Nhibernate Advanced Mappings

리얼웹 개발본부

2011.06

Realweb

목차

NHibernate 소개

1. ORM (Object Relational Mapping) 소개
2. Architecture
3. Benefits

Introduce NHibernate

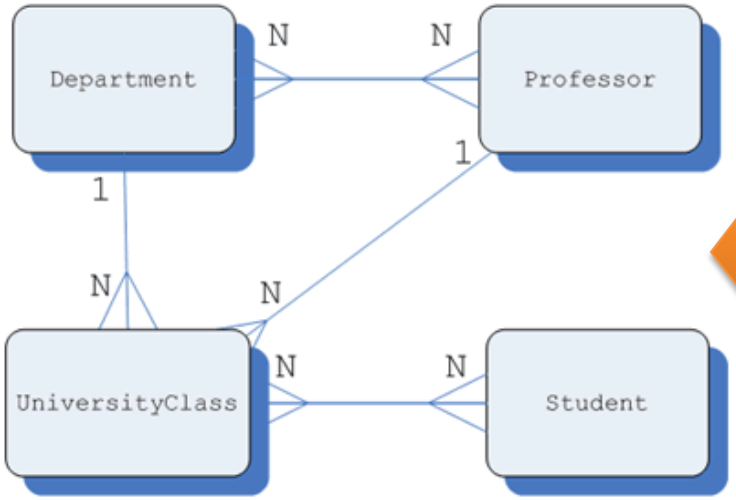
1. NHibernate

- 대표적인 ORM (Object relational mapping) Framework
- Java 기반의 Hibernate 를 .NET 용으로 Porting
- ORM 기본 기능에 가장 충실
- Free/Open Source (LGPL) 로 많은 시스템에 채택되어 안정성 검증
- 최신 버전 : version 3.1.0 GA
- Entity 및 IQuery에 대한 1st, 2nd Cache 지원
- 많은 Contributions 지원

