Chapter 3

DATABASE PROGRAMMING LANGUAGE

This chapter describes an object manipulation language called *JasminelC* [ISHI93a] as a database programming language which integrates a general-purpose programming language (C) and a database language in an object-oriented context, and which allows the user to program advanced applications. The query language as a database language of Jasmine has the following features different from those of SQL [DATE90] (see Appendix for the complete syntax of a query expression). The semantics can be formally defined through query translation by object operators.

A query consists of target and condition parts. The basic unit of a query expression is an object expression, a class name followed by a series of attribute names. The target part is an object expression, or a list of object expressions. The condition part consists of a logical combination of predicates which compare object expressions. The query returns a set of objects. The object expressions denote object joins, or implicit joins. The object expressions can reference complex objects directly. Nonleaf classes in a class hierarchy can be specified in a query. They are expanded or specialized into the leaf subclasses. Like SQL, explicit joins specified by join predicates are supported by the query language. However, the compared objects are not confined to atomic values. Equality of objects can be tested based on their object identifiers. In that case, the compared classes must be either identical or a superclass and a subclass of a class hierarchy. The object expressions can also contain methods, so the user can manipulate objects set-theoretically and filter a set of objects procedurally. The system-defined methods such as put, delete specified in a query can modify a set of objects. A query can invoke demons which implement integrity facilities introduced by QBE [ZLOO78]. The user can specify multiple-valued attributes in a query. The user can control unnesting of multiple values and apply aggregate functions correctly. Multiple-valued attributes are existentially or universally quantified.

Another important feature for advanced applications is the integration of query and programming facilities. First, the user can specify methods in a query as described above. The user can extend the functionality of the query language just by defining and specifying a method in a query, without modifying the query language processor. The
user can develop application programs more compactly without specifying details such as iteration variable declaration and control structures. Making this type of iteration implicit can increase physical data independence [DATE90] of application programs by allowing the system to optimize the query expression.

Second, the user can also define methods by specifying a query for them. This can define so-called virtual attributes and increase logical data independence [DATE90] of application programs when applications evolve.

Third, the fact that the user invokes a query from programs is one of salient aspects of advanced applications. The so-called impedance mismatch problem [DATE90] between the query language and programming language must be solved. We introduce set variables to attain the goal. The set variable has a class defined by an object model as its type and can contain a set of objects returned by a query as its value. The user can fetch an object by sending the scan message to the set variable and operate on the object by sending a dedicated message to the object in an object-oriented programming manner.

Class objects can also be operated set-theoretically for advanced applications. Basic database functions such as transactions, locking, and logging can be provided through system-defined classes. Multimedia data types and operations are provided by implementing them from system-defined primitive classes in a bootstrap manner.

### 3.1 Set-oriented Access

Set-oriented access allows objects to be associatively operated on. It obtains a set of objects by filtering them using attribute conditions.

#### 3.1.1 Object expressions

The basic unit of set-oriented access is an object expression, a class name followed by a series of zero or more attribute names:

```
class
class.attribute-1
class.attribute-1.attribute-2 ...attribute-n
```

The object expression `class` denotes a set of instance objects according to the class semantics. The expression `class.attribute-1` denotes a set of instance objects as the attribute value. In general, the object expression evaluates to a set of objects as the values of the last attribute. As attributes are functions, the object expression corresponds to a functional composition which denotes a functional join or implicit join.