UNIT 14 OBJECTS MAPPING WITH DATABASES

Structure **Page No.**

14.0 INTRODUCTION

In the new era, the project development team uses Object Oriented Technology such as Java or C# to build their business application software and store the data in a relational database. The relational databases are widely used database software in organizations. This unit introduces you to how to implement the object model in the relational model. This unit starts with some introductory concepts on general database as well as relational DBMS concepts. As you have already got updated with the knowledge of Object Modeling in the previous blocks of this course. This unit describes how object modeling relates to the three schema architecture. This schema architecture defines how the data is represented or viewed by the user in the database. In addition to these concepts, this unit explains the mapping of Object Classes to Tables. You have learnt about the associations and generalizations in an earlier block of this course. This unit will revisit how to map different associations such as one-to-one, one-to-many, many-to-many, and ternary association as well as generalizations to database tables. At the end of this unit, you will know about interfacing with the database.

14.1 OBJECTIVES

After going through this unit, you should be able to explain:

- basic relational DBMS concepts,
- how to map object classes to database tables,
- extended three schema Architecture for Object Models,
- how the associations are mapped to the tables, and

- how the generalizations are mapped to the tables.

14.2 RELATIONAL DATABASE SCHEMA FOR OBJECT MODELS

As you know, a database is a group of interconnected data files with structures. It is designed to come across the various information necessities of the organization. The database is created by using DBMS software. The designing of the database is the most significant task for database application software. The term relation schema describes the outline of the database. Relation schema expresses the design and structure of the relation (table), such as table name, set of attributes or field names. This section provides you with a small description of the introductory lesson related to the database concepts.

14.2.1 General DBMS Concepts

A DBMS is a software program which is used for creating, retrieving and controlling access to permanent data. There are various motives for using DBMS, such as Integrity, Security, Crash Recovery, Data Distribution and Sharing between users as well as applications.

To effectively manage data using a DBMS, you may know certain **key terms**:

Database Schema

Database Schema is the design of a database. It is the frame of the database that signifies the structure (table names and their fields/columns name). Each column can hold which type of data, constraints on the data to be stored (if any) and the relationships between the tables. You can also say that the database schema is a logical structure of the database and tells us how the data are organized in a database.

Data Constraints

Sometimes you want to place certain restrictions on the type of data that can be inserted in column(s) of a table. This is done by defining one or more constraints on that column(s) while creating the tables. In other words you can say that it is a condition which applicable on the column(s). It ensures database integrity and a few names of the constraints such as Unique, Primary, Default and Check constraint.

Data Dictionary

The database schema along with numerous constraints on the data is stored in a database catalog or dictionary. It is called meta-data. A meta-data is data about the data.

Database Design

The database design is an important and tough task for any business database application. Database design describes the database structure used for creating, storing and maintaining information. The main objective of designing a database is to build physical as well as logical models of the designs for the planned database system. There are two methods to design a database i.e. **attribute driven design** (ADD) and **entity-driven design** (EDD). In attribute driven design, assemble a list of attributes related to the application and normalize the sets of attributes that preserve functional dependencies (FD). In contrast, in the Entity driven design method, define entities which are significant to the application and explain them. Usually, database design has limited entities as compared to the attributes. Entity design is more considerable due to being easily manageable. Object modeling is a kind of entity design.

Three schema architecture

The three schema architecture provides an abstract view of data to users. The three schema architecture is shown in figure-14.1. The database design contains three layers such as external, conceptual and internal schemas. The **external schema** is external view which represents some portion of the entire database. It is a higher order abstraction which is seen by the user. The next level is **conceptual view** which represents data logically. It defines what data is stored really in the database. The third level of this architecture is an **internal schema** which shows the lowest level of abstraction and closest to physical storage. Object modeling is suitable for designing both the external and conceptual schema. For this, you should construct one object model for each external schema as well as another object model for the conceptual schema.

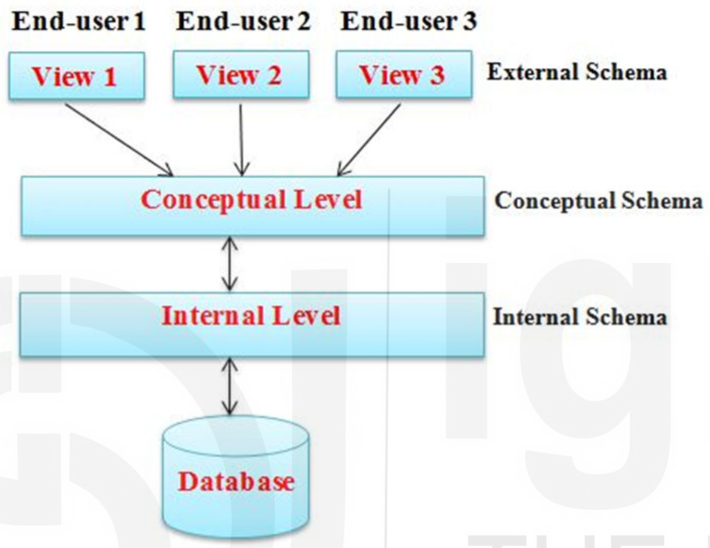

Figure 14.1: The three schema architecture

14.2.2 RDBMS Logical Data Structure

A relational DBMS (RDBMS) is a database management computer program based on a relational data model. Edgar F. Codd invents it. The relational model can store data with one or more relations (tables). A **relation** is a two-dimensional table i.e. row and column. In relations (tables), data is stored in the form of **tuples** (rows) and columns. A **row** is the horizontal portion of the table. One row denotes one **record** of the table. Each row in a table is identified by a **primary key** that does not permit duplicate row. Columns of a table are called as field names or attributes. Column is the vertical portion of the table. The numbers of attributes (columns) in a table is known as **degree** of the relation (table). The number of rows (tuples) in a table is known as **cardinality** of the relation.

