UNIT 2 OBJECT MAPPING WITH DATABASE

Structure **Page Nos.**

2.0 INTRODUCTION

Object oriented designs are **efficient, coherent** and **less prone to the update problems** that are present in many other database design techniques presently. Relational databases are mostly widely used and employed in the organisations.
It is not wrong to say that relational DBMS are increasing their advantage in functionality and flexibility and are also improving their performance. In spite of having many advantages in comparison to other databases, object oriented DBMS have not able to reach the commercial mainstream. Even so, object oriented DBMS is in the development stage. Many DBMS companies have started to support the concept of object oriented Design in their new products.

In this unit, we will discuss data base concept related object orientation which includes discussion on relational database schema for object models, object classes and database tables, and mapping of associations and generalizations to tables.

2.1 OBJECTIVES

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

- explain object modes for relational database schema;
- explain how the object classes are mapped to the database tables;
- explain how the associations are mapped to the tables;
- explain how the generalizations are mapped to the tables;
- describe the interfacing to database, and
- describe the object mapping with databases.

2.2 RELATIONAL DATABASE SCHEMA FOR OBJECT MODES

You have already studied the basics of DBMS in MCS-023. Let us very quickly refresh those concepts.

2.2.1 General DBMS Concepts

The **database management system** (DBMS) is a software system that enables users to define, create, maintain, and control access to the database. Physically, this database is stored in one, or more files.

There are many reasons for using a DBMS, some of which are:

1. Control of data redundancy.
2. Data consistency: by eliminating or controlling redundancy, the inconsistency in the data of database is reduced.
3. Sharing between users: multiple users can access the database at the same time.
4. Sharing between applications: multiple application programs can read and write data to the same database.
5. Improved data integrity: database integrity refers to the validity and consistency of stored data. A DBMS can control the quality of its data over and above facilities that may be provided by application programs.
6. Improved security: database security is the protection of the database from unauthorized users.
7. Improved backup and recovery services: the database is protected from hardware crashes, and disk media failures.
8. Extensibility: data may be added to the database without changing the existing programs.
9. Data distribution: the database may be partitioned across various sites, organizations, and hardware platforms.
10. Economy of scale: combining all the organization's operational data into one database, and creating a set of applications that work on this one source of data can result in cost savings.

The lifecycle of the database applications includes the following steps:

1. Requirement analysis of the application.
2. Design of the application.
3. Devise a specific architecture for coupling the application to a database.
4. Selection of the desired DBMS depending upon the feasibility analysis.
5. Design of the database.
6. Write the application programs which provides a user interface, validate data, and perform computations.
7. Populate the database with the required data.
8. Execute the application and then query, insert, and update the database as needed.

There are two approaches to database design, **attribute driven** and **entity driven**.

- **Attribute driven**: Compile a list of attributes relevant to the application and normalize the groups of attributes that preserve functional dependencies.
- **Entity driven**: Discover entities that are meaningful to the application and describe them.

In a typical database design, there are few entities as compared to the attributes. Entity design is much more easily to manage. Object modeling is a form of entity design.

The three schema architecture on DBMS that was originally proposed by ANSI/SPARC committee is shown in *Figure1*. The database design consists of three layers, which are:

- External schema,
- Conceptual schema, and
- Internal schema.

External schema: an external view is an abstract representation of some portion of the total database. An external schema is a definition of an external view. Each external schema is a database design from the perspective of a single application. The external schema isolates applications from most changes in the conceptual schema.

- **Conceptual schema**: The conceptual view is a logical representation of the database in its entirety. The conceptual schema is a definition of that conceptual view. It integrates related applications and hides the details of the implementation of the underlying DBMS.
- **Internal schema**: The internal schema is the database as it is physically stored. The internal schema is the definition of that internal view. The internal schema level consists of actual DBMS code required for the implementation of the conceptual schema.

