELECT Count(*) AS cnt
FROM INFORMATION_SCHEMA.constraint_column_usage AS ccu
WHERE ccu.constraint_schema=tc.constraint_schema AND
ccu.constraint_name=tc.constraint_name)=1 AND
'ID'=(SELECT Upper(column_name) AS col
FROM INFORMATION_SCHEMA.constraint_column_usage AS ccu
WHERE ccu.constraint_schema=tc.constraint_schema AND
ccu.constraint_name=tc.constraint_name) AND
(SELECT data_type
FROM INFORMATION_SCHEMA.columns AS c
WHERE tc.constraint_schema=c.table_schema
AND tc.table_name=c.table_name
AND Upper(c.column_name)='ID'
) IN ('smallint','integer', 'bigint', 'numeric', 'decimal')
ORDER BY tc.constraint_schema, tc.table_name;
```

Pattern: Leave Out the Constraints

Find base tables that do not participate in any referential constraint (as the referenced table or as the referencing table).

```
SELECT table_schema, table_name
FROM INFORMATION_SCHEMA.tables
WHERE table_type='BASE TABLE' AND
(table_schema, table_name) NOT IN
(SELECT fk_table.table_schema, fk_table.table_name
FROM INFORMATION_SCHEMA.referential_constraints AS rc INNER JOIN
INFORMATION_SCHEMA.key_column_usage AS fk_table
ON (rc.constraint_schema=fk_table.constraint_schema
AND rc.constraint_name=fk_table.constraint_name)
UNION SELECT pk_table.table_schema, pk_table.table_name
FROM INFORMATION_SCHEMA.referential_constraints AS rc INNER JOIN
INFORMATION_SCHEMA.constraint_table_usage AS pk_table ON
(rc.constraint_schema=pk_table.constraint_schema
AND rc.constraint_name=pk_table.constraint_name))
ORDER BY table_schema, table_name;
```

Find pairs of columns of different base tables where the names and types of the columns are the same and there is no referential constraint that connects these columns. In each pair, at least one of the columns is the primary key column or a unique column of a base table. If *x* is the referencing column and *y* is the referenced column in the referential constraint, then the result does not contain combination (y, x) as well as (x, y).

```
SELECT key_columns.table_schema AS primary_table_schema, key_columns.table_name AS
primary_table_name, key_columns.column_name AS primary_column_name,
all_columns.table_schema AS dependent_column_schema, all_columns.table_name AS
dependent_table_name, all_columns.column_name AS dependent_column_name
FROM (SELECT kcu.table_schema, kcu.table_name, kcu.column_name, c.data_type
FROM INFORMATION_SCHEMA.key_column_usage AS kcu INNER JOIN
INFORMATION_SCHEMA.columns AS c
USING (table_schema, table_name, column_name)
WHERE (constraint_schema, constraint_name) IN
(SELECT constraint_schema, constraint_name
FROM INFORMATION_SCHEMA.table_constraints
WHERE constraint_type IN ('PRIMARY KEY','UNIQUE'))) AS key_columns,
(SELECT table_schema, table_name, column_name, data_type
FROM INFORMATION_SCHEMA.columns WHERE
(table_schema, table_name) IN (SELECT table_schema, table_name
FROM INFORMATION_SCHEMA.tables WHERE table_type='BASE TABLE')) AS all_columns
WHERE (key_columns.column_name=all_columns.column_name AND
key_columns.data_type=all_columns.data_type)
AND (NOT (key_columns.table_schema=all_columns.table_schema
AND key_columns.table_name=all_columns.table_name))
EXCEPT
(SELECT kcu_primary.table_schema, kcu_primary.table_name, kcu_primary.column_name,
kcu_dependent.table_schema, kcu_dependent.table_name, kcu_dependent.column_name
FROM INFORMATION_SCHEMA.key_column_usage AS kcu_dependent INNER JOIN
(INFORMATION_SCHEMA.referential_constraints AS rc INNER JOIN
INFORMATION_SCHEMA.key_column_usage AS kcu_primary ON (rc.unique_constraint_schema
=kcu_primary.constraint_schema) AND (rc.unique_constraint_name =
kcu_primary.constraint_name)) ON (kcu_dependent.constraint_schema =
rc.constraint_schema)
AND (kcu_dependent.constraint_name = rc.constraint_name)
UNION
SELECT kcu_dependent.table_schema, kcu_dependent.table_name,
kcu_dependent.column_name,
kcu_primary.table_schema, kcu_primary.table_name, kcu_primary.column_name
FROM INFORMATION_SCHEMA.key_column_usage AS kcu_dependent INNER JOIN
(INFORMATION_SCHEMA.referential_constraints AS rc INNER JOIN
INFORMATION_SCHEMA.key_column_usage AS kcu_primary ON (rc.unique_constraint_schema
=kcu_primary.constraint_schema) AND (rc.unique_constraint_name =
kcu_primary.constraint_name)) ON (kcu_dependent.constraint_schema =
rc.constraint_schema)
AND (kcu_dependent.constraint_name = rc.constraint_name))
ORDER BY primary_table_schema, primary_table_name;
```

Pattern: Use a Generic Attribute Table

Find the base tables, the name of which contains specific substrings (like “object”) that have been suggested as the possible table names in case of the design.

