U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

NISTIR 90-4341

National PDES Testbed
Report Series

Translating Express to SQL: A User's Guide

NIST

U.S. DEPARTMENT OF COMMERCE
National Institute of Standards and Technology

NISTIR 90-4341

National PDES Testbed

Translating Express to SQL: A User's Guide

Katherine C. Morris

U.S. DEPARTMENT OF
COMMERCE
Robert A. Mosbacher,
Secretary of Commerce

National Institute of
Standards and Technology
John W. Lyons, Director

May 8, 1990

NIST

Translating Express to SQL: A User's Guide

1.0 Overview 1

2.0 Mapping Express Constructs into Relational Database Tables 2

2.1 Entity Tables 2
2.1.1 Table Name 3
Table name for a NULL subtype 3
2.1.2 Entity-ID Column 3
2.1.3 Sharable Column 4
2.1.4 Inherited Attributes 4
2.1.5 Explicit Entity Attributes 4
2.2 Attributes 5
2.2.1 Attribute Columns 6
Data Types 6
Optional attributes 7
Unique attributes 7
2.2.2 Attribute tables 8
Table Name 8
ID Column 8
Value Column 8
Aggregate Positioning Columns 8
Nested Aggregate Objects 9
Optional and Uniqueness Concepts 9
2.3 Entity Views 9

3.0 Dictionary 10

3.1 Entity descriptions 10
3.2 Attribute descriptions 10
3.2.1 Aggregate attribute descriptions 11
3.3 Description of defined types 12

4.0 The program: *fedex_sql* 13

4.1 Running the program 13
4.1.1 Generating the SQL schema definition statements 13
Output files 13
4.1.2 Creating the database tables 14
Output files 14
4.2 Different versions of the program 14

1.0 Overview

This document describes the procedure used by the *fedex_sql* software to translate an Express schema into the SQL statements which generate a relational database schema for storing STEP data. The program which loads a STEP physical file into the database is *stepwf_sql* and is describe in a separate document [Nickerson90]. The program uses FED-X, an Express parser, which is documented in [Clark90]. The software has been developed as part of the National PDES Testbed effort and is funded by the Computer Aided Logistics Support (CALS) project.

Three types of issues are involved in translating Express into a relational database schema: translation of the semantic constructs of Express into the data definition language of SQL, resolution of limitiation imposed by the database management system, and development of a data dictionary. The first two are discussed in section 2. First, the constructs of the Express language are translated into relational concepts. The application of this mapping to a particular Express schema, generates the SQL data definition language which is the basis for the database.

Secondly issues involving the particular database management system (DBMS) are resolved. In this case Oracle's SQL*Plus is being used, but the translation also conforms to the SQL standard specification as described in [ANSI86] unless otherwise noted. The basic data types defined in Express are mapped into the data types of the SQL*Plus. The names used by the Express schema need to be modified to be acceptable to the DBMS. For example, they could be too long or the same as key words in the SQL*Plus.

Section 3 discusses the data dictionary. The dictionary holds information from the conceptual specification which is not explicitly captured in the SQL schema. The dictionary captures some of the constraints specified in the conceptual schema which are not directly mapped into the database management system's facilities. For instance, the Uniqueness Rule as defined by the Express language can often be handled directly by the database management system; however, constraints such as the minimum or maximum number of elements in a set are not handled by most database management systems.

The dictionary captures descriptive information provided by the conceptual schema which is also not directly represented in the SQL definitions. For example, Express schemas contain type definitions. Through these definitions semantic information describing attributes is relayed. For instance, the data type "weight_in_pounds" can be defined in an Express schema; a user is then able to associate more meaning with an attribute described as having this data type, than if the type "real" had been assigned to that attribute. However, SQL has no expression available which would allow one to store the depth of this meaning. Therefore, the information is stored in a dictionary.

Section 4 describes how to run the program *fedex_sql* and how to use its output.

2.0 Mapping Express Constructs into Relational Database Tables

This section describes how the Express entity definitions are represented in relational tables. The translation of the entities is summarized as follows: (1) *Every entity defined in the Express schema is translated into a table or view in the relational database. (2) An entity without subtypes is represented as a table. (3) An entity which has subtypes is represented as a view of the tables which represent its subtypes.* Data can be retrieved from these views, but not inserted into them. *(4) If an entity has a "XOR NULL" specified in the Express "Supertype of" statement, it has both a table and a view associated with it.* Data inserted into the table associated with the instances of the NULL subtype entity appears in the view along with the data from the other subtypes' tables.

The attributes of an entity are represented either as columns in the entity's table or as another table. Aggregate attributes are represented as separate tables. The dictionary table EXPRESSYS$ATTRIBUTEDESC indicates how the attribute is represented.

