CHAPTER 1

1. INTRODUCTION

1.1 INTRODUCTION

Database management system (DBMS) is the vital part of current information system technology and they provide reliable and effective mechanism for storing and managing large volumes of information in a multiuser environment. Designing of DBMS becomes essential for managing huge amount of data to meet the informational needs of any user. Recently, database research is aimed at increasing the functionality and performance of DBMS to accommodate the requirements of modern database applications.

1.1.1 Database management systems

A database management system is a software system for storing and managing a large volume of data for allowing multiple users to access and manipulate data in a consistent and secured manner [Connolly and Begg, 2004]. A database system is a database management system together with one or more databases. The three important features of the DBMS are [Tay, 1990]: persistency – if a program modifies certain data, the changes remain even after the program has terminated; sharing – more than one program can concurrently access the data; and reliability – the data must remain correct despite the hardware and software failures.

Typically, a DBMS provides the various facilities such as Data Definition Language (DDL), Data Manipulation Language (DML) and several mechanisms for controlled access of database. DDL allows users to define the database, specify the data types, data structures and the constraints on the data to be stored in the database and it also translates the schema written in a source language into the object schema. DML allows users to insert, update, delete and retrieve data from the database and it also provides general query facilities through query language. The DBMS provides mechanism (software) for controlled access to the database such as security system for preventing unauthorized user trying to access the database, concurrency control
system to allow shared access of the database and recovery system to a database to restore a previous consistent state because of hardware or software failure.

DBMSs are highly complex and sophisticated pieces of software that aim to provide the types of functions and services include data storage, retrieval and update, a user-accessible catalog, transaction support, concurrency control services, recovery services, authorization services support for data communication, integrity services, services to promote data independence and utility services. In order to provide these services, the DBMS should be designed with the different interrelated components include query processor, database manager, file manager, DML processor, DDL processor and catalog manager. Database manager is the central software component of the DBMS which interfaces with user submitted application programs and queries. Database manager is also called as database control system and it handles database access at runtime with the subcomponents such as authorization control, command processor, integrity checker, query optimizer, transaction manager, scheduler, recovery manager and buffer manager and the various components of a database manager are shown in Figure 1.1 [Connolly and Begg, 2004].

![Figure 1.1 Components of a database manager](Connolly and Begg, 2004)
In general database systems are classified based on various criteria such as number of users use the system, the type and extent of use and the database site locations. Classifying a database system based on the number of users are single user DBMS and multiuser DBMS [Elmasri and Navathe, 2011]. Single user DBMS is restricted to personal computer systems with no sharing of data. Multiuser DBMS allows many users to access shared data at the same time. On the basis of the type and the extent of use, the various types of database systems are transactional DBMS, decision support DBMS and data warehouse. The various database systems on the basis of database site locations are centralized DBMS, distributed DBMS and parallel DBMS. The centralized database system consists of a single processor together with its associated storage and peripheral devices and is physically confined to a single location. In a distributed database system, multiple processors are loosely coupled and geographically distributed at several sites communicating via networks with no sharing of physical components. In parallel database systems, multiple processors are tightly coupled which work in parallel and are physically located in a close environment in a building and communicating at very high speed.

1.1.2 Database models

An important aspect in DBMS evolution is the development sophisticated database models. Database models provide a way in which the stored data is organized as specified structure for quick access and efficient management [Connolly and Begg, 2004]. The three different categories of data models are record based, object based and physical data models.

In a record based model, the database consists of a number of fixed format records with different types. Each record type defines a fixed number of fields with a fixed length for each. The three types of record based data model are relational data model, network data model and hierarchical data model. In relational model data and relationships are represented as tables, each of which has a number of columns with unique name. In the network model data is represented as collections of records and relationships are represented by sets. The records are organized as generalized graph structures with records appearing as nodes and sets on edges in the graph. In the hierarchical model data is represented as collection of records and relationship are represented by sets. A hierarchical model is represented by a tree graph with records.
appearing as nodes or segments and sets as edges. Record based data models are used to specify the overall structure of the database and a higher level description of the implementation.

Object based models use concepts such as entities, attributes and relationships. An entity is a distinct object in the organization that is to be represented in the database. An attribute describes the state of the object and a relationship is an association between entities. The various object based data models are entity-relationship (ER) model, semantic model, functional model and object-oriented model. The ER model has emerged as one of the main techniques for database design. Object-oriented data model extends the attribute property that describes both the state and the behavior of the object and it also captures the semantics of objects with object oriented programming concepts. The important data models that have been used in the research and commercial database systems are the relational model, the object-oriented data model and object-relational data model [Bergholt et al., 1998].

Relational data model

Relational data models deals with the data aspect of information models and ignoring the dynamic or behavioral aspects of information modeling. In the relational data model, information is organized in relations (two-dimensional tables). Each relation contains a set of tuples (records). Each tuple contain a number of fields. A field may contain a simple value (fixed or variable size) based on the domain. The relational data model is based on a mathematical foundation, called relational algebra. Relational modeling focuses on the information in the system and not on the behavior. The modeling dimension has been accomplished in relational database systems through well defined aspects like domain, relation, tuple, attribute, relation values, relation keys, views and normal forms [Jackson, 1999]. The various commercial relational DBMSs such as IBM’s DB2, Dynamic Server from Informix, System 10/11 from Sybase, Oracle 7.x from Oracle Corporation have been developed based on the relational model [McClure, 1997].

