11

The Surrogate Data Type

Christian S. Jensen and Richard T. Snodgrass

This short chapter introduces the SURROGATE data type, which is useful when modeling time-varying objects.

Surrogates are unique identifiers that can be compared for equality, but the values of which cannot be seen by the users. In this sense, a surrogate is “pure” identity and does not describe a property (i.e., it has no observable value).

For this reason, a SURROGATE data type cannot be treated identically to how other data types are treated. For example, at tuple cannot be assigned a specific value for a SURROGATE attribute. Rather, the user must indicate that the value to be assigned must be a new value never used before, or, alternatively, must indicate some SURROGATE attribute of some tuple that the assigned value must be identical to.

For example, if surrogates are used for the identification of employees and a new tuple with information about an existing employee is to be entered then it is specified that the surrogate of the new tuple is to be identical to that of the existing tuple(s) for the particular employee.

Beyond the semantics just mentioned, it is the responsibility of the user to give meaning to surrogates. For example, the user may, or may not, use attributes of SURROGATE type as keys.

The SURROGATE data type is treated like any other data type when creating and altering relation schemas. An example follows.

```
CREATE TABLE Employee (Name CHAR,Id SURROGATE,
    Dept CHAR,Salary INT)
AS VALID STATE
```

Due to the special semantics of surrogates (e.g., values of surrogate attributes cannot be seen), update is special. With the relation instance just defined, this is an example of an insertion of a new employee.

```
INSERT INTO r VALUES (’Ben’, NEW, ’Toy’, 30)
VALID PERIOD ’1 Jan 1993 - 31 Mar 1993’
```

The reserved word NEW indicates that the system must supply a new surrogate value that has never before been used in the database. Thus, NEW is not a particular surrogate, but may be thought of as a variable that is instantiated by the system, when the insertion takes place, to a surrogate that has never been used before. For example,

```
INSERT INTO r VALUES (’Ben’, NEW, ’Toy’, 30)
VALID PERIOD ’1 Jan 1993 - 31 Mar 1993’
INSERT INTO r VALUES (’Bill’, NEW, ’Toy’, 30)
VALID PERIOD ’1 Apr 1993 - 30 Jun 1993’
```

results in two tuples, with distinct surrogates, being appended to relation r; further, the two surrogates are distinct from all other surrogates that have been used.

The next example illustrates how new information may be linked to existing information by means of surrogates. In this example, we represent individual employees by means of the surrogate-valued attribute, Id.

```
INSERT INTO Employee
    SELECT (’Benjamin’, Employee.Id, ’Toy’, 30)
    VALID PERIOD ’1 Apr 1993 - 31 May 1993’)
    FROM Employee
    WHERE Employee.Name = ’Ben’
        AND Employee OVERLAPS
            PERIOD ’5 Jan 1993 - 10 Jan 1993’
```

Here, we add more information for the same person (who changed name). We say that the Id of the new tuple should be that of the tuple with the Name value Ben some time during the specified time interval in January 1993, assuming that we know that only one person was named Ben between January 5 and January 10, 1993.

Like attributes of other data types, attributes of type SURROGATE are allowed to have null values. However, like attributes of other data types, constraints such as NOT NULL may also be imposed on attributes of SURROGATE type.

When considering the querying of relations with surrogate-valued attributes, the semantics of surrogates again have some implications. Specifically, since surrogate values cannot be viewed by the user (including application programs), an error results when a surrogate attribute is included in an outer-most target list (i.e., the outer-most SELECT clause) of a query.

Further, the use of “$*$” in the SELECT clause when argument relations contain surrogate-valued attributes needs special attention. Consider an example where we retrieve all information for employees in the Toy department. We would like to formmulate the query as follows.

```
SELECT *
FROM Employee
WHERE Employee.Dept = 'Toy'
```

To make this possible, we adopt the convention that the "*" does not select surrogate valued attributes. Thus `SELECT *` in the query above is equivalent to selecting all attributes in `Employee`, with the exception of attributes of type `SURROGATE`. As a result, the semantics of surrogates as well as the usability of the "*" notation is retained. In order to retrieve a surrogate attribute (in a subquery), the attribute must be referenced explicitly.

Next, only equality comparison is defined for surrogates. Thus, surrogates may be tested for equality (and not-equal, using, `=` and `NOT` or `<>`), but an error results when an attempt is made to apply other (e.g., "greater-than") predicates to surrogates.

Surrogates do not replace keys, but rather supplement them. While surrogates may be used for connecting information within the database, values of surrogates have no real-world meaning. Keys, on the other hand, may be used for relating information in the database with real-world entities. In the sample table `Employee`, the attribute `Name` may be declared as a key (e.g., `PRIMARY KEY ( Name )`). It is via `Name`, rather than via `Id`, that the user establishes the connection between a tuple and the actual employee the tuple is about.