14.2.3 Relational DBMS Concepts

In addition to what has been discussed in section 14.2.2, there are more relational DBMS concepts which are discussed below:

RDBMS Operators/Commands

The Structured Query Language (SQL) is a popular language for RDBMS that enables you to create and manipulate a relational database. SQL provides many operators for manipulating tables. SQL has many processing capabilities, such as Data Definition Language (DDL), and Data Manipulation Language (DML). The SQL DDL provides commands for creating and deleting tables, creating indexes and modifying relation schemas, whereas DML provides commands for data retrieval, data deletion and modification of data stored in the database. Using DML, you can also insert new data into the database. For example, the SQL SELECT command is used to retrieve data from one or more tables. The syntax of the SELECT command is like the following:

```
SELECT <column name> [, <column name>, ...]
FROM <table name> [WHERE <condition>]
```

RDBMS Integrity

For maintaining the quality of the information in the database, relational DBMS provides integrity rules. Data integrity means that the data is accurate and consistent in the database. There are two rules of integrity in Codd's model such as entity integrity and referential integrity.

Integrity Rule 1(Entity integrity): It defines that each table have exactly one primary key. A primary key is a set of one or more attributes that can uniquely identify each row in a relation.

Integrity Rule 2 (Referential integrity): It needs that a foreign key must have a matching primary key. Referential integrity constraint ensures that relationships between records in related tables are valid.

A foreign key is a primary key of one table that is embedded in other table. The primary, as well as foreign keys, are shown in the following tables.

Course Table:

CourseCode	CourseName
MCS-218	Data Communication and Computer Networks
MCS-219	Object Oriented Analysis and Design
MCS-220	Web Technologies

Student Table:

RollNo	Name	Age	CourseCode
1	Ram	20	MCS-218
2	Mohan	25	MCS-219
3	Govind	30	MCS-220
4	Narayan	28	MCS-218

The above example shows Primary and foreign keys. In this example, there are two tables: Student and Course. ***CourseCode*** is the primary key of the Course table, and ***RollNo*** is the primary key of the Student table. ***CourseCode*** is a foreign key in the Student table. It would not be acceptable to change Mohan CourseCode to MCS-221 because MCS-221 is not defined in the Course table. The foreign key is used to create a relationship between two tables.

Normal Forms

Database design is difficult task. Normalization theory is a valuable support in the database design process. Normalization is a data analysis process. While designing the databases, there are unnecessary repetitions of data. The normalization is used to reduce the repetition of data. The normalisation process is related to transforming the conceptual schema (logical data structure) into a computer-representable form. It is used to avoid logical inconsistencies from table update operations. There are various levels of normal form, such as 1NF to 5NF and BCNF.

A table is in **first normal form** (1NF) if and only if all underlying domains of the table contain atomic values. A table is in **second normal form** (2NF) when it satisfies the first normal form and every non-key attribute fully depends on the primary key. A table is in the **third normal form** (3NF) when it satisfies the second normal form, and every non-key attribute is non-transitively dependent on the primary key.

View

View is a virtual table whose contents are derived from one or more tables depending upon the conditions. A view does not physically store data in the database. It is used just like any other table. A view is a query like the **SELECT FROM …clause** that works on the physical table. The purpose of using view is for data security; it permits a user to use the data through the view only, and hidden data is not available to the user. DML operations such as INSERT, DELETE and UPDATE can be applied on the views.

☞ Check Your Progress - 1

1. What are the different integrity constraints in RDBMS?

   ----

2. What is database design in a relational database management system? Explain two methods of database design.

   ----

3. What are the three levels of schema architecture?

14.3 OBJECT CLASSES TO DATABASE TABLES

In the previous section, you have gone through the general DBMS and relational DBMS concepts that are most useful during the database design process. This section provides you with how to transform an object model into a relational database. This section focuses on extended three schema architecture for the object models, use of object IDs, and mapping object classes to tables.

14.4 EXTENDED THREE SCHEMA ARCHITECTURE FOR OBJECT MODELS

This section moves towards the concepts of relational database design as RDBMS technology, which is today's more suitable technology for business applications. It describes how object modeling relates to the three schema architecture.

The three schema architecture is used for designing the structure of database systems. This architecture is used to separate the user applications and physical databases. The three schema architecture covers three levels: External, Conceptual and Internal.

External Schema: It designates high-level data model and closest to the user.

Conceptual Schema: It defines how the database looks in front of users.

Internal Schema: It defines the physical storage structure of the database. The following figure-14.2 illustrates how object modeling connects to the three schema architecture.

In the first step toward this design, you should develop object models for the external and conceptual schema, and after this, you should transform each object model into table model. Views and interface programs which are connected with external tables convert to conceptual tables. Conceptual tables transform these into the internal schema.