The various advantages of using relational model are: most mature and robust data model, good model for handling non-complex data types, providing features such as structured query language (SQL), integrity constraints and access control.
However the disadvantage of the relational model is not object-oriented. The lack of mechanism for supporting object-oriented features in relational model has not been able to handle object-oriented applications with complex data types and their manipulations [Bergholt et al., 1998; Maatuk et al., 2010; Butuner, 2012].

**Object-oriented data model**

The object-oriented data model deals with both the data and the dynamic and behavioral aspects of data modeling [Bergholt et al., 1998]. In the object-oriented data model, information is organized in terms of objects, where each object has a number of attributes. Attributes are simple values, complex values, references to other objects, or methods. Objects are instances of classes, and classes are related to each by means of inheritance. The inheritance mechanism supports generalization and specialization and offers many aspects of structured reuse of models. Inheritance also offers the mechanism for qualified polymorphism, since the resulting type system can allow objects to be recognized as belonging to several different types. A method of an object is a specification of functionality, typically manipulations of the attributes in the same object. The various commercial object-oriented DBMSs such as Jasmine from Computer Associates, Gemstone’s Gemstone, O2’s O2, Object Design’s Object Store, have been developed based on the object-oriented model [McClure, 1997].

The various advantages of using object-oriented data model are [Kim, 1993; Bagui, 2003]: support for handling complex data types with object-oriented programming features, reusability of objects and methods, notion of enforcing semantic integrity constraints for attaching different meanings to generic data types. The main disadvantage of this model has been the lack of standard and thus it has not been compatible with other models.

**Object-relational data model**

Object-relational data model combines the benefits of both relational and object-oriented data model [Bergholt et al., 1998]. This model has been proposed to extend the relational model with the support of object-oriented programming features such as encapsulation, inheritance, class definitions and support for complex unstructured data [Zhang et al., 1997].
An object-relational model can be thought of as a relational model where tables are considered as classes, tuples are object instantiations of classes, and columns of tables are attributes of their corresponding classes. Class inheritance is supported by allowing tables (sub tables) to inherit all or certain columns, constraints and storage options of other tables. Within an object-relational model, attribute domains are abstract data types (ADT), user-defined data types (UDT) or any of the data types supported in relational model [Zhang and Harder, 1997; Zhang et al., 2001; Devarkonda, 2001]. ADT are data types constructed from other built in data types (integer, string char etc.). UDT are data types whose internal structures are completely opaque to the database and the database has no intrinsic mechanism to access such data. Therefore all UDTs must be accompanied by user-defined access methods for reading, writing, indexing and querying information within them. Allowing UDTs to have access these methods has support encapsulation. Encapsulation has also been supported by this model when methods are assigned to tables and serve as mechanisms through which users’ access and manipulate table attributes. As like relational model tables can have constraints, storage options, indexes, methods and triggers.

The various advantages of using object-relational data model are: exploitation of large relational user-based mature technologies and standards, enforcement of semantic integrity constraints, no interface mismatch problem for storing object-oriented data and support for complex data types. The disadvantage of this model has been the difference in implementation among vendors to support for user defined data types, data type extensions. However these difficulties have been rectified by the SQL3 standard [Eisenberg and Melton, 1999]. The SQL3 standard provide a complete model for managing persistent objects and also provide vendors with a neutral model for implementing portable database environments [Eisenberg et al., 2004]. Oracles 8.x from Oracle, Universal Server (Illustra) from Informix, Universal Database from IBM are the various commercial object-relational DBMSs that use both relational and object-oriented model [McClure, 1997].

1.1.3 Architecture and applications of DBMS

The database system architecture is a framework in which the structure of the DBMS is described. The way the components of the DBMS work together to achieve
certain goals is referred as the structure or system architecture. DBMSs are very complex, sophisticated software applications that provide reliable management of large data. The different ways to look at the architecture of a DBMS are logical, physical and multi user DBMS architecture.

**Logical architecture**

The logical architecture deals with the way data is stored and presented to users. The logical architecture describes how data in the database is perceived by users. It is not concerned with how data is handled and processed by the DBMS, but only how it looks. Since a database is a shared resource, each user may require different view of the data held in the database. Users are shielded from the way data is stored on the underlying system and manipulate the data without worrying about where it is located and how it is actually stored. This result in the database having three levels of abstraction based on the ANSI/SPARC architecture includes:

- **Conceptual level** - The conceptual level presents a logical view of the entire database in its totality, which allows one to bring all the data in the database together and see it in a consistent manner. This level describes what data is stored in the database and the relationships among the data. This level represents all entities, their attributes, and their relationships, the constraints on the data, semantic information about the data, security and integrity information.

- **External or view level** - The external view describes that part of the database that is relevant to each user. (i.e.) The users’ view of the database. This level provides a window on the conceptual view which allows the user (application program or end user) to see only the data of interest to them.

- **Physical or internal level** – The physical level described how data is stored in the database. The collection of files permanently stored on secondary storage device is known as the physical storage and it provides a low-level description of the physical database and an interface between the operating file system and the record structures used in higher levels of abstraction.
Physical architecture

The physical architecture is concerned with the software components that make up a DBMS. The physical architecture describes the software components used to enter and process data and how these software components are related and interconnected. Although it is not possible to generalize the component structure of a DBMS, it is possible to identify a number of key functions which are common to most database management systems. At its most basic level the physical DBMS architecture is broken into two parts: the front end and back end. The front end is really any application that runs on top of the DBMS. These applications may be provided by the DBMS vendor, the user or a third party. The user interacts with the front end, and may not even be aware that the back end exists. The back end is responsible for managing the physical database and provides the necessary support and mappings for the internal, conceptual and external levels.

