T u t o r i a l D

(version dated September 22nd, 2016, superseding all previous versions)

This document gives a BNF grammar for **Tutorial D**. It consists of an edited and slightly revised version of material from Chapter 11 ("**Tutorial D**") of the book *Database Explorations: Essays on The Third Manifesto and related topics*, by C. J. Date and Hugh Darwen (Trafford, 2010). Changes with respect to that Chapter are highlighted in blue. The productions make use of several self-explanatory abbreviations: *op* for operator, *inv* for invocation, *exp* for expression, and so on. They're shown in alphabetical order. In Chapter 11 of the book, the section headed **RELATIONS AND ARRAYS** specifies some additional elements that are not properly integrated—for example, there is a production for <*relation set*>, which is not referenced in any other production. These are properly integrated in the grammar given here.

Changes since publication of *Database Explorations*:

Oct 2011: Addition of Boolean operator EQUIV in <*agg op inv*> and <*summary spec*>.

May 2013: Removal of the parentheses surrounding <*grouping*>, <*ungrouping*>, <*wrapping*>, and <*unwrapping*> in the productions for <*group*>, <*ungroup*>, <*wrap*>, <*unwrap*>, <*tuple wrap*> and <*tuple unwrap*>.

Mar 2016: Revision of <*user scalar root type def*> to avoid the requirement for equivalent explicit constraint specifications on each possrep. Replacement of <*statement*> with <*statement sequence*> in various places. Addition of <*pref or inf*> to <*user op def*> to support infix invocation style, with consequent revision to <*user op inv*>.

Sep 2016: Addition of TUPLE{*} to <*tuple selector inv*> as a shorthand for "the current tuple", and <*image*> to <*monadic other built in relation op inv*> to support IMAGE_IN(*r*,*t*), denoting the image of tuple *t* in relation *r*.

```
<agg op inv>
    ::=   <agg op name>
             ( [ <integer exp> , ] <relation exp> [ , <exp> ] )
        | <n-adic count etc>

<agg op name>
    ::=   COUNT | SUM | AVG | MAX | MIN
        | AND | OR | EQUIV | XOR | EXACTLY
        | UNION | D_UNION | INTERSECT | XUNION

<application relation var def>
    ::=   VAR <relation var name> <private or public>
              <relation type or init value> <key def list>

<argument exp>
     ::=   <exp>

<array cardinality>
    ::=   COUNT ( <array var ref> )

<array target>
    ::=   <array var ref>
```

```
<array var def>
    ::=   VAR <array var name> ARRAY <tuple type spec>

<array var ref>
    ::=   <array var name>

<assign>
    ::=   <scalar assign>
        | <nonscalar assign>

<assignment>
    ::=   <assign commalist>

<attribute>
    ::=   <attribute name> <type spec>

<attribute extractor inv>
    ::=   <attribute ref> FROM <tuple exp>

<attribute ref>
    ::=   <attribute name>

<attribute target>
    ::=   <attribute ref>
        | <attribute THE_ pv ref>

<attribute THE_ pv ref>
    ::=   <THE_ pv name> ( <attribute target> )

<begin transaction>
    ::=   BEGIN TRANSACTION

<built in relation op inv>
    ::=   <relation selector inv>
        | <THE_ op inv>
        | <attribute extractor inv>
        | <project>
        | <n-adic other built in relation op inv>
        | <monadic or dyadic other built in relation op inv>

<built in scalar op inv>
    ::=   <scalar selector inv>
        | <THE_ op inv>
        | <attribute extractor inv>
        | <agg op inv>
        | ... plus the usual possibilities

<built in scalar type name>
    ::=   INTEGER | RATIONAL | CHARACTER | BOOLEAN
```

```
<built in tuple op inv>
    ::=   <tuple selector inv>
        | <THE_ op inv>
        | <attribute extractor inv>
        | <tuple extractor inv>
        | <tuple project>