The **object model** defines the static structure of the objects in a system and their relationships. It emphasizes on logical data structures. Each object model contains classes, associations, generalizations and so on. The goal of constructing an object model is to grasp those ideas from the real world which are significant to an application.

You must use any mapping procedure when transforming the object model to the table model. There are many mapping alternatives, and they comprise mapping rules. For example, if you want to map association(s) to the table(s) then there are two alternatives for mapping association. Similarly, when you transform the generalizations to relational tables, you can use different strategies. You will study mapping associations to tables in section 14.5 and mapping generalizations to tables in section 14.6 of this unit. During transformation, you can also insert some points/keys into the table which are missing in the object model, such as the primary key and candidate keys for each table and foreign key wherever require, also put some constraints on the attributes whether each attribute can be **null**, or **not null**. Those attributes are defined as candidate keys and generally should not be null. You must assign a domain to each attribute and list groups of attributes that are frequently accessed.

The **internal schema** of the three schema architecture contains SQL commands which are used to create the tables, attributes, assign constraints, and keys. The latest database software version comes with a wizard facility that automatically creates tables and indexes. While designing a relational table, you should put restrictions on the length and legal characters for names. You can see the conversion process of the external table model to the conceptual table model to the internal schema in the rounded part of the figure 14.2.

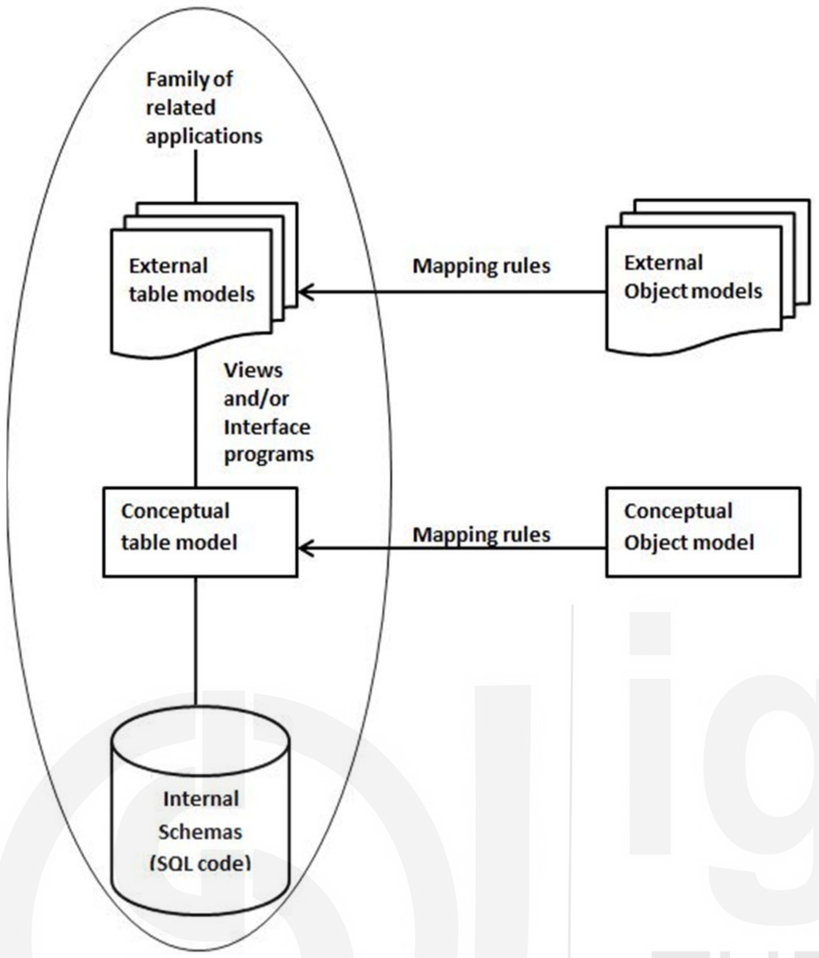

Figure 14.2: Object modeling and the three schema architecture for RDBMS

14.4.1 The use of Object IDs

An object identifier (OID) is a unique name for any type of object or entity. It is a mechanism to identify an entity. When you derive a table from the class, each table has an ID for the primary key. When you derive tables from association(s), then one or more object IDs collectively form the primary key. An ID is equal to the database concept. In a database, an object ID is a set of integers which uniquely identify each row (or record) in a table.

There are some advantages and disadvantages of using IDs. The benefit is that it does not make any changes to the data. The stability of IDs is mainly important for associations as they refer to objects. Any changes in the name of IDs, will update many associations. So, IDs give an unvarying mechanism for referencing objects. The disadvantage is that RDBMS does not support the IDs. Actually, there is no concept for generating the IDs in RDBMS because it emphasizes the data located and manipulated based on attribute values.

You can define IDs as attributes for your tables and accept a mechanism for handling them as per the relational database you have adopted. You cannot use IDs in applications where you directly access the database.

14.4.2 Mapping Object Classes to Tables

When you map class objects to the relational model, you shall start with the attributes of a class. These attributes will translate to one or more columns in a table of relational database because some of the attributes are used for temporary calculations. For example, an employee class may have a *grossPay* attribute that is used for some calculation. It means that all attributes will not be transformed to tables or in persistent storage. While mapping objects to tables, objects in a class may be segregated horizontally and/or vertically for ease of access. For example, if you have a class with many attributes but few are often used, then you can segregate frequently accessed objects in a table and the remaining objects in another table using horizontal partitioning. In a similar manner, if a class has attributes with different access forms, then partition the objects vertically. Both types of partitioning are useful for improving efficiency. The following figure-14.3 illustrates the vertical and horizontal partitioning of the tables.

