NACIONĀLAIS ATTĪSTĪBAS PLĀNS 2020
EIROPAS SAVIENĪBA
Eiropas Sociālais fonds
IEGULDĪJUMS TAVĀ NĀKOTNĒ

Java programmēšanas pamati

17. NODARBĪBA

DATU GLABĀŠANA

DATI

- Dati var būt fakti, kas saistīti ar jebkuru apskatāmo objektu
- Piemēram, jūsu vārds, vecums, garums, svars ir dati saistīti ar Jums

DATU BĀZE

- Datu bāze ir informācijas kopums, kas ir organizēts tā, lai tam varētu viegli piekļūt, izmainīt un izgūt
- Dati tiek kārtoti rindās, kolonās un tabulās
- Dati tiek indeksēti, lai vieglāk būtu atrast nepieciešamo informāciju
- Dati var tikt papildināti, atjaunoti vai dzēsti, kad jauni dati tiek pievienoti

DATU BĀZE VADĪBAS SISTĒMA (DBVS)

- DBVS ir vairākas programmas, kas nodrošina lietotājam
  - piekļuvi datiem
  - ļauj veikt manipulācijas ar datiem
  - sniedz plašas datu atspoguļošanas iespējas
- Kontrolē piekļuvi datu bāzei

DBVS TIPI

- Hierarhiskās DBVS
- Tīmekļa DBVS
- Relāciju DBVS
- Objektu Orientētās Relāciju DBVS

STRUKTURĒTA VAICĀJUMU VALODA

- «Structured Query Language» (SQL) standarta valoda, lai strādātu ar relāciju datu bāzēm
- SQL izmanto,
  - lai veiktu datu pievienošanas, dzēšanas un atjaunošanas operācijas
  - lai veidotu, dzēstu, izmanītu tabulas, indeksus, trigerus, lietotājas tiesības u.c. datu bāzes elementus

