Title: Associativity for <joined table>

Author: Fred Zemke
Source: U.S.A.
Status: Change proposal
Date: October 20, 1998

Abstract

I show that <joined table> is somtimes ambiguous, and propose a solution which resolves the ambiguous cases using left-associativity, as required by 6.2.3.4 “Rule evaluation order” in [Framework FCD3].

References

[Framework FCD3] Jim Melton (ed), “ISO Final Committee Draft (FCD) Database Language SQL - Part 1: SQL/Framework”, WG3:FRA-016 = NCITS H2-98-518

[Foundation FCD3] Jim Melton (ed), “ISO Final Committee Draft (FCD) Database Language SQL - Part 2: SQL/Foundation”, WG3:FRA-017 = NCITS H2-98-519

[FRA-022] Jim Melton (ed.), “ Unresolved SQL3 FCD comments after round 2”, ISO DBL:FRA-022 = ANSI NCITS H2-98-451

[98-239] Fred Zemke, “Ambiguities in <joined table>”, ANSI NCITS H2-98-239

1. Problem statement

This paper shows that there are certain ambiguities in <joined table>. This is a problem discovered during the editing process, and so is presented under the rubric of the catch-all comment.

The basic problem is that the following are all valid cases of <joined table>:

```
<table reference> CROSS JOIN <table reference>
<table reference> NATURAL INNER JOIN <table reference>
<table reference> NATURAL LEFT JOIN <table reference>
<table reference> NATURAL RIGHT JOIN <table reference>
<table reference> NATURAL FULL JOIN <table reference>
<table reference> UNION JOIN <table reference>
```

and one of the possibilities for <table reference> is <joined table>.

For example,

```
A UNION JOIN B UNION JOIN C
```

This can be parsed either

```
<joined table>
       ::= <qualified join>
       ::= <table reference> UNION JOIN <table reference>
       ::= A UNION JOIN <joined table>
       ::= A UNION JOIN B UNION JOIN C
```

or

```
<joined table>
       ::= <qualified join>
       ::= <table reference> UNION JOIN <table reference>
       ::= <joined table> UNION JOIN C
       ::= A UNION JOIN B UNION JOIN C
```

Using parentheses, the example is equivalent either to

```
(A UNION JOIN B) UNION JOIN C
```

or

```
A UNION JOIN (B UNION JOIN C)
```

This example can be repeated with any pair of the alternatives listed above, for example,

```
A NATURAL LEFT JOIN B CROSS JOIN C
```

The problem also occurs if the first join is one of the problematic joins, and the second has a <join specification>. For example

```
A CROSS JOIN B LEFT JOIN C ON X = Y
```

can be interpreted either as

```
A CROSS JOIN (B LEFT JOIN C ON X = Y)
```

or

```
(A CROSS JOIN B) LEFT JOIN C ON X = Y
```

Note that <join specification> only helps to disambiguate on the right but not on the left. The reason is that ON must be paired with the closest unmatched JOIN to left. This pairing will delimit the right operand but not the left operand.

Do these ambiguities actually matter semantically? Yes. For example, consider A UNION JOIN B CROSS JOIN C. Suppose A has a single column A.A with a single row (1); B has a single col-

umn B.B with two rows (2) and (3); and C has a single column C.C with two rows (4) and (5). Then the two possible results are

Table 1: A UNION (B CROSS C)

A	B	C
1	NULL	NULL
NULL	2	4
NULL	2	5
NULL	3	4
NULL	3	5

and

Table 2: (A UNION B) CROSS C

A	B	C
1	NULL	4
NULL	2	4
NULL	3	4
1	NULL	5
NULL	2	5
NULL	3	5

2. Solution overview

This is a classic problem with expressions, and we propose the classic solution, using precedence, associativity, and parentheses.

