SQLAlchemy **infers** whether a relationship is *one-to-many* or *many-to-one* automatically  
based on **where the foreign key (ForeignKey)** is declared.

**In our code:**

**Parent class:**

class Teacher(Base):

\_\_tablename\_\_ = 'teachers'

id = Column(Integer, primary\_key=True)

name = Column(String(50))

subject = Column(String(50))

students = relationship("Student", back\_populates="teacher")

**Child class:**

class Student(Base):

\_\_tablename\_\_ = 'students\_one\_to\_many'

id = Column(Integer, primary\_key=True)

name = Column(String(50))

teacher\_id = Column(Integer, ForeignKey('teachers.id'))

teacher = relationship("Teacher", back\_populates="students")

**How SQLAlchemy Infers the**

**Relationship Type**

1. **It looks for the ForeignKey column.**

Here:

teacher\_id = Column(Integer, ForeignKey('teachers.id'))

means —  
each Student record refers to **one specific Teacher**.

So from the **Student side**, SQLAlchemy knows:  
→ This is a **many-to-one** relationship.

1. **When you define relationship() on the other side (Teacher.students)**,  
   SQLAlchemy automatically **reverses** that logic —  
   if the Student table points to Teacher via a foreign key,  
   then the Teacher table must be related to *multiple students*.

Hence, Teacher.students becomes a **one-to-many** collection.

**About uselist=True**

you can specify it manually:

students = relationship("Student", back\_populates="teacher", uselist=True)

But:

* uselist=True is the **default** behavior when SQLAlchemy detects a one-to-many relationship.
* So unless you explicitly set uselist=False, it assumes that the relationship returns a **list-like collection** (because one teacher → many students).

**Quick test to verify:**

You can print the type of each relationship:

print(type(t1.students)) # → <class 'sqlalchemy.orm.collections.InstrumentedList'>

print(type(s1.teacher)) # → <class 'models.Teacher'>

So:

* t1.students → a list of Student objects
* s1.teacher → a single Teacher object