Parul® University
Vadodara, Gujarat

NAAC GRADE A++

Information and Communication Technology

Hibernate (ORM)

Study

ARNIKA PATEL
CSE, PIT
Parul University

4.1 Introduction to Hibernate

- Hibernate is a framework which is used to develop persistence logic which is independent of Database software.
- In JDBC to develop persistence logic we deal with primitive types. Whereas Hibernate framework we use Objects to develop persistence logic which are independent of database software.

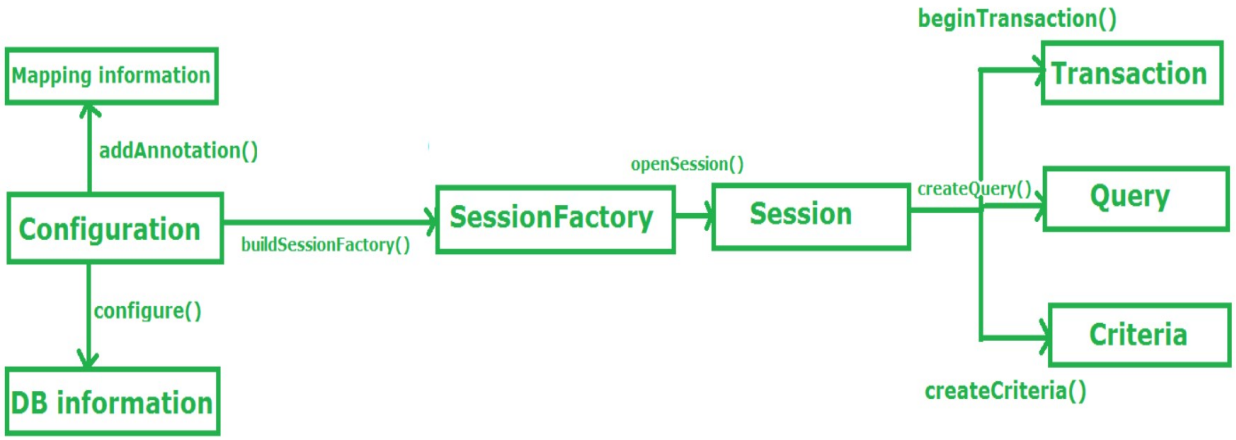

Fig. 4.1 Hibernate Architecture

- **Configuration:**
  - Configuration is a class which is present in org.hibernate.cfg package. It activates Hibernate framework. It reads both configuration file and mapping files.
  - It checks whether the config file is syntactically correct or not.
  - If the config file is not valid then it will throw an exception. If it is valid then it creates a meta-data in memory and returns the meta-data to object to represent the config file.
  - It activate Hibernate Framework

    Configuration cfg=new Configuration();
  - It read both cfg file and mapping files

    cfg.configure();
- **SessionFactory:**
  - SessionFactory is an Interface which is present in org.hibernate package and it is used to create Session Object.
  - It is immutable and thread-safe in nature.
  - buildSessionFactory() method gathers the meta-data which is in the cfg Object. From cfg object it takes the JDBC information and create a JDBC Connection.

    SessionFactory factory=cfg.buildSessionFactory();

- **Session:**
  - Session is an interface which is present in org.hibernate package. Session object is created based upon SessionFactory object i.e. factory.
  - It opens the Connection/Session with Database software through Hibernate Framework.
  - It is a light-weight object and it is not thread-safe.
  - Session object is used to perform CRUD operations.

    Session session = factory.openSession();
  - **openSession()** is a method provided by the SessionFactory that creates and returns a new Session instance.
  - This session is not bound to any transaction or context and is independent of any ongoing transactions in the application.
  - We can also use getCurrentSession, that returns a Session bound to the current context, which is usually managed by a transaction manager or a framework like Spring.

    Session session = sessionFactory.getCurrentSession();
- **Transaction:**
  - Transaction object is used whenever we perform any operation and based upon that operation there is some change in database.
  - Transaction object is used to give the instruction to the database to make the changes that happen because of operation as a permanent by using commit() method.

    Transaction tx=session.beginTransaction();

    tx.commit();
- **Query:**
  - Query is an interface that present inside org.hibernate package.
  - A Query instance is obtained by calling Session.createQuery().
  - This interface exposes some extra functionality beyond that provided by Session.iterate() and Session.find():

    A particular page of the result set may be selected by calling setMaxResults(), setFirstResult().