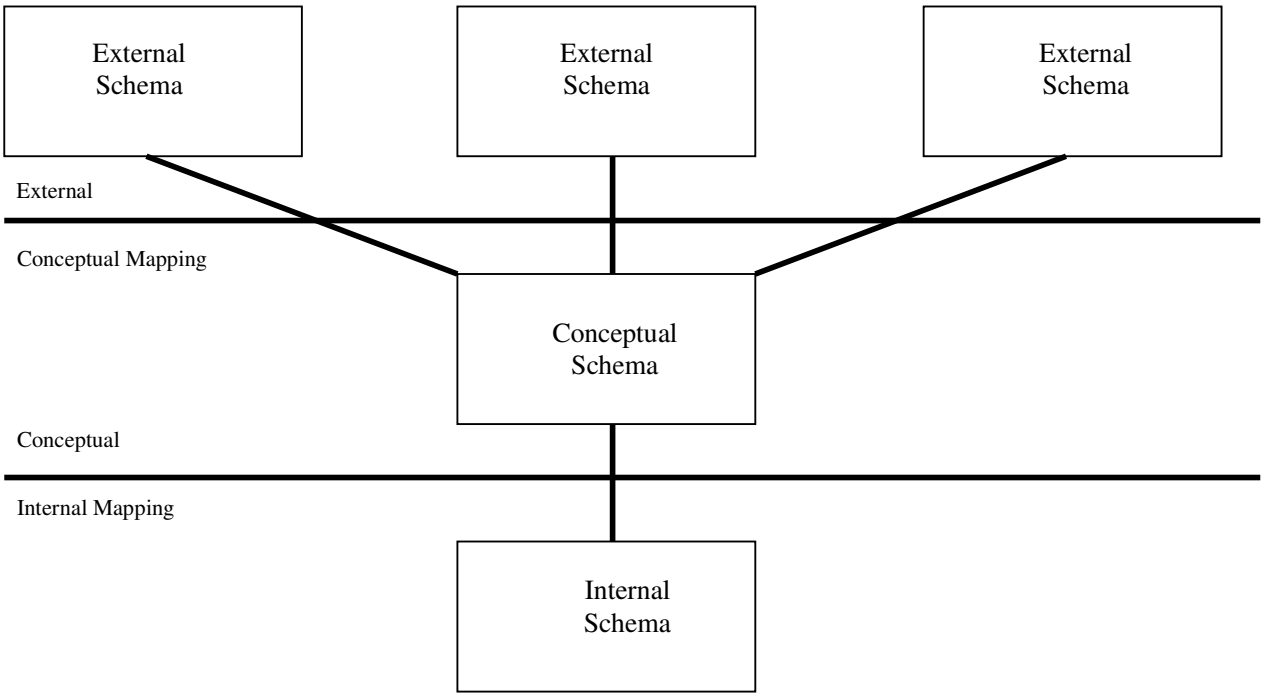


Figure 1: ANSI-SPARC Three level Architecture

Object modeling is useful for designing both the **external** and **conceptual** schema. For this, one should construct **one object model for each external schema, and another object model for the conceptual schema**.

2.2.2 Relational DBMS Concepts

RDBMS, as defined by Codd, has three major parts:

1. Data that is presented as tables
2. Operators for manipulating tables
3. Integrity rules on tables.

Now, let us discuss the concepts covering these three parts.

2.2.3 RDBMS Logical Data Structure

A relational database logically appears **simply as a collection of tables**. Tables have a **specific number** of columns and an **arbitrary number** of rows. The columns of tables are called attributes and directly correspond to attributes in object models. The rows are called **tuples** and correspond to **object instances and links**. A value is stored at each table row and column intersection.

Each value in a table **must belong to the domain of its attribute** or **can be null**. Null means that an attribute value is unknown or indeterminate.

The RDBMS decides if tuning is required for processing a query, and if so, automatically uses it. The RDBMS automatically updates tuning information

whenever the corresponding tables are updated. Indexing, hashing, and sorting are common tuning techniques.

RDBMS Operators

SQL is the most popular language for RDBMS. It follows ANSI as well as ISO standards. SQL provides operators for manipulating tables. The SQL 'SELECT' statement queries tables. The syntax of the select command looks something like:
SELECT attribute-list {as Alias}
FROM table-1, table-2, . . .
WHERE predicate-is-true
Logically Table-1, Table-2, and any others are combined into one temporary table. The attribute list specifies which columns should be retained in the temporary table. The predicate expression specifies which rows should be retained.
SQL can be classified into two categories:
SQL DDL commands: for creating the tables.
SQL DML commands : for inserting,updating, deletion operations.
SQL commands create Tables, insert rows into tables, delete rows from tables, and perform other functions.

RDBMS Integrity

A primary key is a combination of one, or more attributes whose unique value locates each row in a table. The primary key is always a candidate key. In the table shown below, **emp_no** is the primary key of the employee table; **dept_no** is the primary key of the department table.

Table 1: Employee Table

emp_no	Emp_name	Address	dept_no
1	Ram Kumar	20/780, Hauz Khas, New Delhi	02
2	Shyam Kumar	2/462, Sector 62,Noida	01

Table 2: Department Table

dept_no	Department Name	Address
01	Sales	2/2 Sector 2, Noida
02	Finance	1/2 Hauz Khas, New Delhi

Referential integrity requires that the RDBMS keep each foreign key consistent with its corresponding primary key. A foreign key is a primary key of one table that is references in another table. Referential integrity is useful when we are mapping object models to tables.

Normal Forms

Normal forms are rules developed to avoid logical inconsistencies from table update /insert operations. Each normal form avoids a form of redundancy in table organization that could yield ambiguity results if one table is updated independently of other tables. There are many levels of normal form, i.e., 1NF to 5NF and BCNF. Each higher level of normal form adds a constraint to the normal below it. As the database designer satisfies higher normal form, tables tend to become fragmented, higher normal forms improve database consistency, but at the cost of added navigation and slower query execution.

A table is in first normal form when each attribute will have **atomic value**. A table is in second normal form when **it satisfies first normal form and the partial dependency(ies) in the table are removed**.