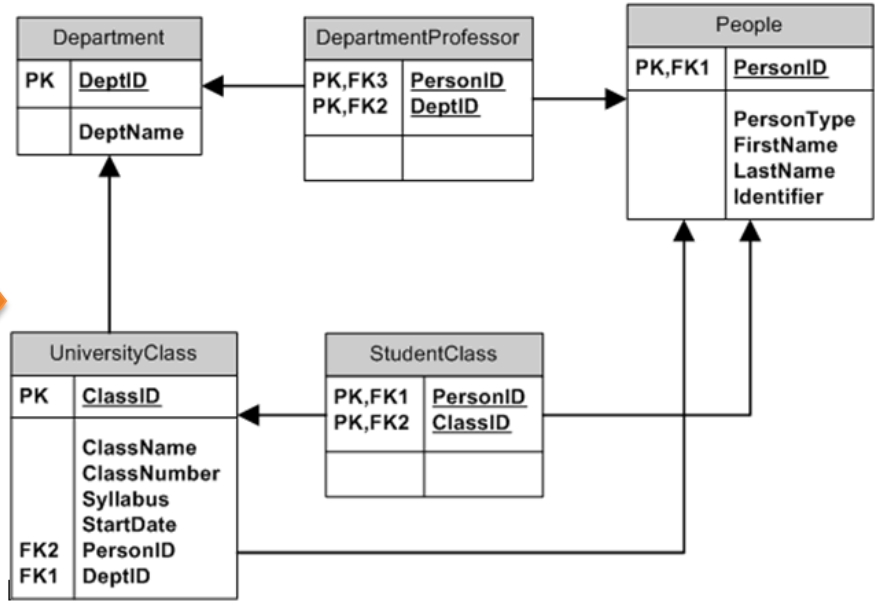
1.1 ORM 개념

Object Model in OOP

Data Model in RDBMS

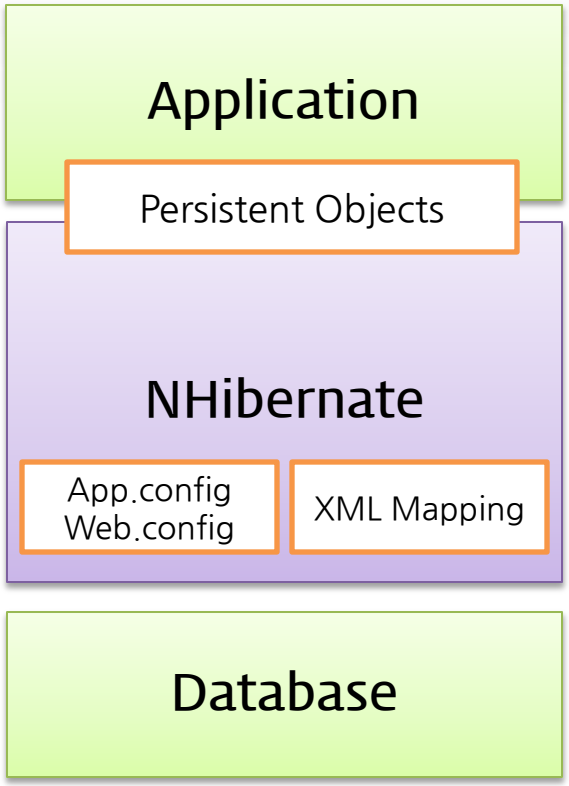


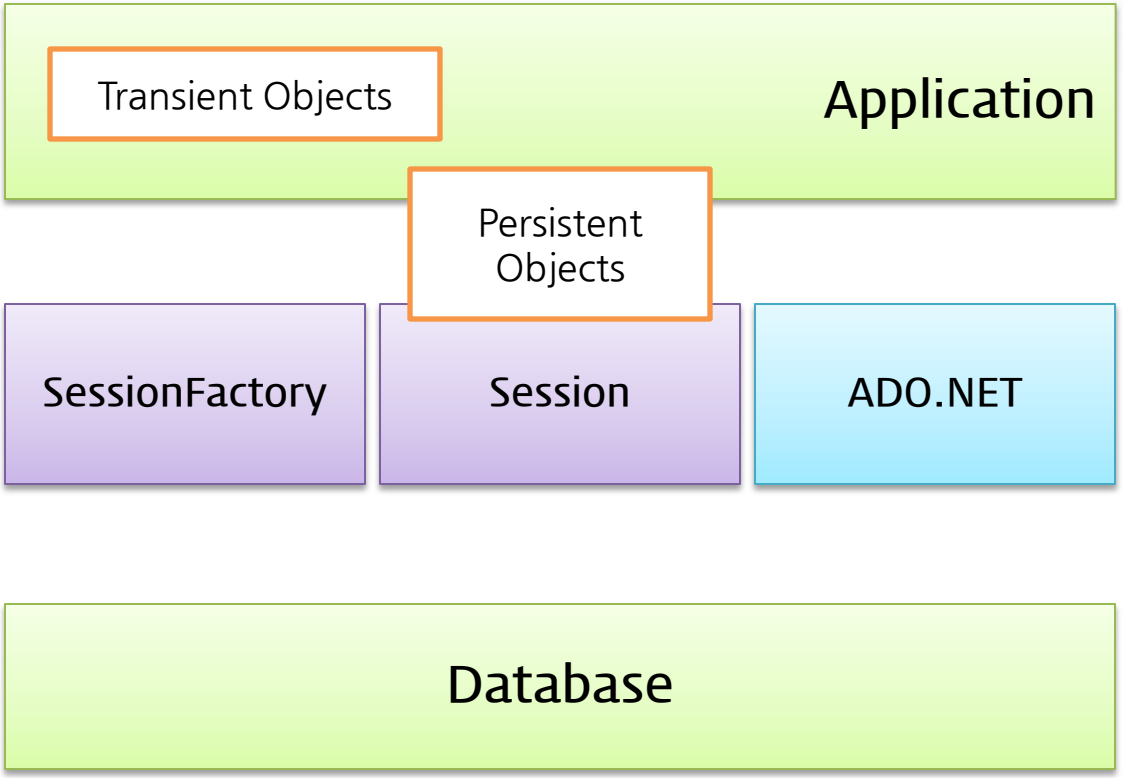
Mapping



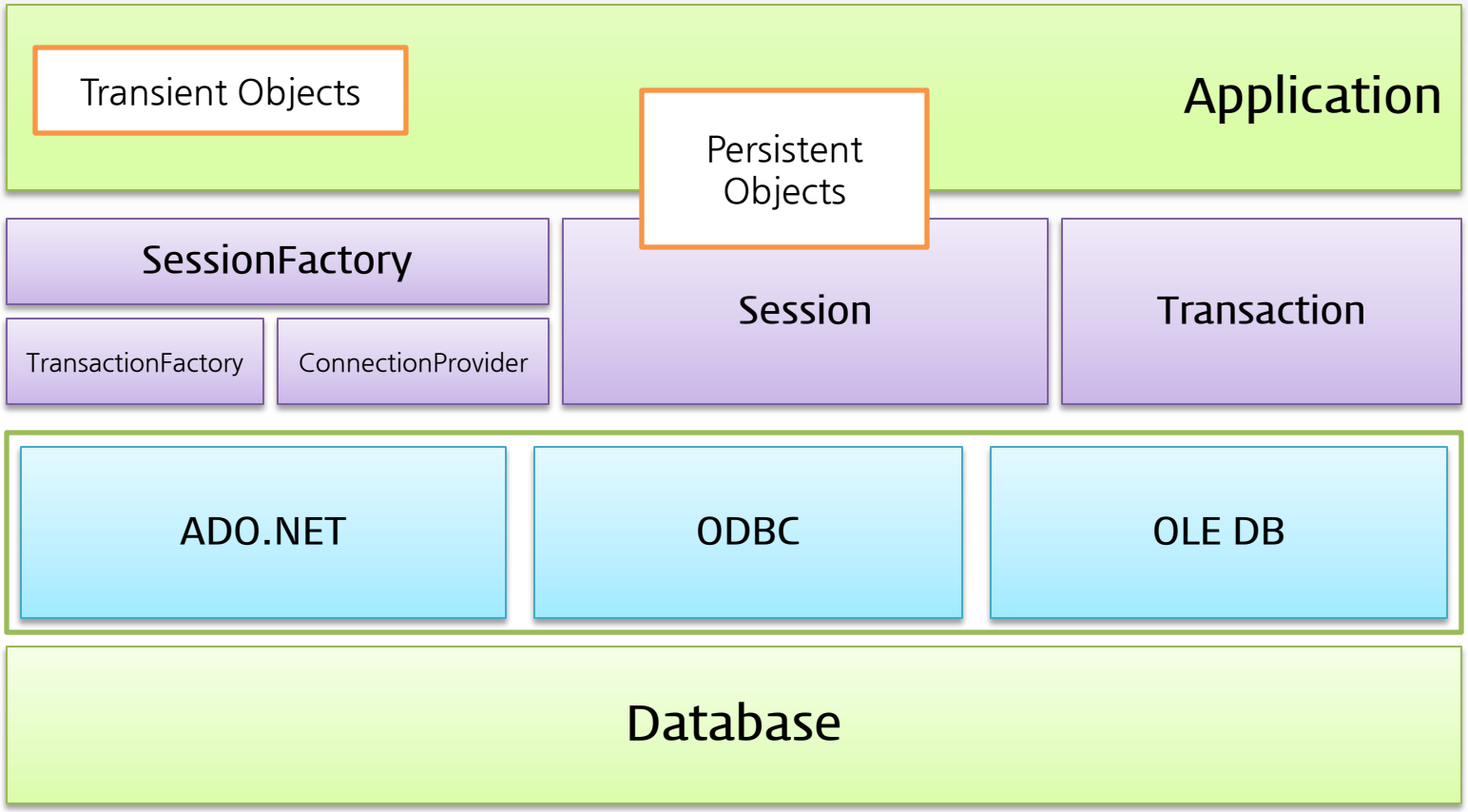
1.2 NHibernate Architecture

NHibernate는 ORM Framework으로서, Data 저장소 (RDBMS)와 object graph를 가진 Application 간의 매핑을 담당한다.
다양한 Query 기능 및 Persister를 지원하여, Application 개발자가 Data와 관련된 개발에 신경쓰지 않도록 도움을 준다.
Session은 NHibernate 의 중추적인 역할인 RDBMS와 Application의 매핑을 관장한다.





1.2 NHibernate Architecture



ISession (NHibernate.ISession)
A single-threaded, short-lived object representing a conversation between the application and the persistent store.
Wraps an ADO.NET connection. Factory for ITransaction. Holds a mandatory (first-level) cache of persistent objec ts, used when navigating the object graph or looking up objects by identifier.