JDBC

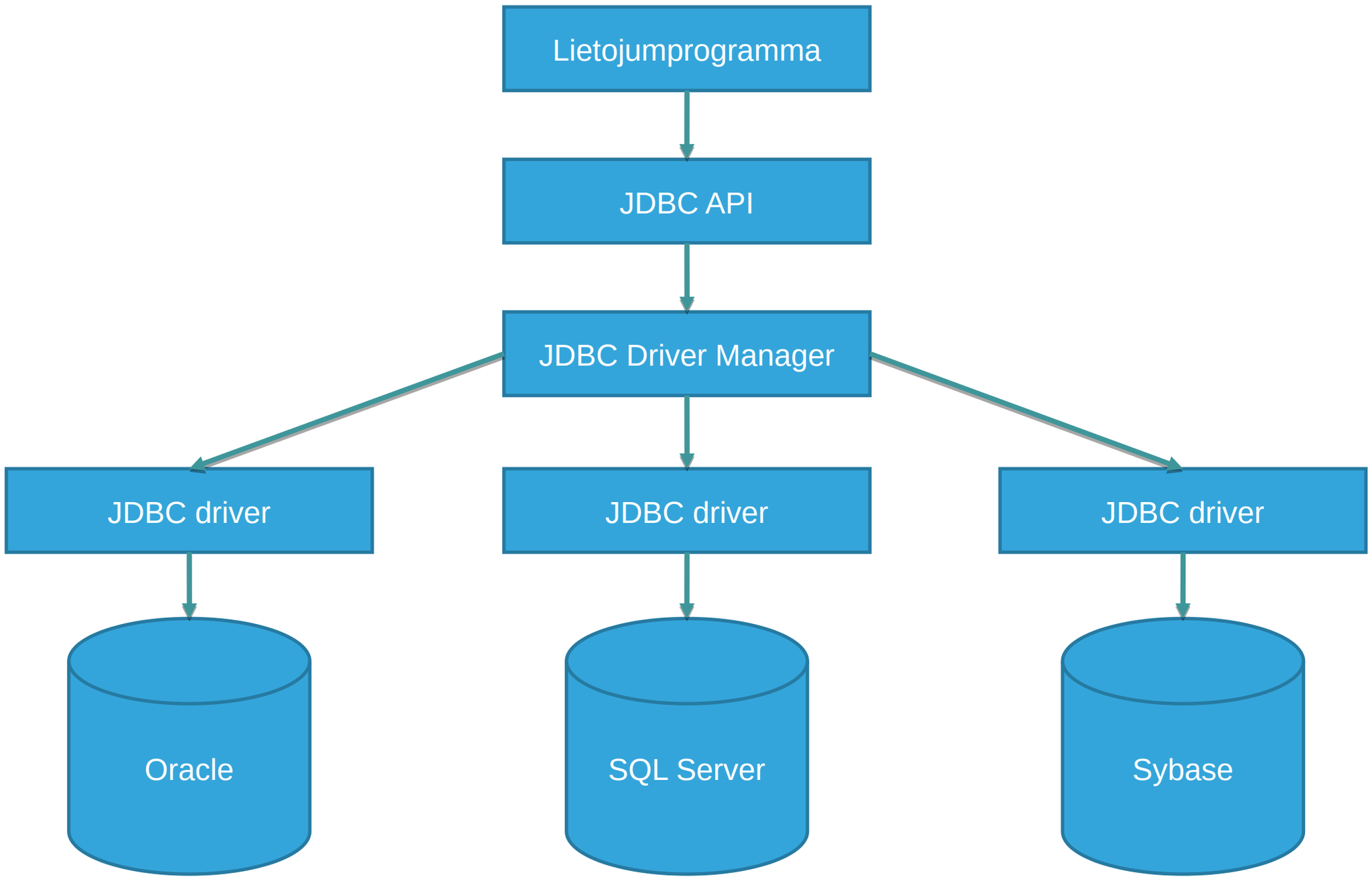
JAVA DATU BĀZES SAVIENOJAMĪBA

- Atšifrējums - JDBC (Java Database Connectivity)
- JDBC ir Java API (Lietojumprogrammas saskarne) darbam ar datu bāzēm
- JDBC bibliotēka ir iekļauta JDK
- JDBC API izmanto JDBC draiverus:
  - JDBC-ODBC Bridge Driver
  - Native Driver
  - Network Protocol Driver
  - Thin Driver

JDBC ARHTEKTŪRA

- JDBC arhitektūra pamatā sastāv no diviem slāņiem:
  - JDBC API – nodrošina savienojumu lietojumprogrammas -> «JDBC Manager»
  - JDBC Driver API – nodrošina savienojumu «JDBC Manager» -> «Driver»
- JDBC API izmanto katrai datu bāzei citu draiveri, lai nodrošinātu savienojamību ar datu bāzi
- JDBC Driver Manager – nodrošina, ka tiek izmantots atbilstošs driveris katrai datu bāzei

IEVADS JDBC


Lietojumprogramma
JDBC API
JDBC Driver Manager
JDBC driver
JDBC driver
JDBC driver
Oracle
SQL Server
Sybase

JDBC PIEMĒRS

```
Connection con = null;
  PreparedStatement prSt = null;
  try {
      Class.forName("oracle.jdbc.driver.OracleDriver");
      con = DriverManager.
          getConnection("jdbc:oracle:thin:@<hostname>:<port num>:<DB name>"
              ,"user","password");
      String query = "insert into emp(name,salary) values(?,?)";
      prSt = con.prepareStatement(query);
      prSt.setString(1, "John");
      prSt.setInt(2, 10000);
      //count will give you how many records got updated
      int count = prSt.executeUpdate();
      //Run the same query with different values
      prSt.setString(1, "Cric");
      prSt.setInt(2, 5000);
      count = prSt.executeUpdate();
} catch (ClassNotFoundException e) {
      e.printStackTrace();
} catch (SQLException e) {
      e.printStackTrace();
} finally{
      try{
          if(prSt != null) prSt.close();
          if(con != null) con.close();
      } catch(Exception ex){}
}
```

