EGE ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

SUNUCU YAZILIM TEKNOLOJİLERİ PROJESİ

Enterprise Java Beans ve Java Persistence API

Grup Üyeleri

İlker Semih Boztepe
Osman Korcan Andaç
Deniz Sezer
Resul Çetin
Aygül Acar
Cem Yücel
Mustafa Gamsız
Hakan Kırkgül
Onur Keskin
Ali Yüncü
Rıdvan Çelebioğlu

18 Mayıs 2010

İZMİR

İÇİNDEKİLER

▪ Enterprise Java Beans (EJB)

Enterprise JavaBeans (EJB) veya Türkçe karşılığı ile işe yönelik yeniden kullanılabilir Java elemanları iş uygulamalarının modüler bir şekilde kurulması için tasarlanmış yönetilebilir *sunucu-yanı* (server-side) eleman mimarisidir.

EJB Java Enterprise Edition içinde yer alan çok sayıdaki Java API (*Java uygulama programlama arayüzü*) tanımlamalarından bir tanesidir. EJB sunucu-yanı bir model olup uygulamaların iş mantığını içerir.

EJB tanımlamasının ana amacı, genelde iş uygulamalarında bulunan arka plandaki (back-end) kodun yerine getirilmesi için standart bir yol sağlamaktır. Bu kodlar yazılımda aynı tür problemin sıklıkla meydana geldiği ve programcılar tarafından çözümlerinin tekrarlanarak yerine getirildiği kısımları teşkil eder. Bu sebeple, Enterprise Java Beans devamlılık, işlemsel bütünlük ve güvenlik gibi yaygın konuları standart yollarla desteklemek ve programcılara belli bir problemi inceleme serbestliği vermek üzere tasarlanmıştır.

Bu mimari kullanılarak geliştirilmiş yazılımlar; ölçeklenebilir, birim işlem yeteneğine sahip, güvenli ve dayanıklı olup kurumların önemli gereksinimlerini karşılayacak nitelikte, çok kullanıcılı uygulamalardır. Bu gereksinimler EJB çalışma zamanı ortamında EJB kabı (container) servisleri tarafından karşılanmaktadır. Yani eşanlı veri erişimi ve güvenlik desteği otomatik olarak EJB kabı tarafından tüm kurumsal çekirdekler(beans) için sağlanmaktadır. EJB ile oluşturulan uygulamalar, bir kez yazıldıktan sonra EJB belirtimlerine uygun herhangi bir sunucu ile kullanılabilir.

Günümüzde EJB 3.0 kullanılmaktadır. EJB 3.0 teknolojisinin misyonu karmaşık EJB gereksinimlerini çıkarmak ve mümkün olduğunca eski basit Java nesnelerine (*Plain Old Java Objects*) dönerek uygulama geliştirebilmektir. Bu teknolojide, kaptan ziyade çekirdeğin tasarımına odaklanılmış ve kabın bugünkü görünümü değiştirilmiştir. Sonuçta çekirdek ve kabın rolleri yeniden tasarlanmıştır.

Bileşen (Component) Model

Bileşen model tüm sunucu-taraflı nesnelerin uyması gereken bir standarttır. Tanımı genişletmek gerekirse, uygulama sunucularıyla uygulama geliştiriciler arasındaki yapılması gereken işleri kesin çizgilerle ayıran bir belirtimdir (specification). Aslında buna bir çeşit kontrat da denilebilir. Çünkü hem uygulama sunucusu satıcısı, hem de EJB geliştirici bu belirtime uyacaklarını garanti etmişlerdir. İki taraftan birinin buna uymaması durumunda geliştirilen sistem ya hiç çalışmaz, ya da sorun çıkartır. Peki uygulama geliştiriciler ile uygulama sunucularının görevleri nelerdir? EJB belirtimine göre uygulama sunucuları bazı düşük seviyeli servisleri sunucu tarafta çalışan nesnelere sağlamalıdır. Bunların bazılarını örnek olarak vermek gerekirse:

-İsimlendirme (naming)
-Dağıtık nesne teknolojileri (RMI,RMI-IIOP,CORBA vs.)
-Süreklilik (persistence)
-Nesne havuzu (instance pooling)

-Hareket (transaction)
-Eş zamanlı kullanım (concurrency)
-Güvenlik (security)

Yukarıda belirtilen servisler her zaman yazılımcıların başını ağrıtmıştır. Çünkü bu servislerin gerçekleştirimi (implementation) gerçekten zordur. Uygulama sunucularının ortaya çıkışının temel nedeni de burada yatmaktadır: Yazılımcıyı alt seviyedeki servislerle uğraşmaktan kurtarmak ve tamamen iş mantığına (business logic) yoğunlaştırmak. Burada tanımını yaptığımız bileşen model kavramının birçok avantajları vardır. Bunlar sistem geliştirme hayat devresinin kısalması, EJB bileşenlerinin uygulama sunucusundan bağımsız olması ve bileşen model çeşitliliğidir.

EJB'nin İskeleti

EJB'nin öncelikle bilinmesi gereken özelliği dağıtık nesne teknolojilerini kullanmasıdır ve şu anda J2EE platformunun standart olarak kullandığı dağıtık nesne protokolü IIOP'dur. Ancak herhangi bir J2EE platformu (uygulama sunucusu da denilebilir) satıcısı kendi tescilli protokolünü bunun yanında kullanabilir. Bilindiği gibi dağıtık nesneler istemci tarafındaki bir vekil (stub) vasıtasıyla sunucu tarafındaki bir nesneye ulaşır. Tipik bir EJB bileşeni en azından şu üç sınıfı içerir:

1- Home Interface
2- Remote Interface
3- Bean Class

1) Home Interface

Bu "interface" bir EJB'nin yaşamsal döngüsünü belirleyen bir arayüzdür. Burada iki çeşit metod tanımlanabilir:

-create (yaratma) metodları
-finder (arama) metodları

Şunu baştan söylemek gerekir ki bir istemci bir *bean*'e her zaman uzaktan (remote) erişir. Bu kritere dayanarak *home interface*'i en basit ifadeyle şöyle tanımlayabiliriz. Sunucu tarafında var olan bir *bean* sınıfına erişimi sağlayan *remote interface*'i yaratmaya ya da bulmaya yarar. Aslında karmaşık gibi görünen bu ifade *home interface*'in görevlerinin büyük bir bölümünü oluşturur. Bu metotların özellikleri bileşenden bileşene (entity, stateful session, stateless session) değişir. Örneğin *finder* metotlar *session bean*'lerde bulunmazlar , fakat *entity bean*'lerde en az bir tane (findByPrimaryKey) bulunmak zorundadır.