        | <n-adic other built in tuple op inv>
        | <monadic or dyadic other built in tuple op inv>

<call>
    ::=   CALL <user op inv>

<case>
    ::=   CASE ; <when spec list> [ ELSE <statement sequence> ]
          END CASE

<commit>
    ::=   COMMIT

<compound statement body>
    ::=   BEGIN ; <statement list> END

<constraint def>
    ::=   CONSTRAINT <constraint name> <bool exp>

<constraint drop>
    ::=   DROP CONSTRAINT <constraint name>

<database relation var def>
    ::=   <real relation var def>
        | <virtual relation var def>

<direction>
    ::=   ASC | DESC

<divide>
    ::=   <relation exp> DIVIDEBY <relation exp> <per>

<do>
    ::=   [ <statement name> : ]
          DO <scalar var ref> := <integer exp> TO <integer exp> ;
             <statement sequence>
          END DO

<dyadic compose>
    ::=   <relation exp> COMPOSE <relation exp>

<dyadic disjoint union>
    ::=   <relation exp> D_UNION <relation exp>

<dyadic intersect>
    ::=   <relation exp> INTERSECT <relation exp>

<dyadic join>
    ::=   <relation exp> JOIN <relation exp>
```

```
<dyadic other built in relation op inv>
    ::=   <dyadic union> | <dyadic disjoint union>
        | <dyadic intersect> | <minus> | <included minus>
        | <dyadic join> | <dyadic times> | <dyadic xunion>
        | <dyadic compose> | <matching> | <not matching> | <divide>
        | <summarize>

<dyadic other built in tuple op inv>
    ::=   <dyadic tuple union> | <dyadic tuple compose>

<dyadic times>
    ::=   <relation exp> TIMES <relation exp>

<dyadic tuple compose>
    ::=   <tuple exp> COMPOSE <tuple exp>

<dyadic tuple union>
    ::=   <tuple exp> UNION <tuple exp>

<dyadic union>
    ::=   <relation exp> UNION <relation exp>

<dyadic xunion>
    ::=   <relation exp> XUNION <relation exp>

<exp>
    ::=   <scalar exp>
        | <nonscalar exp>

<extend>
    ::=   EXTEND <relation exp> : { <attribute assign commalist> }

<group>
    ::=   <relation exp> GROUP <grouping>

<grouping>
    ::=   { [ ALL BUT ] <attribute ref commalist> }
                                        AS <introduced name>

<heading>
    ::=   { <attribute commalist> }

<if>
    ::=   IF <bool exp> THEN <statement sequence>
                       [ ELSE <statement sequence> ]
          END IF

<image>
    ::=   IMAGE_IN ( <relation exp> [ , <tuple exp> ] )

<included minus>
    ::=   <relation exp> I_MINUS <relation exp>

<key def>
    ::=   KEY { [ ALL BUT ] <attribute ref commalist> }
```

```
<leave>
    ::=   LEAVE <statement name>

<matching>
    ::=   <relation exp> MATCHING <relation exp>

<minus>
    ::=   <relation exp> MINUS <relation exp>

<monadic or dyadic other built in relation op inv>
    ::=   <monadic other built in relation op inv>
        | <dyadic other built in relation op inv>

<monadic or dyadic other built in tuple op inv>
    ::=   <monadic other built in tuple op inv>
        | <dyadic other built in tuple op inv>

<monadic other built in relation op inv>
    ::=   <rename> | <where> | <extend> | <wrap> | <unwrap>
        | <group> | <ungroup> | <tclose> | <image>

<monadic other built in tuple op inv>
    ::=   <tuple rename> | <tuple extend>
        | <tuple wrap> | <tuple unwrap>

<n-adic compose>
    ::=   COMPOSE { <relation exp commalist> }

<n-adic count etc>
    ::=   <agg op name> { <exp commalist> }

<n-adic disjoint union>
    ::=   D_UNION [ <heading> ] { <relation exp commalist> }

<n-adic intersect>
    ::=   INTERSECT [ <heading> ] { <relation exp commalist> }

<n-adic join>
    ::=   JOIN { <relation exp commalist> }

<n-adic other built in relation op inv>
    ::=   <n-adic union> | <n-adic disjoint union>
        | <n-adic intersect> | <n-adic join> | <n-adic times>
        | <n-adic xunion> | <n-adic compose>

