Objectivity/DB

What is Objectivity/DB?

- Objectivity/DB is an *Object Database Management System* (ODBMS).

What it offers?

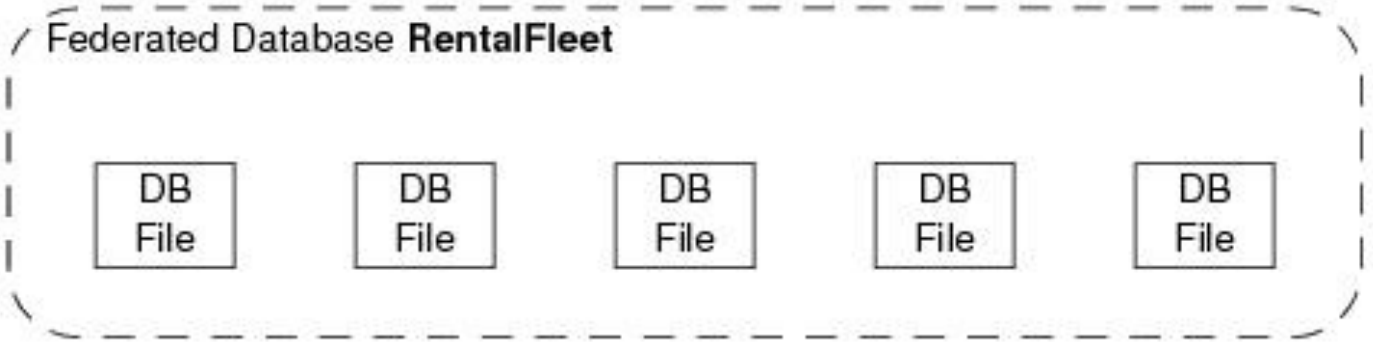
- **Object-oriented:** Objectivity/DB stores data that is created by applications written in any of the main *object-oriented* programming languages—for example, C++, Java, and .NET/C#. The data produced by such applications are in the form of *objects*.
- **Objectivity/SQL++** provides a standard structured query language (SQL) interface that makes it possible to run industry-standard database query and visualization tools with Objectivity/DB objects.
- **DBMS:** Objectivity/DB has all the usual database-management services: concurrency, transactions, recovery management, and queries.
- **Highly scalable:** Objectivity/DB provides control over the clustering of data within files and memory, distribution of data files and client applications across the network, and replication of files and DBMS resources. These capabilities support improved runtime performance of client applications, very large databases (VLDBs), and high availability of data.