2) Remote Interface

Bu arayüz *bean* sınıfında tanımlanan tüm iş mantığı metotlarının aynısını içinde barındırır. Bu sayede istemci *bean* sınıfına direk olarak ulaşmadan bu arayüz vasıtasıyla *bean* sınıfının metotlarını çağırabilir. Buradan da anlaşılabildiği gibi bu arayüz bir kontrol mekanizması (controller) sağlar.

Böylece “container” tarafından oluşturulan bu arayüzün implementasyon sınıfında çeşitli servisler sağlanabilir. Bu servisler böylece *bean* geliştiriciden bağımsız olarak gerçekleştirilmiş olur.

3) Bean Class

İş mantığının asıl gömüldüğü sınıftır. Bu sınıf sunucu tarafında bulunur ve sadece *remote interface* aracılığıyla erişilebilir. Üç çeşittir.

1- Entity Bean
2- Session Bean
3- Message-Driven Bean

Entity Bean: Herhangi bir veri kaynağındaki veriyi temsil eden bean’lerdir. Örneğin ilişkisel bir veritabanında bulunan bir tablonun bir satırı bir *entity bean instance*’ına denk gelir. JPA ile birlikte kullanımında önemli gelişmeler kaydedilmiştir.

Session Bean: Uygulamadaki iş akışını kontrol eden bileşenlerdir. *Entity bean*’in veriyi objeleştirdiğini yukarıda belirtmiştik. İşte bir uygulamanın ortaya çıkması için bu verinin bir şekilde işlenmesi gerekir. Çünkü bir uygulamanın çalışması demek sistemde bulunan verilerin anlamlı şekilde değişmesi demektir. Bu görevi *session bean* gerçekleştirir. *Session bean, entity bean*’leri kullanarak ve işleyerek, örneğin, bir ticari işletmenin iş akış süreçlerini gerçekleştirebilir. Bir diğer değişle ünlü iş mantığı (business logic) çoğunlukla *session bean*’ler tarafından gerçekleştirilir. *Session bean* de kendi içinde iki parçaya ayrılır.

- Stateful Session Bean (SFSB)
- Stateless Session Bean (SLSB)

Stateful Session Bean

Bu *session bea*n çeşidi sunucu tarafındaki bir *bean*’nin istemci tarafındaki uzantısı olarak düşünülebilir. Sadece bir istemcinin isteklerini gerçekleştirir. SFSB’nin özelliği *bean instance'*ın yaşam süresi boyunca bir durum bilgisi (state) tutmasıdır. Kısaca *bean* üzerinde çağrılan farklı metotlar birbirlerinden haberdar olabilirler. Bu *state bean* sınıfının değişkenlerinde tutulur. Bunu basit bir örnekle açıklayalım. Örneğin bir istemci bir e-ticaret sitesine girsin. Bu ziyaretçinin bu siteden istediği malları alış-veriş sepetine attığını düşünelim. Bu istemci siteden malları seçtikçe bu alışveriş sepeti dolmaya başlayacaktır. İşte bunun anlamı bu istemci için sunucuda tutulan bir SFSB’nin bu malları bellekte tutması demektir. Eğer istemci bu malları almaktan vazgeçerse ya da satın alma işlemini başlatmazsa bu alış-veriş sepetindeki mallar istemci sunucuyla ilişkiyi kestiğinde kaybolur. (Aslında çoğu e-ticaret sitesi bu malları bir veritabanına yazar ve aynı istemcinin bir sonraki ziyaretinde karşısına çıkartabilir. Fakat biz burada bunların kaydedilmediğini varsayıyoruz). Bunun anlamı sunucudaki SFSB’nin yok edilmesidir.

SFSB bir diğer ifadeyle iş akışlarındaki senaryoların gerçekleştirilmesi anlamına da gelir. Bu senaryoları yazılımcı SFSB olarak tasarlar ve *entity bean*’leri bu senaryolara göre işleyerek işletmenin ihtiyaçlarını karşılar.

Stateless Session Bean

Stateless session bean küçük iş parçacıklarıdır. SFSB'den farkı hiçbir istemciye bağımlı olmayışıdır. İstemcilerle alakalı bilgi tutmaz. Fonksiyonel dillerde kullanılan RPC (Remote Procedure Call) ile eşdeğerdedir. Çünkü istemci metot çağrımını yapar ve sonucunu alır. Sistemde bu işlemle alakalı hiçbir durum bilgisi tutulmaz.

Performans bakımından en verimli olan *bean*'dir. Tüm ihtiyaç duyduğu bilgiyi parametre olarak alır ya da bir veri kaynağından elde eder. SLSB'lere en güzel örneklerden biri onaylama işlemleridir. Mesela bir *transaction* içerisinde bir öğrencinin bir dersten yeterli notu alıp almadığını kontrol etmek istersek bunu bu tip işlerin toplandığı bir *stateless session bean* içerisindeki bir metot vasıtasıyla gerçekleştirebiliriz. Bu metot çağrımı sonucunda bize sadece doğru ya da yanlış diye bir yanıt döndürülür.

Message-Driven Beans: Artık mesajlama (Messaging) teknolojileri EJB'ler aracılığıyla da kullanılabilirler. EJB 2.0 belirtimiyle EJB bileşenleri ailesine katılan "Mesaj Bileşenleri", mesaja dayalı orta kat yazılımlarının (MOM - Message Orieted Middleware) sağladıkları olanakları, EJB bileşenleri katında sunarlar. Mesaj bileşenleri JMS'e (Java Messaging Service) dayalı olarak tasarlanmışlardır. JMS'in ardındaki prensip, JDBC ya da JNDI'daki ile aynıdır. JMS, farklı mesaja dayalı orta kat yazılımlarına aynı kodu kullanarak standart bir şekilde ulaşmayı sağlar.

▪ Java Persistence API (JPA)

Java Persistence API EJB 2.1 teknolojisine katılarak EJB teknolojisine güç veren varlık kalıcılığı modelini basitleştirir. Java Persistence API ilişkisel verilerin Java nesneleriyle nasıl ilişkilendirileceği, bu nesnelerin ilişkisel veri tabanında daha sonra erişilmeye elverişli şekilde nasıl tutulacağıyla ilgilenir. Ayrıca bu kullanan uygulama sona erdikten sonra da varlığın devamlılığının nasıl sağlanacağı ile de ilgilenir. Varlık kalıcılık modelinin basitleştirilmesi yanında Java Persistence API nesne ilişkisel eşlemeyi standartlaştırır.