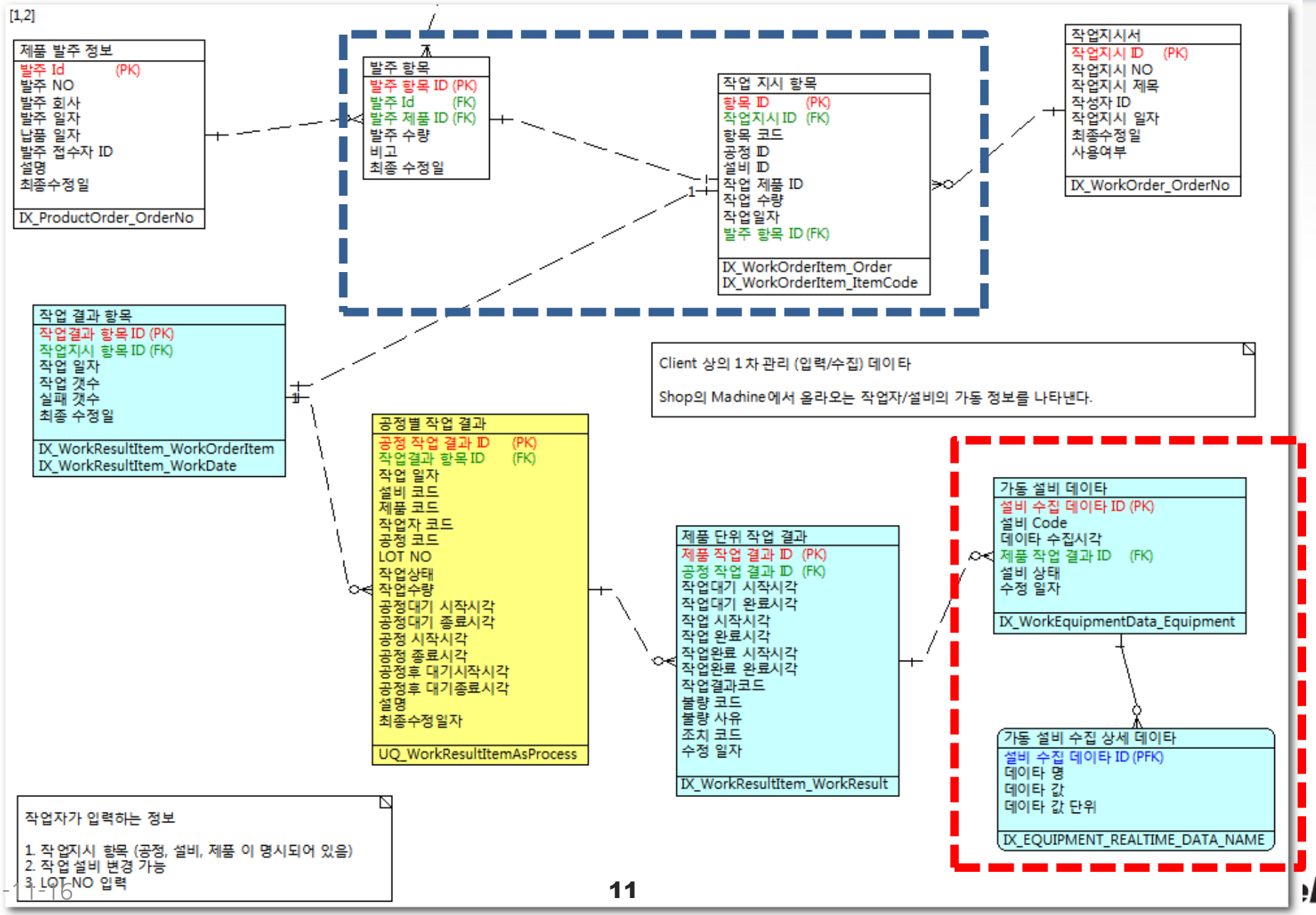
1.3 NHibernate - Benefits

- Enterprise 환경의 개발에서 ORM 도입에 따른 생산성 증가
  - Object = Table 이 아닌 relational mapping 을 자동 지원하므로, data 조작의 수작업이 없음.
  - Data 처리 작업용 개발 공수를 95% 까지 감소시키는 것이 목표
- Enterprise Application을 Data-Centric 이 아닌 Business Logic 에 중점을 두는 CBD 개발에 기여
- 다양한 RDBMS Vendor를 지원 (DB에 종속적이지 않다.)
- ICriteria, HQL (Hibernate Query Language), Native Query, LINQ 지원
- Projections, aggregation, group, subqueries 지원
- 다양한 Fetching strategy 지원
- First, Second Cache 기본 지원 (MemCached, SharedCache, Velocity)

1. Collection Mappings
    1. one-to-many
    2. many-to-many
    3. one-to-one
    4. many-to-any (not recommended)
2. Inheritance Mappings
    1. Table per concrete class (union-subclass)
    2. Table per class hierarchy (subclass & discriminator)
    3. Table per subclass (joined-subclass)

2. ADVANCED MAPPINGS

2.1 Collection Mappings



2.1 Collection Mappings

Mapping	.NET Language	비고
bag	IList<T>	중복 가능
set	ISet<T>	종복 불가능
list	IList<T>	정렬 인덱스 존재 (index 처리 부담)
idbag	IList<T>	Update 시 장점. Index 처리 부담
map	IDictionary<TKey,TValue>	Key 수형이 다른 수형일 경우