2.1 Entity Tables

A primary table, called an entity table, is associated with each entity with no subtypes and with each entity which has 'NULL' as one of its subtypes. The following template shows the structure of the entity tables. It is described in the sections which follow.

Table Name = Entity's abbreviated Name

ID	SHARABLE	INHERITED ATTRIBUTE COLUMNS	EXPLICIT ATTRIBUTE COLUMNS

This mapping is based on the mapping used by the STEP physical file representation of an Express schema [Altemueller88]. Specifically, the decision to represent only entities with no subtypes in tables is based primarily on the fact that these are the only entities which can be populated in a physical file. Furthermore, the order of the columns is based on the ordering of attributes in the physical file and the inheritance rules for attributes are applied in the same way. The use of 'AND' and 'OR' in the supertype declarations is also unaccounted for just as in the current STEP physical file mapping.

2.1.1 Table Name

The table representing an entity is named after the entity. When the entity's name is too long, it is abbreviated. This mapping of the entity names and table names is found in the EXPRESSYS$NAMES table and also in a file generated by the *fedex_sql* program. The name of this file is `TABLE_NAMES.txt` by default. The algorithm for generating the new names is given below.

1. If the name is one of the key words reserved by the DBMS, the last character is changed to "#".
2. If name is less than 20 characters, no abbreviation is needed.
3. Otherwise, the last vowel or repeated character is removed from the name until the name is less than 20 characters or all these characters have been removed.
4. If the name is still not less than 20 characters, the last character of the longest subword is dropped until the shortened name is less than 20 characters. A subword is a portion of the word which is separated by underscores or pound signs.
5. If the name is still not less than 20 characters, the character "_" is used as the name abbreviation. (Thus a numeric name is generated for the table in the following step.)
6. Append a unique three digit number on the end of the abbreviation to guarantee that the name is unique.

2.1.1.1 Table name for a NULL subtype

The name of the table representing an entity with NULL as one of its subtypes is formed as follows: (1) The entity is abbreviated as described above, (2) The string "_NULL" is appended to the end of the abbreviated entity name. This table is included in the view of that entity, which is described in section 2.3.

2.1.2 Entity-ID Column

The first column of every entity table is ID. It contains a unique identifier for every instance of an entity. This identifier is used as the primary key of the entity table; and it is likely to be referenced in two situations outside of this table. An entity referenced by another entity as an attribute is represented by this identifier in that attribute's column of the entity table. The EXPRESSYS$FRNKEYREFERENCES table can be used to find out which tables reference other tables or, conversely, to find out where a table is referenced. For an entity with aggregate attributes (attributes whose type is array, bag, list or set), the same entity identifier is also used in the tables which contain the data for these attributes. Detail about aggregate attribute tables is given below in section 2.2.2.

The identifiers generated from the program *stepwf_sql*, which loads data into the tables, take the following form:

table_name!*00000000*

Table_name is the name of the entity table, and *00000000* is a unique integer.

2.1.3 Sharable Column

Every entity table contains a column called SHARABLE. This column is currently used as an indicator of whether or not the entity instance can be used by more than one other entity instance. This is currently interpreted to mean whether the instance is embedded in another instance in the input STEP physical file.

In future versions of the database this could be used in checking uniqueness and equality of entity instances. For example, the question of whether two points with the same coordinates are the same point or two distinct instances of a point is unclear. If the SHARABLE column is FALSE, the points are definitively not the same; however, if the column is TRUE, the two points may be considered the same. Furthermore, the field could be used as a reference counter to ascertain whether a shared instance is to be deleted when a referencing instance is deleted.

2.1.4 Inherited Attributes

The next group of columns to appear in an entity's table represent the non-aggregate attributes inherited from the entity's supertype(s). The columns are specified in the order of inheritance defined by the STEP physical file structure. The origin of the attribute, the name of the entity in which the attribute is specified in the Express schema, is found in the EXPRESSYS$SRC table.

2.1.5 Explicit Entity Attributes

Finally the non-aggregate attributes declared directly in the Express definition of the entity are columns the table.

In the example that follows the portion of the Express schema shown produces the SQL statement to create a table.