Objectivity/DB and Traditional DBMS

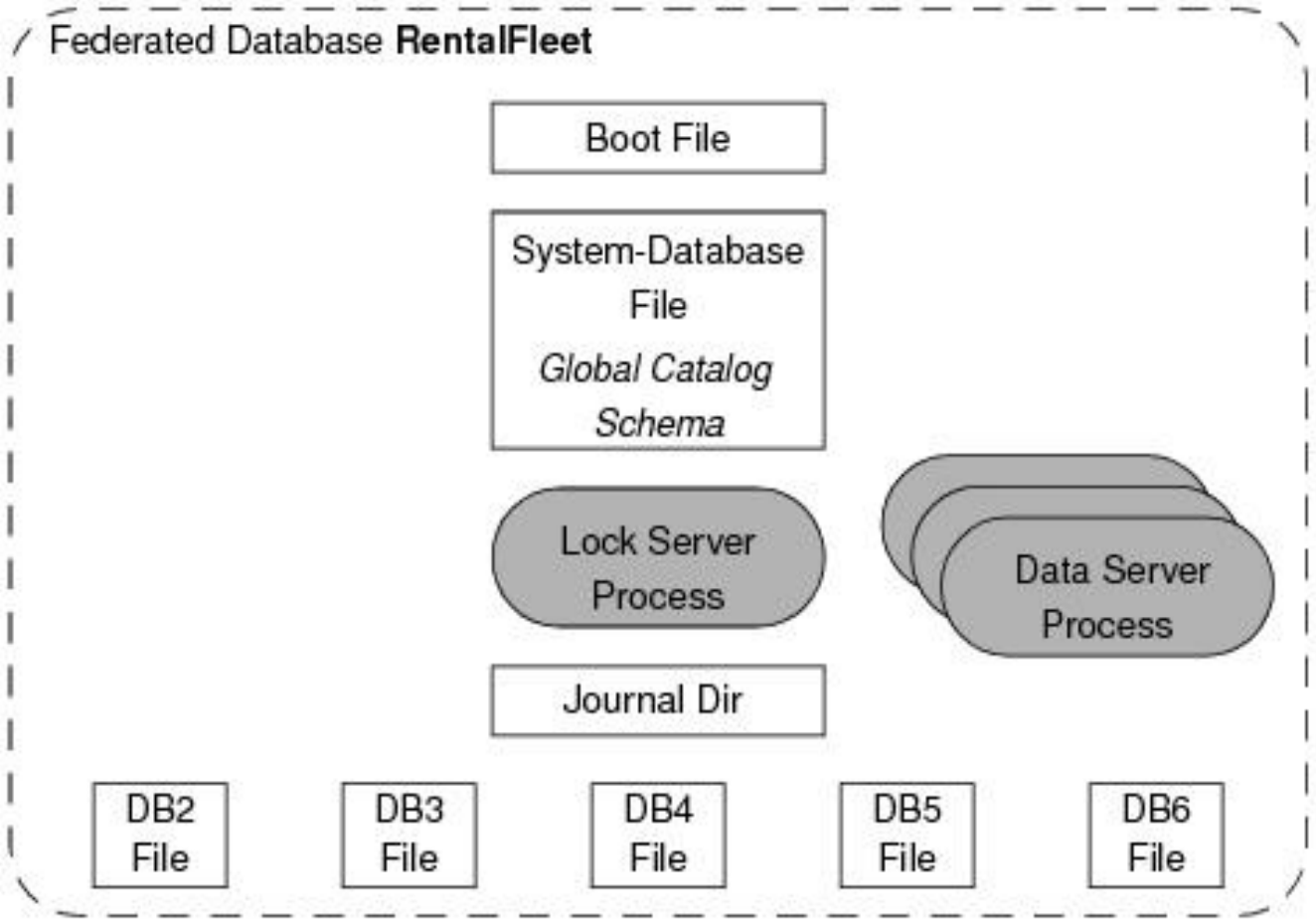
- **Data format:** Objectivity/DB stores persistent objects as they are defined by user application which can therefore have any number of attributes whose values are primitive data types, structures **whereas** a traditional DBMS stores data as tables of *records*—fixed-length sequences of primitive values.
- **Data organization:** In Objectivity/DB, objects are connected to each other by references and relationships **whereas** An RDBMS can store objects, but imposes its organization on them. That is, object attributes and relationships must be broken into rigid rows and columns for storage in tables.

Federated Database: Central concept of objectivity/DB

- With a traditional DBMS, you store related data in a single database. With Objectivity/DB, you typically distribute related data in many individual databases, which are members of a single *federated database* (also called a *federation*).