In an earlier presentation of this problem to X3H2 in paper [98-239], Jim Melton directed our attention to [Framework FCD2] 6.2.3.4 “Rule evaluation order” where we find the following paragraphs:

> Where the precedence of operators is determined by the Formats of ISO/IEC 9075 or by parentheses, those operators are effectively applied in the order specified by that precedence.
>
> Where the precedence is not determined by the Formats or by parentheses, effective evaluation of expressions is generally performed from left to right. However, it is implementation- dependent whether expressions are actually evaluated left to

> right, particularly when operands or operators might cause conditions to be raised or if the results of the expressions can be determined without completely evaluating all parts of the expression.

The consensus when [98-239] was reviewed was that these paragraphs indicate that the correct resolution of the ambiguities found above is left to right evaluation. Nevertheless, it was felt that this resolution should be expressed through the BNF. This paper provides that solution.

In more detail, we introduce <table primary> as the foundation of table expression syntax, analogous to <value expression primary>, which is the foundation of scalar expression syntax. Intuitively, <table primary> is any table expression which is either atomic (cannot be decomposed into more basic table expressions) or is enclosed in parentheses. As it happens, the only table expressions that require parenthesization are <joined table>s. In fact, there is currently a BNF that

```
<joined table> ::=
     <left paren> <joined table> <right paren>.
```

This BNF was a start in the right direction, but it did not succeed because it did not recognize the need to stratify the rules and separate out the case of <table primary>.

Today, table expression syntax is found in two subclauses, 6.11 <table reference> and 7.6 <joined table>. Ideally, all table expression syntax should be in a single subclause, in order to make the precedence relationships clear by collecting them on a single page. However, merging these two subclauses is a big task. As it happens, every case of <table reference> is a <table primary> except for <joined table>, and only <joined table> requires parenthesization. This means that all of the precedence issues can still be relegated to a single subclause, 7.6 <joined table>, while keeping this separate from 6.11 <table reference>.

As shown above, use of a <join specification> (either ON clause or USING clause) can disambiguate the right operand, though not the left. Thus we can continue to allow arbitrary <table reference>s in this case.

The summary of these changes is the following (schematized) BNF in 6.11 <table reference>:

```
<table reference> ::=
       <table primary>
     | <joined table>

<table primary> ::=
       <table or query name> [ [ AS ] . . . ]
     | <derived table> [ AS ] . . .
     | <collection derived table> [ AS ] . . .
     | ONLY <left paren> <table or query name> . . .
           <right paren>
     | <left paren> <joined table> <right paren>
```

and the following BNF in 7.6 <joined table>

```
<joined table> ::=
```

```
        <cross join>
      | <qualified join>
      | <natural join>
      | <union join>

<cross join> ::=
     <table reference> CROSS JOIN
       <table primary>

<qualified join> ::=
     <table reference> [ <join type> ] JOIN
     <table reference> <join specification>

<natural join> ::=
     <table reference>
     NATURAL [ <join type> ] JOIN
     <table primary>

<union join> ::=
     <table reference> UNION JOIN <table primary>

 <join type> ::=
        INNER
      | <outer join type> [ OUTER ]

 <outer join type> ::=
        LEFT
      | RIGHT
      | FULL
```

Notice that as a consequence of these changes, <natural join> and <union join> have been separated out of <qualified join>. This is because the precedence rules for <natural join> and <union join> are fundamentally different from a join containing a <join specification>. <natural join> and <union join>, like <cross join>, require a <table primary> argument on the right, which is not necessary when there is a <join specification>.

The separation implies that UNION is no longer a <join type>, and there are some incidental changes to the Syntax Rules.

Since the meaning of <qualified join> has changed, I undertook a search for all occurrences. There was only one, in 7.11 <query specification> SR 1)g)iii)6), which I amend to restore its meaning.

I also searched for all occurrences of <join type>. In addition to 7.11 <query specification> SR 1)g)iii)6), I found occurences in 4.18 “Functional dependencies”. I will be fixing that section in a succeeding paper (indeed, I was led to this work precisely by my investigation of functional dependencies).