```
SELECT table_schema, table_name
FROM INFORMATION_SCHEMA.tables
WHERE
table_type='BASE TABLE' AND
(table_name LIKE '%object_type%' OR
table_name LIKE '%entity_type%' OR
table_name LIKE '%thing_class%' OR
table_name LIKE '%class%' OR
table_name LIKE '%attribute%' OR
table_name LIKE '%attribute_assignment%' OR
table_name LIKE '%object%' OR
table_name LIKE '%entity%' OR
table_name LIKE '%entities%' OR
table_name LIKE '%thing%' OR
table_name LIKE '%value%' OR
table_name LIKE '%object_attribute%' OR
table_name LIKE '%property%' OR
table_name LIKE '%properties%' OR
table_name LIKE '%relationship%' OR
table_name LIKE '%link%')
ORDER BY table_schema, table_name;
```

Pattern: Use Dual-Purpose Foreign Key

Find pairs of different columns of the same base table where the names of the columns are similar (for instance, the Levenshtein distance between the names of the columns is below a certain threshold. In this query, the Levenshtein distance between the two names should not be bigger than 4), one of the columns has an associated check constraint that limits values in the column, and another column does not participate in any referential constraint as the referencing column. For this task, we use the *fuzzystrmatch* module of PostgreSQL ™ that provides several functions to determine similarities and distance between strings.

```
CREATE EXTENSION IF NOT EXISTS fuzzystrmatch;
```

```
SELECT table1.table_schema, table1.table_name, table2.column_name AS
polymorphic_column, table1.column_name AS classifier_column
FROM (SELECT table_schema, table_name, column_name
FROM INFORMATION_SCHEMA.constraint_column_usage
WHERE (constraint_schema, constraint_name) IN
(SELECT constraint_schema, constraint_name
FROM INFORMATION_SCHEMA.check_constraints
WHERE check_clause~*'^.+=.*ANY.*[(].*ARRAY[[].+[])].*$')
UNION
SELECT cdu.table_schema, cdu.table_name, cdu.column_name
FROM INFORMATION_SCHEMA.column_domain_usage AS cdu INNER JOIN
INFORMATION_SCHEMA.tables AS t USING (table_schema, table_name)
WHERE t.table_type='BASE TABLE' AND (domain_schema, domain_name) IN
(SELECT domain_schema, domain_name
FROM INFORMATION_SCHEMA.domain_constraints
WHERE (constraint_schema, constraint_name) IN
(SELECT constraint_schema, constraint_name
FROM INFORMATION_SCHEMA.check_constraints
WHERE check_clause~*'^.+=.*ANY.*[(].*ARRAY[[].+[])].*$'))) AS table1,
(SELECT c.table_schema, c.table_name, c.column_name
FROM INFORMATION_SCHEMA.columns AS c INNER JOIN INFORMATION_SCHEMA.tables AS t
USING (table_schema, table_name)
WHERE t.table_type='BASE TABLE') AS table2
WHERE table1.table_schema=table2.table_schema AND
table1.table_name=table2.table_name
AND table1.column_name<>table2.column_name AND
levenshtein(table1.column_name,table2.column_name)<=4 AND
(table2.table_schema, table2.table_name, table2.column_name) NOT IN (SELECT
kcu_dependent.table_schema, kcu_dependent.table_name, kcu_dependent.column_name
FROM INFORMATION_SCHEMA.key_column_usage AS kcu_dependent
INNER JOIN INFORMATION_SCHEMA.referential_constraints AS rc ON
(kcu_dependent.constraint_schema = rc.constraint_schema) AND
(kcu_dependent.constraint_name = rc.constraint_name))
ORDER BY table1.table_schema, table1.table_name;
```

Pattern: Create Multiple Columns

Find pairs of different columns of the same base table that have the same type. In addition, after the removal of numbers from the names of the columns the names must be equal in case of each pair. If a column is specified in terms of a domain, then the query takes into account the base type of the domain.