A table is in third normal form **when it satisfies second normal form and transitive dependency is removed if any.**

Views

View is a virtual table that is dynamically computed as needed. A **view does not physically exist in the database** but is derived from one or more underlying tables. In

theory, views are the **means for deriving external schema from conceptual schema** for the ANSI three schema architecture. Commercial, RDBMS **usually support reading** through views, but **rarely support writing** through views.

Student Table

Table 3: Different views from Student Table

Name_id	Course_No	Grade	Phoneno	Major_no	Major-elective	prof
N1	C1	A	232456	M1	M1E1	SANJAY
N2	C2	B	256665	M1	M1E1	RAM
N3	C2	D	267677	M2	M2E1	RAM

View 1

Course_No	Prof.
C1	Sanjay
C2	ram

View 2

Name_id	Course_No	grade
N1	C1	A
N2	C2	B
N3	C2	D

View 3

Major_no	Major-Elective
M1	M1E1
M2	M2E1

View 4

Name_id	Phoneno	Major_no
N1	232456	M1
N2	256665	M1
N3	267677	M2

☞ Check Your Progress 1

1) What are the advantages of object oriented databases in comparison with others? Why it is still not widely used?

……

……

……

2) What are three components of the schema architecture proposed by ANSI/SPARC?

……

……

……

3) What are the different approaches for database design?

……

……

……

4) What are the different integrity constraints in RDBMS?

……

……

2.3 OBJECT CLASSES TO DATABASE TABLES

Here, we will consider relational database design as RDBMS technology is gaining more acceptance among organisation and companies and it dominates the marketplace.

2.3.1 Extended Three Schema Architecture for Object Models

First, we should formulate object models for the external, and conceptual schema. Then we should translate each object model to ideal tables, that is, the table model. Views and interface programs connect external tables to conceptual tables. Conceptual tables convert these to the internal schema.

The object model focuses on logical data structures. Each object model consists of many classes, associations, generalizations, and attributes. Object models are effective for communicating with application experts and reaching a consensus about the important aspects of a problem. Object models help developers achieve a consistent, understandable, efficient, and correct database design.

Each table model consists of many ideal tables. These ideal tables are generic and DBMS independent. Ideal tables abstract the common characteristics of RDBMS implementations. The table model decouples DBMS from object model to table model mapping rules.

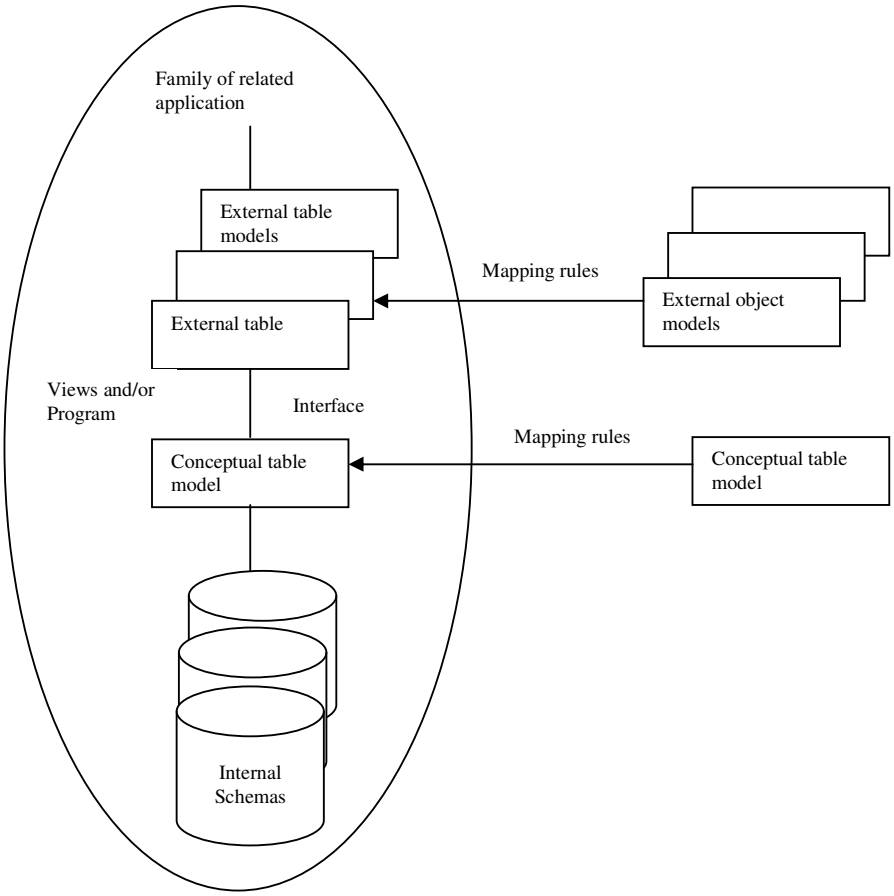


Figure 2: Object modeling and three schema architecture for relational DBMS.