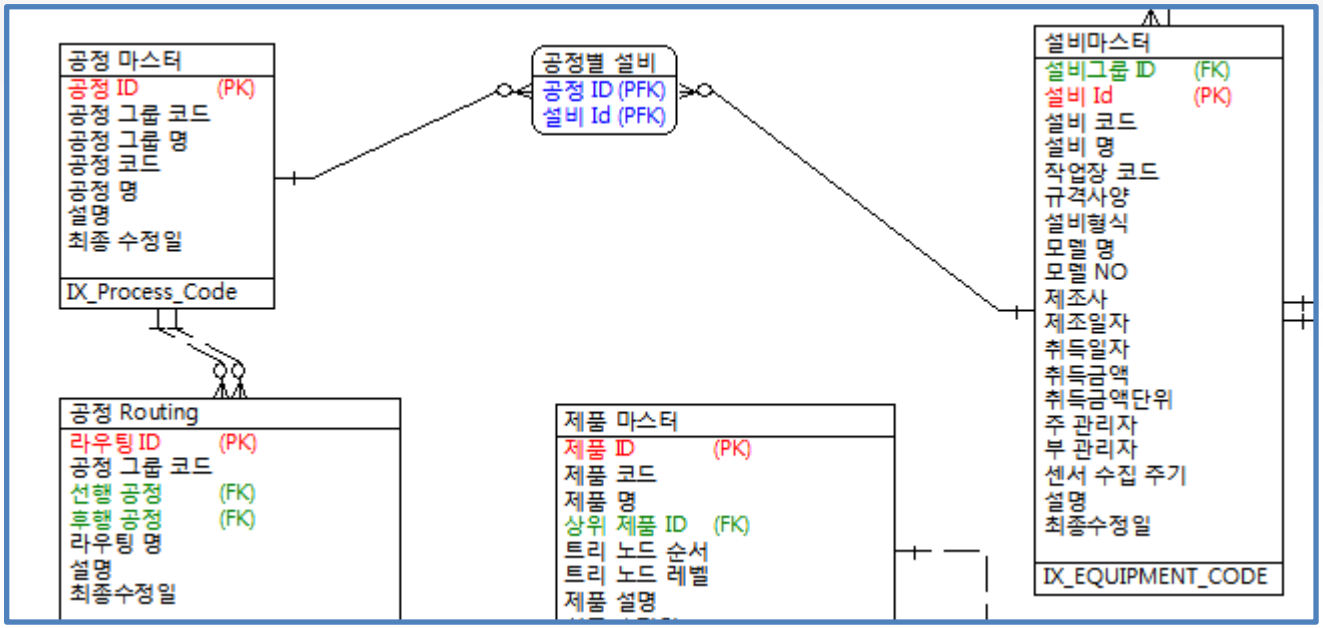
2.1.1 one-to-many

Code
CODE ID (PK)
CODE NAME
CODE TITLE
Product Id (FK)
EnterpriseId (FK
LANGUAGE TYPE
IS SYS DEFINED

```
<class name="CodeItem" table="RAT_CODE_ITEM" dynamic-insert="true" dynamic-update="true">

    <cache usage="read-write" include="all"/>

    <id name="Id" column="ITEM_ID" type="Guid" unsaved-value="none">
        <generator class="guid.comb"/>
    </id>

    <many-to-one name="Code" class="Code" column="CODE_ID" lazy="false" not-null="true" index="IX_CODE_ITEM_NAME_CODE" />

    <property name="Name" column="ITEM_NAME" type="AnsiString" not-null="true" index="IX_CODE_ITEM_NAME_CODE"/>
    <property name="Value" column="ITEM_VALUE" type="AnsiString"/>
```

```
<class name="C

    <cache usa

    <id name="Id" column="CODE_ID" type="Guid" unsaved-value="none">
        <generator class="guid.comb"/>
    </id>

    <many-to-one name="Application" class="Application" column="APPLICATION_ID"
                 lazy="proxy" fetch="select" unique-key="AK_CODE_APPLICATION_CODE_NAME"/>

    <many-to-one name="Enterprise" class="Enterprise" column="ENTERPRISE_ID"
                 lazy="proxy" fetch="select"/>


    <property name="Name" column="CODE_NAME" type="AnsiString" not-null="true" unique-key="AK_CODE_APPLICATION_CODE_NAME"/>
    <property name="LanguageType" column="LANGUAGE_TYPE" type="String"/>

    <property name="IsEnabled" column="IS_ENABLED" type="Boolean"/>
    <property name="IsSysDefined" column="IS_SYS_DEFINED" type="Boolean"/>

    <property name="Title" column="CODE_TITLE" type="String"/>
    <property name="Description" column="Description" type="String"/>

    <!-- Items -->
    <set name="CodeItems" access="field.camelcase-underscore" inverse="true" lazy="false" cascade="all-delete-orphan" fetch="subselect">
        <key column="CODE_ID" />
        <one-to-many class="CodeItem"/>
    </set>
```

2.1.2 many-to-many



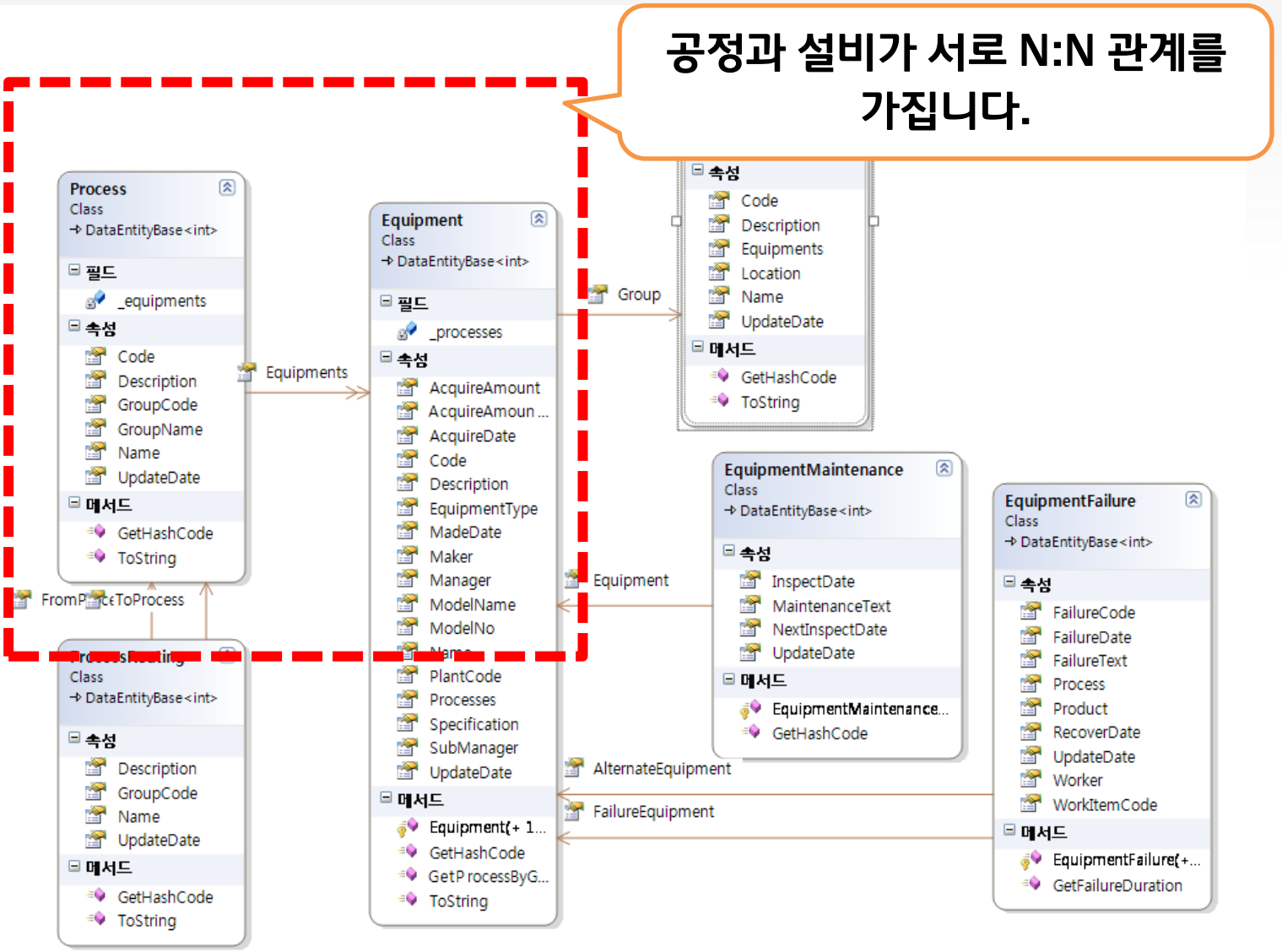
Process

```
<!-- 해당 공정에서 사용할 수 있는 설비들 -->
<set name="Equipments" table="ProcessEquipmentLink" access="field.camelcase-underscore" inverse="false" lazy="true" cascade="save-update">
    <key column="ProcessId" foreign-key="FK_ProcessEquipmentLink_Process" />
    <many-to-many class="Equipment" column="EquipmentId" fetch="select" lazy="proxy" foreign-key="FK_ProcessEquipmentLink_Equipment" />
</set>
```