EXPRESS:

```
ENTITY geometry (* GEOM-1 *)
  SUPERTYPE OF (point XOR
                vector XOR
                curve XOR
                surface XOR
                coordinate_system XOR
                transformation XOR
                axis_placement);
  local_coordinate_system : OPTIONAL coordinate_system;
  axis                    : OPTIONAL transformation;
END_ENTITY;

ENTITY vector (* GEOM-3 *)
  SUPERTYPE OF (direction XOR
                vector_with_magnitude)
  SUBTYPE OF (geometry);
END_ENTITY;

ENTITY direction (* GEOM-14 *)
  SUBTYPE OF (vector);
  x  :  REAL;
```

```
    y : REAL;
    z : OPTIONAL REAL;
END_ENTITY;
```

SQL:

```
CREATE TABLE DIRECTION  (
        ID                  CHAR(40)        PRIMARY KEY,
        SHARABLE            INTEGER          NOT NULL,
        LOCAL_COORDINATE_SYSTEM  CHAR(40) /*  FOREIGN KEY  */,
        AXIS                CHAR(40) /*  FOREIGN KEY  */,
        X                   FLOAT           NOT NULL,
        Y                   FLOAT           NOT NULL,
        Z                   FLOAT
);
```

TABLE:

DIRECTION

SYSTEM ATTRIBUTES		INHERITED ATTRIBUTES		EXPLICIT ATTRIBUTES		
ID	SHARABLE	local_coordinate _system	axis	x	y	z

2.2 Attributes

The attributes of an entity are represented as either a column in the entity table or as a table of their own. If the attribute is aggregate (an array, bag, list, or set), it has its own table; otherwise, the attribute is represented as a column.

The EXPRESSYS$DEFINEDTYPES dictionary table describes the attributes of the entity tables. It includes a short name for the attribute and information about the type of the attribute as it is given in the Express schema. The short name is used in assigning a name for the attribute in the database. The column EXPRESS_TYPE contains a code which can be used to determine whether the attribute is represented as a column in the entity table or as an aggregate table. The valid values for this field are AGGREGATE, ENTITY, SELECT, ENUMERATION, INTEGER, REAL, BOOLEAN, LOGICAL, STRING, and NUMBER.

When the type is AGGREGATE or ENTITY, the value of the attribute is represented in another table. In the case of ENTITY the owning entity table has a column for the attribute. The column contains a key (an entity identifier) into an entity table. In the case of the type AGGREGATE the owning entity table does not contain

a column for this attribute. The entity identifier from the owning entity table is used to identify the aggregate data items in the aggregate table as belonging to that entity.

2.2.1 Attribute Columns

Non-aggregate attributes are represented as columns in the entity tables. The columns have the same name as the attribute when this name is less than 30 characters (the maximum length allowed by SQL); otherwise, the name is abbreviated by truncating the attribute name to 27 characters and appending a unique 2 digit integer to the end. The same abbreviated name is used for attributes with the same name in different tables.

2.2.1.1 Data Types

Oracle data types are assigned to the Express base types as follows for the purpose of representing attributes as columns in the database. The table shows all the base types of Express as described in [Schenck90]. Note that the default length of an attribute with type string is 240.

EXPRESS	ORACLE
Integer	INTEGER
Integer(n)	NUMBER(n)
Real	DECIMAL
Real(n)	NUMBER(n)
Number	NUMBER
String	CHAR(240)
String(n)	CHAR(n) for n <= 240, LONG for strings up to 64 K
Boolean	INTEGER
Logical	INTEGER

The last two base types above, boolean and logical, are treated as special cases of enumerated types which are described below.

Attributes with the following complex Express types are also represented as columns in the database. Below is a mapping of these Express types to Oracle data types.

Entity	CHAR(40) FOREIGN KEY
Select	CHAR(40)
Enumeration	INTEGER

Enumeration type

Both enumeration and select types imply the specification of a domain for attributes. An enumerated type specifies the possible values for the domain explicitly; a select type specifies the possible values indirectly. The values of an enumeration are stored in the dictionary table EXPRESSYS$ENUMERATION. This table assigns integer values to the values of an enumeration. The integer values are what is then stored in the attribute columns. The dictionary table is consulted to see what the integer values represent. The reason for storing the integer values, rather than the string values that they represent, is the fact that an enumeration type implies an ordering on its possible values. In order to enforce the ordering the integer values are used.

```
TYPE
  b_spline_curve_form = ENUMERATION OF
  (line_segment,
  circular_arc,
  elliptic_arc,
  parabolic_arc,
  hyperbolic_arc);
END_TYPE;
```

After this type is entered into the EXPRESSYS$ENUMERATION table, the table looks as follows:

TYPE_NAME	ORDER_ID	VALUE
B_SPLINE_CURVE_FORM	0	LINE_SEGMENT
B_SPLINE_CURVE_FORM	1	CIRCULAR_ARC
B_SPLINE_CURVE_FORM	2	ELLIPTIC_ARC
B_SPLINE_CURVE_FORM	3	PARABOLIC_ARC
B_SPLINE_CURVE_FORM	4	HYPERBOLIC_ARC

Select type

With a select type the possible values of an attribute come from the group of the possible values of several other types. When these types are entities, the values are entity identifiers, as if the attribute's type had been an entity. From the entity identifiers generated by the database loader it is possible to tell which table contains the entity instance information for a given entry. The first part of the entity identifier is the name of the entity table.