    Named query parameters may be used.
- **Criteria:**
  - Criteria is a simplified API for retrieving entities by composing Criterion objects.

- The Session is a factory for Criteria. Criterion instances are usually obtained via the factory methods on Restrictions.

- **Hibernet Architecture Workflow**

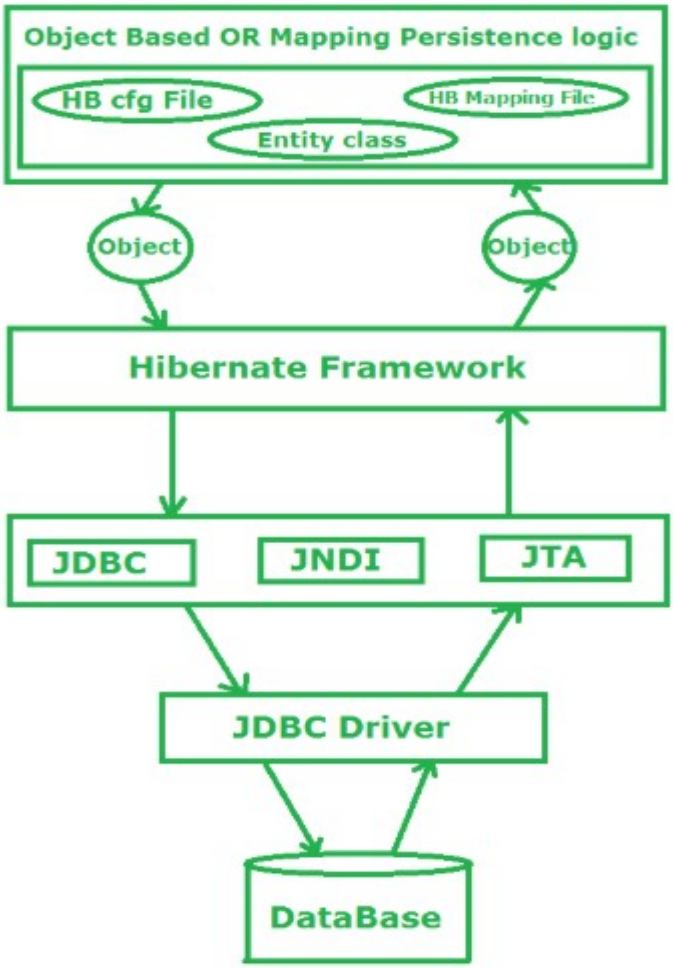

Fig. 4.2 Architecture Workflow

- **Stage I:** In first stage, we will write the persistence logic to perform some specific operations to the database with the help of Hibernate Configuration file and Hibernate mapping file. And after that we create an object of the particular class on which we wrote the persistence logic.
- **Stage II:** In second stage, our class which contains the persistence logic will interact with the hibernate framework where hibernate framework gives some abstraction do perform some task. Now here the picture of java class is over. Now Hibernate is responsible to perform the persistence logic with the help of layers which is below of Hibernate framework or we can say that the layers which are the internal implementation of Hibernate.
- **Stage III:**In third stage, our hibernate framework interact which JDBC, JNDI, JTA etc to go to the database to perform that persistence logic.

- **Stage IV & V:**In fourth & fifth stage, hibernate is interact with Database with the help of JDBC driver.

4.2 JPA

- JPA stands for Java Persistence API (Application Programming Interface).
- It was initially released on 11 May 2006. It is a Java specification that provides functionality and standards for ORM tools.
- It is used to examine, control, and persist data between Java objects and relational databases. It is regarded as a standard technique for Object Relational Mapping.
- **Key Features of JPA**
  - It is a lightweight persistence API.
  - Supports object-relational mapping with simple annotations.
  - Provides support for polymorphism and inheritance.
  - Allows dynamic and named queries.
  - Enables seamless switching between different ORM implementations.
- **The main difference between JPA and Hibernate is that,**
  - JPA is a specification that defines how ORM tools should function.
  - Hibernate is an actual implementation of JPA that provides ORM capabilities.