Inspection showed that all the nested joined tables in the Information Schema remain valid under this proposal. However, to encourage the use of parentheses in complex table expressions, I have inserted parentheses in all nested table expressions of more than two operands.

3. Proposal for [Foundation FCD3]

This proposal uses the following conventions:

SMALLCAPS	denote editorial instructions;
~~strikeout~~	denotes existing text to be deleted;
boldface	denotes new text to be inserted;
plain	denotes existing text to be retained,
[Note:...]	brackets enclose italicized notes to the proposal reader
⌊‾‾‾‾⌋	boxes surround "editing tags," which are part of the document (not instructions to the editor) and may be deleted, inserted, modified or retained, depending on the typeface within the box

3.1 Changes to 6.15 <table reference>

1. EDIT THE FORMAT FOR <TABLE REFERENCE> AS FOLLOWS :

```
<table reference> ::=
       <table primary>
     | <joined table>

<table primary> ::=
       <table or query name>
         [ [ AS ] <correlation name>
           [ <left paren> <derived column list>
             <right paren> ]
         ]
     | <derived table>
         [ AS ] <correlation name>
         [ <left paren> <derived column list>
           <right paren> ]
     | <joined table>
     | <lateral derived table>
         [ AS ] <correlation name>
         [ <left paren> <derived column list>
           <right paren> ]
     | <collection derived table>
         [ AS ] <correlation name>
         [ <left paren> <derived column list>
           <right paren> ]
     | ONLY <left paren> <table or query name> <right paren>
```

```
        [ [ AS ] <correlation name>
          [ <left paren> <derived column list>
            <right paren> ] ]
      | <left paren> <joined table> <right paren>
```

[NOTE to the proposal reader: I have decomposed <table reference> into the 'primary' cases, which are self-delimiting, and <joined table>, which must be recognized as a kind of expression.]

3.2 Changes to 7.6 <joined table>

1. EDIT THE FORMAT AS FOLLOWS:

```
<joined table> ::=
       <cross join>
     | <qualified join>
     | <natural join>
     | <union join>
     | <left paren> <joined table> <right paren>
```

[NOTE to the proposal reader: these parentheses are a failed attempt to deal with the ambiguities. They can be used to remove ambiguities, but this placement in the BNF does not guarantee that they will be used. This guarantee is now provided by the use of <table primary>.]

```
<cross join> ::=
     <table reference> CROSS JOIN
       <table reference primary>

<qualified join> ::=
     <table reference>
     [ NATURAL ] [ <join type> ] JOIN
     <table reference> [ <join specification> ]

<natural join> ::=
     <table reference>
     NATURAL [ <join type> ] JOIN
     <table primary>

<union join> ::=
     <table reference> UNION JOIN <table primary>

<join specification> ::=
        <join condition>
      | <named columns join>
```

```
<join condition> ::= ON <search condition>

<named columns join> ::=
     USING <left paren> <join column list> <right paren>

<join type> ::=
       INNER
     | <outer join type> [ OUTER ]
     ~~| UNION~~

<outer join type> ::=
       LEFT
     | RIGHT
     | FULL

<join column list> ::= <column name list>
```

2. DELETE SYNTAX RULE 4)

   ~~4) If a <qualified join> is specified, then~~

   ~~Case:~~

   ~~a) If NATURAL is specified, then a <join specification> shall not be specified.~~

   ~~b) If UNION is specified, then neither NATURAL nor a <join specification> shall be specified.~~

   ~~c) Otherwise, a <join specification> shall be specified.~~

   [NOTE to the proposal reader: these restrictions are now enforced by the BNF.]

3. EDIT SYNTAX RULE 5) AS FOLLOWS:

   5) If a <qualified join> **or <natural join>** is specified and a <join type> is not specified, then INNER is implicit.

4. EDIT GENERAL RULE 1)A) AS FOLLOWS:

   1) Case:

      a) If ~~<join type> is UNION~~ **<union join> is specified**, then let *T* be the empty set.