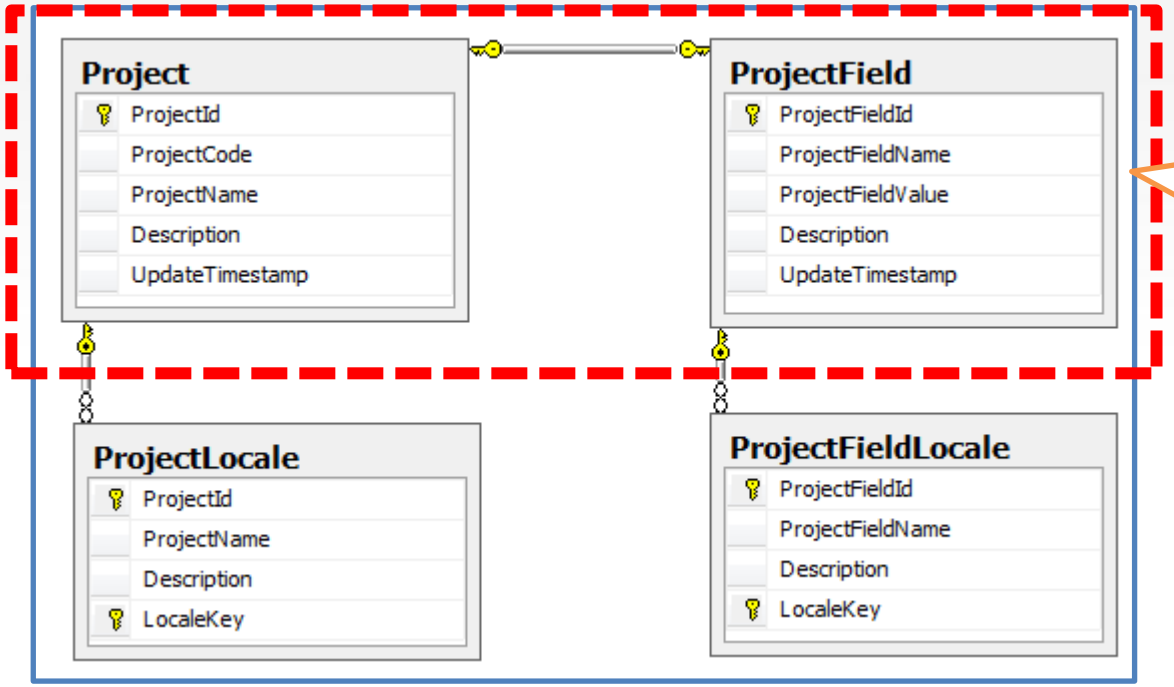
Equipment

```
<!-- 설비가 사용되는 공정들-->
<set name="Processes" table="ProcessEquipmentLink" access="field.camelcase-underscore" inverse="true" lazy="true" fetch="select" cascade="save-
    <key column="EquipmentId" not-null="true" />
    <many-to-many class="Process" column="ProcessId" lazy="proxy" fetch="select" />
</set>
```

2.1.2 many-to-many



2.1.3 one-to-one (join)



Project와 ProjectField는
1:1 관계를 가집니다.

```
<class name="ProjectField" table="ProjectField" dynamic-insert="true" dynamic-update="true">

    <id name="Id" column="ProjectFieldId">
        <generator class="foreign">
            <param name="property">Project</param>
        </generator>
    </id>

    <one-to-one name="Project" class="Project" constrained="true" foreign-key="FK_ProjectField_Project" />

    <property name="Name" column="ProjectFieldName" type="String" />
    <property name="Value" column="ProjectFieldValue" type="String" length="9999" />
    <property name="Description" type="String" length="9999" />
```