On the other hand, when the types are not entities, then the values must be of the same base type as these defined types. In Express it is possible to create an object which does not have a single base type through the use of a select type; *fedex_sql,* which implements this design, does not deal with this situation. We assume that in the majority of instances of select types the selection is amongst entity types; therefore, a select type attribute maps to the SQL type CHAR(40) just as do entity type attributes. When the values of the selection are not entity identifiers, the field representing these type attributes is still confined to CHAR(40). The choices of the selection for a select type are stored in the dictionary table EXPRESSYS$SELECT.

2.2.1.2 Optional attributes

Each attribute column in the entity tables is specified to be NOT NULL unless the key word OPTIONAL is specified for that attribute in the Express definitions. The DBMS then only allows rows which contain values for all non-optional attributes to be inserted in the database.

2.2.1.3 Unique attributes

When an attribute is characterized as being unique in the Express definition, a unique index is created on the column which represents that attribute in the entity table. The indices are named after the table to which they apply. An integer is appended to the end of the table name to create a unique name for the index. Every

table has at least one index on the ID column. In this way the Express uniqueness construct is directly supported by the DBMS for non-aggregate attributes.

2.2.2 Attribute tables

Attributes with aggregate data types are represented as tables, called aggregate tables, in the database. The valid Express aggregate data types for attributes are array, bag, list, and set. Each item of the aggregate object is represented by a row in the aggregate table.

2.2.2.1 Table Name

Aggregate tables' names are created by combining the abbreviated name of the owning entity (which is the name of the entity table) with the name of the attribute, which it represents. The two names are separated by a pound sign (#), and then the new string is abbreviated using the same algorithm described above for naming entity tables. The unabbreviated and abbreviated string pair are entered into the dictionary table EXPRESSYS$NAMES.

2.2.2.2 ID Column

The first column in every aggregate table is called ID. The values in this column correspond to the values in the ID column of the owning entity's table. Whereas, in the entity table there is only one entry for each unique entity identifier, in this table there are multiple rows for a given entity identifier. The value of the ID column is the same for all the items contained in a a single aggregate attribute.

2.2.2.3 Value Column

The last column in all aggregate tables is called VALUE. This column contains the data for the individual data items of the aggregate object. For example, if the aggregate is a "list of integer", this column contains integers; if the aggregate is a "set of cartesian points", this column contains entity identifiers from the cartesian point entity table.

2.2.2.4 Aggregate Positioning Columns

The second column in an aggregate table indicates the position of the individual data item in the aggregate object. The name of this column is determined by the type of the aggregate object. For example, if the object is an array, this column is called SUBSCRIPT_1. The column name can be determined from the following table:

AGGREGATE TYPE	COLUMN NAME
Array	SUBSCRIPT_*n*
Bag	ELEMENT_ID_*n*
List	POSITION_ID_*n*
Set	ELEMENT_ID_*n*

The *n* in the column name is an integer, which is always 1 for a simple, not nested, aggregate attribute.

When the aggregate is a list, the following column is PREVIOUS_ID_*n*.

2.2.2.5 Nested Aggregate Objects

If the aggregate attribute is nested, or multi-dimensional, more positioning columns follow the initial one. These columns are named in the same way as the initial positioning column, as described above. The integer *n* in the column name indicates the nesting level that this column represents. For example, for a two-dimensional array there are two positioning columns SUBSCRIPT_1 and SUBSCRIPT_2. The first column contains the value of the first subscript of the array and the second column contains the values for the second subscript of the array. The data for the item at position [1,2] of the array would have a row in the table which contains the following entry:

ID	SUBSCRIPT_1	SUBSCRIPT_2	VALUE
entity-id	1	2	data item

2.2.2.6 Optional and Uniqueness Concepts

The Express concepts, optional and unique, are not directly supportable for aggregate attributes by a relational database system under this mapping. The information is stored in the dictionary tables EXPRESSYS$ARRAY, EXPRESSYS$BAG, EXPRESSYS$LIST, and EXPRESSYS$SET.

With non-aggregate objects the Express key word UNIQUE is represented in SQL by creating a unique index on an attribute. However, the translation of UNIQUE with complex objects involves comparing the objects element by element. Furthermore, equality is not defined for aggregate objects; therefore, uniqueness for nested aggregate objects is not enforceable.

In Express the key word OPTIONAL within aggregate attributes indicates that not all the data elements of the attribute must be specified. This is different than designating that the object is an optional attribute of the entity, which is modeled by the non-use of the NOT NULL clause in the entity table definition.