```
SELECT table1.table_schema AS table_schema, table1.table_name AS table_name,
table1.column_name as column1, table2.column_name AS column2, table1.type AS
data_type
FROM (SELECT c.table_schema, c.table_name, c.column_name, c.data_type ||
coalesce(c.character_maximum_length::text, c.numeric_precision ||'.'||
c.numeric_scale, '0') AS type
FROM INFORMATION_SCHEMA.columns AS c INNER JOIN INFORMATION_SCHEMA.tables AS t
USING (table_schema, table_name)
WHERE t.table_type='BASE TABLE') AS table1,
(SELECT c.table_schema, c.table_name, c.column_name, c.data_type ||
coalesce(c.character_maximum_length::text, c.numeric_precision ||'.'||
c.numeric_scale, '0') AS type
FROM INFORMATION_SCHEMA.columns AS c INNER JOIN INFORMATION_SCHEMA.tables AS t
USING (table_schema, table_name)
WHERE t.table_type='BASE TABLE') AS table2
WHERE table1.table_schema=table2.table_schema AND
table1.table_name=table2.table_name AND
table1.column_name<>table2.column_name AND
translate(table1.column_name,'0123456789','')=translate(table2.column_name,'0123456
789','') AND
table1.type=table2.type
ORDER BY table_schema, table_name;
```

Pattern: Clone Tables or Columns

Clone Tables: Find pairs of different base tables, in case of which both tables have the same ordered set of pairs of column names and data types. In addition, after the removal of numbers from the names of the tables the names must be equal in case of each pair. If a column is specified in terms of a domain, then the query takes into account the base type of the domain.

```
SELECT table1.table_schema, table1.table_name, table2.table_schema,
table2.table_name
FROM (SELECT table_schema, table_name, string_agg(column_spec, ',') AS columns
FROM (SELECT c.table_schema, c.table_name, c.column_name ||' '|| c.data_type ||
coalesce(c.character_maximum_length::text, c.numeric_precision ||'.'||
c.numeric_scale, '0') AS column_spec
FROM INFORMATION_SCHEMA.columns AS c INNER JOIN INFORMATION_SCHEMA.tables AS t
USING (table_schema, table_name)
WHERE t.table_type='BASE TABLE'
ORDER BY c.table_schema, c.table_name, c.ordinal_position) AS sq
GROUP BY table_schema, table_name) AS table1,
(SELECT table_schema, table_name, string_agg(column_spec, ',') AS columns
FROM (SELECT c.table_schema, c.table_name, c.column_name ||' '|| c.data_type ||
coalesce(c.character_maximum_length::text, c.numeric_precision ||'.'||
c.numeric_scale, '0') AS column_spec
FROM INFORMATION_SCHEMA.columns AS c INNER JOIN INFORMATION_SCHEMA.tables AS t
USING (table_schema, table_name)
WHERE t.table_type='BASE TABLE'
ORDER BY c.table_schema, c.table_name, c.ordinal_position) AS sq
GROUP BY table_schema, table_name) AS table2
WHERE table1.columns=table2.columns AND
translate(table1.table_name,'0123456789','')=
translate(table2.table_name,'0123456789','')
AND (NOT(table1.table_schema=table2.table_schema AND
table1.table_name=table2.table_name))
ORDER BY table1.table_schema, table1.table_name;
```

Clone Columns: Find pairs of different columns of the same base table where the types of the columns are the same. In addition, after the removal of numbers from the names of the columns the names must be equal in case of each pair. If a column is specified in terms of a domain, then the query takes into account the base type of the domain.

```
SELECT table1.table_schema AS table_schema, table1.table_name AS table_name,
table1.column_name as column1, table2.column_name AS column2, table1.type AS
data_type
FROM (SELECT c.table_schema, c.table_name, c.column_name, c.data_type ||
coalesce(c.character_maximum_length::text, c.numeric_precision ||'.'||
c.numeric_scale, '0') AS type
FROM INFORMATION_SCHEMA.columns AS c INNER JOIN INFORMATION_SCHEMA.tables AS t
USING (table_schema, table_name)
WHERE t.table_type='BASE TABLE') AS table1,
(SELECT c.table_schema, c.table_name, c.column_name, c.data_type ||
coalesce(c.character_maximum_length::text, c.numeric_precision ||'.'||
c.numeric_scale, '0') AS type
FROM INFORMATION_SCHEMA.columns AS c INNER JOIN INFORMATION_SCHEMA.tables AS t
USING (table_schema, table_name)
WHERE t.table_type='BASE TABLE') AS table2
WHERE table1.table_schema=table2.table_schema AND
table1.table_name=table2.table_name AND
table1.column_name<>table2.column_name AND
translate(table1.column_name,'0123456789','')=translate(table2.column_name,'0123456
789','') AND
table1.type=table2.type
ORDER BY table_schema, table_name;
```

Pattern: Use FLOAT Data Type

Find the columns of base tables, the type of which is an approximate numeric type (FLOAT, REAL, or DOUBLE PRECISION). The query also detects columns that are defined in terms of a domain, the base type of which is an approximate numeric type.