HIBERNATE

OBJEKTU RELĀCIJU KARTĒŠANA

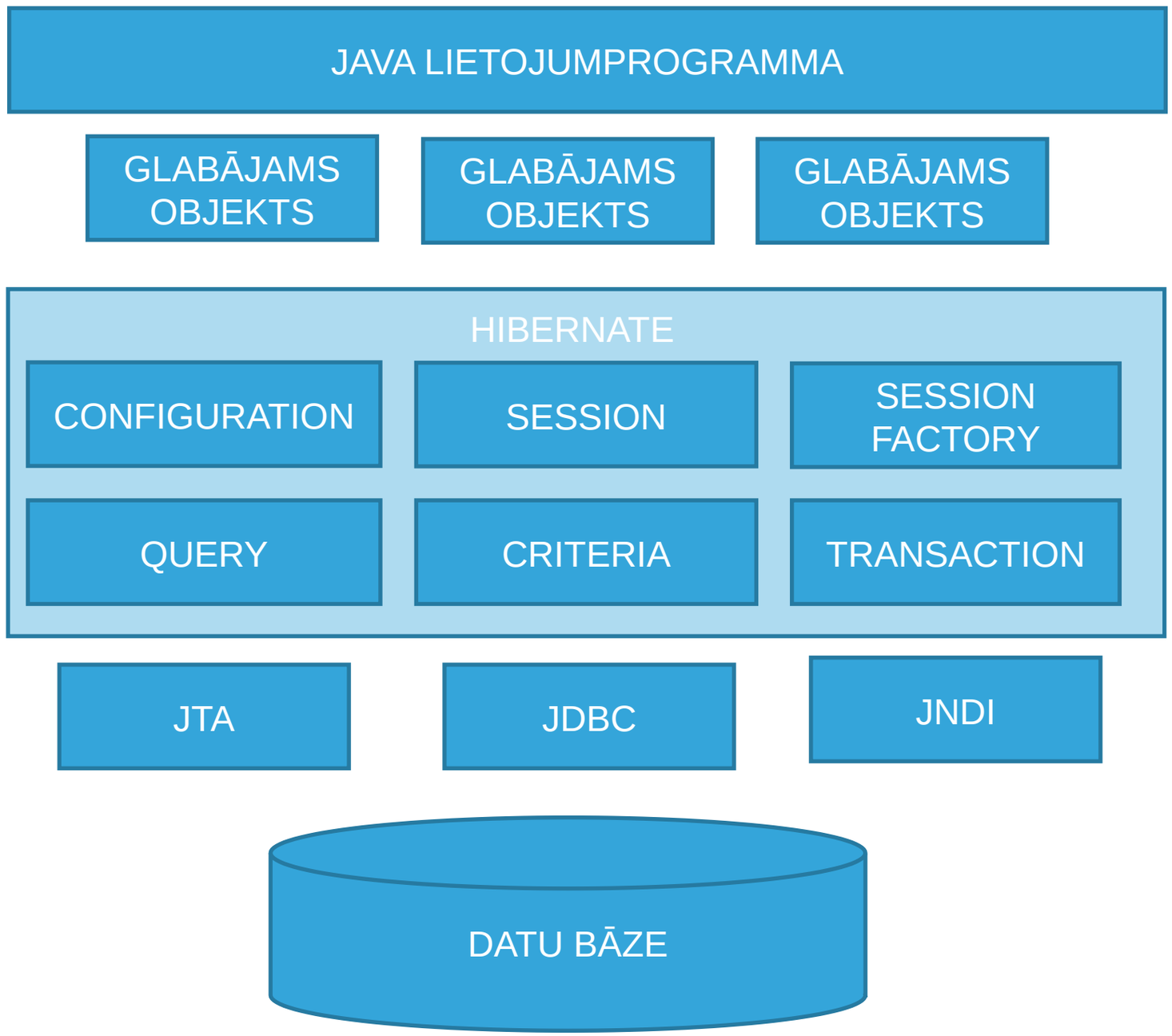
- Object-Relational-Mapping (ORM) ir programmēšanas tehnika datu konvertēšanai starp relāciju datu bāzi un objekt orientētu programmēšanas valodu.
- Hibernate ir ORM risinājums valodai Java
- Hibernate piesaista Java klases datu bāzes tabulām
- Hibernate piesaista Java tipus SQL datu tipiem
- Hibernate ir vidus slānis starp ierastajiem Java objektiem un datu bāzes tabulām, kas nodrošina šo objektu glabāšanu datu bāzē

DATU PLŪSMA

PRIEKŠROCĪBAS

- Hibernate sasaista Java klases ar datu bāzes tabulām, izmantojot XML failus/Anotācijas, nerakstot papildus kodu.
- Nodrošina vienkāršu API java objektu glabāšanai un izgūšanai no datu bāzes
- Gadījumā ja tiek izmainīta datu bāzes tabula, tad nepieciešamas izmaiņas tikai XML failā/klases anotācijās
- Nav nepieciešams strādāt ar SQL tipiem, tā vietā piedāvā Java tipus
- Hibernate darbam nav nepieciešams aplikāciju serveris
- Nodrošina vienkāršu datu izgūšanu
- Minimizē piekļuvi datu bāzei, izmantojot viedās datu ielādēšānas stratēģijas