2.3 Entity Views

Entities which have subtypes are represented as views of the entity tables in the database. Views serve as tables for the purpose of retrieving data from the database, but data can not be inserted in or deleted from the views directly. Entity views are named using the algorithm given earlier for naming entity tables. Also as with entity tables the original and abbreviated name are entered into the EXPRESSYS$NAMES table. The entity views contain an ID column and columns for all the non-aggregate, explicit and inherited attributes which belong to the entity being viewed. There are no views for the aggregate attributes.

The dictionary table EXPRESSYS$CLASSES shows the class hierarchy and can be used to see which entity tables are included in a view.

3.0 Dictionary

Fourteen dictionary tables are used to store semantic information found in Express schemas. Identical tables are established for each and every schema. The SQL statements for creating these tables are found in the beginning of the main output file of *fedex_sql*. The dictionary tables summarized below are described in detail in the in the document *Translation of an Express Schema into SQL*. In the database the names of these tables are prefixed by "EXPRESSYS$" to indicate that they are dictionary tables.

Four of the tables involve the handling of aggregate data types. One table maintains information pertaining to subtype and supertype relations. Another table stores the logical names of tables. Finally there are tables for representing the Express type definitions and another for recording descriptions of entity attributes in the terms of these definitions.

3.1 Entity descriptions

- NAMES: maps Express names to the names used by the database system

NAME	SHORT_NAME
BOUNDARY_LOCATION_SHAPE_ASPECT	BNDRY_LCTN_SHP_SPCT_494
BNDRY_LCTN_SHP_SPCT_494#REPRESENTATIONS	BND_LCT_SHP_SP_49#RP_509
BOUNDED_CURVE	BOUNDED_CURVE
BOUNDED_SURFACE	BOUNDED_SURFACE
B_SPLINE_CURVE	B_SPLINE_CURVE
B_SPLINE_CURVE#CONTROL_POINTS	B_SPLN_CRV#CNTR_PNTS_512
B_SPLINE_CURVE#KNOT_MULTIPLICITIES	B_SPLN_CRV#KNT_MLTPL_513
B_SPLINE_CURVE#KNOTS	B_SPLINE_CURVE#KNOTS
B_SPLINE_CURVE#WEIGHTS	B_SPLINE_CURVE#WEIGHTS

- CLASSES: captures the class structure of the Express schema

SUBTYPE	SUPERTYPE
BNDRY_LCTN_SHP_SPCT_494	DMNSNLTY_0_SHP_SPCT_495
BOUNDED_CURVE	CURVE
BOUNDED_SURFACE	SURFACE
B_SPLINE_CURVE	BOUNDED_CURVE
B_SPLINE_SURFACE	BOUNDED_SURFACE
CURVE	GEOMETRY

3.2 Attribute descriptions

The tables used to describe attributes are the following:

- ATTRIBUTEDESC: contains Express type information, whether the attribute is optional, unique, or sharable, and the name used to represent the attribute in the database (See the attached table.)

- FRNKEYREFERENCES: maps attributes to the table which would represent them

BASE_TABLE_NAME	REFERENCING_TABLE_NAME	REFERENCING_TABLE_COLUMN
B_SPLINE_CURVE	UNSTRCT_GMTRY_SM_RP_536	DEFINITION
B_SPLINE_CURVE	UNSTR_GMTRY_PRMTR_RP_921	DEFINITION
B_SPLINE_CURVE	UNSTR_GMTR_DM_0_S_RP_510	DEFINITION
B_SPLINE_CURVE	UNSTRUCT_GMTRY_R_RP_558	DEFINITION
B_SPLINE_CURVE	TRIMMED_CURVE	BASIS_CURVE
B_SPLINE_CURVE	SWP_PRFL_NL#CRV_PRFL_949	VALUE
B_SPLINE_CURVE	SURFACE_CURVE	CURVE_1
B_SPLINE_CURVE	SIZ_CHRCTRSTC_DMNSN_943	CENTER_OF_SYMMETRY
B_SPLINE_CURVE	RCTNGL_PRFL#CRV_PRFL_936	VALUE
B_SPLINE_CURVE	RCTRCK_PRFL#CRV_PRFL_935	VALUE
B_SPLINE_CURVE	POINT_ON_CURVE	BASIS_CURVE
B_SPLINE_CURVE	PCURVE	BASIS_CURVE
B_SPLINE_CURVE	OTHER_SWEEP_PATH	PATH
B_SPLINE_CURVE	OTHER_SWEEP_PATH	PROFILE
B_SPLINE_CURVE	N_GON_PRFL#CRV_PRFL_915	VALUE
B_SPLINE_CURVE	OTHR_CLS_PRF#CRV_PRF_916	VALUE
B_SPLINE_CURVE	INTERSECTION_CURVE	BASIS_CURVE