Vertical Partition

StudentID	Name
1	Ram
2	Shyam
3	Govind

StudentID	Address
1	R.K. Puram, Delhi
2	Papum Para, Itanagar
3	Jalukbari, Gawahati

Horizontal Partition

StudentID	Name	Address
1	Ram	R.K. Puram, Delhi
2	Shyam	Papum Para, Itanagar

StudentID	Name	Address
3	Govind	Jalukbari, Gawahati

Figure 14.3: Vertical and horizontal partition of tables

You can see in the figure14.3 where an object class is converted into a table. Class 'Student' has attributes such as Name and Address. You can list these attributes in the table model and insert the implicit object ID into it. During making a table model, you can mention more details. You can specify StudentID, which cannot be null because it is a candidate key. The name attribute cannot be null, a name must be entered for every student. You cannot define name of student as candidate key because two students may have the same name, but the address attribute may be null. You can also assign a primary key to each table. The Structured Query Language is used to create a Student table. The following figure 14.4 demonstrates you how to map object class to a table.

Object Model $\longrightarrow$

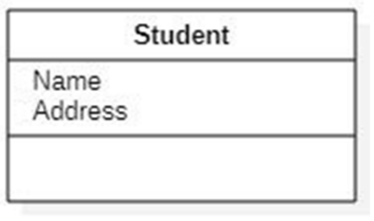

Table Model ⟶

Attribute Name	Null	Domain
StudentID	N	ID
Name	N	Name
Address	Y	address

Figure 14.4: Mapping Object Class to Table

Creating Student table:

SQL Code ⟶

```
CREATE TABLE Student (
  StudentID    ID            not null,
  Name         varchar(30)  not null,
  Address      varchar(30),
  PRIMARY KEY (StudentID)    );
```

☞ **Check Your Progress - 2**

1. What is an object IDs? What is the advantage of using object IDs?

   ----

2. Explain how the object classes are mapped to database tables.

   ----

3. Explain an object model with a simple example model and transform it to the relational table.

14.5 MAPPING ASSOCIATIONS TO TABLES

As you already know, association is a relationship between two separate classes through their objects. Associations are generally denoted as continuous lines between the participants' classes in the relationship. An association may be one-to-one, one-to-many, many-to-many or a ternary association. The association mapping to tables depends on the association type and multiplicity of the association. This section will discuss about the mapping of different types of associations into database tables.

14.5.1 Mapping Binary Associations to Tables

This section describes the mapping of one-to-one association and one-to-many association to the database tables.

In **one-to-one Association**, one occurrence of a class is related to exactly one occurrence of the associated class. For example, the following class diagram shows one-to-one association between Country and City. In the figure 14.5, relationship type *has_capital* shows how each Country holds exactly one City.

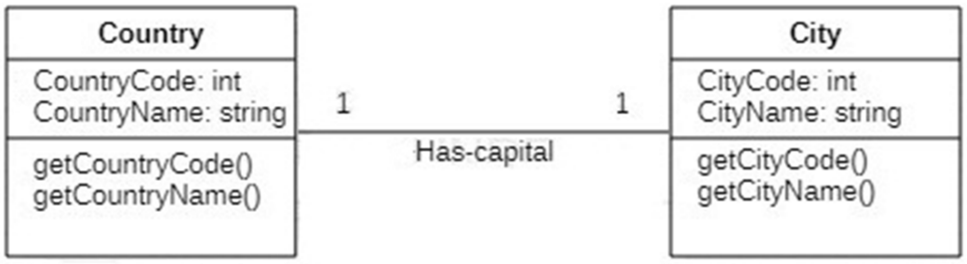

Figure 14.5: Object model for one-to-one Association

While mapping such a relationship to the tables, the primary key of one table is included in another table, which means that you can handle one-to-one association by using Foreign Key. Like in the above example, *CountryCode* is a primary key of the Country table. *CityCode* is a primary key of the City table. The foreign key that is primary key of the Country table is inserted into the City table.

The figure14.6 is shown as a SQL code for the above one-to-one association example. While mapping to tables, first, you create a table model or list the attributes as you have learnt in section 3.3.3 as mapping object classes to tables; after that, you may create SQL code for the table. The following SQL code (figure 14.6) is generated for SQL Server database.

```
Create table country
( countrycode int not null,
countryName varchar(30)not null,
Primary key(countrycode));

Create table city
( CityCode int not null,
CityName varchar(30)not null,
countrycode int not null,
Primary key(CityCode),
Foreign key (countrycode) References country );
```

Figure 14.6: SQL code for mapping one-to-one association to tables

You may also merge the above example (one-to-one association) into a table and store both objects and association all in one table. This approach is useful for improving performance and reducing database storage. But you should be careful while using this approach as it may introduce redundancy.

In **one-to-many association**, one occurrence of a class is related to many occurrence of the associated class. For example, the class diagram for one-to-many association between University and Employee classes is shown in the figure 14.7.

