CHAPTER 7

CONCLUSION

Even though the conventional database systems have excellent capabilities for managing large amount of data, they are limited in their structuring facilities. Object-Oriented programming languages provide excellent facilities for handling complex entities, but they are limited in their secondary storage capabilities. Object-Oriented database systems attempt to synthesize object-oriented programming languages and conventional database systems.

In an object-oriented DBMSs, new data types and operations can be added to define new entities and relationships. This provides a way for interfacing specialized hardware or software processors to the DBMS. The data types and operations implemented by these processors are defined in the schema as objects, classes and operations. By virtue of this, these systems have the potential of being used as the main vehicle for implementing DBMSs for Office Automation, CAD/CAM and other similar application environments.

In Chapter 2, we reviewed the research efforts in the area of object-oriented database modelling. The object-oriented paradigm and abstraction mechanisms were discussed. Also, the conceptual schema and data operations in object-oriented data model were described.
Further, the advantages of object-oriented database systems over conventional record-based database systems are discussed. Also, in this chapter we classified existing efforts in creating object-oriented DBMSs into three categories: (i) those that are directly based on the object-oriented paradigm (ORION, OZ+, IRIS, CACTIS); (ii) extensions to relational systems (POSTGRES, STARBURST); (iii) experimental toolkits or storage systems (EXODUS). The salient features of the above prototype models were reviewed. This was followed by a comparison of these prototype models.

In Chapter 3, we have suggested a new scheme that integrates declarative approach to represent graphics and object-oriented data modelling techniques to built a constraint-based graphics database system. This approach has a number of advantages:

- The schema is intuitive and easy to understand.
- It has rich modelling constructs to describe graphics data.
- It provides input mechanisms which enable both 2-dimensional and 3-dimensional graphical objects to be put to real use. The hierarchical composition allows graphics primitives to be grouped together. The higher-level object is used to define common properties.
- It provides a useful level of abstraction. The graphical objects can be abstracted and mapped naturally to this schema.

- This approach allows the sharing of representation rather than sharing of drawings.

- It allows individual objects to be addressed and manipulated.

- It enables constraints to be specified on some instances of a class. This is essential, as the designer often wants to freeze some parts of the design at certain stage.

- It supports part hierarchies and allows parts to share common geometric description. It also enables constraints to be imposed on the components of complex objects. The components can be manipulated individually under these constraints.

- It provides useful mechanisms for management of constraints and consistency problems occurring at the time of insertion, deletion and modification operations.

- It caters for multiple output devices and works in world coordinates.
In Chapter 4, we have also identified important classes of constraints in the context of object-oriented graphics database systems. These are class constraints, instance constraints and structural constraints. In class constraints; domain constraints, intra-object constraints, inter-object constraints and reference constraints are one-way constraints whereas association constraints are two-way constraints. Instance constraints are two-way constraints. Under structural constraints, we have further identified geometric and appearance constraints, connectivity constraints and sharing constraints. Our scheme also provides useful mechanisms for management of integrity constraints. Algorithms were given for maintenance of constraints at the time of insertion, deletion and modification.

In Chapter 5, we have presented a formal model for object-oriented databases. In our approach we have separated the concepts of types and classes. This gives rise to clear semantics and thus enables us to define classes which share the same type, and separate the two meanings of "is-a" relationship into subtype/supertype and subclass/superclass relationships. We have shown that these relations are partial order.

We have also defined operations on classes that give rise to new classes along the lines of relational databases. Union, Intersection, Difference and Select operations give rise to new classes whereas Cartesian
Product and Partition operations give rise to new types in addition to new classes. Since new types and classes cannot exist in isolation we have also identified their position in the type hierarchy and class hierarchy respectively. These operations allow queries against multiple target classes and enable us to extract attributes similar to relational project.

In Chapter 6, following the approach of separating types and classes we have extended the relational algebra operations and set theoretic operations on complex objects. We allowed new class hierarchies to be defined as a result of applying these operations. This enables us to have queries that involve:

- retrieval of a class of complex objects. Such queries involve CHOOSE operations which is similar to relational select operation;

- retrieval of a class of complex objects with selected attributes from complex object hierarchy rooted at the given class. PARTITION operation is used for this purpose and it is similar to relational project;

- retrieval of a class of complex objects by combining complex objects from two component-class hierarchies rooted at the given classes. Such queries involve CARTESIAN PRODUCT and CHOOSE operations. The CARTESIAN PRODUCT operation has limitation because the objects belonging only to the root classes can be
combined; and

- conditions on two or more branches of the complex object hierarchy. These queries involve set-theoretic operations and choose operation. The cartesian product and set-theoretic operations enable multiple target queries.

The constraint maintenance is considered only at insertion, deletion and modification. The constraint enforcement at the end of transaction, or after completion of certain design phase may also be considered. Insertion, deletion and modification operations invoke several functions. Optimal combinations of functions may be considered.

Operations on complex objects can be generalized by allowing a complex attribute to be set-valued. We have restricted component-type hierarchies and component-class hierarchies to be rooted directed trees. These can be generalized to rooted directed graphs or graphs with cycles. Further, we may allow a component object to have more than one parent thereby allowing sharing of components.