3.3 Changes to 7.11 <query expression>

1. INSERT SYNTAX RULE 1)G)III)6.1) FOLLOWING SR 1)G)III)6) AS FOLLOWS:

   1) If <with clause> is specified, then:

   . . .

g) For a potentially recursive <with list> *WL* with *n* elements, and for *i* ranging from 1 (one) to *n*, let *WLEi* be the *i*-th <with list element> of *WL*, let *WQNi* be the <query name> immediately contained in *WLEi*, let *WQEi* be the <query expression> immediately contained in *WLEi*, let *WQTi* be the table defined by *WQEi*, and let *QNDG* be the query name dependency graph of *WL*.

. . .

iii) For each *WLEi*, for *i* ranging from 1 (one) to *n*, and for each *WQNj* that belongs to the stratum of *WQEi*:

. . .

6) *WQi* shall not contain a <qualified join> *QJ* ~~with~~ **in which**:

A) *QJ* immediately contains a <join type> that specifies FULL and a <table reference> that contains *WQNj*.

B) *QJ* immediately contains a <join type> that specifies LEFT and a <table reference> following the <join type> that contains *WQNj*.

C) *QJ* immediately contains a <join type> that specifies RIGHT and a <table reference> preceding the <join type> that contains *WQNj*.

6.1) *WQi* shall not contain a <natural join> *QJ* in which:

A) *QJ* immediately contains a <join type> that specifies FULL and a <table reference> or <table primary> that contains *WQNj*.

B) *QJ* immediately contains a <join type> that specifies LEFT and a <table primary> following the <join type> that contains *WQNj*.

C) *QJ* immediately contains a <join type> that specifies RIGHT and a <table reference> preceding the <join type> that contains *WQNj*.

3.4 Changes to 20.12 ATTRIBUTES view

1. EDIT THE FROM CLAUSE AS FOLLOWS

```
FROM
  ( DEFINITION_SCHEMA.ATTRIBUTES AS A
    LEFT JOIN
      ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D1
          LEFT JOIN
```

```
              DEFINITION_SCHEMA.COLLATIONS AS C1
              ON ( ( C1.COLLATION_CATALOG,
                     C1.COLLATION_SCHEMA,
                     C1.COLLATION_NAME )
                 = ( D1.COLLATION_CATALOG,
                     D1.COLLATION_SCHEMA,
                     D1.COLLATION_NAME ) )
        )
        ON ( ( A.UDT_CATALOG, A.UDT_SCHEMA, A.UDT_NAME,
               'ATTRIBUTE', A.ATTRIBUTE_NAME,'', 0 )
           = ( D1.OBJECT_CATALOG, D1.OBJECT_SCHEMA,
               D1.OBJECT_NAME, D1.OBJECT_TYPE,
               D1.OBJECT_TYPE_NAME,
               D1.METHOD_SPECIFICATION_IDENTIFIER,
               D1.ORDINAL_POSITION ) )
    )
  LEFT JOIN
    ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D2
        LEFT JOIN
          DEFINITION_SCHEMA.COLLATIONS AS C2
            ON ( ( C2.COLLATION_CATALOG, C2.COLLATION_SCHEMA,
                   C2.COLLATION_NAME )
               = ( D2.COLLATION_CATALOG, D2.COLLATION_SCHEMA,
                   D2.COLLATION_NAME ) )
    )
    ON ( ( A.DOMAIN_CATALOG, A.DOMAIN_SCHEMA, A.DOMAIN_NAME,
           'DOMAIN', '', '', 0 )
       = ( D2.OBJECT_CATALOG, D2.OBJECT_SCHEMA,
           D2.OBJECT_NAME, D2.OBJECT_TYPE,
           D2.OBJECT_TYPE_NAME,
           D2.METHOD_SPECIFICATION_IDENTIFIER,
           D2.ORDINAL_POSITION ) )
```