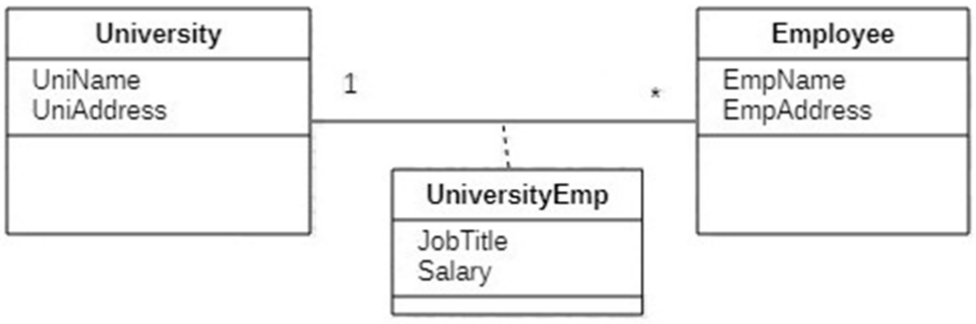

Figure 14.7: Object model for one-to-many Association

When you're mapping an one-to-many association to tables, you can have two options; you may use a **foreign key** in the table for 'many' class or create a **distinct table** for each association class. Using the first option, you can create a table model from the object model, as shown in figure 14.7.

The following figure 14.8 is shown the table model for the object model, which is defined in figure 14.7.

University Table Model

Attribute Name	Null	Domain
UniID	N	ID
UniName	N	Name
UniAddress	Y	address

Employee Table Model

Attribute Name	Null	Domain
EmpID	N	ID
EmpName	N	Name
EmpAddress	Y	Address
UniID	N	ID
Job_title	Y	Title
Salary	Y	Salary

Figure 14.8: Table model for one-to-many Association(using foreign key)

You can see the figure-14.8, there are two tables as University and Employee tables. The *UniID* is the primary key of the University table, and EmpID is the primary key of the Employee table. The foreign key that is the primary key of the University table is inserted into the Employee table. You can create relational tables using the table model defined in figure-14.8. The SQL code generation of this example is not shown here, and you may create this easily with the help of other examples.

While mapping to tables, first, you can create a table model and then write SQL code for association. Now using the second option, you may create a distinct table for one-to-many association. Here, take the above example of the object model, which is shown in figure 14.7 for transforming to a table model.

University Table Model

Attribute Name	Null	Domain
UniID	N	ID
UniName	N	Name
UniAddress	Y	address

Employee Table Model

Attribute Name	Null	Domain
EmpID	N	ID
EmpName	N	Name
EmpAddress	Y	Address

UniversityEmp

Attribute Name	Null	Domain
UniID	N	ID
EmpId	N	ID
Job_title	Y	Title
Salary	Y	Salary

Figure 14.9: Table model for one-to-many association – using distinct table

In figure 14.9, you can see the table model for the object model defined in figure14.7. This table model consists of three tables: University, Employee and UniversityEmp table. Here, all the classes in the association are mapped to a distinct table. The *UniID* is the primary key of the University table, and EmpID is the primary key of the Employee table. The attributes *UniID* and EmpID join together to form the only candidate key for the *UniversityEmp* table. Now, you can create relational tables like the following SQL code in figure 14.10.

```
Create table University
( UniID varchar(10) not null,
  UniName varchar(30) not null,
UniAddress varchar(30),
Primary key(UniID));

Create table Employee
( EmpID varchar(10) not null,
  EmpName varchar(30) not null,
EmpAddress varchar(30),
Primary key(EmpID));

Create table UniversityEmp
( UniID varchar(10) not null,
  EmpID varchar(10) not null,
  Job_title varchar(30) ,
  Salary int
Primary key (UniID,EmpID),
Foreign key (UniID) References University,
Foreign key (EmpID) References Employee);
```

Figure 14.10: SQL code for the one-to-many Association (using distinct table)

14.5.2 Mapping Many-to-Many Association to Tables

In many-to-many association, M occurrence of a class is in a relationship with N occurrence of the associated class, as shown in the following figure 14.11.

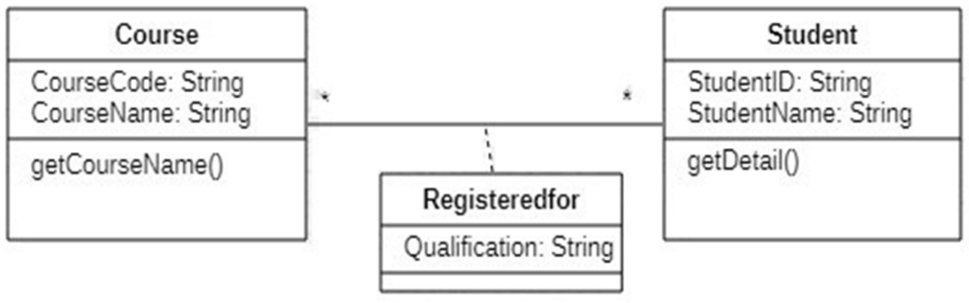

Figure 14.11: Object model for many-to-many Association

While mapping such relationship to the tables, you require an associative table that references the primary keys of the associated records. The name of this associative table is usually either the combination of the names of the tables that it associates or the name of the association that it implements. If you need to store any extra information for this relationship, you can add columns to the associative table. Let's see an example of such a relationship in the class diagram (figure-14.11). This example contains two classes, Course and Student, and one associated class. The relationship type *Registeredfor* shows how each course has M student registered for it whereas depicted other, each student Registered for N different courses. Now you can create a table model by taking the help of the above examples described in this unit.