In order to translate from an object model to ideal tables, we must choose from different mapping alternatives. For example, there are **two ways to map an association to tables,** and **four ways to map a generalization**. We must also supply details that are missing from the object model, such as the primary key and candidate keys for each table and whether each attribute can be null, or not null. Attributes in

candidate keys usually 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 consists of SQL commands that create the tables, attributes, and performance tuning structures.

2.3.2 The use of Object IDs

Each class-derived table has an ID for the primary key, one or more object IDs form the primary key for association derived tables. An ID is the equivalent database construct. There are benefits of using the IDs. IDs are never changing and completely independent of changes in data value and physical location. The stability of IDs is particularly important for associations since they refer to objects. A changing in a name requires update of many associations. IDs provide a uniform mechanism for referencing objects.

On the other hand, IDs have some disadvantages also. There is no support by the RDBMS for generating the IDs. For example, it is difficult to track previously allocated IDs, and, to reclaim deleted IDs for reuse.

Object oriented programming uses IDs of many bits, so that reuse of IDs is avoided. We can define IDs as attributes, and adopt a mechanism for handling them.
IDs are not used in applications where users directly access the database. A user thinks in terms of descriptive properties such as names, and not in terms of numbers. The advantage of IDs may prevail when database access is restricted via programs. Restricted access often occurs because application software is needed to compensate for DBMS deficiencies, to enforce integrity, and to provide a user interface.

2.3.3 Mapping Object Classes to Tables

Each **object class** maps to **one,** or **more tables** in the database. The objects in a class may be **partitioned horizontally** and/or **vertically**. For instance, if a class has many instances from which a few are often referenced, then horizontal partitioning may improve efficiency by placing the frequently accessed objects in a table, and the remaining objects in another tables. Similarly, if a class has attributes with different access patterns, it may help to partition the objects vertically.

Employee Table

Table 4: Horizontal Partition

emp_no	Emp_name	Address
1	Ram Kumar	20/780, Hauz Khas, New Delhi
2	Shyam Kumar	2/462, Sector 62, Noida

emp_no	Emp_name	Address
3	Avinash Kumar	F32, Madangir, New Delhi

Table 5: Vertical Partition Horizontal and Vertical Partitioning of Tables

emp_no	Emp_name
1	Ram Kumar
2	Shyam Kumar

emp_no	Address
1	20/780, Hauz Khas, New Delhi
2	2/462, Sector 62, Noida

An object class is converted to one table. Class employee has attributes emp_name and address. The table model lists these attributes, and adds the implicit object ID. As part of formulating the table model, we also add details. We specify that emp_ ID, ID cannot be null since it is a candidate key. Emp_name cannot be null_name must be

entered for every employee. Attribute address can be null. We assign a domain to each attribute, and specify the primary key for each table. Finally, the SQL DDL statements creates the employee table.

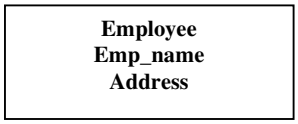


Table Model

Attribute Name	Nulls?	Domain
Emp_ID	N	ID
Emp_name	N	Name
Address	Y	address

Employee Table
Candidate Key: (Emp_ ID)
Primary key : (Emp_ ID)
Frequently accessed: (Emp_ ID)(emp_name)

Create TABLE Employee

(Emp_ID)	ID	not null,
emp_name	char(20)	not null,
address	char(20)	

PRIMARY KEY (Emp_ ID));
CREATE SECONDRY INDEX employee-index-name

ON Employee (emp_name);

MAPPING A CLASS TO A TABLE

2.4 MAPPING ASSOCIATIONS TO TABLES

In this section, we will discuss the mapping of different kinds of associations into database tables.

2.4.1 Mapping Binary Associations to Tables

In general, an association may, or may **not map to a** table. It depends on the type and multiplicity of the association, and the database designer's preferences in terms of extensibility, number of tables, and performance tradeoffs. Let us see one example of mapping a binary association into tables.

Object Model

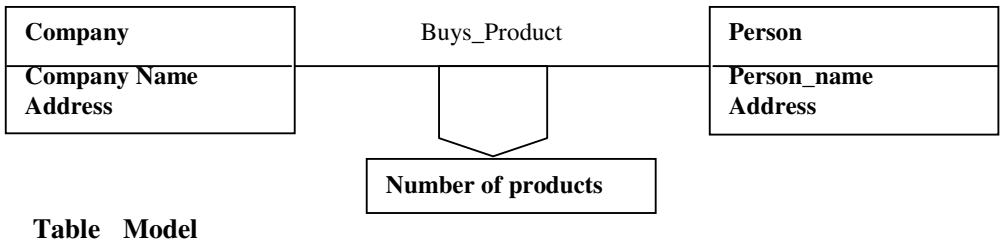


Table Model

Attribute Name	Nulls?	Domain
person_ ID	N	ID
person_name	N	Name
Address	Y	address

Person Table

Candidate Key: (person_ ID)
Primary key : (person_ ID)
Frequently accessed: (person_ ID)(person_name)

Attribute Name	Nulls?	Domain
company_ ID	N	ID
company_name	N	Name
Address	Y	address