3.5 Changes to 20.16 COLUMN_DOMAIN_USAGE

1. EDIT THE FROM CLAUSE AS FOLLOWS

```
FROM DEFINITION_SCHEMA.COLUMNS AS C
     JOIN
       (  DEFINITION_SCHEMA.DOMAINS AS D
          JOIN
          DEFINITION_SCHEMA.SCHEMATA AS S
            ON ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA )
               = ( S.CATALOG_NAME, S.SCHEMA_NAME ) ) )
     ON ( ( D.DOMAIN_CATALOG, D.DOMAIN_SCHEMA,
```

```
            D.DOMAIN_NAME )
        = ( C.DOMAIN_CATALOG, C.DOMAIN_SCHEMA,
            C.DOMAIN_NAME ) )
```

[NOTE to the proposal reader: Note also the stylistic improvement to insert AS in three places.]

3.6 Changes to 20.19 COLUMNS view

1. EDIT THE FROM CLAUSE AS FOLLOWS

```
FROM ( DEFINITION_SCHEMA.COLUMNS AS C
    LEFT JOIN
      ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D1
        LEFT JOIN
          DEFINITION_SCHEMA.COLLATIONS AS C1
          ON
          ( ( C1.COLLATION_CATALOG, C1.COLLATION_SCHEMA,
              C1.COLLATION_NAME )
          = ( D1.COLLATION_CATALOG, D1.COLLATION_SCHEMA,
              D1.COLLATION_NAME ) )
      )
    ON
    ( ( C.TABLE_CATALOG, C.TABLE_SCHEMA, C.TABLE_NAME,
          'COLUMN', C.COLUMN_NAME, 0 )
      = ( D1.OBJECT_CATALOG, D1.OBJECT_SCHEMA,
          D1.OBJECT_NAME, D1.OBJECT_TYPE, D1.COLUMN_NAME,
          D1.ORDINAL_POSITION ) )
    )
  LEFT JOIN
    ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D2
      LEFT JOIN
        DEFINITION_SCHEMA.COLLATIONS AS C2
        ON
        ( ( C2.COLLATION_CATALOG, C2.COLLATION_SCHEMA,
            C2.COLLATION_NAME )
        = ( D2.COLLATION_CATALOG, D2.COLLATION_SCHEMA,
            D2.COLLATION_NAME ) )
    )
  ON
   ( ( C.DOMAIN_CATALOG, C.DOMAIN_SCHEMA,C.DOMAIN_NAME,
       'DOMAIN', '', 0 )
    = ( D2.OBJECT_CATALOG, D2.OBJECT_SCHEMA, D2.OBJECT_NAME,
      D2.OBJECT_TYPE, D2.COLUMN_NAME,
      D2.ORDINAL_POSITION ) )
```

3.7 Changes to 20.26 DOMAINS view

1. EDIT THE FROM CLAUSE AS FOLLOWS

```
FROM DEFINITION_SCHEMA.DOMAINS
     JOIN
       ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D
         LEFT JOIN
         DEFINITION_SCHEMA.COLLATIONS AS S
         USING ( COLLATION_CATALOG, COLLATION_SCHEMA,
                 COLLATION_NAME )
       )
     ON
       ( ( DOMAIN_CATALOG, DOMAIN_SCHEMA, DOMAIN_NAME,
           '' ,0 )
       = ( OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
           COLUMN_NAME , ORDINAL_POSITION ) )
```