ARHITEKTŪRA



KONFIGURĀCIJAS OBJEKTS

- Pirmais Hibernate objekts, kas tiek izveidots Hibernate lietojumprogrammā
- Parasti tiek izveidots tikai viens tās eksemplārs, programmas inicializācijas procesā
- Apzīmē Hibernate nepieciešamos konfigurācijas un resursu failus
- Konfigurācijas fails dod divas galvenās komponentes:
  - Datu bāzes savienojuma apraksts
  - Klašu «mapinga» iestatīšana – starp datu bāzes klasēm un Java klasēm

KONFIGURĀCIJAS OBJEKTS: PIEMĒRS

```
@Configuration
@ComponentScan(basePackages = "lv.lu.todolist")
@PropertySource("classpath:application.properties")
public class AppConfig {

    @Value("${jdbc.url}")
    private String jdbcUrl;

    @Value("${driverClass}")
    private String driverClass;

    @Value("${database.user.name}")
    private String userName;

    @Value("${database.user.password}")
    private String password;

    @Bean
    public static PropertySourcesPlaceholderConfigurer propertySourcesPlaceholderConfigurer() {
       return new PropertySourcesPlaceholderConfigurer();
    }

}
```

SESIJU FABRIKAS OBJEKTS

- Sesiju fabrikas (SessionFactory) objektu izveido Konfigurācijas objekts
- Izmantojot konfigurācijas failu Sesijas fabrikas objekts tiek instancēts
- Sesijas Fabrikas objekts ir plūsmu drošs objekts
- Sesijas fabrika ir «smags» objekts, tādēļ tas tiek izveidots programmas palaišanas brīdi un tiek atkal izmantots pēc nepieciešamības
- Katra datu bāzei būs nepieciešams viens šāds objekts

SESIJU FABRIKAS OBJEKTS: PIEMĒRS

```java
@Bean
public Properties hibernateProperties(
        @Value("org.hibernate.dialect.MySQLDialect") String dialect,
        @Value("true") boolean showSql,
        @Value("true") boolean formatSql,
        @Value("validate") String hbm2ddl) {

    Properties properties = new Properties();
    properties.put("hibernate.dialect", dialect);
    properties.put("hibernate.show_sql", showSql);
    properties.put("hibernate.format_sql", formatSql);
    properties.put("hibernate.hbm2ddl.auto", hbm2ddl);

    return properties;
}

@Bean
public SessionFactory sessionFactory(DataSource dataSource,
                                     @Value("com.javaguru.todolist") String packagesToScan,
                                     Properties hibernateProperties) throws Exception {

    LocalSessionFactoryBean sessionFactoryBean = new LocalSessionFactoryBean();
    sessionFactoryBean.setDataSource(dataSource);
    sessionFactoryBean.setPackagesToScan(packagesToScan);
    sessionFactoryBean.setHibernateProperties(hibernateProperties);
    sessionFactoryBean.afterPropertiesSet();
    return sessionFactoryBean.getObject();
}

@Bean
public PlatformTransactionManager transactionManager(SessionFactory sessionFactory) {
    return new HibernateTransactionManager(sessionFactory);
}
```

SESIJAS OBJEKTS

- Sesija tiek izmantota, lai iegūtu fizisku savienojumu ar datu bāzi
- Sesijas objekts ir viegls un tiek veidots no jaunu katru reizi, kad ir nepieciešams komunicēt ar datu bāzi
- Datu bāzē saglabātie objekti tiek glabāti un izgūti izmantojot Sesijas objektu
- Sesija objekts nav plūsmu drošs, tādēļ tas ir jāaizver līdzko darbs ir padarīts

TRANSAKCIJAS OBJEKTS

- Transakcija ir viena vesela darba vienība ar datu bāzi
- Transakcijas Hibernate sistēmā tiek realizētas transakciju menedžeriem (JDBC vai JTA)
- Šis nav obligāts objekts un lietojumprogramma var arī neizmantot to, ja tas nav nepieciešams

VAICĀJUMA OBJEKTS

- Vaicājuma objekts izmanto SQL vai Hibernate Vaicājumu vallodu (HQL) simbolu virkni, lai izgūtu datus no datu bāzes un veidotu objektus
- Vaicājuma objekta instanci izmanto, lai nodotu vaicājumam parametrus, ierobežotu rezultāta ierakstu skaitu un lai izpildītu vaicājumu

```
public boolean existsByName(String name) {
    String query = "select case when count(*)> 0 " +
            "then true else false end " +
            "from Task where name=“ + name;
    return (boolean) sessionFactory.getCurrentSession().createQuery(query)
            .setMaxResults(1)
            .uniqueResult();
}
```