Company Table

Candidate Key: (company_ ID)
Primary key: (company_ ID)
Frequently accessed: (company_ ID)
(company_name)

Attribute Name	Nulls?	Domain
company_ ID	N	ID
person_ ID	N	ID
Number_of _products	Y	product-count

Buys_products Table

Candidate Key: (company_ID, person_ ID)
Primary key : (company_ID, person_ ID)
Frequently accessed: (company_ID, person_ ID)

SQL Code

```
Create TABLE Person
(person_ID          ID          not null,
person_name         char(20)    not null,
address             char(20)
PRIMARY KEY (person_ ID));

CREATE SECONDARY INDEX person-index-name
ON Person (person_name);

Similarly, table and indexes can be created for company also
Create TABLE Buys_product
```

SQL Code

```
(company_ID         ID          not null,
person_ID           ID          not null,
number_of_products integer      ,
PRIMARY KEY ((company_ID, person_ ID)),
FOREIGN KEY (company_ID) references Company,
FOREIGN KEY (person_ ID)) references Person);,

CREATE SECONDARY INDEX buys_product-index-
company
 ON buys_product (company_ID);

CREATE SECONDARY INDEX buys_product-index-person
ON buys_product (person_ID);
```

2.4.2 Mapping Many-to-Many Association to Tables

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. Attributes company ID, and person ID combine to form the only candidate key for the Buys_product table. In general, an association may be traversed starting from either

class so that both company ID and Person ID could be frequently accessed. The foreign key clauses to the SQL code indicates that each Buys_product table tuples must reference a company and person that had been defined in their respective tables.

An association table always sets the foreign keys from the related objects to not null. If a given pair of objects does not have a link, we omit an entry in the association table.

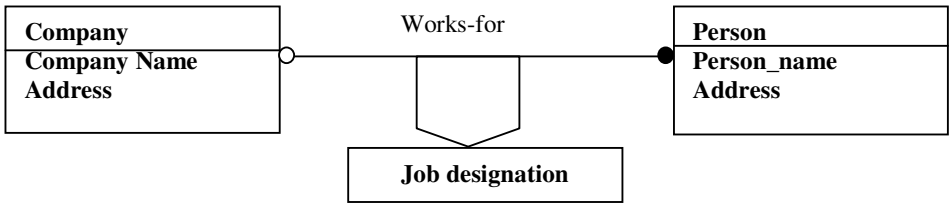


Figure 3: Object Model for one-to-many Association

Company Table (similar to the Company table in the last example)
Person Table (similar to Person table in the last example)

Attribute Name	Nulls?	Domain
company_ ID	N	ID
person_ ID	N	ID
Job_designation	Y	Designation

Works_for Table

Candidate Key: (person_ ID)
Primary key : (person_ ID)
Frequently accessed: (company_ID, person_ ID)

Table Model for one-to-many Associations-distinct association table
Company Table (similar to the Company table in the last example)

Attribute Name	Nulls?	Domain
person_ ID	N	ID
person_name	N	Name
Address	Y	Address
company_ ID	Y	ID
Job_designation	Y	Designation

Person Table

Candidate Key: (person_ ID)
Primary key: (person_ ID)
Frequently accessed: (person_ ID) (person_name)(company_ID)

Table Model for one-to-many Associations-added association table

The *Figure 3* shows two options for mapping a one-to-many association to the tables. We can create a distinct table for the association, or add a foreign key in the table for many class. The advantages of merging an association into a class are:

1. Fewer tables
2. Faster performance due to fewer tables to navigate.

The disadvantage of merging an association into a class are:

1. **Less design firmness:** associations are between independent objects of equal weight. In general, it seems inappropriate to mix objects with knowledge of other objects.
2. **Reduced Extensibility**: it is difficult to get multiplicity right on the first few design passes. One-to-one and one-to-many associations may be externalised. Many-to-many associations must be externalised.

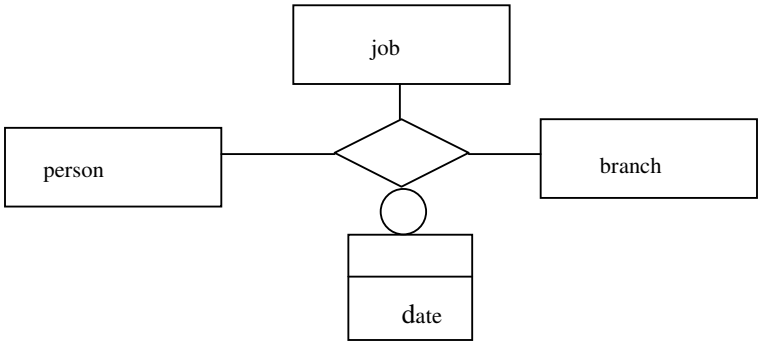
3. **Increased complexity**: an assymetrical representation of the association complicates search and update.

The final decision on whether to merge an association into a related class depends on the application. For a one-to-many association you may also merge both classes and the association all into one table.