- ATTRSRC: indicates the entity from which an attribute originated in the inheritance hierarchy

ENTITY_SHORT_NAME	ATTRIBUTE_NAME	COLUMN_NAME
GEOMETRY	AXIS	AXIS
GEOMETRY	LOCAL_COORDINATE_SYSTEM	LOCAL_COORDINATE_SYSTEM
B_SPLINE_CURVE	DEGREE	DEGREE
B_SPLINE_CURVE	UPPER_INDEX_ON_CONTROL_POINTS	UPPER_INDEX_ON_CONTROL_POINTS
B_SPLINE_CURVE	KNOT_MULTIPLICITIES	KNOT_MULTIPLICITIES
B_SPLINE_CURVE	KNOTS	KNOTS
B_SPLINE_CURVE	SELF_INTERSECT	SELF_INTERSECT
CURVE	BASIS_SURFACE	BASIS_SURFACE

3.2.1 Aggregate attribute descriptions

The following tables apply to aggregate attributes:

- ATTRBEXPRESSTYPE: holds information for reconstructing type information for nested aggregate (i.e. multi-dimensional) attributes
- ARRAY:

OBJECT_TABLE	SEQUENCE_NUMBER	LOW_BOUND	HIGH_BOUND	OPTIONAL	UNIQUE_ELEMENTS
B_SPLINE_CURVE#KNOTS	1	1		0	0
B_SPLINE_CURVE#WEIGHTS	1	0		0	0
B_SPLINE_SURFACE#U_KNOTS	1	1		0	0
B_SPLINE_SURFACE#V_KNOTS	1	1		0	0
B_SPLINE_SURFACE#WEIGHTS	1	0		0	0
B_SPLINE_SURFACE#WEIGHTS	2	0		0	0
B_SPLN_CRV#CNTR_PNTS_512	1	0		0	0
B_SPLN_CRV#KNT_MLTPL_513	1	1		0	0
B_SPLN_SRFC#CNTR_PNT_514	1	0		0	0
B_SPLN_SRFC#CNTR_PNT_514	2	0		0	0
B_SPLN_SRFC#V_MLTPLC_516	1	1		0	0
B_SPLN_SRFC#_MLTPLCT_515	1	1		0	0

- BAG, LIST, and SET are very similar to the array table and are described in detail in [Metz89].

3.3 Description of defined types

The following tables contain information about schema-defined types:

- DEFINEDTYPES: records definition of type defined within the Express schema

This table contains two columns which map a type name to a data type as declared in the TYPE block of the Express schema. The value of the DEFINITION column of this table is either the name of an Express base type, the name of an aggregate type, the key word "ENUMERATION," the key word "SELECT," the key word "AGGREGATE", an Express entity name, or the value of the NAME column from another row in the table.

TYPE	DEFINITION
INFINITY	NUMBER
INTERSECTION_ENUMERATION	ENUMERATION
LIST_OF_EDGE	LIST
SET_OF_VERTEX	SET
SHAPE_OR_DERIVED	SELECT
SURF_FORM	ENUMERATION
SURF_TYPE	ENUMERATION
TOL_IBO	ENUMERATION
TOL_MLSN	ENUMERATION
TRUE_FALSE_OR_UNDEFINED	BOOLEAN

- ENUMERATION: records the possible values of a type which is an enumeration

TYPE_NAME	ORDER_ID	VALUE
SURF_FORM	0	BOUNDED_PLANAR
SURF_FORM	1	BOUNDED_RULE
SURF_FORM	2	BOUNDED_COMPLEX
SURF_FORM	3	UNBOUNDED_PLANAR
SURF_FORM	4	UNBOUNDED_RULE
SURF_FORM	5	UNBOUNDED_COMPLEX
SURF_TYPE	0	CIRCULAR
SURF_TYPE	1	FLAT
SURF_TYPE	2	GENERAL

- SELECT: records the types of a selection

TYPE_NAME	CHOICE
SELECT_FACE_OR_SUBFACE	FACE
SHAPE_OR_DERIVED	DT_SHAPE_ASPECT
SHAPE_OR_DERIVED	GEOMETRIC_DERIVATION

The following table is used by the program *stepwf_sql* which loads a STEP file into the database:

- INSTANTIATEDTABLES: keeps track of which tables are actually populated.

4.0 The program: *fedex_sql*

The program *fedex_sql* is part of the NIST/PDES Fed-x toolkit. This module translates an Express schema into a relational database schema using the methodology described in this document. The Fed-x toolkit is described in detail in the documents [Clark90].

4.1 Running the program

Two steps are needed to use the software for translating an Express schema into a relational database schema. First the SQL statements for creating the database schema are generated. Then these statements are loaded into the database. The document [Strouse90] describes this process for the NIST/PDES Testbed environment. To run the program on your own follow the instructions given here.

