#1. Table per class hierarchy

descriminator-value: ip

```
class InsurancePlan {
  int insurancePlanNo;
  String planName;
  int minTenure;
  double minInsuranceAmount;
}
```

[since all the entity classes in the hierarchy or persisted into one single table, we need descriminator-column to differentiate or identify which record belongs to which entity class in the hierarchy]

```
Table: insurance_plan
columns:
insurance_plan_no
plan_nm
min_tenure
min_insurance_amt
coverage_type
partial_disability_coverage_percentage
death_coverage_percentage
copay
network_type
plan_type VARCHAR(10) #descriminator_column
                       additional column
```

descriminator-value: aip

```
class AccidentalInsurancePlan
          extends InsurancePlan {
  String coverageType;
  double partialDisabilityCoveragePercentage;
  double deathCoveragePercentage;
}
```

descriminator-value: mip

```
class MedicalInsurancePlan extends
InsurancePlan {
  int copay;
  String networkType;
}
```

Let us understand how to perform persistence operations for these entity classes of hierarchial relationship into the underlying database tables.

#1. persist
parent entity class (InsurancePlan):
Store the data of the parent entity class object into the same one single-table (InsurancePlan:object into insurance_plan:table). But while storing the superclass object, insert NULL values into the subclass represented columns of that table

any of the subclass entities (AccidentalInsurancePlan or MedicalInsurancePlan):
Every subclasses has all the attributes of the data of its parent. So while persisting the subclass object, along with its class attributes, even the inherited attributes also should be persisted into the underlying table

since we have only one-single table all the attributes of data pertaining to the subclass (including inherited properties) should be persisted into the single table leaving other subclass attributes as “NULL”

Note: while storing any of the entity classes objects in the hierarchy into the insurance_plan table, persist the descriminator-value of the entity class into descriminator-column of the insurance_plan table. So that we can identify which record belongs to which entity class of the hierarchy.
InsurancePlan = ip, AccidentalInsurancePlan = aip, MedicalInsurancePlan = mip

---

#2 querying the object (fetching)
Let us consider we have persisted objects of data within the hierarchy into the underlying tables as below
insurance_plan_no:1 = InsurancePlan object
insurance_plan_no:2 = AccidentalInsurancePlan object
insurance_plan_no:3 = MedicalInsurancePlan object

parent entity class (InsurancePlan):
session.get(InsurancePlan.class, 1);

In order to fetch the data for insurance_plan_no:1 of entity class InsurancePlan type we need to go and query the data from insurance_plan table and populate the data into InsurancePlan object and return

any of the subclass entities (AccidentalInsurancePlan or MedicalInsurancePlan):
session.get(AccidentalInsurancePlan.class, 2);
(or)
session.get(MedicalInsurancePlan.class, 3);

since all the entity class objects data is stored in same table insurance_plan, we need to go and fetch the data from insurance_plan table only and populate into appropriate entity class object as requested and return

Note: while querying the data for any of the sub-class types we need to consider the descriminator-column value as well to verify whether the record belongs to the corresponding subclass or not as below

session.get(AccidentalInsurancePlan.class, 2);
select * from insurance_plan where insurance_plan_no=2 and plan_type=‘aip’; = If record found, return it as AccidentalInsurancePlan object

Till now all the things seems to quite easy and straight forward, let us analyze the challenging path:

MedicalInsurancePlan mip = session.get(MedicalInsurancePlan.class, 2);
Here the insurance_plan_no: 2 asper the above assumption belongs to AccidentalInsurancePlan, when we are querying it as MedicalInsurancePlan, we should be able to detect this and report as no object of Type: MedicalInsurancePlan with plan_no: 2 found should be reported by returning “null” object

But how do we identify which record in the insurance_plan table belongs to which entity class type, since all the objects are persisted into same table since we have descriminator-column using it, we can identify the record: 2 belongs to MedicalInsurancePlan class object or not

session.get(MedicalInsurancePlan.class, 2);
select * from insurance_plan where insurance_plan_no=2 and plan_type=‘mip’ = this doesnt not return any record as plan_type of the record is not matching.
So we can detect and avoid fetching one sub-class type object as another sub-class type.

---

insurance_plan_no = 1 belongs to InsurancePlan (object)
insurance_plan_no = 2 belongs to AccidentalInsurancePlan (object)
insurance_plan_no = 3 belongs to MedicalInsurancePlan (object)

AccidentalInsurancePlan aip = session.get(AccidentalInsurancePlan.class, 2);
Yes since insurance_plan_no = 2 record has descriminator_column(plan_type) value as ‘aip’ which means the record belongs to AccidentalInsurancePlan type only so we can fetch the record into AccidentalInsurancePlan class object

internal query:
select * from insurance_plan where insurance_plan_no = 2 and plan_type = ‘aip’;

InsurancePlan ip = session.get(InsurancePlan.class, 2);
Yes, this way of querying the data should be supported, as it is polymorphic retrieval of data.
Polymorphic retrieval = should support querying any of the subclass object types in the hierarchy as parent class reference Type

Even though here we are querying insurance_plan_no=2 record as InsurancePlan class type, we should detect the underlying record descriminator-column value based on that we should identify the record belongs to which classType and return an appropriate class object rather than superClass type only

internal query:
select *, plan_type from insurance_plan where insurance_plan=2;
based on the plan_type of the record returned by the query: we should determine
ip = InsurancePlan
aip = AccidentalInsurancePlan
mip = MedicalInsurancePlan

create appropriate object, populate the data and return it.

Note: so while always querying the data for a superclass type, inorder to support polymorphic retrieval we should not return superclass reference type object, we should identify the underlying record belongs to which classType based on descriminator-column value and return an appropriate object by populating the data.

The way we insert, update, delete or query the data for the entity classes in inheritance hierarchy into underlying database tables when choosing the strategy as “Table per class hierarchy” is same for any of the entity classes. Seems to be pretty standard way

How to handle persistence operations for the entity classes (inheritance hierarchy): Table per class hierarchy strategy is well known to hibernate and jpa api as well, because it is same irrespective of who does it.
So already the code for carrying the persistence operations are pre-coded by these orm frameworks or jpa api, all that we need to do is tell them how did we mapped the hierarchy classes into table model.