2.4.3 Mapping Ternary Associations to Tables

Whenever a ternary relationship is there between the different classes then each class is mapped to a table with the inclusion of object ID. Apart from this a new ternary table is also created which has attributes from the different classes involved in the relationship. The attributes will the object ID of all the classes involved in the relationship and the attributes of the relation. Let us look at one example of mapping a ternary association to the tables.

Object model



Candidate key: (person_ID, branch_ID, job_ID)

Ternary Table:

Attribute Name	Nulls?	Domain
person_ID	N	ID
job_ID	N	ID
branch_ID	N	ID
hours	Y	Time

Table Model

Candidate key: (person_ID, branch_ID, job_ID)
Primary key: (person_ID, branch_ID, job_ID)
Frequently accessed: (person_ID, branch_ID, job_ID)

Person Table:

Attribute Name	**Nulls?**	**Domain**
person_ ID	N	ID
person_name	N	Name
Address	Y	Address

Candidate Key: (person_ ID)
Primary key: (person_ ID)
Frequently accessed: (person_ ID)(person_name)

Similarly, the job and branch tables will have the required attributes.

SQL Code

```
Create TABLE Person
(person_ID          ID          not null,
person_name         char(20)    not null,
address             char(20)
PRIMARY KEY (person_ ID));
```

Similarly, create the other table for job and branch

```
Create TABLE Person-job-branch-ternary
(person_ID          ID          not null,
job_ID              ID          not null,
branch_ID           ID          not null,
date                date                ,
                PRIMARY KEY (person_ID, branch_ID, job_ID),
                Foreign key (person_ID) references person,
                Foreign key (job_ID) references job,
                Foreign key (branch_ID) references branch);
```

Also, create the indexes in the same manner

2.5 MAPPING GENERALIZATIONS TO TABLES

There are different approaches for mapping generalizations to table.

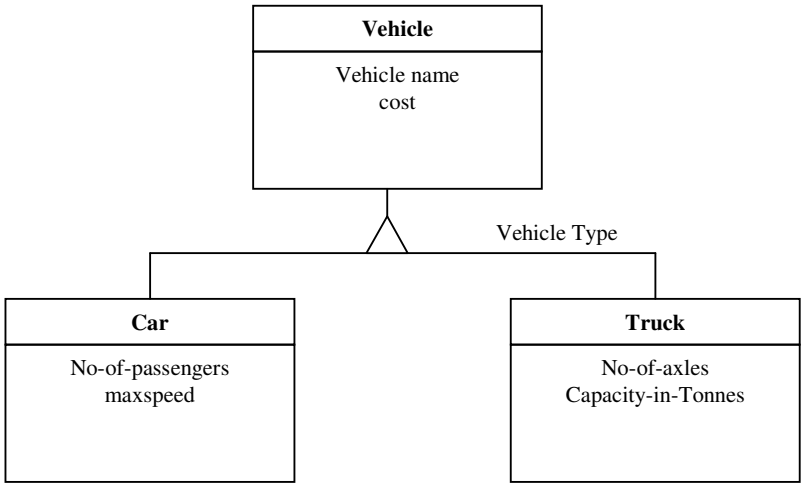


Figure 4: Object Model for Generalization

The first approach is shown in the *Figure 4*. The superclass and the subclass each map to a table. The identity of an object across a generalization is preserved through the use of a shared ID. Thus, vehicle car may have one row in the vehicle table with ID 1001, and another row in the Car table also with ID 1001.This approach is simple and extensible. However, it involves many tables, and super class to subclass navigation may be slow.

The navigation of the tables is as follows:

1. The user gives a vehicle name.
2. Find the Vehicle row that corresponds to vehicle name.
3. Retrieve the vehicle ID and vehicle type for this row.
4. Go to the subclass table indicated by vehicle type, and find the subclass row with the same ID as the Vehicle row.

Table: Model for Generalization Superclass and Subclass Tables:

Vehicle table:

Attribute name	Nulls	Domain
Vehicle-ID	N	ID
Vehicle-name	N	Name
Cost	Y	Money
Vehicle-type	N	Vehicle type

Candidate Key: (Vehicle-ID, Vehicle-name)
Primary key: (Vehicle-ID)
Frequently accessed: (Vehicle-ID, Vehicle-name)

Car table:

Attribute name	Nulls	Domain
Vehicle-ID	N	ID
No-of-passengers	Y	Person
Maxspeed	Y	speed

Candidate Key: (Vehicle-ID)
Primary key : (Vehicle-ID)
Frequently accessed: (Vehicle-ID)

Truck table:

Attribute name	Nulls	Domain
Vehicle-ID	N	ID
No-of-axles	Y	count-axles
Capacity-in-Tonnes	Y	capacity

Candidate Key: (Vehicle-ID)
Primary key: (Vehicle-ID)
Frequently accessed: (Vehicle-ID)

The user may specify vehicle type name "vehicle car." The application looks in the vehicle table and finds that vehicle car has ID 1001 and vehicle type car. The application then searches the car table and retrieves additional data for ID 1001.
The SQL code for the generalization of the super-class and sub-class as mentioned above is as follows:

Create table vehicle

```
(Vehicle-ID          ID          not null,
Vehicle-name         char (40)   not null,
cost                 number,
Vehicle-type         char (8)    not null,
primary key (Vehicle-ID));

Create Secondary index vehicle-index-name on vehicle (vehicle-name);

Create table car
(Vehicle-ID          ID          not null,
no-of-passengers     number,
maxspeed             real,
primary key (Vehicle-ID) ,
foreign key (Vehicle-ID) references vehicle);

Create table truck
(Vehicle-ID          ID          not null,
no-of-axles          number,
Capacity-in-Tonnes   real,
primary key (Vehicle-ID),
foreign key (Vehicle-ID) references Vehicle);
```

In the other approach, the navigation from superclass to subclass is eliminated and thus speed performance is achieved. The improved performance incurs a price by inserting the attributes of Vehicle table in both the subclasses.

Table Model for Generalisation Many Subclass Tables:

Car table:

Attribute name	Nulls	Domain
Vehicle-ID	N	ID
Vehicle-name	N	Name
Cost	Y	Money
No-of-passengers	Y	Person
Maxspeed	Y	Speed

Candidate Keys: (Vehicle-ID, Vehicle-name)
Primary key: (Vehicle-ID)
Frequently accessed: (Vehicle-ID, Vehicle-name)

Truck table:

Attribute name	Nulls	Domain
Vehicle-ID	N	ID
Vehicle-name	N	Name
Cost	Y	Money
No-of-axles	Y	count-axles
Capacity-in-Tonnes	Y	Capacity

Candidate Keys: (Vehicle-ID, Vehicle-name)
Primary Key: (Vehicle-ID)
Frequently accessed: (Vehicle-ID, Vehicle-name)

The above Table (Car Table, Truck Table) illustrates various subclass approaches. This approach eliminates the superclass table and replicates all the superclass attributes in each subclass table. We may use this approach if a subclass has many attributes, the superclass has few attributes, and the application knows which subclass to search.

Table Model for Generalization one Super Class Table

Vehicle table:

Attribute name	Nulls	Domain
Vehicle-ID	N	ID
Vehicle-name	N	Name
Cost	Y	Money
No-of-passengers	Y	Person
Maxspeed	Y	Speed
No-of-axles	Y	count-axles
Capacity-in-Tonnes	Y	Capacity

Candidate Keys: (Vehicle-ID, Vehicle-name)
Primary Key: (Vehicle-ID)
Frequently accessed: (Vehicle-ID, Vehicle-name)

The one superclass table approach shown above brings all subclass attributes up to the superclass level. Each record in the superclass table uses attributes present in one subclass, the other attribute values are null. The table in the above table (Vehicle table) violates third normal form. Vehicle-ID or vehicle-name is the primary key, but attributes values also depend on equipment type. This may be a useful approach if there are only two or three subclasses with few attributes.

The best way to handle generalization relationships that exhibits multiple inheritance from disjoint classes is to use **one table per super-class**, **one table per subclass**. The best way to handle multiple inheritance from overlapping classes is to use one table for the superclass, one table for each subclass, and one table for the generalization relationship.

☞ Check Your Progress 2

1) What is the main advantage of the object model? What are its different components?

……

……

2) What are advantages of object models?

……

……

3) What is an object ID?

……

……

4) What are the advantages of object ID?

……

……

5) How the object classes are mapped to tables?

……

……

6) What are the advantages and disadvantages of merging an association into a class?

……

……

2.6 INTERFACING TO DATABASES

For many applications, the only adequate solution to providing persistency of large amounts of data is to make use of an existing database system. As well as being able to handle effectively unlimited amounts of data, databases provide a number of valuable services, such as support for multiple users, which cannot realistically be implemented again and again, for every applications.

Object oriented databases provide seamless database support for applications designed using object oriented methods. However, affordable object oriented databases are not easily writable for object oriented applications. Object oriented applications may be written using backend relational database. However, this approach can create significant problems because the relational model of data is in some ways quite different from the object model.

Suppose that we have a class diagram describing the data model of an application, containing a number of persistent classes related by associations and generalization relationships. To provide database support for this application, we will have to create a relational database schema enabling the same information to be stored. This activity

involves **translating one notation into another**, and is similar in principle to that of implementing a model in a programming language: we need to find a way of expressing each UML construct using the concepts and notations of the target environment.

Representing associations

Associations can be translated into relational schemas in a number of ways. In the simplest case, a unidirectional link from object X to object Y can be implemented by storing the key value of Y in the row of the table corresponding to X. This is the relational equivalent of one object holding a reference to another; the embedded reference is known as a foreign key in relational database terminology.

Representing generalization

Implementing generalizations in relational databases is slightly problematic, as there is no single feature or technique which provides the required semantics. The most straightforward approach is to represent both the superclass and the subclass in the generalization relation as tables, with attributes of each class defined in the corresponding table.