A many-to-many association always maps to a distinct table. The primary keys for both related classes and any link attributes become attributes of the association table. The attributes (classes i.e. Course and Student) *CourseCode* and *StudentID* join together to form the only candidate key for the *Registeredfor* table.

The following is a SQL code for the above many-to-many association mapping to tables.

```
Create table Course
( CourseCode varchar(10) not null,
  CourseName varchar(30) not null,
Primary key(CourseCode));

Create table Student
( StudentID varchar(10) not null,
  StudentName varchar(30) not null,
Primary key(StudentID));

Create table Registeredfor
( CourseCode varchar(10) not null,
  StudentID varchar(10) not null,
  Qualification varchar(30) not null,
Primary key(CourseCode,StudentID),
Foreign key (CourseCode) References Course,
Foreign key (StudentID) References Student );
```

Figure 14.12: SQL code for mapping many-to-many association to tables

You have seen the above example of many-to-many association. While mapping to the tables primary key of both the tables i.e. Course and Student are included in another new table i.e. *Registeredfor* as foreign keys along with the attributes of relationship. These keys combine to form the new table's collective primary key.

14.5.3 Mapping Ternary Associations to Tables

When the three classes are involved in a relationship, it is called ternary associations. When you are mapping these associations to the tables, all classes shall be mapped onto the distinct tables, and their relationship shall be taken by creating another table with a primary key which must be combined primary keys of all three classes. It would also include the relationship attributes such as start and end date. Let's see the following Ternary association diagram and then map these associations to tables.

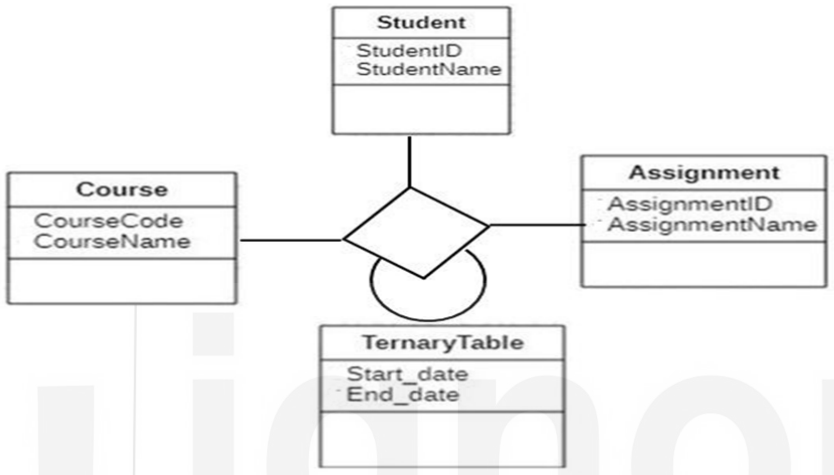

Figure 14.12: Object model for Ternary Association

The following are the table models for the above Ternary association.

Course Table Model

Attribute Name	Null	Domain
CourseCode	N	ID
CourseName	N	Name

Student Table Model

Attribute Name	Null	Domain
StudentID	N	ID
StudentName	N	Name

Assignment Table Model

Attribute Name	Null	Domain
AssignmentID	N	ID
AssignmentName	N	Name

TernaryTable Table Model

Attribute Name	Null	Domain
CourseCode	N	ID
StudentID	N	ID
AssignmentID	N	ID
Start_date	Y	Start date
End_date	Y	End date

Figure-14.13: Table models for Ternary Association.

The following is a SQL code for the above ternary association mapping to tables. After mapping, you have four RDBMS tables, as shown in figure-14.13.

```
Create table Course
( CourseCode varchar(10) not null,
  CourseName varchar(30) not null,
Primary key(CourseCode));

Create table Student
( StudentID varchar(10) not null,
  StudentName varchar(30) not null,
Primary key(StudentID));

Create table Assignment
( AssignmentID varchar(10) not null,
  AssignmentName varchar(30) not null,
Primary key(AssignmentID));

Create table TernaryTable
( CourseCode varchar(10) not null,
  StudentID varchar(10) not null,
  AssignmentID varchar(10) not null,
  Start_date Datetime,
  End_date Datetime,
Primary key(CourseCode,StudentID,AssignmentID),
Foreign key (CourseCode) References Course,
Foreign key (StudentID) References Student,
Foreign key (AssignmentID) References Assignment);
```

Figure 14.15: SQL code for mapping ternary association to tables

Kindly note that the keyword VARCHAR is used in the above examples in this unit. It is a datatype in SQL databases which indicates the values of the column are variable-length character string and the length of the string are not more than the number of characters specified in parentheses.

14.6 MAPPING GENERALIZATIONS TO TABLES

As you already know, a generalization is a superclass - subclass type of relationship. This section describes different strategies which are used to map generalization relationships to tables.

The Employee superclass is a generalization of the entity subsets such as salaried or hourly employees. For example, if you are modeling an organization's database and want to store information about their employees, either salaried employees or hourly employees. In this case, you can create an Employee set that would be called as superclass and salaried/hourly employee subset is called subclasses.

The following figure-14.14 is shown above mention generalization relationship and is used to demonstrate different tactics.