Client/server architecture

The construction of high performance database systems that combine the aspects of relational and object-oriented approaches requires the design of flexible architecture that can fully exploit the resources available in the system. The most common and flexible architecture that is used to implement multiuser database management system is client/server architecture [Elmasri and Navathe, 2011]. Client/server architecture refers to the way in which software components interact. The client/server architecture of a database system has two logical components namely client and server, each executes on different systems and are connected into a network. The client process that requires certain resource and the server process that provides the resource. The applications and tools act as clients of the DBMS and the DBMS software act as the server. The client takes the user’s request, checks the syntax and generates the database requests appropriate to the application logic and then it transmits the message to the server, waits for response and formats the response for the end-user. The server accepts and processes the database requests, then transmits the results back to the client. The processing involves checking authorization, ensuring integrity, maintaining system catalog, performing query and update processing, concurrency control and recovery.
Client/server architecture has been traditionally designed with 2-tier client/server model consists of the client (tier 1) responsible to handle user interface actions and the business application logic. The need for enterprise scalability and the complexity of advanced applications 2-tier model has been replaced by the 3-tier architecture. The 3-tier model consists of a client (tier 1) responsible for user interface, a business logic or application server (tier-2) and a DBMS or database server (tier-3).

**Database applications**

Generally database management systems are accessed either by the user program or application. Users interact with the database system by executing applications, where each application consists of set of operations to be performed on the stored data. Database application stores its data in a database and uses the services of a DBMS to retrieve and update the data and also to protect and maintain its integrity. Database applications construct and submit transactions to manipulate the persistent data stored by a DBMS. Recently DBMSs have increasingly been used for advanced database application domains and different tools are used to support the requirements of the application. However database systems are beginning to be applied to a different range of application domains associated with highly information processing with complex data, conventional database environment has proved to be unsatisfactory [Feuerlicht, 2010].

The database technology has to support applications that involve complex models and analysis methods in combination with high requirements on computational efficiency. Therefore database systems require more comprehensive facilities for modeling both structural and behavioral aspects of advanced applications. Hence researches in database technology have focused on extending conventional database systems to enhance their functionality and to accommodate the requirements of advanced applications.

**1.1.4 Characteristics and objectives of database systems**

The main objectives of a DBMS include an efficient, flexible, reliable and secure management of data [Bergholt et al., 1998]. Certainly, the value and
importance of these aspects vary among application areas as well as for specific purposes within application areas. However to meet these objectives a DBMS can include the software that provide:

- **Data modeling capabilities** - A data model includes a set of predefined constructs for structuring data that can involve predefined data types, basic operations, and user defined data structures. An important aspect of data modeling is the ability of the user to define complex data objects and relationships. A data model also supports the design and use of standardized data representations that can facilitate reuse, exchange of data among applications or users.

- **High-level database language** - A high-level database language (DBL) usually referred to as the query language that provides the interface to the database. The DBL is used directly and interactively for defining, manipulating and querying of data. DBL statements can also be embedded in a host language (the implementation language of the application) for indirectly accessing data in the database. In a DBL, data management is usually specified more declaratively than in a conventional programming language.

- **Persistent storage of data** - Data and program procedures are stored permanently on secondary storage for later retrieval after the termination of program execution. Persistency implies a process of how to store objects as well as a persistence mechanism. Transferring complex data between a DBMS and applications can give rise to an impedance mismatch problem which has to be solved. Impedance mismatch problem has been solved by the integration of object-oriented programming systems into databases (e.g. OODBMS, ORDBMS) to store and manipulate programming language objects persistently in the database.

- **Efficient accessibility of data** - The DBMS supports facilities for creating access structures, or indexes, which make access of data elements efficient. There are general indexing techniques such as various tree data structures and hash tables, and techniques specialized for certain types of
data, such as quad trees for spatial data. The DBMS usually has facilities for optimizing queries, i.e. transforming a query into a form that has an effective execution order.

- Improved data integrity - Database integrity refers to the validity and consistency of stored data. The DBMS can specify integrity in terms of constraints, which are consistency rules that the database is not permitted to violate.

- Handling of hardware and software failures - To assure that the database is recovered from various types of hardware and software failures, the DBMS should include the facilities for logging of transactions, making backup of the database, and recovery procedures to recover from failures and restore the database into its last consistent state.

- Data sharing through concurrency control and transaction processing - Operations in a DBMS are performed as transactions that are logical operation units on the database. The transaction processing software controls the state of transactions and guarantees that the database has always return to a consistent state. Further, the DBMS can use concurrency control to ensure that several users can access and update the same data element in a controlled manner and guaranteeing a correct result. Thus, transactions and concurrency control ensure multiuser transactions to be performed correctly and several users can share data without bothering about interfering with other users.

- Access control through authorization - The DBMS must include authorization mechanisms to access the database in terms of privileges that provide security for protecting the database from unauthorized users.

- Supports for active behavior – Enriching database systems with active behavior provide the ability to detect the occurrence of events and respond to them automatically in a timely manner without user or application explicit request. Active behavior of a database is defined through active rules which has in the form of event-condition-action (ECA) rules.
Database systems coupled with active rules are known as active database systems.

- Support for architecture and applications - The DBMS should provide an effective architecture for the interaction among the database software components as well as provide the facility for modeling, storing and manipulating complex data that are required by the advanced applications.

1.2 ADVANCED DATABASES WITH OBJECT-ORIENTED SYSTEMS

The field of database research is defined by its previous successes, and the current research is focused to increase the functionality and performance of DBMSs [Stonebraker et al., 2013]. During the past years DBMSs have undergone drastic changes as a result of the increasing requirements of emerging database applications.