EJB 3.0'e Java Persistence API'nin eklenmesiyle uygulama geliştiricilere kullanımı daha kolay ve içeriği daha zengin bir varlık programlama modeli sunmuştur.

Sun Java platformuna Java Persistence API'yi iki sebeple tanıtmıştır. İlk neden, bu yeni API JavaEE ve Java SE uygulamalarının geliştirilmesini veri kalıcılığı kullanarak basitleştirmesidir. Diğer neden ise bütün Java topluluğunu tek ve standart bir persistence API altında toplamaktır.

Java Persistence API'nin Avantajları

Java Persistence API, Hibernate, TopLink, JDO gibi kalıcılık teknolojilerinin en iyi yönleri ve son zamanlardaki EJB kap (container) yönetimine dayalı kalıcılık üzerine kurulmuştur. Java Persistence API varolan tek bir yapı iskeletine dayanmaz aksine var olan

pek çok güncel yapı iskeletinin desteklediği fikirleri içine alır, beraber çalışır ve geliştirir. Bu sayede uygulama geliştiriciler artık nesne ilişkisel modeli gerçekleştirmede uyumsuz ve standart olmayan kalıcılık modelleriyle yüzleşmek durumunda kalmamaktadır. Bununla beraber Java Persistence API daha çok uygulama geliştiriciye standart bir persistence API'ye sahip olma şansını vererek hem Java SE hem de Java EE içinde kullanılabilmektedir.

Java Persistence API uygulama geliştiricilere şu avantajları sağlamaktadır:

- Veri kalıcılığının teknik geçmişinden bağımsızlık
- Veri depolama araçlarından bağımsızlık
- Yüksek performans
- İş mantığı
- Test etmede daha az güç harcama
- Çoklu kullanıcı desteği
- Gelecekteki gelişimlere katılım
- Garantilenen geliştirme, yükseltme desteği

Java Persistence API'nin Bazı Temel Özellikleri

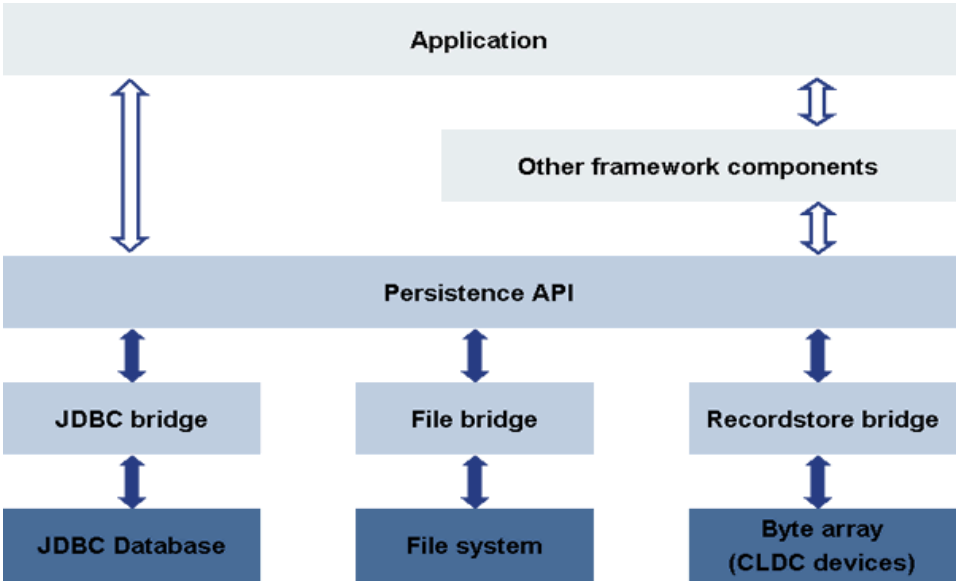
Java Persistence API nesne/ilişkisel eşlemede bir basit java nesnesi kalıcılık uygulama programı arayüzü (POJO persistence API)'dür. Java nesneleri ve ilişkisel veri tabanı arasındaki eşlemeyi tanımlamak için Java dili ara veri (metadata) için kod içine yazılan özel açıklamaları (annotations) ve / veya XML tanımlayıcılarının kullanımını destekleyen eksiksiz bir nesne/ilişkisel eşleme belirtimi içermektedir. Sabit ve dinamik sorgular (static and dynamic queries) için EJB QL üzerine kurulmuş önemli bir uzantı olan SQL benzeri zengin bir sorgu dilini destekler.

Sonuç olarak Java Persistence üç alan içerir:

- Java kalıcılık uygulama programı arayüzü.(Java Persistence API)
- Sordu dili (query language)
- Nesne/ ilişkisel eşlemele ara verisi. (Object/relational mapping metadata) API, Java sınıflarını tanımlar.

Java Persistence API'nin Java EE platformu dışında kullanımı da mümkündür. RI ve TCK belirtimleri Java Persistence API'nin Java SE ile JavaEE de olduğu gibi çalışabilmesini sağlar. TCK'yi JavaSE bölümüne geçirmek JavaEE sertifikası olmadan Java Persistence API kullanımına izin verir. Var olan EJB CMP uygulamalarını Java Persistence API'ye taşımaya gerek yoktur bu uygulamalar değişmeden çalışmaya devam ederler. Ayrıca Java Persistence API kullanımı sağlamak için, aynı uygulama içinde kullanımına devam edilen EJB CMP varlık çekirdeklerini (entity beans) yeni EJB bileşenleriyle birleştirmek mümkündür.

Aşağıdaki grafik genel mimariyi açıklamaktadır:



JPA' da Varlıklar (Entities)

Varlık hafif bir persistence temel nesnesidir. Bir varlık tipik olarak ilişkisel veri tabanında bir tabloyu temsil eder. Her bir varlık nesnesi tablonun bir satırına karşılık gelir. Varlığın birincil programlama adımı varlık sınıfı tanımlamaktır. Varlığın kalıcılık durumu (persistent state) ya kalıcılık alanları(persistent field) yada kalıcılık özellikleri ile ifade edilir. Bu alanlar ya da özellikler varlıkları ve varlık ilişkilerini öncelikli veri stokundaki ilişkisel veriye eşlemek için nesne/ilişkisel eşleme açıklamalarını kullanırlar.