<n-adic other built in tuple op inv>
    ::=   <n-adic tuple union>

<n-adic times>
    ::=   TIMES { <relation exp commalist> }

<n-adic tuple union>
    ::=   UNION { <tuple exp commalist> }

<n-adic union>
    ::=   UNION [ <heading> ] { <relation exp commalist> }
```

```
<n-adic xunion>
    ::=   XUNION [ <heading> ] { <relation exp commalist> }

<name intro>
    ::=   <introduced name> := <exp>

<no op>
    ::=   ... zero or more “white space” characters

<nonscalar assign>
    ::=   <tuple assign>
        | <relation assign>

<nonscalar exp>
    ::=   <tuple exp>
        | <relation exp>

<nonscalar selector inv>
    ::=   <tuple selector inv>
        | <relation selector inv>

<nonscalar type spec>
    ::=   <tuple type spec>
        | <relation type spec>

<nonscalar var ref>
    ::=   <tuple var ref>
        | <relation var ref>

<nonwith statement body>
    ::=   <previously defined statement body commalist>
        | <begin transaction> | <commit> | <rollback>
        | <call> | <return> | <case> | <if> | <do> | <while>
        | <leave> | <no op> | <compound statement body>

<not matching>
    ::=   <relation exp> NOT MATCHING <relation exp>

<order item>
    ::=   <direction> <attribute ref>

<ordering>
    ::=   ORDINAL | ORDERED

<parameter def>
    ::=   <parameter name> <type spec>

<per>
    ::=   PER ( <relation exp> [ , <relation exp> ] )

<per or by>
    ::=   <per>
        | BY { [ ALL BUT ] <attribute ref commalist> }

<possrep component def>
    ::=   <possrep component name> <type spec>
```

```
<possrep component ref>
    ::=   <possrep component name>

<possrep component target>
    ::=   <possrep component ref>
        | <possrep THE_ pv ref>

<possrep constraint def>
    ::=   CONSTRAINT <bool exp>

<possrep def>
    ::=   POSSREP [ <possrep name> ]
                  { <possrep component def commalist>
                          [ <possrep constraint def> ] }

<possrep THE_ pv ref>
    ::=   <THE_ pv name> ( <possrep component target> )

<previously defined statement body>
    ::=   <assignment>
        | <user op def> | <user op drop>
        | <user scalar type def> | <user scalar type drop>
        | <scalar var def> | <tuple var def>
        | <relation var def> | <relation var drop>
        | <constraint def> | <constraint drop>
        | <array var def> | <relation get> | <relation set>

<private or public>
    ::=   PRIVATE | PUBLIC

<project>
    ::=   <relation exp> { [ ALL BUT ] <attribute ref commalist> }

<real or base>
    ::=   REAL | BASE

<real relation var def>
    ::=   VAR <relation var name> <real or base>
              <relation type or init value> <key def list>

<relation assign>
    ::=   <relation target> := <relation exp>
        | <relation insert>
        | <relation d_insert>
        | <relation delete>
        | <relation i_delete>
        | <relation update>

<relation comp>
    ::=   <relation exp> <relation comp op> <relation exp>

<relation comp op>
    ::=   = | ≠ | ⊆ | ⊇ | ⊂ | ⊃
```

```
<relation d_insert>
    ::=   D_INSERT <relation target> <relation exp>

<relation delete>
    ::=   DELETE <relation target> <relation exp>
        | DELETE <relation target> [ WHERE <bool exp> ]

<relation exp>
    ::=   <relation with exp>
        | <relation nonwith exp>

<relation get>
    ::=   LOAD <array target> FROM <relation exp>
                              ORDER ( <order item commalist> )

<relation i_delete>
    ::=   I_DELETE <relation target> <relation exp>

<relation insert>
    ::=   INSERT <relation target> <relation exp>

<relation nonwith exp>
    ::=   <relation var ref>
        | <relation op inv>
        | ( <relation exp> )

<relation op inv>
    ::=   <user op inv>
        | <built in relation op inv>