The achievements in database research and the opportunities for future database research related with the major categories such as technological infrastructure, support for complex data types, support for advanced databases, handling of complex application environment, redesigning of database architecture and easy use of database management system have been discussed [Silberschatz et al., 1995; Silberschatz and Zdonik et al., 1996; Bernstein et al., 1998; Abiteboul et al., 2005; Agrawal et al., 2009]. A brief overview of different research aspects in various database systems such as object database, active database, real-time database, mobile and multimedia databases have been presented [Kumar, 2012].

The conventional relational DBMS (RDBMS) based on relational model has proven to be very successful in supporting business data processing applications such as inventory control system, payroll system, financial system, and reservation system. The advantages of relational database systems include data independence, multiple views of data and a non-procedural set oriented language that is suitable for atomic query optimization. RDBMS allows the physical design of database to change without impacting existing applications, eliminating the need for all future applications to be anticipated in the original database design.
Even though RDBMS has proved to be an effective solution for structured data management requirements in large and small organizations, the emergence of nontraditional advanced application areas such as computer-aided design (CAD), computer-aided manufacturing (CAM), computer-aided software engineering (CASE), office information system (OIS), multimedia system, digital publishing, interactive and dynamic websites, scientific and medical applications, handle complex object-oriented data which are difficult to represent in RDBMS, since RDBMS has supported only structured, pre-defined data types with simple modeling and not designed to manage arbitrary user-defined data types with complex modeling. Using a relational database system for an object-oriented application requires, additional processing and complexity for moving objects and mapping complex data into the relational structure. The limitations of relational database system demand the need for advanced database systems integrated with object-oriented programming system for modeling, storing and manipulating large complex data as well as the object support at the database level [Kim, 1990; Premerlani et al., 1990].

Object-oriented programming system (OOPS) is a paradigm that uses objects, data structures consisting of data fields and methods together with the support of object-oriented concepts. The set of object-oriented concepts found in OOPS forms a good basis for an object-oriented data model which extends the domain of database applications beyond the traditional business data processing applications [Kim, 1990]. The integration of database systems and OOPS provides an integrated application development environment with many advantages including the support for: rich data types, object-oriented programming features such as objects, methods and classes, abstraction, encapsulation, inheritance, polymorphism, overriding, maintainability, ease of modeling real world objects and database facilities for developing, modifying, and extending complex data intensive software systems.

The limited modeling capabilities of RDBMS and the increasing complexity of advanced database applications, object-oriented database management system (OODBMS) has been proposed as a technology for supporting the object-oriented advanced applications mainly by providing the capabilities for modeling, storing and manipulating complex objects with object-oriented programming concepts [Kim, 1993]. The promises and reality of object-oriented database system compared with
relational database system has been reviewed [Kim, 1993]. Object-oriented database systems improve relational database systems by offering complex data structures, object identity, encapsulation, inheritance between classes, and extensibility through object-oriented data model.

Even though OODBMS is well suited for storing and retrieving complex data and also capture the state as well as behavior of the objects still it lacks the standard model as known in the relational model. Also OODBMS is not compatible with RDBMS and do not include several RDBMS features such as a complete declarative query language, metadata management, views and authorization [Stonebraker et al., 1999]. The limitations of OODBMS and RDBMS made database developers generate another type of database system namely object-relational database systems. Object-relational database management system (ORDBMS) has been proposed as a unification of relational database technology with object-oriented programming concepts through both relational and object-oriented data model to support the management of complex data for advanced applications [Kim, 1996; Stonebraker et al., 1999]. ORDBMS provides declarative object-oriented query languages and SQL3 for defining, accessing and updating objects. Object classes are defined as types and object attributes and methods are defined as functions.

In general, the above database systems have been viewed as a passive repository for managing data in response to the user requests or application program and not initiating any operations on their own. However certain applications includes financial applications, workflow management, computer integrated manufacturing, stock market systems, air traffic control, threat assessment and analysis, nuclear process control, and inventory control system, require monitor changes to the database state and initiate appropriate actions. For example, an inventory control system needs to monitor the quantity of stock of items in the inventory database, so that when the quantity in stock of the item falls below the threshold value, a reordering activity may be initiated.

In conventional DBMS, two approaches are used to support monitoring for the above category of applications. One approach is, special purpose program can be written to poll the database periodically to check for relevant conditions and other approach is, embedding the code for checking the relevant condition in application
programs updating the database. However the above two approaches cannot efficiently support reactive situation monitoring. Since the disadvantage of the first approach is, if the polling is too frequent, the system performance is degraded because of overloaded queries and if the polling frequency is too low, the relevant condition may not be detected in time. The second approach leads to redundancy, distribution and difficult maintainability as a result of the check being replicated and distributed among several application programs. Hence the need for reactive mechanisms has been recognized as an important feature in next generation database systems [Stonebraker et al., 1990; Dittrich et al., 1995].

The integration of active rule systems by ECA rule paradigm into database systems called active database systems that have been proposed to move the reactive behavior from the application into the DBMS [Stonebraker, 1992]. In other words, ECA rules provide active database systems, the capability to monitor and react to specific situations that are relevant to an application. Active database system has been proposed as a reactive mechanism to perform certain operations automatically, when specified events occur and particular conditions being satisfied based on the knowledge stored inside the database system in the form of event-condition-action (ECA) rule paradigm [Dittrich et al., 1995].