3.8 Changes to 20.30 METHOD_SPECIFICATION_PARAMETERS

1. EDIT THE FROM CLAUSE AS FOLLOWS

```
FROM
  ( DEFINITION_SCHEMA.METHOD_SPECIFICATION_PARAMETERS P
     INNER JOIN
       ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR D
         LEFT JOIN DEFINITION_SCHEMA.COLLATIONS C
           ON
             ( ( C.COLLATION_CATALOG, C.COLLATION_SCHEMA,
                 C.COLLATION_NAME )
             = ( D.COLLATION_CATALOG, D.COLLATION_SCHEMA,
                 D.COLLATION_NAME ) )
       )
     ON
       ( P.USER_DEFINED_TYPE_CATALOG,
         P.USER_DEFINED_TYPE_SCHEMA,
         P.USER_DEFINED_TYPE_NAME,
         'METHOD_SPECIFICATION_PARAMETER', '',
         P.METHOD_NAME,
         P.METHOD_SPECIFICATON_IDENTIFIER,
         P.ORDINAL_POSITION )
     = ( D.OBJECT_CATALOG, D.OBJECT_SCHEMA, D.OBJECT_NAME,
         D.OBJECT_TYPE, D.COLUMN_NAME, D.METHOD_NAME,
         D.METHOD_SPECIFICATION_IDENTIFIER,
D.ORDINAL_POSITION)
  )
```

```
INNER JOIN DEFINITION_SCHEMA.METHOD_SPECIFICATIONS M
ON ( ( P.USER_DEFINED_TYPE_CATALOG,
       P.USER_DEFINED_TYPE_SCHEMA,
       P.USER_DEFINED_TYPE_NAME, P.METHOD_CATALOG,
       P.METHOD_SCHEMA, P.METHOD_NAME,
       P.METHOD_SPECIFICATION_IDENTIFIER )
   = ( M.USER_DEFINED_TYPE_CATALOG,
       M.USER_DEFINED_TYPE_SCHEMA,
       M.USER_DEFINED_TYPE_NAME, M.METHOD_CATALOG,
       M.METHOD_SCHEMA, M.METHOD_NAME,
       M.METHOD_SPECIFICATION_IDENTIFIER ) )
```

3.9 Changes to 20.31 PARAMETERS view

1. EDIT THE FROM CLAUSE AS FOLLOWS:

```
FROM ( DEFINITION_SCHEMA.PARAMETERS AS P1
  LEFT JOIN
  ( DEFINITION_SCHEMA.DATA_TYPE_DESCRIPTOR AS D1
    LEFT JOIN DEFINITION_SCHEMA.COLLATIONS AS C1
      ON ( ( C1.COLLATION_CATALOG, C1.COLLATION_SCHEMA,
             C1.COLLATION_NAME )
         = ( D1.COLLATION_CATALOG, D1.COLLATION_SCHEMA,
             D1.COLLATION_NAME ) )
  )
  ON
    ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA, SPECIFIC_NAME,
      'PARAMETER', '', P1.ORDINAL_POSITION )
  = ( OBJECT_CATALOG, OBJECT_SCHEMA, OBJECT_NAME,
      OBJECT_TYPE, COLUMN_NAME, D1.ORDINAL_POSITION )
  )
  INNER JOIN DEFINITION_SCHEMA.ROUTINES AS R1
    ON ( ( P1.SPECIFIC_CATALOG, P1.SPECIFIC_SCHEMA,
           P1.SPECIFIC_NAME )
       = ( R1.SPECIFIC_CATALOG, R1.SPECIFIC_SCHEMA,
           R1.SPECIFIC_NAME ) )
```

3.10 Changes to 20.38 ROUTINE_COLUMN_USAGE

1. EDIT THE FROM CLAUSE AS FOLLOWS

```
FROM ( DEFINITION_SCHEMA.ROUTINE_COLUMN_USAGE
       JOIN DEFINITION_SCHEMA.ROUTINES
         USING ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA,
                 SPECIFIC_NAME )
     )
```

```
        JOIN DEFINITION_SCHEMA.SCHEMATA AS S
          ON ( ( TABLE_CATALOG, TABLE_SCHEMA ) =
               ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
```