Interfacing to databases

Once an object oriented data model has been implemented as a relational database, it is necessary to write the code that provides the functionality of the system. This code must be able to read and write data from the database, interpreting it wherever necessary, in terms of the model used by the program.

To achieve this, programming environments typically support an interface that allows programmers to abstract away from the details of individual databases, and enables an application to work with a variety of databases, or data sources. A typical example of such an interface is the Java Database Connectivity (JDBC) API that enables Java programmers to write programs, which interface to relational databases.

Essentially, an API like JDBC enables programmers to manipulate a database by constructing commands in the database query language SQL, executing them on the database, and then dealing with the data that is returned as a result.

☞ Check Your Progress 3

1) Explain how the object classes are mapped to tables.

……

……

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

……

……

3) Explain how the generalizations are mapped to the tables.

……

……

2.7 SUMMARY

In this unit, we have discussed mapping object classes to tables, i.e., each class maps to one, or more tables. Mapping associations to tables, each many-to-many association maps to a distinct table.

We have also seen that each one-to-many association maps to a distinct table, or may be buried as a foreign key in the table for the many class.

- Each one-to-one association maps to a distinct table, or may be buried as a foreign key in the table for either class.
- For one-to-many, and one-to-one associations, if there are no cycles, you have the additional option of storing the association, and both related objects, all in one table. Be aware that this may introduce redundancy, and violate normal forms.
- Role names are incorporated as a part of the foreign key attribute name.
- N-ary (n>2) associations map to a distinct table. Sometimes, it helps to promote an n-ary association to a class.
- A qualified association maps to a distinct table with at least three attributes, the primary key of each related class, and the qualifier.
- Aggregation follows the same rules as association.

Further, we have discussed mapping single inheritance generalizations to tables which includes:

- The superclass and each subclass map to a table.
- No superclass table, superclass attributes are replicated for each subclass.
- No subclass tables, bring all subclass attributes up to the superclass level.

We have also seen that in mapping disjointed multiple inheritance to tables, the super-class and each subclass map to a table. In the mapping overlapping multiple inheritance to tables the superclass and each subclass map to a table, the generalization relationship also maps to a table.

2.8 SOLUTIONS/ANSWERS

Check Your Progress 1

1) Object oriented designs are efficient, coherent, and less prone to the update problems that are not present in many other database design techniques presently.

   It is not used widely because only few database vendors have supported it. It is still not in the commercial stream. It is in the development stage.

2) The three component schema architecture proposed by ANSI/SPARC is as follows:

   External schema: an external view is an abstract representation of some portion of the total database.
   Conceptual schema: the conceptual view is an logical representation of the database in its entirety.
   Internal schema: the internal schema is the database as it is physically stored.

3) There are two basic approaches for database design which are as follows:

   Attribute driven: compile a list of attributes relevant to the application, and normalize the groups of attributes that preserve functional dependencies.

   Entity driven: discover entities that are meaningful to the application, and describe them.

4) The different integrity constraints in RDBMS are as follows:

Primary Key: a primary key is a combination of one or more attributes whose unique value locates each row in a table.

Referential integrity (A foreign key): foreign key is an attribute of one table that references to attribute in another table.

Check Your Progress 2

1) The object model focuses on logical data structures. Each object model consists of many classes, associations, generalizations, and attributes.

2) Object models are effective for communicating with application experts and reaching a consensus about the important aspects of a problem. Object models help developers achieve a consistent, understandable, efficient, and correct database design.

3) Each class-derived table has an ID for the primary key, one or more object IDs form the primary key for association derived tables. An object ID is the equivalent database construct.

4) The advantages of Object IDs are as follows:

   i) IDs are never changing
   ii) IDs are completely independent of changes in data value and physical location
   iii) IDs provide a uniform mechanism for referencing objects.

5) Each object class maps to one or more tables in the database. The objects in a class may be partitioned horizontally and/or vertically.

6) Advantages:

   i) Fewer tables
   ii) Faster performance due to fewer tables to navigate.

Disadvantages:

   i) Less design rigor
   ii) Reduced Extensibility
   iii) Increased complexity

Check Your Progress 3

1) Each object class maps to one, or more tables in the database. The objects in a class may be partitioned horizontally and/or vertically. For instance, if a class has many instances of which, a few are often referenced. In this case, horizontal partitioning may improve efficiency by placing the frequently accessed objects in a table, and the remaining objects in another tables. Similarly, if a class has attributes with different access patterns, it may help to partition the objects vertically.

2) Whenever a ternary relationship is there between the different classes, then each class is mapped to a table with the inclusion of object ID. Apart from this, a new ternary table is also created which has attributes from the different classes involved in the relationship. The attributes will include the object ID of all the classes involved in the relationship, and the attributes of the relation.

3) There are two approaches in which generalizations are mapped to the tables:

   The **superclass** and the **subclass** each map to a table. The identity of an object across a generalization is preserved through the use of a **shared ID.** The one superclass table will have all the subclasses attributes up to the super-class level. Each record in the superclass table uses attributes present in one subclass, the other attribute values are null.