Moving the reactive behavior in the form of ECA rules into the DBMS enable the application developers to easily find and modify the rules, when a policy change is needed or when application requirements change. This makes it possible to quickly change the rules without modifying the rest of the application code, thereby enhancing maintainability. The integration of active rules facility into the database system support a number of database features including integrity constraint enforcement, derived data maintenance, implementation of triggers and alerters, protection and version control, gather statistics for query optimization [Dayal et al., 1994]. Active database systems have been proposed as a very promising technology [Dittrich et al., 1995; Dittrich and Simon, 1999] and its benefits and realities have been discussed [Simon and Dittrich, 1995]. The utility and functionality of the active capability has been well established in database systems and a lot of research has been conducted in this field [Chakravarthy, 1992]. Applications which depend on data monitoring activities such as CIM, telecommunications network management,
medical and financial decision support systems are greatly benefited from the integration of active database systems.

Even though the database research community has responded to numerous challenges from changes in application requirements and advances in hardware and software, still object-relational database systems with reactive mechanisms has been the successful and the dominant database technology for handling advanced applications that handle data with complex structures and manipulated by complex operations [Gray, 2004; Feuerlicht, 2010]. Hence this research work focuses on extending the functionality of active object-relational database systems for satisfying the requirements of emerging advanced applications.

1.3 ACTIVE DATABASE SYSTEMS

Active database systems (ADBS) that extend conventional database systems by supporting mechanisms to automatically monitor and react to events that are taking place either inside or outside of the database systems by using active rules in the form of event-condition-action (ECA) paradigm [Dittrich et al., 1995]. In order to achieve the reactive mechanism, an active database system has to perform the following two functions: (a) specifying the reactive behavior – rule languages are used to define the syntax and specification of ECA rules, (b) executing the reactive behavior – mechanisms for monitoring relevant situations and reacting to them are required and this function has been carried out by the ADBS through a strategy called rule processing.

1.3.1 Specification of reactive behavior

ADBS are centered on the notion of rule. The reactive behavior of a system is specified by means of rules which describe the situations to be monitored and the reactions of the system when these situations are encountered. Thus rules are syntactic constructs by means of which the reactions of the system are specified. Active rules are defined based on the ECA rule paradigm. A rule definition language provides constructs for specifying rules. The result of the rule specification activity is a set of rule definitions or rules. Depending on the rule language the elements of a rule is
defined and the general syntax for defining an ECA-rule is shown in Figure 1.2 [Dayal et al., 1994].

![Defined rule syntax](image)

**Figure 1.2 General syntax for defining an ECA rule**

The define rule clause of a rule defines a unique rule-name of the rule. The on-clause of a rule defines the types of events that may trigger the rule. The if-clause of a rule defines the conditions that are evaluated when the rules are triggered. The action after then-clause of a rule is a sequence of operations to be performed when the rule is triggered and its condition is satisfied. However in certain ADBS systems the rule format has been specified in different formats: EC-A (Condition-Action, where only the condition and action are specified. If the event is omitted then the rule is executed when the condition becomes true) and E-CA (Event-Action, where only event and action are specified. If the condition is omitted it is always true).

**Event specification**

An event is a relevant happening that has to be monitored by the ADBS. The event part of a rule specifies the situations to which the rule may be able to respond. Rules are triggered on the occurrence of various events such as database operations, occurrence of database states and transitions between states. An event may be categorized as either primitive or composite (Dayal et al., 1994; Buchmann et al., 1995; Gatziu and Dittrich, 1993).

Primitive events are atomic events which are associated with a point in time. Primitive events are events that are defined in the system. Primitive events are classified based on their origin as internal and external. Internal events are associated with the access to the database and the external events are produced by the occurrences outside the database in its environment.
The various primitive internal events are:

- Database operation events occur at the beginning or at the end of a database operation. Thus the event definition has to specify if the instances have to occur before or after the operation has been performed. In relational database systems these events are SQL modification operations such as insert, update, delete and the retrieval operation select applied on a certain table. In object oriented database systems data operations are performed through the methods invocation that access persistent collection of objects. Method events are signaled when the persistent objects are created (constructor is called) or deleted (destructor is called) or when methods modifying or retrieving objects are invoked.

- Transaction events occur before or after a transaction operation such as begin, end, abort and commit.

The various primitive external events are:

- Temporal (absolute/relative) events occur either at absolute time points (e.g., 6.8.2013, 10.00) or periodically (e.g., every day at 10.00).

- Abstract events are user-defined events whose occurrences are directly signaled and they are explicitly raised by the user and associated with a point in time.

Composite events express more semantics than primitive events. Composite events are defined as a combination of primitive and possibly by other composite events. The meaningful ways to build composite events from its constituent events are usually specified through event algebra that defines certain event constructors. Thus composite events consist of primitive or composite events combined with logical operators. The various composite event constructors are:

- Disjunction – Events are combined using the “or” operator. The disjunction of two events, event1 and event2, is raised when either of event1 or event2 occurs.
- **Conjunction** – Events are combined using the “and” operator. The conjunction of two events, event1 and event2, is raised when both event1 and event2 occurs.

- **Sequence** – Events are to be raised one after another and in the right order. The sequence of two events, event1 and event2, is raised when event1 and event2 occur in that order.

- **Closure** – The closure of an event, event1 is raised exactly once regardless of the number of times event1 occurs.

- **Negation** – The negation of an event, event1 is raised if event1 does not occur in a given time interval.

- **History** – The history of an event, event1 is raised, if event1 occurs a given number of times.

For the last three event constructors, it is appropriate to define time intervals in which composition of events should take place. The definition of a time interval is mandatory for negation and optional for history and closure. Triggering events can also be parameterized. When a parameterized event occurs, values related to the event are bound to the event parameters and these parameter values are referred in rule conditions or actions.