Table 4.1 JPA vs. Hibernate

JPA	Hibernate
JPA is described in **javax.persistence** package.	Hibernate is described in **org.hibernate** package.
It describes the handling of relational data in Java applications.	Hibernate is an Object-Relational Mapping (ORM) tool that is used to save Java objects in the relational database system.
It is not an implementation. It is only a Java specification.	Hibernate is an implementation of JPA. Hence, the common standard which is given by JPA is followed by Hibernate.
It is a standard API that permits to perform database operations.	It is used in mapping Java data types with SQL data types and database tables.
As an object-oriented query language, it uses Java Persistence Query Language (JPQL) to execute database operations.	As an object-oriented query language, it uses Hibernate Query Language (HQL) to execute database operations.
To interconnect with the entity manager factory for the persistence unit, it uses	To create Session instances, it uses the **SessionFactory** interface.

the **EntityManagerFactory** interface. Thus, it gives an entity manager.	
To make, read, and remove actions for instances of mapped entity classes, it uses the **EntityManager** interface. This interface interconnects with the persistence condition.	To make, read, and remove actions for instances of mapped entity classes, it uses **Session** interface. It acts as a runtime interface between a Java application and Hibernate.

4.3 Generator Class

- A **generator** class is used to generate an **ID** for an object, which is going to be inserted in the database as a primary key.
- The **ID** is a unique identifier for the objects of the persistent class.
- We can use any **generator** classes in our application as per our requirements.
- Hibernate provides a **<generator>** tag to represent the generator.
- However, it is a sub-element of <id>.
- There is a shortcut name for each predefined **generator** class so that the **<generator>** tag has to configure with this shortcut name.
- **Hibernate** provides many predefined **generator** classes. Some of the important predefined generator classes in hibernate are:

  1. assigned generator

  - The assigned generator is the default generator. By default hibernate consider assigned as a generator and assigned is a shortcut name given for the Assigned class.
  - This class returns the same id set by us to the hibernate which in turn store an object with that id in the database.

  ```
  <hibernate-mapping>
      <class …>
          <id name=”studentId” column=”sid”>
              <generator class=”assigned”></generator>
          </id>
      </class>
  </hibernate-mapping>
  ```

2. foreign generator

- **foreign** is a shortcut name given for the **ForeignGenerator** class.

- The **ForeignGenerator** is only suitable for one-one relationships.
- It returns the id of the **parent object** as the id for the **child object.**

```
<hibernate-mapping>
    <class …>
        <id name="studentId" column="sid">
            <generator class="foreign"></generator>
        </id>
    </class>
</hibernate-mapping>
```

3. hilo generator

- **hilo** is a shortcut name for the **TableHiloGenerator** class.
- While configuring **hilo** generator class in the **hbm.xml** file, we pass **"table", "column"**, and **"max_lo"** as parameters.
- The default value of above three parameters given by **Hibernate** are:

table = **hibernate_unique_key**

column = **next_hi**

max_lo = **32767**

```
<hibernate-mapping>
    <class …>
        <id name="studentId" column="sid">
            <generator class="hilo">
                <param
                    name="table">document</param>
                <param name="column">column</param>
                <param name="max_lo">13210</param>
            </generator>
        </id>
    </class>
</hibernate-mapping>
```

4. identity generator

- **identity** is a shortcut name given for the **IdentityGenerator** class.
- It reads an auto-incremented column algorithm of the database and takes that value and returns it as id to hibernate.
- **Identity** generator supports databases like **Sybase, My SQL, MS SQL Server,**

DB2, **and HypersonicSQL** but doesn't support **Oracle** database because there is no autoincrement functionality in **Oracle** database.

- Hence identity generator can be considered as a database-dependent generator.

```
<hibernate-mapping>
    <class …>
        <id name="studentId" column="sid">
            <generator class="identity"></generator>
        </id>
    </class>
</hibernate-mapping>
```

5. increment generator

- An **increment** is a shortcut name given for the **IncrementGenerator** class.
- The **IncrementGenerator** class reads the max value of the existing ID in the database table and then increments it by one and returns the ID value to hibernate.

```
<hibernate-mapping>
    <class …>
        <id name="studentId" column="sid">
            <generator class="increment"></generator>
        </id>
    </class>
</hibernate-mapping>
```