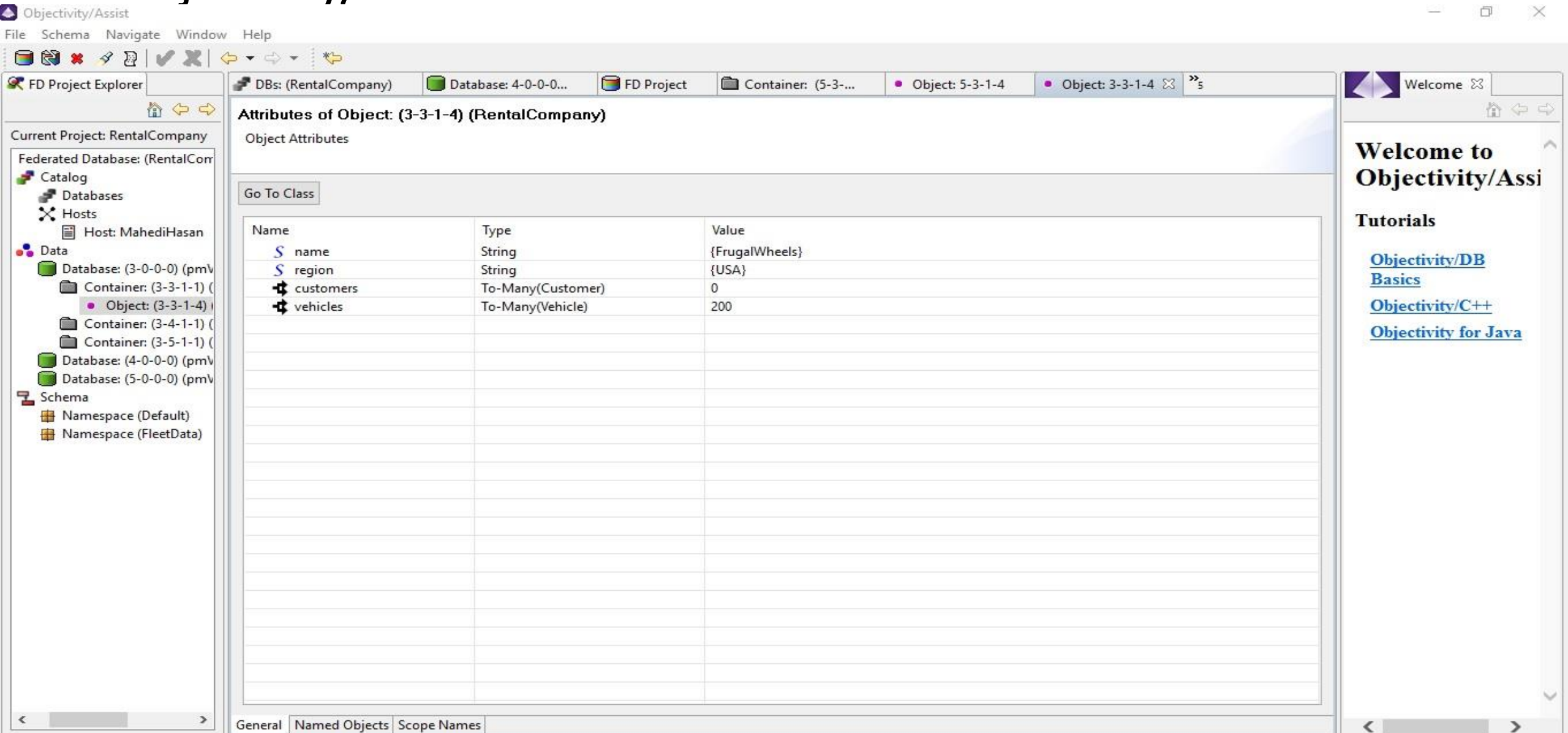
Federated Database: Architecture



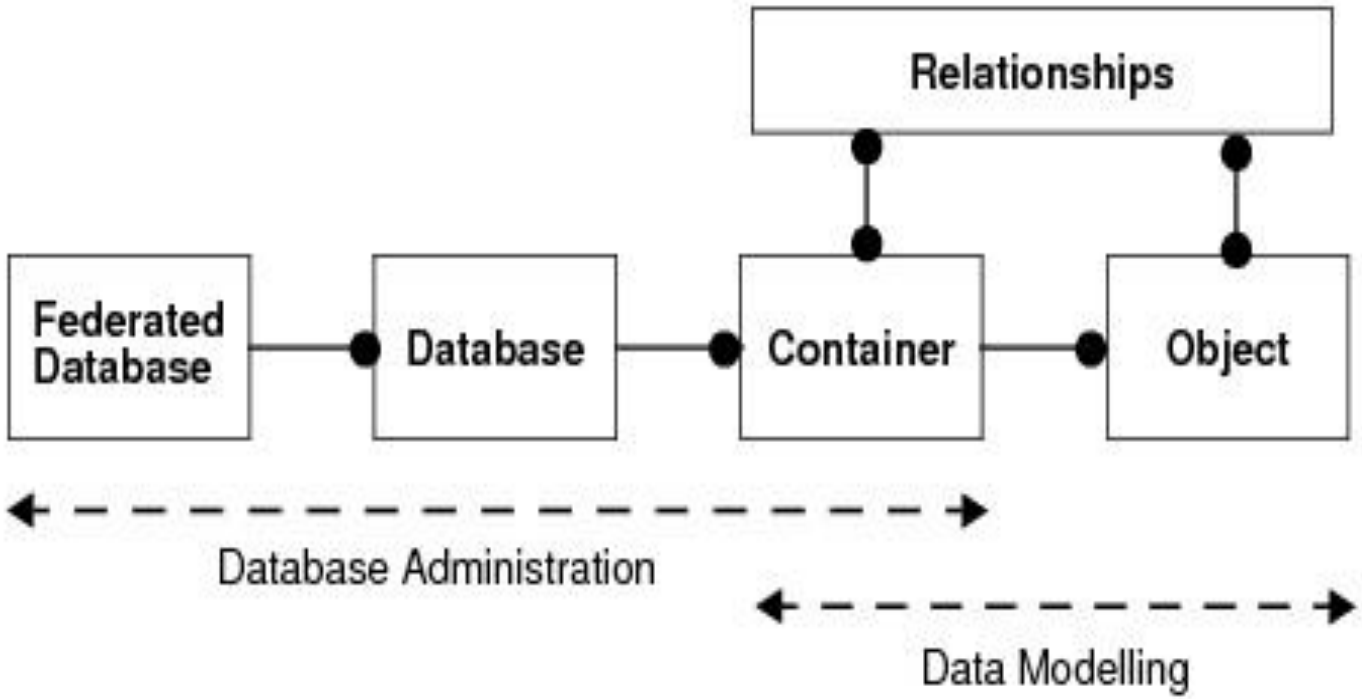
Federated Database: Architecture(contd.)