<relation selector inv>
    ::=   RELATION [ <heading> ] { <tuple exp commalist> }
        | TABLE_DEE
        | TABLE_DUM

<relation set>
    ::=   LOAD <relation target> FROM <array var ref>

<relation target>
    ::=   <relation var ref>
        | <relation THE_ pv ref>

<relation THE_ pv ref>
    ::=   <THE_ pv name> ( <scalar target> )

<relation type name>
    ::=   RELATION <heading>

<relation type or init value>
    ::=   <relation type spec> | INIT ( <relation exp> )
        | <relation type spec> INIT ( <relation exp> )
```

```
<relation type spec>
    ::=   <relation type name>
        | SAME_TYPE_AS ( <relation exp> )
        | RELATION SAME_HEADING_AS ( <nonscalar exp> )

<relation update>
    ::=   UPDATE <relation target> [ WHERE <bool exp> ] :
              { <attribute assign commalist> }

<relation var def>
    ::=   <database relation var def>
        | <application relation var def>

<relation var drop>
    ::=   DROP VAR <relation var ref>

<relation var ref>
    ::=   <relation var name>

<relation with exp>
    ::=   WITH ( <name intro commalist> ) : <relation exp>

<rename>
    ::=   <relation exp> RENAME { <renaming commalist> }

<renaming>
    ::=   <attribute ref> AS <introduced name>
        | PREFIX <character string literal>
              AS <character string literal>
        | SUFFIX <character string literal>
              AS <character string literal>


<return>
    ::=   RETURN [ <exp> ]

<rollback>
    ::=   ROLLBACK

<scalar assign>
    ::=   <scalar target> := <scalar exp>
        | <scalar update>

<scalar comp>
    ::=   <scalar exp> <scalar comp op> <scalar exp>

<scalar comp op>
    ::=   = | ≠ | < | ⩽ | > | ⩾

<scalar exp>
    ::=   <scalar with exp>
        | <scalar nonwith exp>
```

```
<scalar nonwith exp>
    ::=   <scalar var ref>
        | <scalar op inv>
        | ( <scalar exp> )

<scalar op inv>
    ::=   <user op inv>
        | <built in scalar op inv>

<scalar selector inv>
    ::=   <built in scalar literal>
        | <possrep name> ( <argument exp commalist> )

<scalar target>
    ::=   <scalar var ref>
        | <scalar THE_ pv ref>

<scalar THE_ pv ref>
    ::=   <THE_ pv name> ( <scalar target> )

<scalar type name>
    ::=   <user scalar type name>
        | <built in scalar type name>

<scalar type or init value>
    ::=   <scalar type spec> | INIT ( <scalar exp> )
        | <scalar type spec> INIT ( <scalar exp> )

<scalar type spec>
    ::=   <scalar type name>
        | SAME_TYPE_AS ( <scalar exp> )

<scalar update>
    ::=   UPDATE <scalar target> :
                { <possrep component assign commalist> }

<scalar var def>
    ::=   VAR <scalar var name> <scalar type or init value>

<scalar var ref>
    ::=   <scalar var name>

<scalar with exp>
    ::=   WITH ( <name intro commalist> ) : <scalar exp>

<selector inv>
    ::=   <scalar selector inv>
        | <nonscalar selector inv>

<statement>

::=   <statement body> ;

<statement body>
    ::=   <with statement body>
        | <nonwith statement body>
```

```
<statement sequence>
    ::=   <statement list>

<subscript>
    ::=   <integer exp>

<summarize>
    ::=   SUMMARIZE <relation exp> [ <per or by> ] :
                                   { <attribute assign commalist> }

<summary>
    ::=   <summary spec> ( [ <integer exp> , ] [ <exp> ] )

<summary spec>
    ::=   COUNT | COUNTD | SUM | SUMD | AVG | AVGD | MAX | MIN
        | AND | OR | XOR | EXACTLY | EXACTLYD
        | UNION | D_UNION | INTERSECT | XUNION

<tclose>
    ::=    TCLOSE ( <relation exp> )

<THE_ op inv>
    ::=   <THE_ op name> ( <scalar exp> )

<tuple assign>
    ::=   <tuple target> := <tuple exp>
        | <tuple update>

<tuple comp>
    ::=   <tuple exp> <tuple comp op> <tuple exp>