4.1.1 Generating the SQL schema definition statements

1) The command line for *fedex_sql* is the following:

`fedex_sql -e` *express-schema-file*

The *express-schema-file* is the file where an Express schema is stored. The program may then print out some warning messages regarding the schema.

2) Next the program prompts for the name of an output file. The SQL statements to create the database schema are stored in this file, so it should probably end in the `.sql` extension.

3) The program then prompts for a file containing a list of entity and table names. The default the file is `TABLE_NAMES.txt`. If the file name provided is not found, no file will be used and the program will generate unique abbreviations for the entity names.

4) Finally the program prompts for the name of a file in which to store the list of entity table names. If no file name is provided, the names are stored in the file `TABLE_NAMES.txt` in the working directory.

4.1.1.1 Output files

When this program is finished, six files will have been created. The names of two of these are supplied by the user in the steps above: the names supplied when prompted for an output file (step 2) and a file for the table names (step 4), `TABLE_NAMES.txt` by default. The others are `DICT_DATA.sql`, `DICT_INDICES.sql`, `SUPERTYPES.sql`, and `INDICES.sql` and are found in the working directory. *Warning: if any of these files existed in the working directory before the program was run, they would have been replaced by the new files.*

The main output file contains the statements for creating all the tables. The beginning of this file creates the dictionary tables; the remainder creates the entity and aggregate tables.

Each line of the file of table names contains two words. The first is the abbreviated entity name to be used in generating table names, and the second is the entity name used in the Express schema. The line following the list of names contains "*** ***" which signifies the end of the list. The next and last line contains an integer which is the first number the program will use in generating unique abbreviations, if they are needed.

The file `DICT_DATA.sql` contains the SQL statements that populate the Express data dictionary, the other dictionary file `DICT_INDICES.sql` contains the statements for generating indices on the data dictionary.

The file `SUPERTYPES.sql` contains the SQL statements to generated views for the supertype entities defined in the Express schema.

The file `INDICES.sql` contains SQL statements to create indices on the entity and aggregate attribute tables.

4.1.2 Creating the database tables

To load the database schema do the following:

5) Log into the database management system.

6) At the `SQL>` prompt type `run` *`sql-schema-file`*, where *sql-schema-file* is the name the user provided in step 3 above. This file takes some time to load depending on the size of the schema. Over 30 minutes in the PDES Testbed environment is not unusual.

The process loads the files `DICT_DATA.sql`, `DICT_INDICES.sql`, and `SUPERTYPES.sql` automatically if they are in the working directory.

At this point the database is ready to be populated. After the database has been completely populated, the indices on the tables should be created. This is done by typing `run INDICES.sql` at the `SQL>` prompt.

4.1.2.1 Output files

The creation of the tables generates a file called `errors.lst`. This file is a listing of what appeared on the screen during the process. It should not contain anything of significance, but if there were any problems the error message will be in this file.

4.2 Different versions of the program

The data definition produced by *fedex_sql* is designed to work with an Oracle database. Due to physical design considerations two tablespaces are used. In the current configuration these are named `ts0` and `ts1`. Entity tables are assigned to these tablespaces alternately. The indices for an entity table and any tables that represent aggregate attributes of that entity are created on the opposite tablespace.

An alternate version of *fedex_sql* is available which does not include designations for tablespaces. The output of this program is therefore easier to port to other relational database systems which have different configurations. This version is stored as *fedex_standardsql*.

Another version of *fedex_sql, fedex_oracle*, also exists. This version outputs the Express dictionary data in a flat file format rather than as SQL INSERT statements. One file is created for each dictionary table, and each file is named for the table that it represents. A specialized tool, such as Oracle's SQL*Loader can be used to load the dictionary tables from this output.

