K. Poonyanuch and T. Senivongse, “SQL Antipatterns Detection and Database Refactoring Process”,2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, Ishikawa, Japan,2017

SQL Antipatterns Detection and Database Refactoring Process

TABLE I. DEFINITIONS, EXAMPLES, AND HEURISTICS TO DETECT SQL ANTIPATTERNS IN TRANSACT-SQL

SQL Antipattern	Definition by Karwin [1]	Detection Heuristic in Transact-SQL
Clone Table or Columns	“Split a single long table into multiple smaller tables, using table names based on distinct data values in one of the table’s attributes” e.g. CREATE TABLE Bugs_2008 (. . .); CREATE TABLE Bugs_2009 (. . .); CREATE TABLE Bugs_2010 (. . .); “Querying Across Tables : You can reconstruct the full set of bugs using a UNION of all the split tables. As the years go on and you create more tables such as Bugs_2011, you need to keep your application code up-to-date to reference the newly created tables.”	(a) Potential zone INSERT INTO @TMP_TABLE SELECT ROW_NUMBER() OVER(ORDER BY COUNT(C.COLUMN_NAME)) AS _RK , T.TABLE_NAME , COUNT(C.COLUMN_NAME) AS CNT_COL FROM INFORMATION_SCHEMA.TABLES T INNER JOIN INFORMATION_SCHEMA.COLUMNS C ON (T.TABLE_NAME = C.TABLE_NAME) WHERE T.TABLE_TYPE = 'BASE TABLE' GROUP BY T.TABLE_NAME (b) Dynamic SQL SET @p_stmt = ' INSERT INTO T_CHECK_CLONE_TABLE SELECT T1.TABLE_NAME , T1.COLUMN_NAME , ' + CAST (@cnt_col AS VARCHAR) + ' , T2.TABLE_NAME , T2.COLUMN_NAME FROM INFORMATION_SCHEMA.COLUMNS T1 LEFT OUTER JOIN INFORMATION_SCHEMA.COLUMNS T2 ON (T1.ORDINAL_POSITION = T2.ORDINAL_POSITION AND T1.DATA_TYPE = T2.DATA_TYPE AND ISNULL(T1.CHARACTER_MAXIMUM_LENGTH ,0) = ISNULL(T2.CHARACTER_MAXIMUM_LENGTH, 0) AND ISNULL(T1.CHARACTER_OCTET_LENGTH ,0) = ISNULL(T2.CHARACTER_OCTET_LENGTH, 0) AND ISNULL(T1.NUMERIC_PRECISION ,0) = ISNULL(T2.NUMERIC_PRECISION, 0) AND ISNULL(T1.NUMERIC_SCALE,0) = ISNULL(T2.NUMERIC_SCALE , 0) AND T2.TABLE_NAME = ''' + @next_table_name + ''') '+ ' WHERE T1.TABLE_NAME = ''' + @table_name + '''and T2.TABLE_NAME is not null '
Create Multiple Columns	“We still have to account for multiple values in the attribute, but we know the new solution must store only a single value in each column. It might seem natural to create multiple columns in this table, each containing a single tag. As you assign tags to a given bug, you’d put values in one of these three columns. Unused columns remain null. Most tasks you could do easily with a conventional attribute now become more complex.” bug_id | description | tag1 | tag2 | tag3 1234 | Crashes while saving | crash | NULL | NULL 3456 | Increase performance | printing | performance | NULL 5678 | Support XML | NULL | NULL | NULL “Handling Growing Sets of Values: When you add a column in the set for a multicolumn attribute, you must revisit every SQL statement in every application that uses this table, editing the statement to support new columns.”	SELECT a.TABLE_NAME , a.COLUMN_NAME ,a.DATA_TYPE , a.IS_NULLABLE , a.CHARACTER_MAXIMUM_LENGTH , a.CHARACTER_OCTET_LENGTH, a.NUMERIC_PRECISION, a.NUMERIC_SCALE from INFORMATION_SCHEMA.COLUMNS a INNER JOIN (SELECT TABLE_NAME , LEFT(COLUMN_NAME,3) AS PREFIX_COL , DATA_TYPE , IS_NULLABLE , CHARACTER_MAXIMUM_LENGTH , CHARACTER_OCTET_LENGTH, NUMERIC_PRECISION, NUMERIC_SCALE FROM INFORMATION_SCHEMA.COLUMNS GROUP BY TABLE_NAME , LEFT(COLUMN_NAME,3) , DATA_TYPE , IS_NULLABLE , CHARACTER_MAXIMUM_LENGTH , CHARACTER_OCTET_LENGTH, NUMERIC_PRECISION, NUMERIC_SCALE HAVING COUNT(*) > 1) c ON (a.table_name = c.table_name and LEFT(a.COLUMN_NAME,3) = LEFT(c.PREFIX_COL,3)) WHERE ASCII(RIGHT(COLUMN_NAME,1)) BETWEEN 48 AND 57