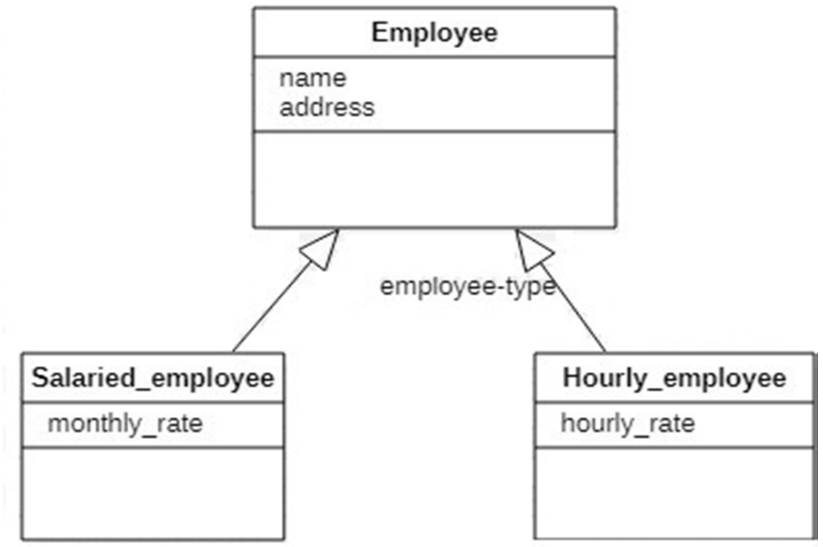

Figure 14.14: Object Model for Generalization

These are different tactics or strategies used for mapping generalizations relationship to tables.

Strategy 1 - Map the superclass and each subclass to a table

According to this strategy, all the entities in the relationship are mapped to individual tables.

To illustrate this strategy, you may map figure 14.14 as shown below in the box. You can create a table for the superclass entity as EMPLOYEE, which includes all the attributes of this employee entity in the table and underline the primary key attribute. You can also create a separate table for each subclass entity. Same as a superclass entity, include the attributes of the respective subclasses in the subclass tables. Take the primary key from the superclass table and insert into the subclass tables as a primary key and underline it.

Employee(<u>empID</u>, name, address, phoneNo)

Salaried_employee (<u>empID</u>, monthly_rate)

Hourly_employee(<u>empID</u>, hourly_rate)

Strategy 2 - Map each subclass to a table

Using this method, you can only map subclasses to tables. You do not map the superclass to a table. The attributes in the superclass are duplicated in all subclasses.

To illustrate this strategy, you will map figure-14.14 as shown below in the box. Here, you can create an isolated table equivalent to each subclass entity with the involvement of attributes of the corresponding subclasses in the corresponding subclass tables. You can also add the superclass entity's primary key and other attributes in all the subclass tables and underline the primary key fetched from the superclass entity to the subclass table. This strategy is most chosen when inheritance is disjoint and complete, e.g. every employee is either salaried or hourly or none.

Salaried_employee (<u>empID</u>, name, address, phoneNo, monthly_rate)

Hourly_employee(<u>empID</u>, name, address, phoneNo, hourly_rate)

Strategy 3 - Map the superclass to a single table

In this case, only the superclass is mapped to a table and does not map the subclasses to tables. The attributes in the subclasses are brought to the superclass.

To illustrate this strategy, you may map figure14.14 as shown below in the box. Using this strategy, you can create a single table containing the superclass entity and its attributes, along with the subclasses and their attributes. This strategy will introduce null values. When you insert a salaried employee record in the table, the *hourly_rate* column value will be null. Similarly, when you insert an hourly employee record in the table, the *monthly_rate* value will be null. This strategy may be useful when there are only two or three subclasses consisting few attributes.

Employee(<u>empID</u>, name, address, phoneNo, monthly_rate, hourly_rate)

14.7 INTERFACING TO DATABASES

A DBMS is a software program which is used for creating, retrieving and controlling access to permanent data. As the focus of this unit is on object mapping to databases. This section is to discuss the relationship between object orientation and relational databases and some interfacing points with the database.

Object-oriented technology is unanimously recognized methodology for developing business applications since it provides more suitable methods and services for modeling objects from the real world. Object orientation and relational databases are not compatible. Both represent two different views: in an Object-Oriented system, concepts are surrounded by the discrete objects which combine both data structure and behaviour in a single entity; in an RDBMS, all about the data or data-centric. The Object-Oriented model is suited with applications with state-specific behaviour, and the data is a secondary point. The RDBMS model is well-matched to reporting applications in which the relationships are dynamic and is still used to preserve the persistence of enterprise data.

The discussion point here is that a lot of data is stored in relational databases. If Object-Oriented applications are used to store data in RDBMS, then it is more able to read and write data in the relational database. Besides, Object-Oriented applications can share data with non-object-oriented applications by using RDBMS as the sharing mechanism.

Connecting these two standards is not an easy task and requires organized mapping from one to another. This mapping from object model to the relational database needs some rules and strategies, which we have already discussed in the above sections of this unit.

Once you have transformed object oriented data model to relational database, you shall write the interfacing code responsible for the application's functionality. This code must contain commands capable of reading and writing data from the database and interpreting it wherever requisite in terms of the model used by the program. To achieve this, you need interfacing software such as JDBC API, which supports your programming environment and allows programmers to shorten the details of individual databases and allows an application to work with a variety of databases or data sources. The JDBC API is a Java API which can be used to access any kind of tabular data, specifically data stored in a relational database. It allows Java programmers to write programs which interface with relational databases. It has three simple steps to connect a database, such as connecting to a data source, like a database; sending queries and update statements to the database and retrieving and processing the results received from the database in answer to the query.