Varlıkların Yönetimi

Varlıklar varlık yönetici (entity manager) tarafından yönetilir. Varlık Yönetici **javax.persistence.EntityManager** olguları ile temsil edilir**.** Her EntityManager olgusu bir kalıcılık bağlamı ile ilişkilidir. Bir kalıcılık bağlamı, belirli bir varlık olgusunun hangi etkinlik alanında yaratılıp, devam ettirilip, kaldırıldığını tanımlar.

- ## Eclipse Ortamında EJB ve JPA

Kurulması Gereken Dosyalar

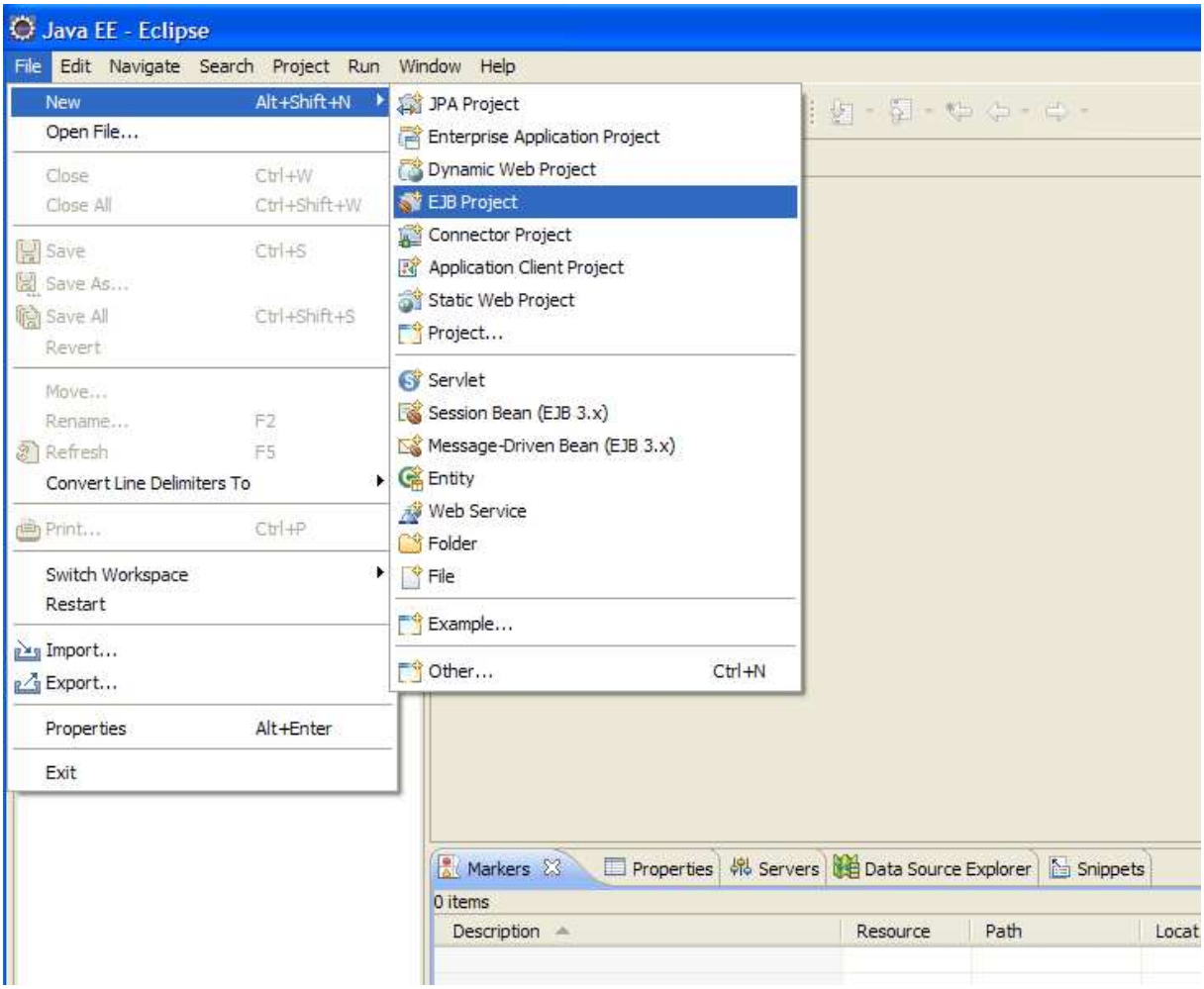
- ***Xampp Server*** (http://www.apachefriends.org/en/xampp-windows.html)
- ***Eclipse IDE for Java EE Developers*** (http://www.eclipse.org/downloads/)
- ***MySQL Connector/J*** –MySQL Driver(http://dev.mysql.com/downloads/mirror.php?id=13598)

- ***Persistence Provider*** – Toplink
  (http://download.java.net/maven/1/toplink.essentials/jars)
  -toplink-essentials-2.1-60f.jar
  - toplink-essentials-agent-2.1-60f.jar

- **JBOSS Application Server (En az 4.2.0 versiyonu)**
  (http://www.jboss.org/jbossas/downloads.html)