6. native generator

- **native** is the shortcut name given for the **NativeGenerator** class.
- If the database supports an **identity** generator, then it acts as **identity** otherwise it will check for the database support of other generators.
- It selects **identity, sequence**, or **hilo** depending upon the capabilities of the underlying database.

```
<hibernate-mapping>
    <class …>
        <id name="studentId" column="sid">
            <generator class="native"></generator>
        </id>
    </class>
</hibernate-mapping>
```

7. sequence generator

- **sequence** is a shortcut name given for the **SequenceGenerator** class. It reads the next value of a database sequence and then returns that value as **id** to the hibernate.
- **Syntax to create a sequence in the database:**

  create sequence <sequence_name> start with <random_number> increment by <random_number>

```
<hibernate-mapping>
  <class …>
      <id name="studentId" column="sid">
          <generator class="sequence">
              <param name="sequence">sequential_datasource</param>
          </generator>
      </id>
  </class>
</hibernate-mapping>
```

8. uuid generator

- **uuid** is a shortcut name given for the **AbstractUUIDGenerator** class.
- It generates a unique string Identifier and returns to hibernate based on the **IP Address of the machine**, **Start-up time of JVM, System time**, and **Counter value in JVM.**

```
<hibernate-mapping>
  <class …>
      <id name="studentId" column="sid">
          <generator class="uuid">    </generator>
      </id>
  </class>
</hibernate-mapping>
```

4.4 Dialects

- **Dialect** is a class that acts as a bridge between **Java JDBC types** and **SQL types**, which contains the mapping between java language data type and database datatype.
- Dialect allows Hibernate to generate SQL optimized for a particular relational database. Hibernate generates queries for the specific database based on the **Dialect** class.
- A hibernate dialect gives information to the framework of how to convert **hibernate queries(HQL)** into native **SQL queries**.

- Dialects can be used in the following ways:
    - To generate Optimized SQL queries
    - To interact with a particular Database if the application works with the help of more than one Database.
    - To set default values for hibernate configuration file properties based on the database software even though they are not specified in the configuration file.
- **SQL Dialects Configuration**
- The SQL dialect converts the **HQL query** which we write in java or any other object-oriented program to the specific database SQL query.
- For connecting any hibernate application with the database, it is required to provide the configuration of SQL dialect.
- We can provide it in **hibernate.cfg.xml (DB2 Dialect)** as :

  <hibernate-configuration>

  <session-factory name="session-factory">

  *<property name="hibernate.dialect">**org.hibernate.dialect.DB2Dialect**</property>*

  </session-factory>

  </hibernate-configuration>
- We can also specify in the properties file as :

  hibernate.dialect=org.hibernate.dialect.DB2Dialect
- The **hibernate.dialect** property should be set to the correct org.hibernate.dialect.
- Dialect subclass for the application database. If the Dialect class is not specified in the configuration, for most of the databases, Hibernate tries to resolve dialect names from the database connections. But It is best to provide dialect so that Hibernate identifies the appropriate Dialect class for specific database versions.

4.5 Mapping

The hibernate works to link the JAVA language to the database table along with this link we can establish relations/mappings.

The main basic types of mapping are:

Primitive Types

Table 5.2 Primitive Types

Mapping Type	Java Type	ANSI SQL Type
integer	int or java.lang.Integer	INTEGER
character	java.lang.String	CHAR(1)
float	float or java.lang.Float	FLOAT
string	java.lang.String	VARCHAR
double	double or java.lang.Double	DOUBLE
boolean	boolean or java.lang.Boolean	BIT
short	short or java.lang.Short	SMALLINT
long	long or java.lang.Long	BIGINT
byte	byte or java.lang.Byte	TINYINT
big_decimal	java.math.BigDecimal	NUMERIC

Date and Time Types

Table 5.3 Date and Time *Types*

Mapping type	Java type	ANSI SQL Type
date	**java.util.Date or java.sql.Date**	**DATE**
time	**java.util.Date or java.sql.Time**	**TIME**
calendar	**java.util.Calendar**	**TIMESTAMP**
timestamp	**java.util.Date or java.sql.Timestamp**	**TIMESTAMP**
calendar_date	**java.util.Calendar**	**DATE**

Binary and Large Object Types

Table 5.4 Binary and large object

Mapping type	Java type	ANSI SQL Type
clob	java.sql.Clob	CLOB
blob	java.sql.Blob	BLOB
binary	byte[]	VARBINARY (or BLOB)
text	java.lang.String	CLOB
serializable	any Java class that implements java.io.Serializable	VARBINARY (or BLOB)