☞ **Check Your Progress - 3**

1. Explain the mapping of one-to-many associations to the database tables.

   ----

2. Explain how the ternary associations are mapped to the database tables.

   ----

3. Explain how the generalizations relationships are mapped to the database tables.

14.8 SUMMARY

In this unit, you have learned about the basic and relational concepts of RDBMS as well as extended 'three schema architecture' for object model, which is the base of database design. The three schema architecture provides an abstract view of data to users. This Unit has typically about the object mapping to database. You have learnt about mapping the object classes to tables. Each class transforms to one or more tables. This Unit described you the conversion process of different association to tables. These different associations are one-to-one association, one-to-many association, many-to-many association as well as ternary association. While transforming classes to tables in one-to-one association, you can use foreign key concepts. In one-to-many association, there are two options for transformation to tables such as distinct table and foreign key whereas in many-to-many association, all the classes are mapped to a distinct table.

You have also studied the mapping generalizations to tables. In this type of mapping, you have learnt different tactics or strategies. Towards this unit's end, the database interfacing has been described.

14.9 SOLUTIONS/ANSWERS TO CHECK YOUR PROGRESS

☞ **Check Your Progress - 1**

1. There are two rules of integrity in the relational model such as entity integrity and referential integrity. Entity integrity defines that each table has exactly one primary key. A primary key is a set of one or more attributes that can uniquely identify each row in a relation. Referential integrity needs a foreign key.

2. Database design is a group of methods that help the database designer design, develop, implement, and maintain the enterprise data management systems. The main objective of designing a database is to build physical as well as logical models of designs for the planned database system. Using this approach, the database designer can decide on data elements correlation and what data must be stored in the database.

   There are two methods to design a database i.e. attribute driven design (ADD) and entity-driven design (EDD). In Attribute driven design, assemble a list of attributes related to the application and normalize the sets of attributes that preserve functional dependencies (FD), while in the Entity driven design method, define entities which are significant to the application and explain them.

3. The three schema architecture provides an abstract view of data to users. The database design contains three layers such as external, conceptual and internal schemas. The **external schema** is an external view which represents some portion of the entire database. It is a higher-order abstraction which the user sees. The next level is **conceptual schema** which represents data logically. Basically, it defines what data is stored really in the database. The third level of this architecture is **internal schema** which shows the lowest level of abstraction and closest to physical storage.

☞ **Check Your Progress - 2**

1. An object identifier (OID) is a unique name for any type of object or entity. It is a mechanism to identify an entity. When you derive a table from the class, each table has an ID for the primary key. When you derived tables from association, one or more object IDs collectively form the primary key. An ID is equal to the database concept. In a database, an object ID is a set of integers uniquely identifying each row (or record) in a table.

   There are some advantages and disadvantages of using IDs. The benefit is that it does not make any changes to the data. The stability of IDs is mainly important for associations as they refer to objects. Any changes in the name of IDs, this will update many associations. So, IDs give an unvarying mechanism for referencing objects.

2. Each object class transforms into one or more tables in the database. The objects in a class may be partitioned horizontally and/or vertically. When you map class objects to relational model, you shall start with attributes of a class. You can list these attributes in the table model and insert the implicit object ID into it. While making a table model, you can add some constraints details like not null as per your need. You can also assign a primary key to each table. The Structured Query Language is used to create code for the table.

3. The object model defines the static structure of the objects in a system and their relationship. It emphasizes on logical data structures. Each object model contains classes, associations, generalizations and so on. The goal of constructing an object model is to grasp those ideas from the real world which are significant to an application. The object model graphically represents the object diagrams which contain the object. Each class defines some attributes'

values carried by object instances and operations performed by each object. For example and mapping object model to tables, please refer to section 3.3.3.

☞ **Check Your Progress - 3**

1. In one-to-many association, there are two options for transformation to tables: distinct table and buried foreign key.
2. When the three classes are involved in a relationship, it is called ternary associations. When you are mapping these associations to the tables, all classes are mapped onto the distinct tables. Their relationship shall be taken by creating another table with a primary key that must be combined with all three classes.
3. Whenever you map generalizations to tables, there are different tactics or strategies. As per the strategy-1, you can map the superclass and each subclass to a relational table. According to this strategy, all the entities in the relationship are mapped to individual tables. The second way to map generalizations to tables is that each subclass mapped to a table. Using this method, you can only map subclasses to tables. You do not map the superclass to a table. The attributes in the superclass are duplicated in all subclasses. You can map the superclass to a single table using the third strategy. In this case, only the superclass is mapped to a table and does not map the subclasses to tables. The attributes in the subclasses are brought to the superclass.

14.10 REFERENCES/FURTHER READING

- James Rambaugh, Michael Blaha, William Premerlam, Frederick Eddy, William Corensen, "Object Oriented Modelling and Design" PHI, New Delhi, 2004.
- Bipin C. Desai, "An Introduction to Database Systems", Galgotia Publications, 2010.
- http://agiledata.org/essays/mappingObjects.html
- https://web.csulb.edu/colleges/coe/cecs/dbdesign/dbdesign.php?page=class.php
  - https://docs.oracle.com/javase/tutorial/jdbc/index.html