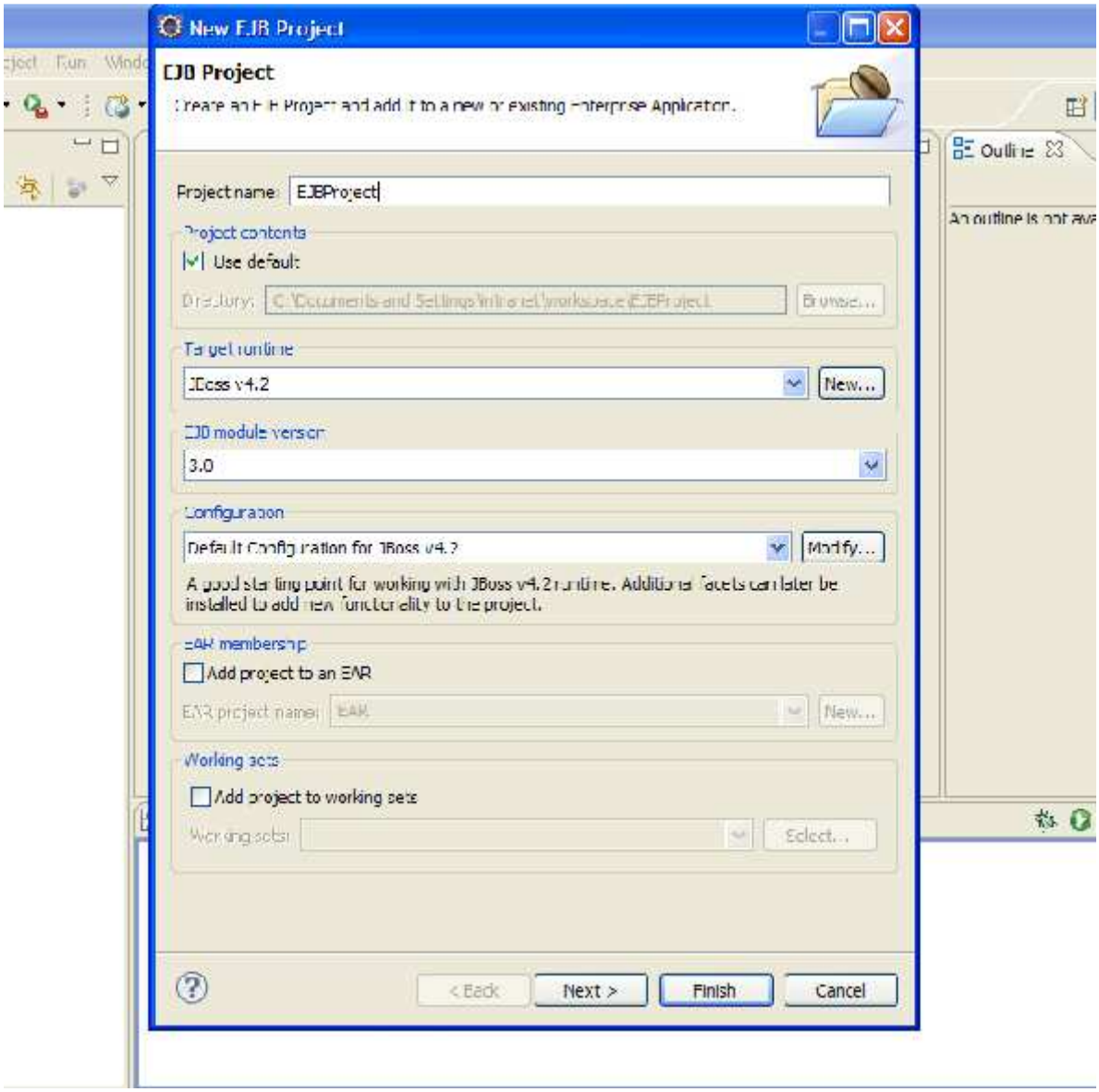
Kurulum ve Konfigürasyon

1. EJB Kurulum ve Konfigürasyon



Şekil Kurulum 1

İlk olarak Eclipse açılır ve Kurulum 1 deki gibi çıkan sayfadan New Project sekmesinde EJB Project seçilir.



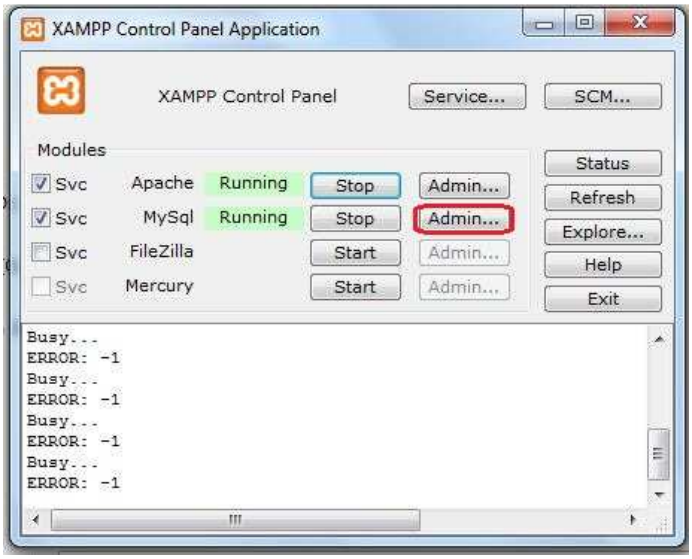
Şekil Kurulum 2

Daha sonra Kurulum 2 gibi çıkan sayfadan EJB projemize isim verilerek Finish sekmesiyle EJB projesi yaratılmış olur.

2. JPA Kurulum ve Konfigürasyon

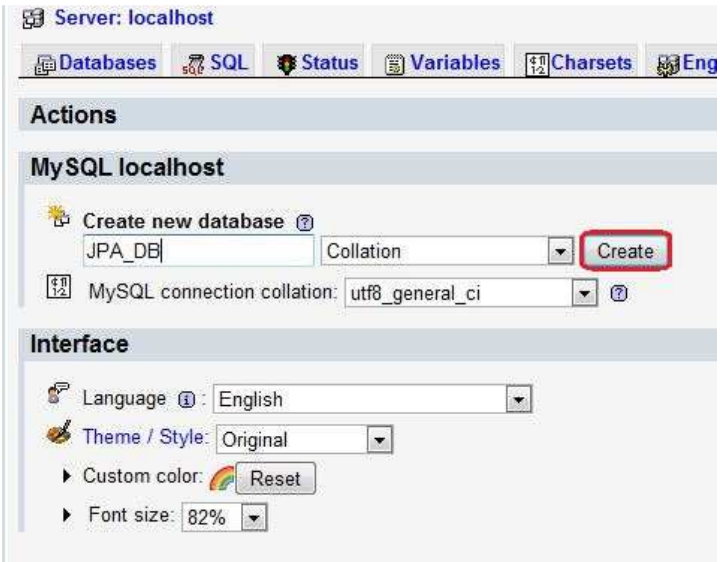
- İlk olarak Başlat-Apache Friends-Xampp-Xampp Control Panel açılarak Apache ve MySql başlatılır. Bunun için Apache ve MySql'in yanındaki start butonuna basılması yeterlidir.

- Servisler başlatıldıktan sonra MySql admin sayfasına geçilir.[Şekil Kurulum 1]



Şekil Kurulum 1

- Admin sayfası oluşturduktan sonra [Şekil Kurulum 2]'deki gibi bir sayfa ile karşılaşacağız. Bu sayfada "Create New Database" den yeni bir veritabanı oluşturulur.