2.2 Mapping class inheritance

- OOP 상속을 RDBMS 에서 어떻게 매핑 시킬 것인가?
- 3가지 대표적 방안

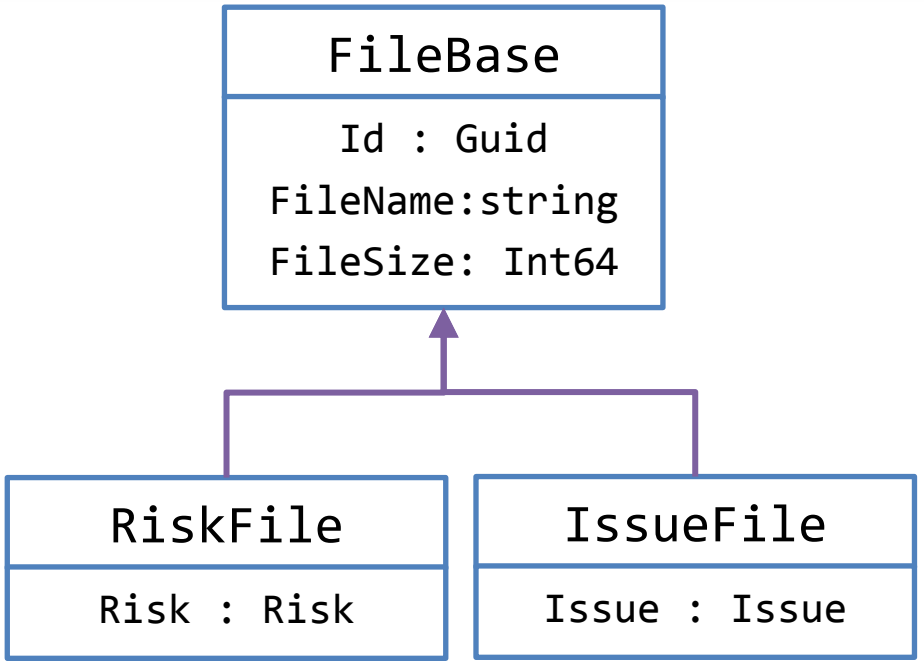
- Table Per Concrete class
  - 모든 엔티티 클래스 별로 TABLE을 1:1로 매핑 (union-subclass)

- Table Per Class Hierarchy
  - 모든 클래스들을 한 테이블에 모두 넣기 (subclass & discriminator)

- Table Per SubClass
  - 클래스 상속 구조와 유사하게 각 클래스 별로 TABLE 만들기 (joined-subclass)

2.2.1 Table Per Concrete class (union-subclass)

모든 Concrete Class(엔티티 클래스) 별로 TABLE이 생성



RISK_FILE

FILE_ID: Guid
FILE_NAME: varchar
FILE_SIZE: Bigint
RISK_ID: Int

ISSUE_FILE

FILE_ID:Guid
FILE_NAME:varchar
FILE_SIZE:Bigint
ISSUE_ID: Int

2.2.1 Table Per Concrete class (union-subclass)

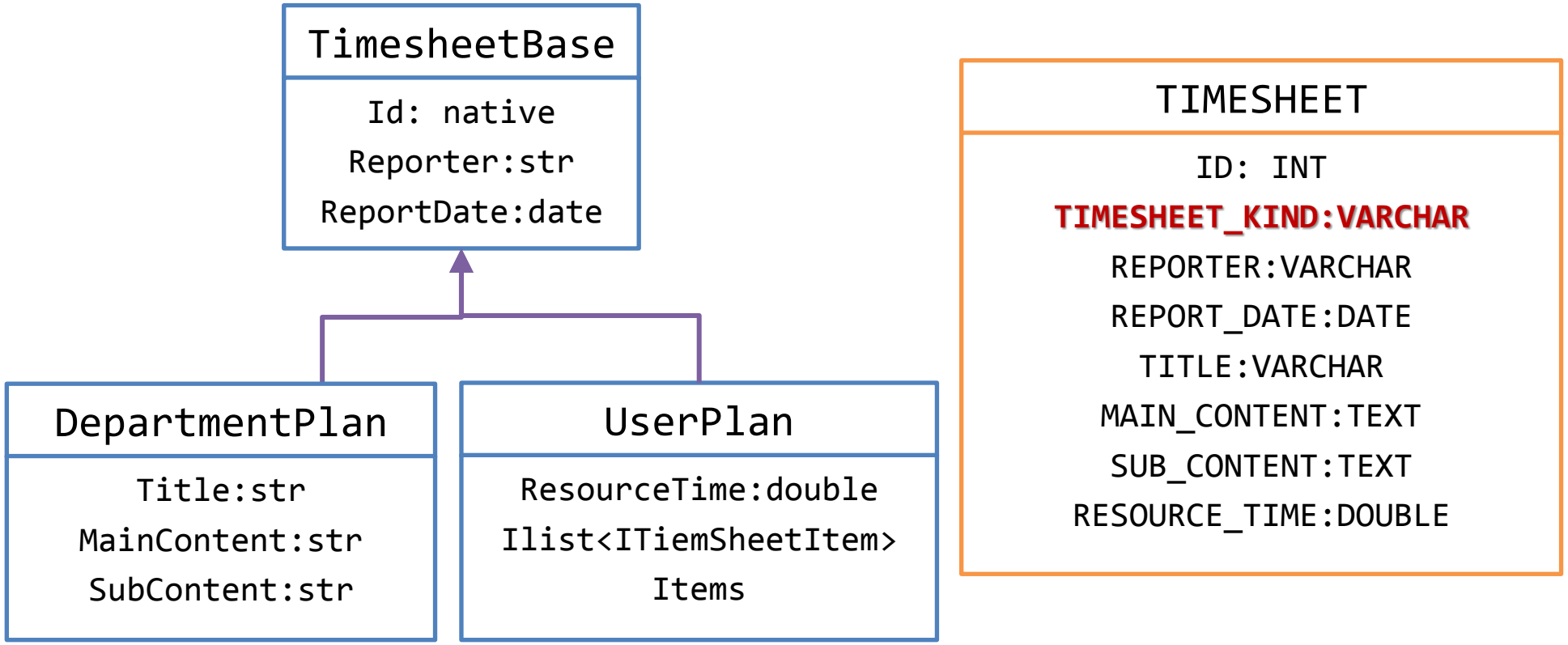
```
<!-- 일반적인 파일 정보에 대한 추상화 클래스 -->
<class name="FileBase" abstract="true" dynamic-insert="true" dynamic-update="true" >

    <id name="Id" column="FileId" type="Guid" unsaved-value="none">
        <generator class="assigned" />
    </id>

    <property name="FileName" type="String"  length="200" not-null="true" />
    <property name="FilePhysicalName" type="String"  length="1024" />
    <property name="FilePath" type="String"  length="2000" />
    <property name="FileExtention" type="String"  length="50" />
    <property name="FileMimeType" type="String"  length="200" />
    <property name="FileSize" type="Int64" />

    <property name="LastUpdateUserId" type="AnsiString" length="50" />
    <property name="LastUpdateDate" type="Timestamp" />

</class>
```

```
<union-subclass name="RiskFile" extends="FileBase" table="RiskFile" dynamic-insert="true" dynamic-update="true" >

    <many-to-one name="Risk" class="Risk" column="RiskId" lazy="proxy" fetch="select" not-null="true"
                 foreign-key="FK_RiskFile_Risk" index="IX_RiskFile_Risk" />

</union-subclass>
```

```
<union-subclass name="IssueFile" table="IssueFile" extends="FileBase" dynamic-insert="true" dynamic-update="true">

    <many-to-one name="Issue" class="Issue" column="IssueId"
                 lazy="proxy" fetch="select" foreign-key="FK_IssueFile_Issue" index="IX_IssueFile_Issue" />

</union-subclass>
```