JDK-related Types

Table 5.5 JDK-related

Mapping type	Java type	ANSI SQL Type
class	java.lang.Class	VARCHAR
locale	java.util.Locale	VARCHAR
currency	java.util.Currency	VARCHAR
timezone	java.util.Currency	VARCHAR

4.6 Annotations

- Annotation in JAVA is used to represent supplemental information. As you have seen @override, @inherited, etc are an example of annotations in general Java language the motive of using a hibernate is to skip the SQL part and focus on core java concepts.
- Generally, in hibernate, we use XML mapping files for converting our POJO classes data to database data and vice-versa. But using XML becomes a little confusing so, in

replacement of using XML, we use annotations inside our POJO classes directly to declare the changes.

- Also using annotations inside out POJO classes makes things simple to remember and easy to use.
- Annotation is a powerful method of providing metadata for the database tables and also it gives brief information about the database table structure and also POJO classes simultaneously.

Table 5.6 Annotations

Annotations	Use of annotations
@Entity	Used for declaring any POJO class as an entity for a database
@Table	Used to change table details, some of the attributes are- • name – override the table name • schema • catalogue • enforce unique constraints
@Id	Used for declaring a primary key inside our POJO class
@GeneratedValue	Hibernate automatically generate the values with reference to the internal sequence and we don't need to set the values manually.

Annotations	Use of annotations
@Column	It is used to specify column mappings. It means if in case we don't need the name of the column that we declare in POJO but we need to refer to that entity you can change the name for the database table. Some

	attributes are- • Name – We can change the name of the entity for the database • length – the size of the column mostly used in strings • unique – the column is marked for containing only unique values • nullable – The column values should not be null. It's marked as NOT
@Transient	Tells the hibernate, not to add this particular column
@Temporal	This annotation is used to format the date for storing in the database
@Lob	Used to tell hibernate that it's a large object and is not a simple object

4.7 Transaction Management

- A transaction is a sequence of operation which works as an atomic unit.
- A transaction only completes if all the operations completed successfully.
- A transaction has the Atomicity, Consistency, Isolation and Durability properties (ACID).
- **Transaction interface:**
- Transaction interface provides the facility to define the units of work or transactions.
- A transaction is associated with a session.
- We have to call beginTransaction()method of Session to start a transaction (Session.beginTransaction()).
- Commonly used methods of Transaction interface:

  1. begin(): It starts a new transaction.

  Syntax:

  public void begin() throws HibernateException

  2. commit(): It ends the transaction and flush the associated session.

  Syntax:

  public void commit() throws HibernateException

  3. rollback(): It roll back the current transaction.

Syntax:

public void rollback()throws HibernateException

4. setTimeout(int seconds): It set the transaction timeout for any transaction started by a subsequent call to begin() on this instance.

Syntax:

public void **setTimeout(int seconds)** throws HibernateException

5. isActive(): It checks that is this transaction still active or not.

Syntax:

public boolean isActive()throws HibernateException

6. wasRolledBack(): It checks that is this transaction roll backed successfully or not.

Syntax:

public boolean wasRolledBack()throws HibernateException

7. wasCommitted(): It checks that is this transaction committed successfully or not.

Syntax:

public boolean wasCommitted()throws HibernateException

8.registerSynchronization(Synchronization synchronization): It register a user synchronization callback for this transaction.

Syntax:

public boolean registerSynchronization(Synchronization synchronization)throws HibernateException

Example:

```
Transaction tx = null;
Session session = HibernateUtil.getSessionFactory().openSession();
try{
        tx = session.beginTransaction();
        //Perform some operation here
        tx.commit();
}catch (HibernateException e) {
        if(tx!=null){
                tx.rollback();
         } e.printStackTrace();
}finally {
        session.close();
```

```
}
```

4.8 HQL

- HQL stands for Hibernate Query Language, a language used within the Hibernate Framework for querying and manipulating data in relational databases.
- Hibernate converts HQL queries into SQL queries, which are used to perform database actions.
- Although Native SQL may be used directly with Hibernate, it is encouraged to utilize HQL wherever feasible to prevent database portability issues.
- HQL has many benefits. Some benefits are:
  - HQL is database-independent.
  - polymorphic queries supported which are type-safe.
  - It is portable and easy to learn for Java programmers.
- There are many HQL clauses available to interact with relational databases, and several of them are listed below:

1. **FROM Clause**

   To load a whole persistent object into memory, the FROM clause is used.