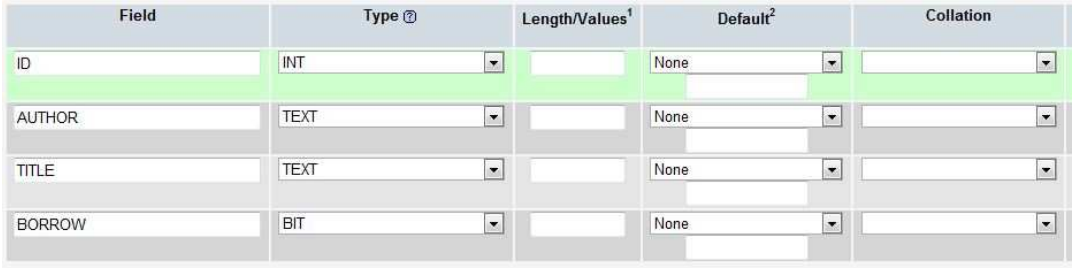


Şekil Kurulum 2

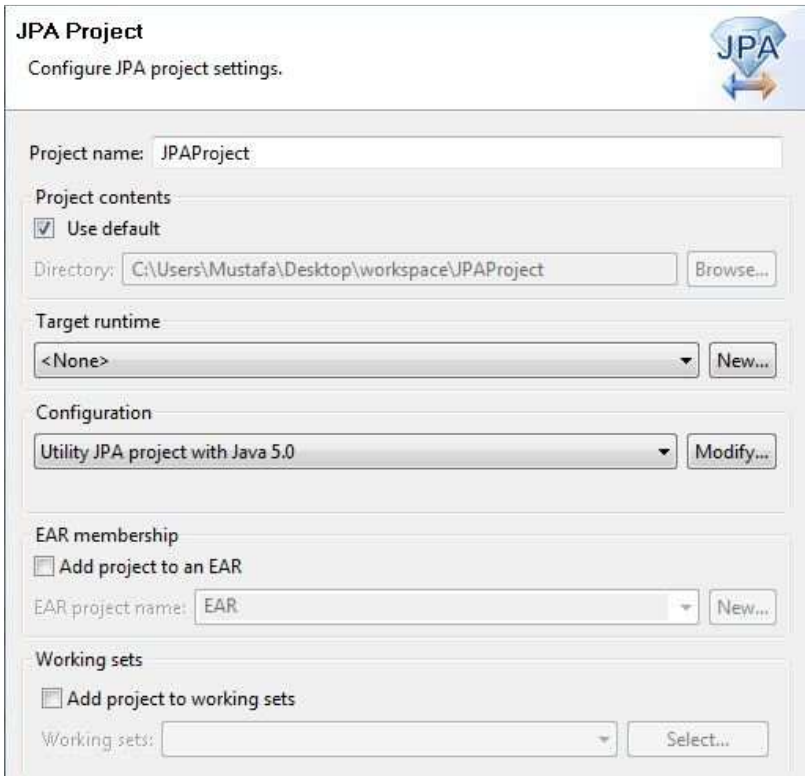
- Oluşturulan veritabanına "book" tablosunu ekleyelim. [Şekil Kurulum 3]

Şekil Kurulum 3

- Tabloda gerekli sahalar aşağıdaki gibi eklenir.



Şekil Kurulum 4

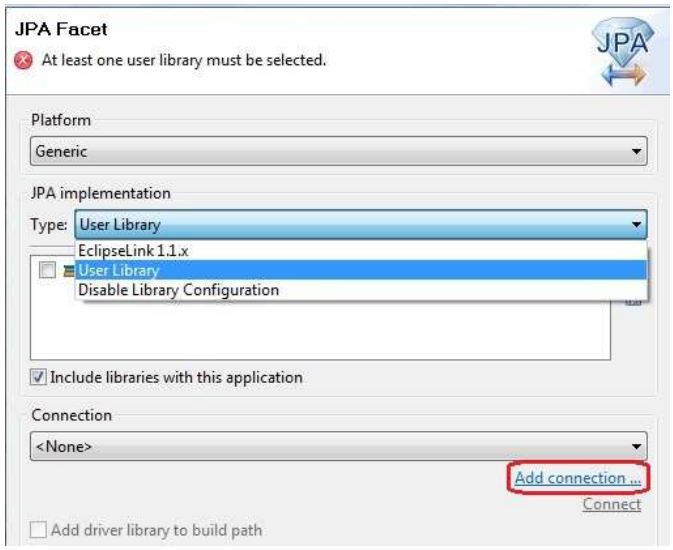


Şekil Kurulum 5

- Veritabanı operasyonlarımız bittikten sonra Eclipse'i açıp yeni bir JPA projesi oluşturalım.[Şekil Kurulum 5]'te görülen sayfada gerekli görülen yerler

işaretlendikten sonra “next” butonu yardımıyla [Şekil Kurulum 6]’da görülen sayfaya geçilebilir.

- Burada ise Type kısmında User Library işaretlenir.

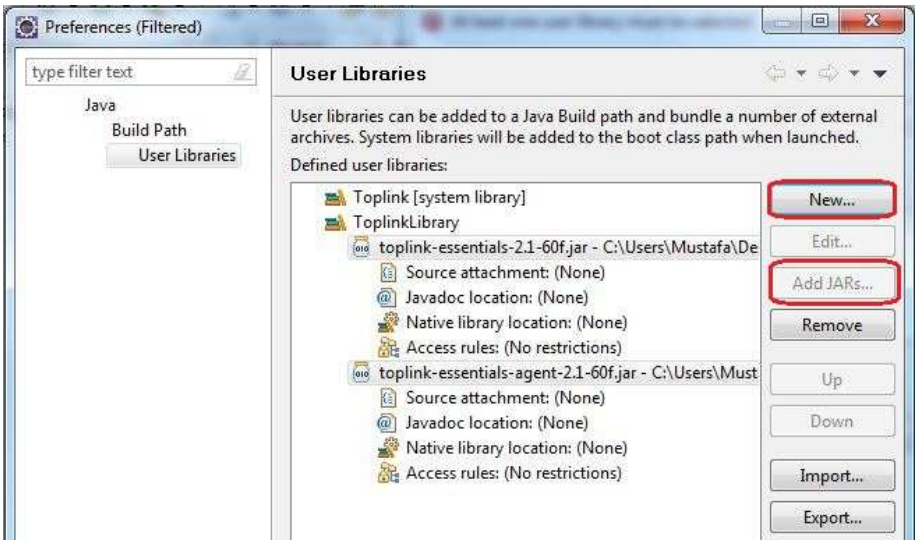


Şekil Kurulum 6

- User Library işaretlendikten sonra [Şekil Kurulum 7]’deki gibi bir sayfa karşımıza çıkıyor. Burada bir user library oluşturmamız gerek. ToplinkLibrary ismi verilebilir.