2.2.2 Table Per Class Hierarchy (subclass)

모든 클래스의 정보가 한 테이블에 저장됩니다.
각 클래스의 구분값을 discriminator 값으로 구분하게 됩니다.
DepartmentPlan 클래스는 TIMESHEET_KIND 값이 “DepartmentPlan” 이고,
UserPlan 클래스는 “UserPlan” 값이 저장됩니다.

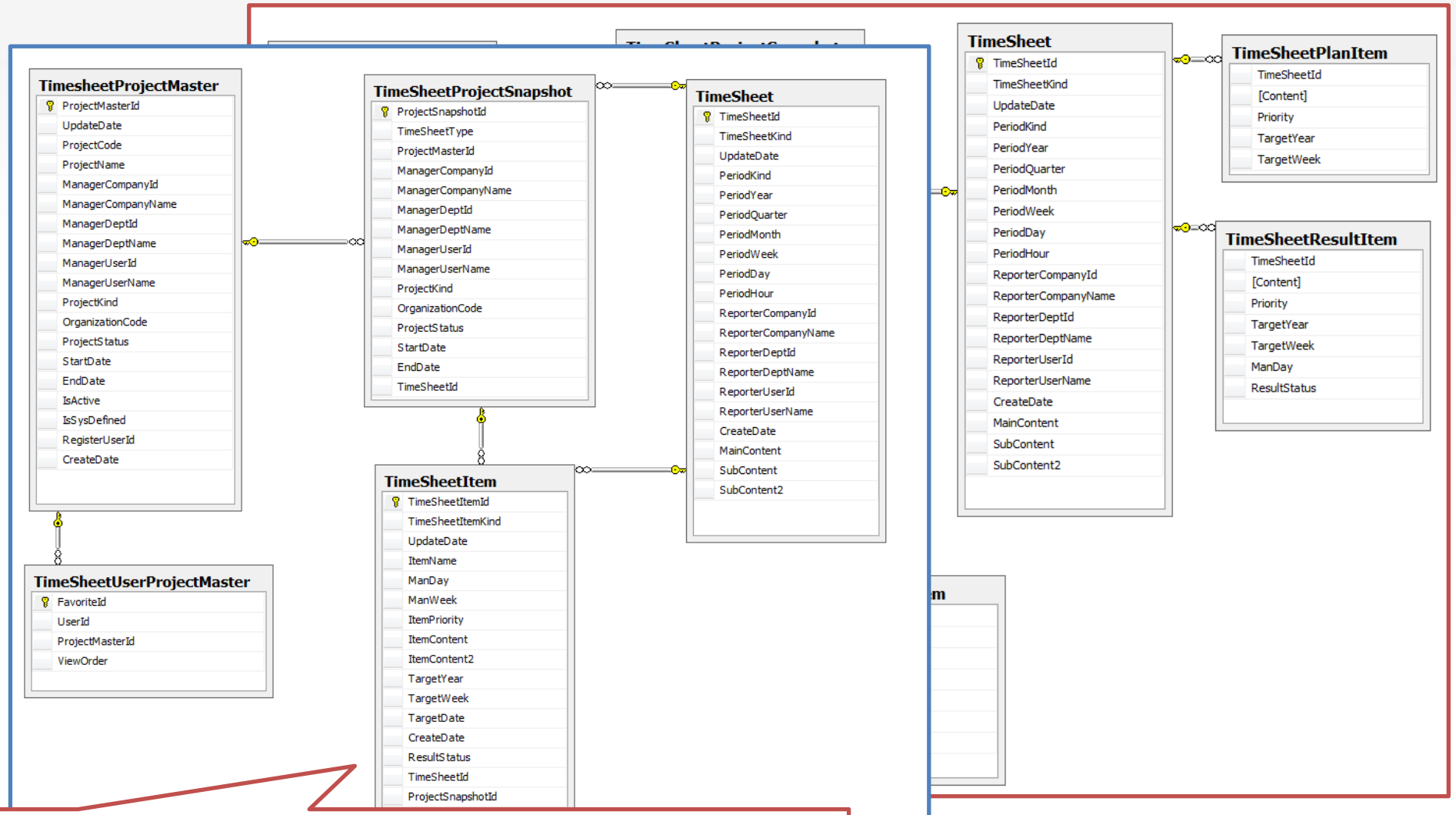


2.2.2 Table Per Class Hierarchy (subclass)

```
<!-- TimeSheet 보고 자료 -->
<class name="TimesheetBase" table="Timesheet" proxy="ITimesheet" abstract="true" discriminator-value="TimesheetBase" dynamic-insert="true" dyna

    <id name="Id" column="TimesheetId" type="Int32" unsaved-value="0">
        <generator class="native"/>
    </id>

    <discriminator column="TimesheetKind" type="AnsiString" length="64" not-null="true" />

    <version name="UpdateDate" column="UpdateDate" type="DateTime" insert="false"/>

    <!-- 보고 시점을 나타낸다. -->
    <component name="ReportTime" class="PeriodTimeValue" lazy="false" access="field.camelcase-underscore">

    <!-- 보고자의 보고시의 정보를 저장한다. -->
    <component name="Reporter" class="UserValue" access="field.camelcase-underscore">

    <!-- 작성완료 여부 (0:임시저장, 1: 작성완료)-->
    <property name="WriteStatus" column="WriteStatus" type="Int32" />

    <property name="CreateDate" column="CreateDate" type="DateTime" update="false"/>

</class>

<!-- 부서 계획 -->
<subclass name="DepartmentPlan" extends="TimesheetBase" discriminator-value="DepartmentPlan"  dynamic-insert="true" dynamic-update="true">
    <property name="MainContent" column="MainContent" type="String" length="9999"/>
    <property name="SubContent" column="SubContent" type="String" length="9999"/>
    <property name="SubContent2" column="SubContent2" type="String" length="9999"/>
</subclass>
<!-- 부서 보고 -->
<subclass name="DepartmentResult" e
    <property name="MainContent" co
    <property name="SubContent" col
    <property name="SubContent2" co
</subclass>
```

```
<!-- 주기별 사용자 계획 정보-->
<subclass name="UserPlan" extends="TimesheetBase" discriminator-value="UserPlan"  dynamic-insert="true

    <!-- 프로젝트의 Activity별로 계획/보고 내용을 가진다.-->
    <set name="Items" access="field.camelcase-underscore" inverse="true" lazy="false" cascade="all">
        <key column="TimeSheetId" />
        <one-to-many class="TimesheetPlanItem"/>
    </set>
</subclass>
```