K. Poonyanuch and T. Senivongse, "SQL Antipatterns Detection and Database Refactoring Process",2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, Ishikawa, Japan,2017

<table>
<tr><th>SQL Antipattern</th><th>Definition
by Karwin [1]</th><th>Detection Heuristic in Transact-SQL</th></tr>
<tr><td>Leave Out the Constraints</td><td>"Even though it seems at first that skipping foreign key constraints makes your database design simpler, more flexible, or speedier, you pay for this in other ways. It becomes your responsibility to write code to ensure referential integrity manually."

"Checking for Mistakes: You can imagine that you'd have to write a similar query for every referential relationship in your database. If you find yourself in the habit of checking for broken references like this, your next question is, how often do you need to run these checks? Running hundreds of checks every day, or even more frequently, becomes quite a chore."</td><td>SELECT T.TABLE_NAME , T.ConstraintType ,c.ConstraintName
, CASE WHEN C.ConstraintName IS NOT NULL THEN 1 ELSE 0 END AS IS_NOT_EXISTS
FROM (
SELECT T1.TABLE_NAME , cons.ConstraintType
FROM INFORMATION_SCHEMA.TABLES T1 ,
(
select 'DEFAULT_CONSTRAINT' as ConstraintType union all
select 'FOREIGN_KEY_CONSTRAINT' as ConstraintType union all
select 'PRIMARY_KEY_CONSTRAINT' as ConstraintType union all
select 'UNIQUE_CONSTRAINT' as ConstraintType union all
select 'CHECK_CONSTRAINT' as ConstraintType) as cons) T
left outer join
(
SELECT OBJECT_NAME(object_id) AS ConstraintName,
SCHEMA_NAME(schema_id) AS SchemaName,
OBJECT_NAME(parent_object_id) AS TableName,
type_desc AS ConstraintType
FROM sys.objects
WHERE type_desc LIKE '%CONSTRAINT') c
on (t.TABLE_NAME = c.TableName
AND T.ConstraintType = C.ConstraintTy</td></tr>
<tr><td>Always Depend on One's Parent</td><td>"The naive solution commonly shown in books and articles is to add a column parent_id. This column references another comment in the same table, and you can create a foreign key constraint to enforce this relationship"
e.g.
<table><tr><td>COMMENT_ID</td><td>PARENT_ID</td></tr><tr><td>1</td><td>NULL</td></tr><tr><td>2</td><td>1</td></tr><tr><td>3</td><td>2</td></tr><tr><td>4</td><td>1</td></tr><tr><td>5</td><td>4</td></tr><tr><td>6</td><td>4</td></tr><tr><td>7</td><td>6</td></tr></table>
"Maintaining a Tree with Adjacency List: However, deleting a node from a tree is more complex. If you want to delete an entire subtree, you have to issue multiple queries to find all descendants. Then remove the descendants from the lowest level up to satisfy the foreign key integrity."</td><td>(a) Potential zone
INSERT INTO @TMP_DEPEND_TABLE
SELECT p.TABLE_NAME , p.COLUMN_NAME AS PARENT_COLUMN_NAME , c.COLUMN_NAME AS CHILD_COLUMN_NAME
FROM (
SELECT T.TABLE_NAME , COLUMN_NAME , DATA_TYPE
, CHARACTER_MAXIMUM_LENGTH
, CHARACTER_OCTET_LENGTH , NUMERIC_PRECISION, NUMERIC_SCALE
FROM INFORMATION_SCHEMA.TABLES T
inner join INFORMATION_SCHEMA.COLUMNS C
ON (T.TABLE_NAME = C.TABLE_NAME)
where T.TABLE_TYPE = 'BASE TABLE'
AND (C.COLUMN_NAME like '%parent%'
or C.COLUMN_NAME like '%upper%'
or C.COLUMN_NAME like '%child%')) P
left outer join INFORMATION_SCHEMA.COLUMNS c
on (p.TABLE_NAME = c.TABLE_NAME
AND P.DATA_TYPE = C.DATA_TYPE
and isnull(p.CHARACTER_MAXIMUM_LENGTH ,0) = isnull(c.CHARACTER_MAXIMUM_LENGTH,0)
and isnull(p.CHARACTER_OCTET_LENGTH,0) = isnull(c.CHARACTER_OCTET_LENGTH ,0)and isnull(p.NUMERIC_PRECISION ,0) = isnull(c.NUMERIC_PRECISION,0)
and isnull(p.NUMERIC_SCALE ,0) = isnull(c.NUMERIC_SCALE ,0))
where P.COLUMN_NAME <> c.COLUMN_NAME
ORDER BY p.TABLE_NAME , p.COLUMN_NAME