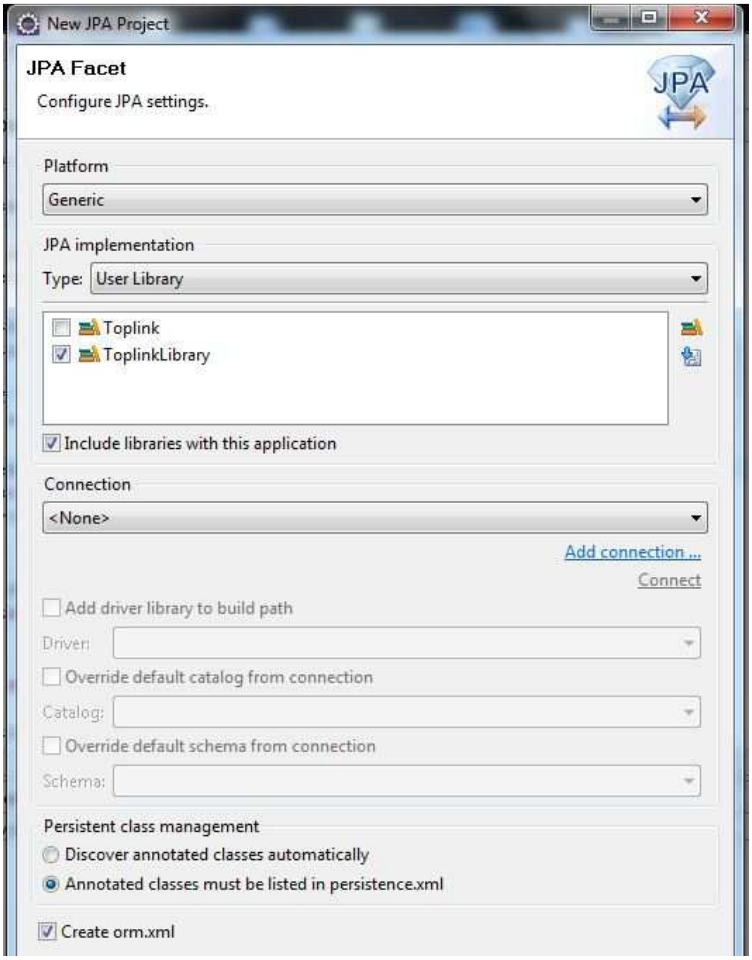
Şekil Kurulum 7

- Library oluşturulduktan sonra “Add Jars” yardımıyla ,[Şekil Kurulum 8]’de de görüldüğü gibi, indirilen (kurulması gereken dosyalar) “TopLink jar” dosyalarımızı import edip oluşturulan library’e ekleyelim.



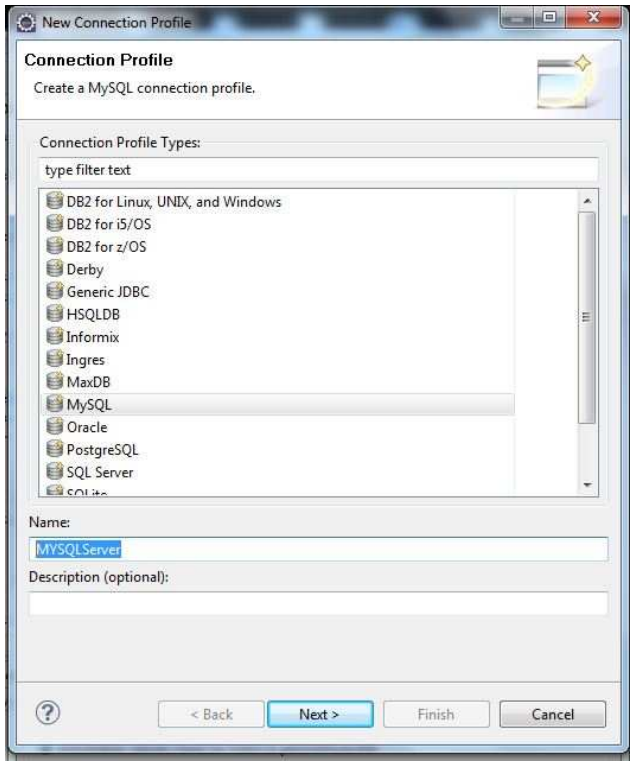
Şekil Kurulum 8

- Yeniden New Project sayfamıza geri döndüğümüzde “TopLinkLibrary”i işaretlememiz gerekir.



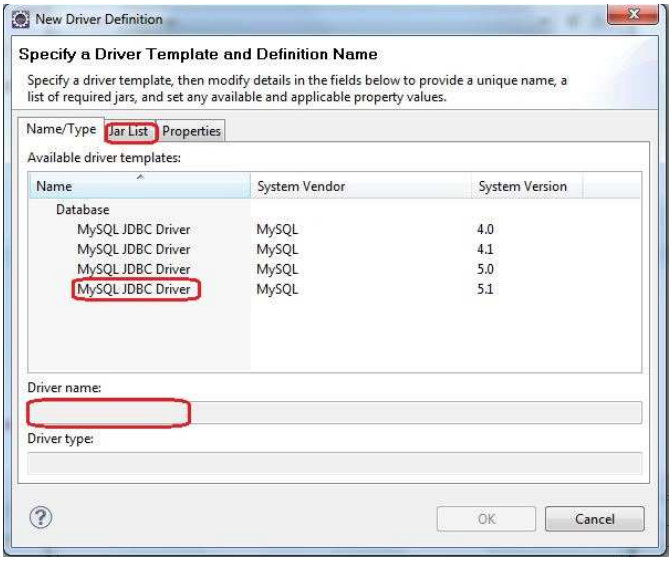
Şekil Kurulum 9

- Bu operasyondan sonra bağlantı kurulması gerekir. Bunun için add connection seçeneğinin tıklanması gerekmektedir.
- Sunucumuzun MySql olması dolayısıyla MySQL işaretlenir ve bağlantı profilimiz için isim seçmemiz gerekir. Bu isim MYSQLServer olabilir[Şekil Kurulum 10]. Daha sonra next butonu yardımıyla bağlantı oluşturma işlemimiz devam edilebilir.