2.2.2 Table Per Class Hierarchy (subclass)



SubClass 적용 후 기존 TimeSheetPlanItem과
TimeSheetResultItem을 TimeSheetItem으로 통합

Table Per SubClass (joined-subclass)

모든 클래스 (Abstract Class 포함)해서 Class : Table을 1:1 대응이 됩니다.
단 실제 사용될 Employee 클래스는 PERSON 테이블과 EMPLOYEE 테이블의 JOIN 을 수행하여야 합니다.

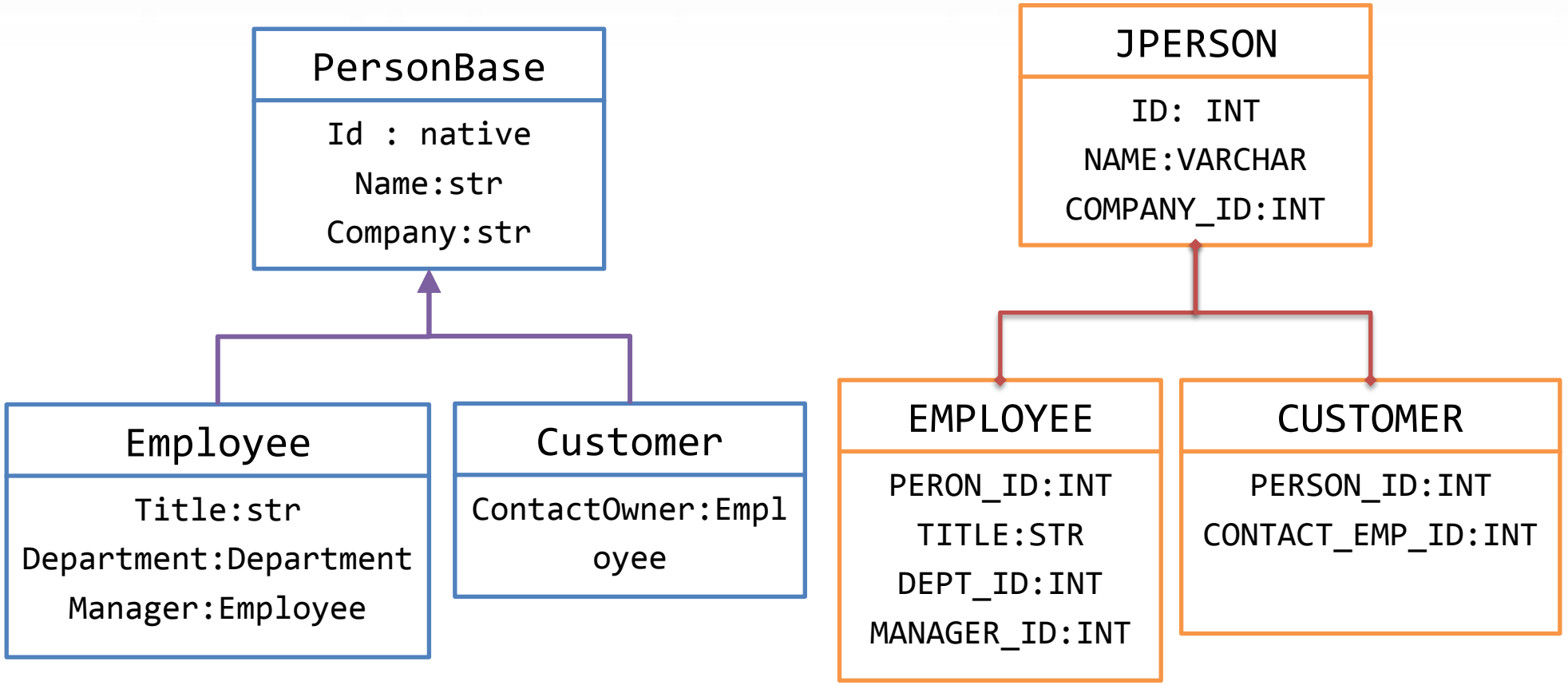


Table Per SubClass (joined-subclass)

```
<class name="Person" table="JPerson">

    <id name="Id" column="person_id">
        <generator class="native"/>
    </id>

    <property name="Name"/>
    <property name="Company"/>
    <property name="Region"/>

    <joined-subclass name="Employee">
        <key column="person_id" />
        <property name="Title"/>
        <property name="Department" column="dept"/>
        <many-to-one name="Manager" column="mgr_id" class="Employee" cascade="none"/>
        <set name="Minions" inverse="true" lazy="true" cascade="all">
            <key column="mgr_id"/>
            <one-to-many class="Employee"/>
        </set>
    </joined-subclass>

    <joined-subclass name="Customer">
        <key column="person_id" />
        <many-to-one name="ContactOwner" class="Employee"/>
    </joined-subclass>

    <filter name="region" condition="Region = :userRegion"/>

</class>
```

1. Resources

3. APPENDIX

Appendix – Resources

- Official Sites
  - http://www.hibernate.org
  - NHibernate Resources
  - http://nhforge.org
  - NHibernate Reference Documentation
- Articles
  - NHibernate Best Practices with ASP.NET
  - NHibernate in ServerSide.NET
- Books
  - NHibernate in Action
  - LINQ in Action
  - Patterns of Enterprise Application Architecture by Martin Fowler
- Other Resources
  - http://www.castleproject.org
  - Inversion of Control and Dependency Injection with Castle Windsor Container
  - Inversion of Control and Dependency Injection: Working with Windsor Container
  - Castle Windsor Configuration Samples

감사합니다

Realweb