        | <tuple exp> ∈ <relation exp>
        | <tuple exp> ∉ <relation exp>

<tuple comp op>
    ::=   = | ≠

<tuple component>
    ::=   <attribute name> <exp>

<tuple exp>
    ::=   <tuple with exp>
        | <tuple nonwith exp>

<tuple extend>
    ::=   EXTEND <tuple exp> : { <attribute assign commalist> }

<tuple extractor inv>
    ::=   TUPLE FROM <relation exp>

<tuple nonwith exp>
    ::=   <tuple var ref>
        | <tuple op inv>
        | <array var ref> ( <subscript> )
        | ( <tuple exp> )
```

```
<tuple op inv>
    ::=   <user op inv>
        | <built in tuple op inv>

<tuple project>
    ::=   <tuple exp> { [ ALL BUT ] <attribute ref commalist> }

<tuple rename>
    ::=   <tuple exp> RENAME { <renaming commalist> }

<tuple selector inv>
    ::=   TUPLE { <tuple component commalist> | * }

<tuple target>
    ::=   <tuple var ref>
        | <tuple THE_ pv ref>

<tuple THE_ pv ref>
    ::=   <THE_ pv name> ( <scalar target> )

<tuple type name>
    ::=   TUPLE <heading>

<tuple type or init value>
    ::=   <tuple type spec> | INIT ( <tuple exp> )
        | <tuple type spec> INIT ( <tuple exp> )

<tuple type spec>
    ::=   <tuple type name>
        | SAME_TYPE_AS ( <tuple exp> )
        | TUPLE SAME_HEADING_AS ( <nonscalar exp> )

<tuple unwrap>
    ::=   <tuple exp> UNWRAP <unwrapping>

<tuple update>
    ::=   UPDATE <tuple target> :
              { <attribute assign commalist> }

<tuple var def>
    ::=   VAR <tuple var name> <tuple type or init value>

<tuple var ref>
    ::=   <tuple var name>

<tuple with exp>
    ::=   WITH ( <name intro commalist> ) : <tuple exp>

<tuple wrap>
    ::=   <tuple exp> WRAP <wrapping>

<type spec>
    ::=   <scalar type spec>
        | <nonscalar type spec>
```

```
<ungroup>
    ::=    <relation exp> UNGROUP <ungrouping>

<ungrouping>
    ::=    <attribute ref>

<unwrap>
    ::=   <relation exp> UNWRAP <unwrapping>

<unwrapping>
    ::=    <attribute ref>

<user inf op inv>
    ::=   <argument exp> <user op name> <argument exp>

<user op def>
    ::=   <user update op def>
        | <user read-only op def>

<user op drop>
    ::=   DROP OPERATOR <user op name>

<user op inv>
    ::=   <user pref op inv> | <user inf op inv>

<user pref op inv>
    ::=   <user op name> ( <argument exp commalist> )

<user read-only op def>
    ::=   OPERATOR [ <pref or inf> ] <user op name>
           ( <parameter def commalist> )
             RETURNS <type spec> ;
                <statement sequence>
          END OPERATOR

<user scalar root type def>
    ::=   TYPE <user scalar type name> [ <ordering> ]
             <possrep def list> [ <possrep constraint def> ]
               INIT ( <literal> )

<user scalar type def>
    ::=   <user scalar root type def>

<user scalar type drop>
    ::=   DROP TYPE <user scalar type name>

<user update op def>
    ::=   OPERATOR <user op name>
           ( <parameter def commalist> )
             UPDATES { [ ALL BUT ] <parameter name commalist> } ;
                <statement sequence>
          END OPERATOR
```

```
<var ref>
    ::=   <scalar var ref>
        | <nonscalar var ref>

<virtual relation var def>
    ::=   VAR <relation var name> VIRTUAL
              ( <relation exp> ) <key def list>

<when spec>
    ::=   WHEN <bool exp> THEN <statement sequence>

<where>
    ::=   <relation exp> WHERE <bool exp>

<while>
    ::=   [ <statement name> : ]
          WHILE <bool exp> ;
             <statement sequence>
          END WHILE

<with statement body>
    ::=   WITH ( <name intro commalist> ) : <statement body>

<wrap>
    ::=   <relation exp> WRAP <wrapping>

<wrapping>
    ::=   { [ ALL BUT ] <attribute ref commalist> }
                                          AS <introduced name>
```