   ```
   String hib = "FROM Student";
   Query query = session.createQuery(hib);
   List results = query.list()
   ```

2. **SELECT Clause**

   The SELECT clause is used when only a *few attributes of an object* are required rather than the entire object.

   ```
   String hib = "SELECT S.roll FROM Student S";
   Query query = session.createQuery(hib);
   List results = query.list();
   ```

3. **WHERE Clause**

   Filtering records is done with the WHERE clause. It's used to *retrieve only the records that meet a set of criteria*.

   ```
   String hib = "FROM Student S WHERE S.id = 5";
   Query query = session.createQuery(hib);
   List results = query.list();
   ```

4. **ORDER BY Clause**

   The ORDER BY clause is used to *sort the results* of an HQL query.

   ```
   String hib = "FROM Student S WHERE S.id > 5 ORDER BY S.id DESC";
   ```

```
Query query = session.createQuery(hib);
List results = query.list();
```

5. **UPDATE Clause**

   The UPDATE clause is required to *update the value* of an attribute.

```
String hib = "UPDATE Student set name=:n WHERE roll=:i";
Query q=session.createQuery(hib);
q.setParameter("n","John");
q.setParameter("i",23);
int status=q.executeUpdate();
System.out.println(status);
```

6. **DELETE Clause**

   It is required to *delete a value* of an attribute.

```
String hib = "DELETE FROM Student WHERE id=10";
Query query=session.createQuery(hib);
query.executeUpdate();
```

7. **INSERT Clause**

   It is required to *Insert values* into the relation.

```
String hib = "INSERT INTO Student(first_name, last_name)" +
"SELECT first_name, last_name FROM backup_student";
Query query = session.createQuery(hib);
int result = query.executeUpdate();
```

4.9 HCQL

- Hibernate Criteria Query Language (HCQL) provides a type-safe, object-oriented approach to querying database entities in Hibernate. It's built using Java code instead of string-based queries like HQL, offering better readability, maintainability, and reduced risk of errors.
- HCQL uses the Criteria interface, Restrictions class, and Order class to define query conditions, filters, and sorting.
- **Criteria Interface**
  - The **Criteria** interface is the main part of HCQL. It provides methods to add conditions, set limits, and define how results should be ordered.
  - You create a **Criteria** object by calling the **createCriteria()** method from the Hibernate **Session**.
  - **Example of creating a Criteria object:**

- Criteria c = session.createCriteria(Emp.class);

- **Advantages of HCQL**
  - HCQL provides built-in methods to add conditions, making it simple for Java developers to use.
  - You can easily combine multiple conditions in a single query without worrying about query syntax.
  - Since queries are built using Java code, it's easier to spot mistakes and fix them.
- **Commonly used methods of Criteria:**
  - **add(Criterion c)**: Adds a condition to the query.
  - **addOrder(Order o)**: Specifies how the results should be ordered.
  - **setFirstResult(int firstResult)**: Sets where to start fetching records (useful for pagination).
  - **setMaxResults(int totalResult)**: Sets how many records to fetch.
  - **list()**: Executes the query and returns the results as a list.

  setProjection(Projection p): Allows fetching specific columns instead of full objects.
- **Order Class:** The **Order** class is used to sort the results in ascending or descending order based on a specific property.
  - **asc(property)**: Sorts in ascending order.
  - **desc(property)**: Sorts in descending order.

References:

1. **Book Reference**

   Jim Farley, William Crawford, David Flanagan. Java Enterprise in a Nutshell, O'Reilly
2. **Book Reference**

   Rocha, R., Purificação, J. (2018). Java EE 8 Design Patterns and Best Practices: Build Enterprise-ready Scalable Applications with Architectural Design Patterns. Germany: Packt Publishing..

3. **Website Reference**

   https://www.scribd.com/document/268349254/Java-8-Programming-Black-Book .

4. **Sources**

   https://developers.redhat.com/topics/enterprise-java

5. **Article**

   https://www.researchgate.net/publication/276412369_Advanced_Java_Programming

Parul® University
Vadodara, Gujarat

NAAC GRADE A++

Information and Communication Technology

NAAC GRADE A++

https://paruluniversity.ac.in/