**Condition specification**

The condition of a rule checks the state of a database at the time when the rule event occurs. The condition part of a rule is usually expressed as a Boolean expression, a predicate, or a set of queries and it is satisfied if the expression evaluates to true or all the queries return non-empty results. In addition to the current state of the database, the condition may access the state of the database at the time of event occurrence by use of event parameters.

**Action specification**

The action part of a rule specifies the operations to be performed when the rule is triggered and its condition is satisfied. In general an action can be database
operations, transaction commands or arbitrary executable routines. Therefore during the execution of an action certain events may also occur. This may lead to the triggering of other rules which is called nested or cascaded rule triggering. The action may access besides the current database state, the database state at the time of event occurrence and the time of condition evaluation which are performed by parameter passing.

1.3.2 Rule processing

The behavior of active database system depends on, not only the set of rules given to the system but also the strategy of rule processing adopted by the system. Applications programs are executed until an event is signaled. If an event is signaled, the control moves to the rule processing. Signaled events trigger appropriate rules which are inserted into the set of triggered rules. Then, rules are selected according to certain criteria from this set, their condition is evaluated and finally their action is executed. If the execution of the action signals further events, further rules are inserted into the set of triggered rules. When the set of triggered rules is empty, the control move backs to the application program. The various activities that have been performed by the ADBS during rule processing are described.

- **Event signaling** - The primitive event detector detects primitive events when it occurs. Also the composite event detector considers the primitive event occurrences that contribute to the composite events. The detection is carried out until the composite event detector arrives to a final state and no more composite events may be detected. The result of event detection is the signaling of events. First primitive events are signaled and then composite events are signaled. Then the system assigns unique timestamp for events.

- **Rule triggering** - This phase is performed immediately after event signaling. It applies to only rule events. Considering the events in the order given by their timestamps, corresponding rule definitions are retrieved from the set. For each retrieved definition, a rule instance is created. Rule instances are inserted into conflict set, which is the set of all triggered and
not yet executed rules. In order to execute the triggered rules a selection criterion has to be applied.

- **Rule selection** - When multiple rules are triggered by a single event or fired by many different events at the same time, active database system has to provide certain techniques to schedule the execution order of multiple triggered rules at a given point in time. In ADBS, rule scheduler uses the deterministic (static) and non-deterministic (dynamic) priority policies to specify the order in which rules are selected for execution. The three techniques that are used by the rule scheduler during rule processing are [Cilia, 2006]: (a) one rule is selected arbitrarily for execution in the set; (b) sequential execution of all rules in a set; (c) parallel execution of all rules in a set. Parallel execution of rules is achieved through concurrent execution of all rules by using a nested transaction model.

- **Rule evaluation and execution** - Rule evaluation evaluates the condition of the selected rule. If the condition is satisfied the action is executed. Rule execution is also called as rule firing which represents the phase where the action of the selected rule is executed.

In general any ADBS must have an underlying DBMS, a facility to define a set of rules, an event detector, a condition evaluator and an action processor and the organization of these components varies over different database systems [Dittrich et al., 1995]. Research and development efforts on active database systems and commercial implementations have focused on integrating active capabilities in the context of relational, object-oriented and object-relational database systems [Campin et al., 1997; Simon and Dittrich, 1995; Bertino et al., 2000; Eisenberg and Melton, 1999; Thome et al., 2005].

In active relational database systems, active rules or triggers monitor changes performed to given tables. Triggers are executed immediately before or after the transactional statement that performs the table change; the condition is typically an SQL predicate, and the action is either a collection of SQL statements or a stored procedure or a program written in a database programming language. Thus active relational database systems have been limited to simple database events like database
modifications such as insert delete and modify operations, events related to transactions such as commit and abort.

However active object-oriented database systems provide richer event type system. It includes any method invocation, temporal events, user defined events and composition of events through event algebra of varying expressiveness. The concept of rule inheritance and overriding provides the facility for reusing and redefining the rules. The integration of active database systems with object database systems provide many benefits than relational database systems that has been recognized [Dayal et al., 1988; Beeri and Milo, 1991; Chakravarthy et al., 1993; Anwar et al., 1993; Dayal et al., 1994; Campin et al., 1997; Bertino et al., 2000].

In the recent past object-relational DBMS effectively support advanced applications than conventional relational and object-oriented database systems [Zhang and Harder, 1997; Feuerlicht, 2010]. Since object-relational DBMS supports object-relational model by combining the benefits of both relational and object-oriented data model. Object-relational database systems support relational technologies (SQL, ODBC, transaction management and concurrency control), user-defined data types, and handling of complex data with object-oriented concepts. Object-relational database systems support active mechanisms in the form of triggers [Eisenberg and Melton, 1999].

1.4 APPLICATIONS OF ACTIVE DATABASE SYSTEMS

Active database systems with ECA rules have been found to provide an elegant mechanism for capturing many real life applications which automatically perform certain operations in response to certain events occurring with certain conditions being satisfied. ADBS are fast growing of research, mainly due to the large number of applications which are benefited from this active dimension. Active database systems provide applications with the possibility of specifying rules that monitor changes in the database that inform applications of interesting changes. Applications of active database systems can range from handling simple integrity constraints to performing complex database changes. In general active database systems are used for three categories of applications [Paton and Diaz, 1999]:

Please purchase PDF Split-Merge on www.verypdf.com to remove this watermark.
• Closed database applications – This type of application involves the use of active functionality within the database and without reference to the external devices or systems. Examples of such applications are automatic statistical analysis (e.g. making market predictions), maintaining high level data consistency (e.g. propagating load requirements between connected elements in an architectural design), and altering users to special conditions (e.g. warning production managers of low inventory levels).