- A ***boot file*** gives applications the information they need to connect to the federated database. The boot file for the tutorial example is RentalFleet.boot.
- A ***system database*** maintains internal Objectivity/DB information such as the global catalog of databases. The system-database file for the tutorial example is RentalFleet.fdb
- A ***lock server*** is a process that manages concurrent access to a federated database's data
- ***Data servers*** enable applications to obtain data from distributed databases on local or remote hosts.
- A ***journal directory*** (by default, the directory containing the federated database) contains files that maintain information for recovering transactions.

Objectivity/Assist



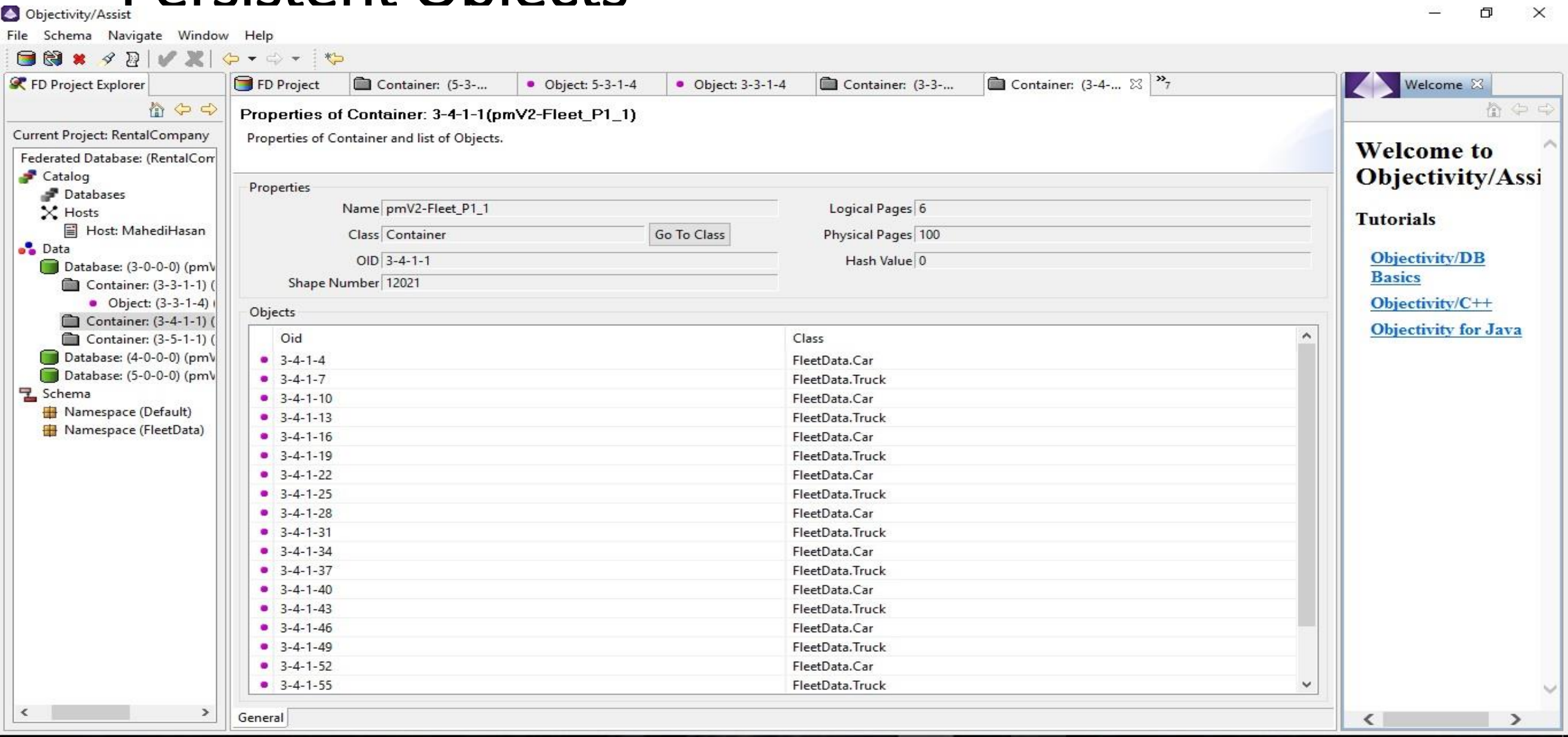
Storage Hierarchy:



Storage Hierarchy(contd.)

- A **federated database** contains one or more databases
- Each database contains one or more **containers**.
- Each container contains one or more **persistent objects**.
- Persistent objects, including **instances of application-defined** container classes, can be linked together using to-one or to-many relationships (associations).

Persistent Objects



Persistent Objects (contd.)

- Persistent objects are the fundamental logical units of data storage in an Objectivity/DB federated database.
- Persistent objects are created by applications. Each persistent object has an object identifier (OID) and is an instance of a class. **The class may be defined in an object-oriented programming language.**

Why the word “Persistence”?

- A persistent object continues to exist and retain its data beyond the duration of the process that creates it. In contrast, a *transient object* exists only within the memory of the process that creates it; when that process terminates, the transient object ceases to exist.

Advantages of OID

- Transparent access at runtime to objects located anywhere in the network.
- Full interoperability across all platforms.
- Access to more objects than a direct memory address permits.
- Ability to move objects to different physical locations without changing their identity.

Locking and Concurrency: a mechanism

- To prevent incompatible operations, Objectivity/DB uses a mechanism, called locks
- Let imagine a scenario where two sessions may read, and then subsequently update, an object. If both sessions perform these actions simultaneously, one of the updates would be overwritten by the other update. It is called incompatible operation
- Lock allows an application to inform Objectivity/DB how it plans to use an object. When an application requests a read lock, the application indicates to Objectivity/DB that it needs read-only access to an object. When an application requests a write lock, the application indicates that it intends to modify the object.
- **Containers are the fundamental unit which implements locking within Objectivity/DB**

Simple Example: Gets the name of the current federated database and tests whether a database resides in that federated database

```
import com.objy.db.DatabaseNotFoundException;
import com.objy.db.DatabaseOpenException;
import com.objy.db.ObjyRuntimeException;
import com.objy.db.app.Connection;
import com.objy.db.app.Session;
import com.objy.db.app.oo;
import com.objy.db.app.storage.ooDBObj;
import com.objy.db.app.ooFDObj;
import com.objy.db.app.storage.ooContObj;
public class example
{public static void main(String[] args){
Connection connection;
try{
connection =
Connection.open("C:/..../RentalCompany.boot",
oo.openReadOnly);
}catch (Exception e1){}
Session session;
session = new Session();
session.setOpenMode(oo.openReadOnly);
session.begin();
ooFDObj fd;
fd = session.getFD();
System.out.println(fd.getName());
System.out.println(fd.hasDB("pmV2-Region_G1_1"));
session.commit();}}
```