(b) Dynamic SQL
SET @p_stmt = ' INSERT INTO T_CHECK_DEPEND_PARENT (TABLE_NAME ,PARENT_COLUMN_NAME,CHILD_COLUMN_NAME,PARENT_DATA,CNT_RELATE_DATA) '
+ ' SELECT '' + @table_name + '','' + @parent_column_name + '','' + @child_column_name + '' + ', LV1.' + @parent_column_name + ' , COUNT(*) AS TOTAL_REC '
+ ' FROM ' + @table_name + ' LV1 INNER JOIN ' + @table_name + ' LV2 '
+ ' ON (LV1.' + @parent_column_name + ' = LV2.' + @child_column_name + ') '
+ ' GROUP BY LV1.' + @parent_column_name + ', LV2.' + @child_column_name
+ ' ORDER BY COUNT(*) DESC '</td></tr>
</table>

K. Poonyanuch and T. Senivongse, “SQL Antipatterns Detection and Database Refactoring Process”,2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, Ishikawa, Japan,2017

SQL Antipattern	Definition by Karwin [1]	Detection Heuristic in Transact-SQL
One Size Fits All	“Every database table must have a primary key column with the following characteristics. The presence of a column named id in every table is so common that this has become synonymous with a primary key. Programmers learning SQL get the false idea that a primary key always means a column defined in this manner.” e.g. CREATE TABLE BugsProducts (id SERIAL PRIMARY KEY, bug_id BIGINT, product_id BIGINT, -- . . .); “Allowing Duplicate Rows: A compound key consists of multiple columns. One typical use for a compound key is in an intersection table like BugsProducts. The primary key should ensure that a given combination of values for bug_id and product_id appears only once in the table, even though each value may appear many times in different pairings. However, when you use the mandatory id column as the primary key, the constraint no longer applies to two columns that should be unique.”	(a) Potential zone SELECT DISTINCT TCS.TABLE_NAME , c.COLUMN_NAME FROM (SELECT OBJECT_NAME(parent_object_id) AS TABLE_NAME, type_desc AS CONSTRAINT_TYPE FROM sys.objects WHERE type_desc in ('PRIMARY_KEY_CONSTRAINT' , 'UNIQUE_CONSTRAINT')) TCS LEFT OUTER JOIN INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE cs on (tcs.TABLE_NAME = cs.TABLE_NAME) LEFT OUTER JOIN INFORMATION_SCHEMA.COLUMNS c ON (TCS.TABLE_NAME = c.TABLE_NAME) WHERE cs.COLUMN_NAME <> c.COLUMN_NAME AND c.COLUMN_NAME NOT IN (SELECT COLUMN_NAME FROM INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE c2 WHERE C2.TABLE_NAME = tcs.TABLE_NAME) AND DATA_TYPE <> 'datetime' (b) Dynamic SQL SET @p_stmt = 'INSERT INTO T_CHECK_ONE_SIZE_FIT_ALL ' + ' SELECT '' + @table_name + '','' + @column_name + '' + ' ,COUNT(*) AS CNT_TABLE_RECORD , COUNT(DISTINCT ' + @column_name + ') AS CNT_COLUMN_RECORD ' + ' FROM ' + @table_name + ' WITH (NOLOCK) '

K. Poonyanuch and T. Senivongse, “SQL Antipatterns Detection and Database Refactoring Process”,2017 18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, Ishikawa, Japan,2017

REFERENCES

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