• Open database applications – In this category, a database is used in conjunction with monitoring devices to record and respond to situations outside the database. Open database applications require a more proactive approach in which system actually controls actions to be taken in the real world. Examples of such application include air traffic flow management, communications network control, medical monitoring, and battlefield threat assessment.

• Database system extensions – Active rules are used as a primitive mechanism for supporting extensions to core database functionality. It includes integrity constraints, materialized views, derived data, data modeling constructs, triggers and automatic screen updating in the context of database change.

Many proposals have been made for using ECA rule paradigm in various application contexts such as cooperative applications, supporting of various transaction models, e-business applications, web applications, business processes system, ubiquitous web services, continuous query system, biometric system and healthcare system. The various notable proposed research works related to these application areas are:

Geppert et al., [1995] proposed an agent style system that has been designed for the realization of cooperative process oriented environments (CPEs) implemented in active object oriented database system. CPEs are modeled as broker objects offering specialized services and the brokers communicate by raising special events to indicate the success or failure of their subtasks.
Multiple transaction models can also be supported using active rules [Anwar et al., 1995]. A framework has been designed to support the various transaction models by using active rules. The use of ECA rules to define the semantics of transactions gives the flexibility to choose transaction models at runtime by activating or deactivating the appropriate rule sets. Events and rules can also be reused across rule sets when defining various transaction models such as nesting, sagas and split transactions.

Bonifati et al., [2001] designed active rules explicitly for the management of XML information. The use of active rules for XML offers a natural paradigm for the development of e-services which has been justified.

An ECA rule based approach has been used to describe business rules in heterogeneous e-business environnement [Cilia and Buchmann, 2002]. Active service functionality has been designed to detect the occurrence of events and provide notifications based on publish/subscribe mechanism in heterogeneous environment.

An ECA rule management method for data management in P2P network has been proposed [Kantere and Tsois, 2004]. Facca et al., [2005] proposed an ECA based approach to adaptive web applications.

Bae et al., [2004] proposed an approach to the automatic execution of business processes using ECA rules that are automatically triggered by an active database without user intervention.

Business processes have been analyzed using ECA rules [Bry et al., 2006]. A flexible and modular ECA rule based approach provides an effective approach for easy integration of rules, methods for verification and validation of rule sets, adapt, alter and maintain the requirements change in business processes system. Database trigger has been used to implement business processes system [Wai Yin Mock et al., 2013].

Schiefer et al., [2007] proposed a rule management system which is able to sense and evaluate events in order to respond to changes in a business environment. Sense and respond rules use ECA model for describing constraints of events. ECA rules automatically perform actions in response to events provided stated conditions
hold. The actions of sense and respond rules generate response events, which are used for triggering business activities or evaluating further rules.

Jung et al., [2007] proposed an ECA based coordination approach, to facilitate inter-operation among web services-enabled devices in the ubiquitous computing environment. An ECA rule based approach has been useful for describing the reactive behaviors of the various ubiquitous web services as well as service interactions. The ECA rules are embedded into the devices and then triggered by internal or external events of devices.

Ding et al., [2007] uses active database systems for optimizing the maintenance of continuous queries for trajectories in moving object databases. Trigger specified in the form of ECA rule is utilized to monitor and maintain the correctness of the answers to continuous spatio-temporal queries. More specifically the trigger’s execution is used to improve the performance of reevaluation of a given set of pending queries.

Active database systems have also been used in biometric system [Rabuzin et al., 2007a]. The paradigm of active rules with complex events has been applied to implement a multimodal biometric system.

Flexible specification and execution of clinical guidelines based on the ECA rule paradigm has been proposed [Dube et al., 2002; Dube et al., 2002a]. A framework and architecture has been designed for supporting the management of ECA rule based clinical protocols. A clinical protocol specification has been expressed as a composition of sets of ECA rules for managing patients’ plans.

Wu et al., [2006] proposed an event driven approach with ECA rule paradigm for computerizing clinical guidelines and it has been centered on the concept of an active electronic health care record (EHCR). In this work, a set of active rules represent clinical guidelines and react to changes within the patient medical record.

The active database system with ECA rule paradigm has been useful for managing and easy integration of clinical practical guidelines with electronic health care records [Dube and Wu, 2009]. A generic approach has been designed for monitoring, coordinating and generating suggestions relevant to patient care. The
ECA rules have been useful to monitor patient conditions and coordinate interventions as well as suggesting further appropriate clinical interventions such as ordering appropriate clinical laboratory tests whose outcome has also been further monitored.

An active database system has also contributed to the field of event-based systems [Hinze et al., 2009]. Schaff et al., [2010] proposed an activity service for cloud computing that adopts the semantics of active database system with ECA rule specification and execution model for the cloud. Active functionalities have also been required for complex event processing (CEP) technology to monitor and to react to the CEP system state changes in real time [Wang et al., 2010].

1.5 RESEARCH WORK IN ACTIVE DATABASE SYSTEMS

There has been a lot of research work in the field of active database systems to support reactive behavior in database systems. A number of excellent surveys of active database systems have been proposed [Dayal et al., 1994; Dittrich et al., 1995; Paton and Diaz, 1999; Buchmann, 1999].