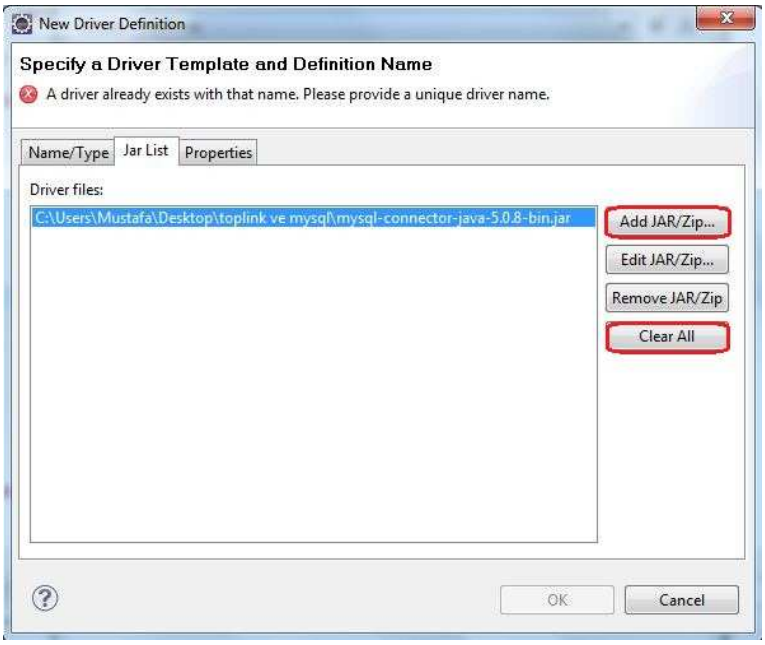
Şekil Kurulum 10

- [Şekil Kurulum 11]'deki gibi bir ekranla karşılaştığımızda ise; amacımız bağlantı sürücüsünü yüklemektir. Bunun için indirdiğimiz MySQL JDBC Driver'ın sürümüne göre bir seçim yapılır.(Örneğin 5.1)



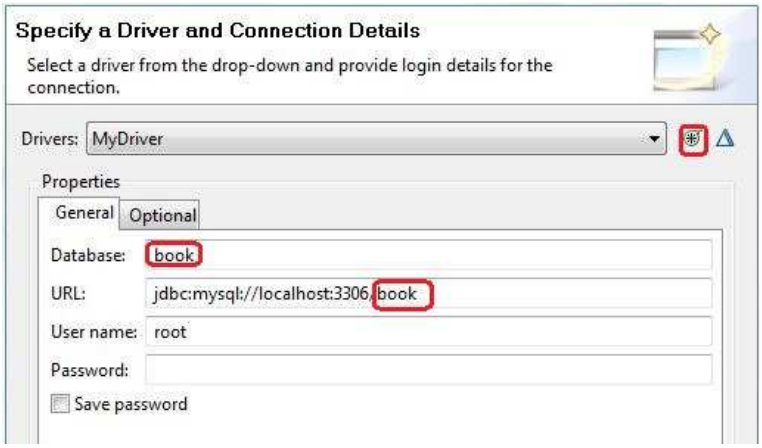
Şekil Kurulum 11

- Bu seçimden sonra indirilen jar dosyasının eklenmesi gerekmektedir. “Clear All” seçeneğinin önemiyse; bazen açılan bu ekranda istenmeyen bir driver görülebilir. Bunun eklenmemesi için ilk olarak “Clear All”ın işaretlenmesi yararlı olabilir.



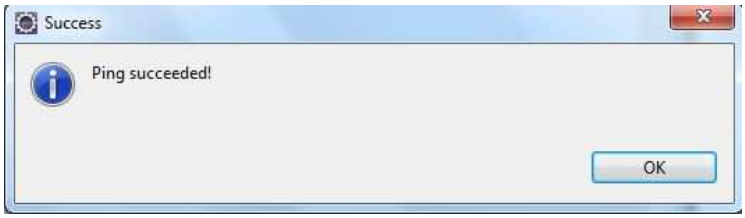
Şekil Kurulum 12

- URL kısmında ise veritabanımızın ismi verilir. Ve oluşturulan MyDriver, Drivers’dan işaretlenir.



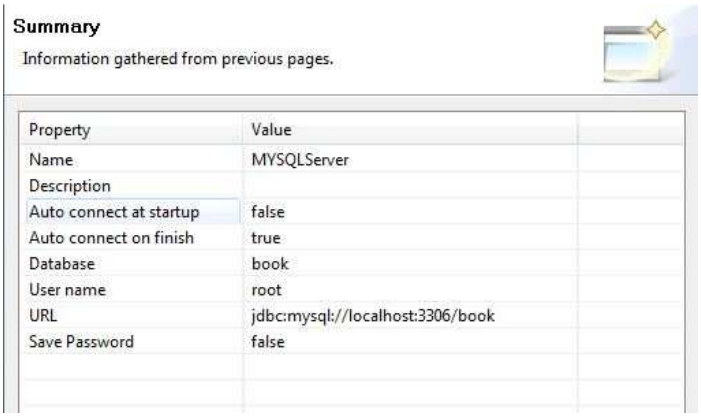
Şekil Kurulum 13

- Normal şartlar altında yapılan bu işlemlerden sonra test connection butonuna basılınca [Şekil Kurulum 14]’teki gibi bir başarılı uyarısının alınması gerekir.



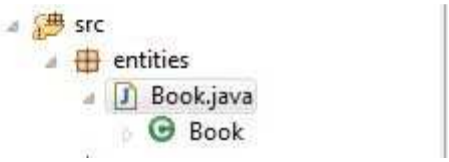
Şekil Kurulum 14

- Bir sonraki kurulum safhasında ise kurulumun özeti görülebilir.[Şekil Kurulum 15]



Şekil Kurulum 15

- Proje oluşturduktan ve bağlantı oluşturduktan sonra bir entities paketi oluşturulmuştur. Ondan sonra bu paketin içine Book.java adlı bir entity class'ı oluşturulur. Veritabanında oluşturduğumuz book tablosuna paralel olarak bu sınıftaki attribute'ler incelenebilir.



Şekil Kurulum 16

- [Şekil Kurulum 17]'de görüldüğü gibi entity sınıfı, veritabanında oluşturduğumuz tabloyla paralellik göstermektedir.

```java
@Entity
@Table(name="book")
public class Book implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)
    @Column(name="id")
    private int ID;

    @Lob()
    @Column(name="author")
    private String AUTHOR;

    @Column(name="borrow")
    private boolean BORROW;

    @Lob()
    @Column(name="title")
    private String TITLE;

    public Book() {
    }

    public int getID() {
        return this.ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }

    public String getAUTHOR() {
        return this.AUTHOR;
    }
```

Şekil Kurulum 17

- **Kaynaklar**

[1] en.wikipedia.org/wiki/Enterprise_JavaBean

[2] en.wikipedia.org/wiki/Java_Persistence_API