Attribute Description Table: EXPRESSYS$ATTRIBUTEDESC

ENTITY_SHORT_NAME	ATTRIBUTE_NAME	EXPRESS_TYPE	EXPRESS_DEFINED_TYPE	OWNERSHIP_CODE	COLUMN_NAME	SEQUENCE_NUMBER
B_SPLINE_CURVE	LOCAL_COORDINATE_SYSTEM	ENTITY	COORDINATE_SYSTEM	2	LOCAL_COORDINATE_SYSTEM	1
B_SPLINE_CURVE	AXIS	ENTITY	TRANSFORMATION	2	AXIS	2
B_SPLINE_CURVE	DEGREE	INTEGER	INTEGER	2	DEGREE	3
B_SPLINE_CURVE	UPPER_INDEX_ON_CONTROL_POINTS	INTEGER	INTEGER	2	UPPER_INDEX_ON_CONTROL_POINTS	4
B_SPLINE_CURVE	CONTROL_POINTS	AGGREGATE	AGGREGATE	2	CONTROL_POINTS	5
B_SPLINE_CURVE	UNIFORM	ENUMERATION	UNIFORM_TYPE	2	UNIFORM	6
B_SPLINE_CURVE	UPPER_INDEX_ON_KNOTS	INTEGER	INTEGER	2	UPPER_INDEX_ON_KNOTS	7
B_SPLINE_CURVE	KNOT_MULTIPLICITIES	AGGREGATE	AGGREGATE	2	KNOT_MULTIPLICITIES	8
B_SPLINE_CURVE	KNOTS	AGGREGATE	AGGREGATE	2	NOTS	9
B_SPLINE_CURVE	WEIGHTS	AGGREGATE	AGGREGATE	2	WEIGHTS	10
B_SPLINE_CURVE	FORM_NUMBER	ENUMERATION	B_SPLINE_CURVE_FORM	2	FORM_NUMBER	11
B_SPLINE_CURVE	SELF_INTERSECT	BOOLEAN	TRUE_FALSE_OR_UNDEFINED	2	SELF_INTERSECT	12

A References

[Altemueller88a] Altemueller, J., The STEP File Structure, ISO TC184/SC4/WG1 Document N279, September, 1988

[Altemeuller88b] Altemeuller, J., Mapping from Express to Physical File Structure, ISO TC184/SC4/WG1 Document N280, September, 1988

[ANSI86] American National Standards Institute, Database Language SQL, Document ANSI X3.135-1986

[Clark90a] Clark, S. N., An Introduction to The NIST PDES Toolkit, NISTIR 4336, National Institute of Standards and Technology, Gaithersburg, MD, May 1990

[Clark90b] Clark, S.N., Fed-X: The NIST Express Translator, NISTIR, National Institute of Standards and Technology, Gaithersburg, MD, forthcoming

[Clark90c] Clark, S.N., The NIST Working Form for STEP, NISTIR 4351, National Institute of Standards and Technology, Gaithersburg, MD, June 1990

[Clark90d] Clark, S.N., The NIST PDES Toolkit: Technical Fundamentals, NISTIR 4335, National Institute of Standards and Technology, Gaithersburg, MD, May 1990

[Clark90e] Clark, S.N., NIST Express Working Form Programmer's Reference, NISTIR, National Institute of Standards and Technology, Gaithersburg, MD, forthcoming

[Metz89] Metz, W.P., and K.C. Morris, Translation of an Express Schema into SQL, PDES Inc. internal document, November 1989

[Nickerson90] Nickerson, D., The NIST Database Loader: STEP Working Form to SQL, NISTIR 4337, National Institute of Standards and Technology, Gaithersburg, MD, May 1990

[Schenck89] Schenck, D., ed., Information Modeling Language Express: Language Reference Manual, ISO TC184/SC4/WG1 Document N362, May 1989

[Schenck90] Schenck, D., ed., Information Modeling Language Express: Language Reference Manual, ISO TC184/SC4/WG1 Document N466, March 1990

[Strouse90] Strouse, Kathleen and M. Mclay, PDES Testbed User's Guide, NISTIR, National Institute of Standards and Technology, Gaithersburg, MD, forthcoming

Oracle is a registered trademark of Oracle Corporation

No approval or endorsement of any commercial product by the National Institute of Standards and Technology is intended or implied

The work described was funded by the United States Government, and is not subject to copyright.

ORDER and INFORMATION FORM

MAIL TO:

National Institute of Standards and Technology
Gaithersburg MD., 20899
Metrology Building, Rm-A127
Attn: Secretary National PDES Testbed
(301) 975-3508

NATIONAL PDES TESTBED

Please send the following documents and/or software:

- ☐ Clark, S.N., An Introduction to The NIST PDES Toolkit
- ☐ Clark, S.N., The NIST PDES Toolkit: Technical Fundamentals
- ☐ Clark, S.N., Fed-X: The NIST Express Translator
- ☐ Clark, S.N., The NIST Working Form for STEP
- ☐ Clark, S.N., NIST Express Working Form Programmer's Reference
- ☐ Clark, S.N., NIST STEP Working Form Programmer's Reference,
- ☐ Clark, S.N., QDES User's Guide
- ☐ Clark, S.N., QDES Administrative Guide
- ☐ Morris, K.C., Translating Express to SQL: A User's Guide
- ☐ Nickerson, D., The NIST SQL Database Loader: STEP Working Form to SQL
- ☐ Strouse, K., McLay, M., The PDES Testbed User Guide

OTHER (PLEASE SPECIFY)

These documents and corresponding software will be available from NTIS in the future. When available, the NTIS ordering information will be forthcoming.

NIST