Many notable research works have been proposed to support individual aspects of active database systems that include: Data modeling issues [Navathe et al., 1992; Beeri and Milo, 1991], Architectures and monitoring techniques [Chakravarthy, 1995], Semantics of event and rule specification and event detection [Gehani et al., 1992; Gehani et al., 1992a; Anwar et al., 1993; Gatziu and Dittrich, 1993; Gatziu and Dittrich, 1994; Chakravarthy et al., 1994a; Zimmer and Unland, 1999; Zimmer et al., 1997], Rule analysis [Kim and Chakravarthy, 1997], Rule execution model [Kim and Chakravarthy, 1995; Coupaye and Collet, 1995; Baralis and Bianco, 1997; Saygin et al., 1998], Termination of rule execution [Bailey et al., 2004], Rule inheritance and overriding [Bertino et al., 2000], Transaction model and concurrency control [Kappel et al., 1996a; Kangsabanik et al., 2007], Rule scheduling [Ceri et al., 2003; Jin et al., 2007; Rasoolzadegan et al., 2006; Rasoolzadegan and Meybodi, 2010].

Early the research on ADBS has focused on active capabilities within the context of relational database systems. A number of research prototypes have been developed for the integration of active behavior into relational database systems
namely Starburst [Widom and Finkelstein, 1990; Widom, 1992], Ariel [Hanson, 1996] and an extended relational database system POSTGRES [Stonebraker and Kemnitz 1991].

Also several research prototypes have been proposed to incorporate active rules into object-oriented systems include HiPAC [Dayal et al., 1988; McCarthy and Dayal, 1989], Adam [Diaz et al., 1991], Ode [Gehani and Jagadish, 1991], ACOOD [Berndtsson and Lings, 1992; Berndtsson, 1994], NAOS [Collet et al., 1994], SAMOS [Gatziu et al., 1991; Gatziu and Dittrich, 1992; Gatziu and Dittrich, 1993, Gatziu and Dittrich, 1994; Gatziu et al., 1994; Geppert et al., 1995a], Sentinel [Chakravarthy et al., 1994], Chimera [Ceri et al., 1996], REACH [Buchmann et al., 1995], EXACT [Diaz and Jaime, 1997] and TriGS [Kappel and Retschitzegger, 1998].

A model for integrating the execution of triggers with the evaluation of declarative constraints in SQL database systems has been proposed [Cochrane et al., 1996]. The object-relational database standard SQL3 and the various commercial object-relational database products such as Oracle, Ingres, Informix and Sybase support active features in the form of triggers [Eisenberg and Melton, 1999].

1.6 OBJECTIVES OF THE RESEARCH WORK

Active database systems provide an ideal platform for supporting today’s complex applications in order to express an event-driven and constraint-driven system environment. The use of active rules in advanced applications creates new challenges for the development of rule execution model to provide the effective rule processing mechanism at runtime in active database systems. Given the support for effective rule execution model for advanced applications, an emerging need in the development of an active ORDBMS is to provide a rule scheduling mechanism to perform both sequential and concurrent rule execution to speed up the ADBS rule execution process for improving the system performance. To avoid ambiguity in terminology, in this thesis the term rule means as active rules or ECA rules.
The objectives of this research work have been to:

- Study of ECA rules processing and its issues in active database systems.
- Analyse the existing rule scheduling mechanisms for handling the execution of multiple triggered rules raised by an event in active database systems.
- Design and develop a rule scheduling mechanism for ordering rules based on static priority scheme and execution of rules concurrently using the nested transaction model with a locking protocol as the correctness criterion as well as sequentially by immediate and deferred coupling modes of rule execution in active ORDBMS.
- Implement the functionalities of rule scheduler using multi-threaded mechanism and also test the performance evaluation of the developed scheduler with a case study application.

1.7 THESIS ORGANIZATION

The basic terminology of database systems which includes the definition, functional components, various data models, architecture and applications and objectives of database systems are described. The need for advanced databases with object-oriented programming concepts and the specification of ECA rule syntax as well as the rule processing strategy for the execution of reactive behavior has also been described. The importance of active database systems in various application domains has also been looked and the various notable proposed research works in the area of active database systems has been outlined. Finally the objectives of this research work have been specified. The remaining chapters of the thesis are organized as follows:

Chapter 2 analyses the importance of active rule execution semantics in active database systems. The various phases and the issues for developing a rule execution model of an active database system, the abstract architecture of the active database system and the need for active information system are discussed.
This chapter reviews the existing rule scheduling research works in active database systems and identifies the need for developing a rule scheduler for processing trigger rule conflicts in object-relational database systems. The concept of object-relational database systems with active features in SQL3 standard is described. The role of transaction management mechanism and concurrency control protocols for transaction processing in database systems is presented.

Chapter 3 describes the importance of handling trigger rule conflicts in active ORDBMS and specifies the problem statement of this research work. The importance of concurrency with the nested transaction model for modeling concurrent rule execution and the concurrency control protocol as the correctness criterion for the execution of concurrent rules are described. The modeling of rule execution based on active object-relational view of data with the major components such as manager process, scheduler process, evaluation process and transaction process is described.

Chapter 4 introduces an algorithm for designing and developing an rule scheduling component for performing user-defined priority based sequential, concurrent and both concurrent and sequential rule execution and also describes the multi-threaded mechanism implemented by the rule scheduler component for achieving the rule execution.

Chapter 5 presents the design and development of the prototype to test validity of the proposed concepts and algorithm for rule scheduling. The software prototype includes two phases: rule maintenance component for defining and maintaining rules for specifying reactive behavior of the application system and the rule processing component shows the functions of the rule scheduler.

Chapter 6 presents the experimental results. The performance of the rule scheduling algorithm has been examined and the results are compared to show the effectiveness of the developed rule scheduler.

Chapter 7 summarises the research, describes the achievements and limitations and presents recommendations for future work.