```
SELECT table_schema, table_name, column_name, data_type
FROM INFORMATION_SCHEMA.columns
WHERE data_type IN ('real','float', 'double precision') AND
(table_schema, table_name) IN (SELECT table_schema, table_name
FROM INFORMATION_SCHEMA.tables WHERE table_type='BASE TABLE')
ORDER BY table_schema, table_name, ordinal_position;
```

Pattern: Specify Values in the Column Definition

Find the columns of base tables, which have a directly associated check constraint that specifies possible values in the column. In addition, show the name of the check constraint as well as the check clause of the constraint.

```
SELECT ccu.table_schema, ccu.table_name, ccu.column_name, ccu.constraint_schema,
ccu.constraint_name, cc.check_clause
FROM INFORMATION_SCHEMA.constraint_column_usage AS ccu INNER JOIN
INFORMATION_SCHEMA.check_constraints AS cc
USING (constraint_schema, constraint_name)
WHERE cc.check_clause~*'^.+=.*ANY.*[(].*ARRAY[[].+[])].*$'
ORDER BY ccu.table_schema, ccu.table_name;
```

Find the columns of base tables, which have been defined by using a domain, the specification of which includes a check constraint that specifies possible values in the column. In addition, show the name of the check constraint as well as the check clause of the constraint. One can use the UNION operator to merge the results of the two queries.

```
SELECT cdu.table_schema, cdu.table_name, cdu.column_name, dc.constraint_schema,
dc.constraint_name, cc.check_clause
FROM ((INFORMATION_SCHEMA.column_domain_usage AS cdu INNER JOIN
INFORMATION_SCHEMA.tables AS t USING (table_schema, table_name)) INNER JOIN
INFORMATION_SCHEMA.domain_constraints AS dc
USING (domain_schema, domain_name)) INNER JOIN INFORMATION_SCHEMA.check_constraints
AS cc USING (constraint_schema, constraint_name)
WHERE t.table_type='BASE TABLE' AND
cc.check_clause~*'^.+=.*ANY.*[(].*ARRAY[[].+[])].*$'
ORDER BY cdu.table_schema, cdu.table_name;
```

Pattern: Assume You Must Use Files

Find all the columns of base tables with the type VARCHAR or TEXT and for each found column *c* try to determine, based on the actual values in the column, whether *c* contains paths to the files. If *c* is defined in terms of a domain, the base type of which is VARCHAR or TEXT, then the function analyses the column as well. The search conditions of the dynamically generated SELECT statements contain regular expressions.

```
CREATE OR REPLACE FUNCTION f_assume_you_must_use_files()
RETURNS TABLE (table_schema VARCHAR(128), table_name VARCHAR(128), column_name
VARCHAR(128)) AS $$
DECLARE
       sql_stmt TEXT;
       cnt BIGINT;
       varchar_columns RECORD;
BEGIN
       RAISE NOTICE 'Detecting possible occurrences of the antipattern "Assume You
Must Use Files"';
       FOR varchar_columns IN SELECT c.table_schema, c.table_name, c.column_name
FROM INFORMATION_SCHEMA.columns AS c INNER JOIN INFORMATION_SCHEMA.tables AS t
USING (table_schema, table_name) WHERE c.data_type IN ('character varying', 'text')
AND t.table_type='BASE TABLE' ORDER BY c.table_schema, c.table_name LOOP
              table_schema:= varchar_columns.table_schema;
              table_name:= varchar_columns.table_name;
              column_name:= varchar_columns.column_name;
              sql_stmt:='SELECT Count(' || quote_ident(column_name) || ') AS c FROM
' || quote_ident(table_schema) ||'.' || quote_ident(table_name) || ' WHERE '||
quote_ident(column_name) || '~''^(?:[a-zA-
Z]\:|\\\\[\w\.]+\\[\w.]+)\\(?:[\w]+\\)*\w([\w.])+$''';
/*The source of the regular expression:
http://stackoverflow.com/questions/6416065/c-sharp-regex-for-file-paths-e-g-c-test-
test-exe*/
              EXECUTE sql_stmt INTO cnt;
              IF cnt>0 THEN
                     RETURN NEXT;
              END IF;
       END LOOP;
       RAISE NOTICE 'Detection completed';
       RETURN;
END;
$$ LANGUAGE plpgsql SECURITY DEFINER
SET search_path = information_schema, pg_temp;
```

```
SELECT * FROM f_assume_you_must_use_files();
```

References

[1] B. Karwin, *SQL Antipatterns. Avoiding the Pitfalls of Database Programming*, The Pragmatic Bookshelf, 2010.