3.11 Changes to 20.40 ROUTINE_TABLE_USAGE

1. EDIT THE FROM CLAUSE AS FOLLOWS:

```
    FROM ( DEFINITION_SCHEMA.ROUTINE_TABLE_USAGE
           JOIN
           DEFINITION_SCHEMA.ROUTINES
           USING ( SPECIFIC_CATALOG, SPECIFIC_SCHEMA,
                 SPECIFIC_NAME )
         )
         JOIN
         DEFINITION_SCHEMA.SCHEMATA S
         ON
           ( ( TABLE_CATALOG, TABLE_SCHEMA ) =
             ( S.CATALOG_NAME, S.SCHEMA_NAME ) )
```

3.12 No changes required for the following Information Schema views:

20.10 APPLICABLE_ROLES view - a single table and a two-table query. (Someone call out the style police, this view does not use a JOIN in the second FROM clause when it could have)

20.11 ASSERTIONS view- only two tables

20.13 CHARACTER_SETS view - only one table

20.14 CHECK_CONSTRAINTS view- only two tables

20.15 COLLATIONS view - only one table

20.17 COLUMN_PRIVILEGES - only one table

20.18 COLUMN_USER_DEFINED_TYPE_USAGE- only two tables

20.20 CONSTRAINT_COLUMN_USAGE view- many JOINS, but each of them is okay

20.21 CONSTRAINT_TABLE_USAGE view- This is a very complex FROM clause, but there are no <joined table> operators with more than two operands.

20.22 DIRECT_SUPERTABLES view - only two tables

20.23 DIRECT_SUPERTYPES - only one table

20.24 DOMAIN_CONSTRAINTS view - only two tables

20.25 DOMAIN_USER_DEFINED_TYPE_USAGE view - only two tables

20.27 ENABLED-ROLES view - only two tables

20.28 KEY_COLUMN_USAGE view- only two tables

20.29 METHOD_SPECIFICATIONS view - only two tables

20.32 REFERENTIAL_CONSTRAINTS view- only two tables

20.33 ROLE_COLUMN_GRANTS view - only one table

20.34 ROLE_ROUTINE_GRANTS view - only one table

20.35 ROLE_TABLE_GRANTS view - only one table

20.36 ROLE_USAGE_GRANTS view - several FROM clauses, each with at most two tables

20.36 ROLE_USER_DEFINED_TYPE_GRANTS view - several FROM clauses, each with at most two tables

20.39 ROUTINE_PRIVILEGES - only two tables

20.41 ROUTINES view - only two tables

20.42 SCHEMATA view - only one table

20.43 SQL_FEATURES view - only one table

20.44 SQL_IMPLEMENTATION_INFO view - only one table

20.45 sql_PACKAGES view - only one table

20.46 SQL_SIZING view - only one table

20.47 SQL_SIZING_PROFILES view - only one table

20.48 SQL_LANGUAGES view - only one table

20.49 TABLE_CONSTRAINTS view - only two tables

20.50 TABLE_METHOD_PRIVILEGES - only one table

20.51 TABLE_PRIVILEGES view - only one table

20.52 TABLES view - only one table

20.53 TRANSFORMS view - only one table

20.54 TRANSLATIONS view - only one table

20.55 TRIGGERED_UPDATE_COLUMNS view - only two tables

20.56 TRIGGER_COLUMN_USAGE - only two tables

20.57 TRIGGER_TABLE_USAGE - only two tables

20.58 TRIGGERS view - only two tables

20.59 USAGE_PRIVILEGES view - only one table

20.60 USER_DEFINED_TYPE_PRIVILEGES view - only one table

20.61 USER_DEFINED_TYPES view - only one table

20.62 VIEW_COLUMN_USAGE view- only two tables

20.63 VIEW_TABLE_USAGE view- only two tables

20.64 VIEWS view - only one table

21.3 EQUAL_KEY_DEGREES assertion - no <joined table>

21.4 KEY_DEGREE_GREATER_THAN_OR_EQUAL_TO_1 assertion - only two tables

21.5 UNIQUE_CONSTRAINT_NAME assertion - no <joined table>

- End of paper -