KRITĒRIJA OBJEKTS

- Kritērija objektu izmanto, lai izveidotu objekta orientētu Kritērija vaicājumu datu bāzes ierakstu izgūšanai

```
public List<Task> findAll() {
    return sessionFactory.getCurrentSession().createCriteria(Task.class)
            .list();
}
```

HIBERNATE ANOTĀCIJAS

IESKATS

- Hibernate Anotācijas ir efektīvs veids kā nodot meta datus Objektu un Relāciju tabulu sasaistei.
- Visi meta dati tiek ievietoti Java POJO objektā, kas palīdz uzreiz izprast tabulas struktūru

PIEMĒRS

```java
@Entity
@Table(name = "tasks")
public class Task {

    @Id
    @Column(name = "id")
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column(name = "name")
    private String name;
    @Column(name = "description")
    private String description;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getDescription() {
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

}
```

ANOTĀCIJA @Table

- Anotācija @Table apraksta tabulu, kura tiek izmantota šī objekta glabāšanai datu bāzē
- Anotācija @Table ir četri atribūti:
  - nosaukums
  - katalogs
  - shēma
  - papildus unikāli ierobežojumi

ANOTĀCIJA @Entity

- Anotācija @Entity tiek izmantota, lai apzīmētu glabājamo objektu
- Šādai klasei ir jābūt bez parametru konstruktoram, kuram redzamība ir vismaz protected

ANOTĀCIJAS @Id un @GeneratedValue

- Katrai vienībai (entity) primārā atslēga, kura tiek anotēta ar @Id anotāciju
- Primārā atslēga varu būt viens lauks vai kombinācija no vairākiem laukiem
- @Id pēc noklusējuma izmantos viss piemērotāko primārās atslēgas ģenerēšanas stratēģiju
- Izmantojot @GeneratedValue ir iespējams primārās atslēgas ģenerēšanas algoritma stratēģiju mainīt

ANOTĀCIJA @Column

- @Column anotāciju izmanto, lai detalizēti aprakstītu kā lauks ar kolunnu ir sasaistīts:
  - «name» - ļauj atsevišķi noteikt kolonas nosaukumu
  - «length» - ļauj noteikt vērtības izmēru, īpaši simbolu virknēm
  - «nullable» - ļauj kolonai tikt atzīmētai kā NOT NULL, kad shēma tiek ģenerēta
  - «unique» - atzīmē kolonu, kā kolonu kas satur tikai unikālas vērtības

TRANSAKCIJA

TRANSAKCIJA

- Transakcija datu bāzēs ir viena loģiskā vienība, kas izpilda datu izgūšanu vai atjaunošanu datu bāzē
- Transakcijām relāciju datu bāzēs ir jābūt nemainīgām, konsekventām, izolētām un izturīgām (atomic, consistent, isolated, durable) – ACID
- Transakcijas tiek pabeigtas ar komandu COMMIT vai atceltas ar komandu ROLLBACK

ACID PRINCIPS

- Nemainīgums – Transakcijai var būt pilnīgi pabeigta vai neizpildīta vispār. Nav iespējama daļēji izpildīta transakcija.
- Konsistence – Transakcijai ir jāatbilst datu bāzes ierobežojumiem, tā nedrīkst to pārkāpt.
- Izolēta – Transakcijas dati nedrīkst būt pieejami citām transakcijām līdz brīdim kamēr dati ir saglabāti datu bāzē vai transakcija ir atcelta.
- Izturība – Veiksmīga transakcija izmaina sistēmas stāvokli un, pirms transakcija ir pabeigta, sistēmas stāvokļa izmaiņas tiek ierakstītas pastāvīgā transakciju logā.

PEMĒRS

- JPA izmantošanas piemērs ārpus atkarību injekcijas sistēmām

```
UserTransaction utx = entityManager.getTransaction();

try {
    utx.begin();
    businessLogic();
    utx.commit();
} catch(Exception ex) {
    utx.rollback();
    throw ex;
}
```

- Transakcijas apzīmēšana izmantojot @Transactional

```
@Transactional
public void businessLogic() {
... use entity manager inside a transaction ...
}
```

ATSAUCES

- http://hibernate.org/
- http://hibernate.org/orm/documentation/
- https://www.tutorialspoint.com/hibernate/
- https://dzone.com/articles/how-does-spring-transactional

LATVIJAS UNIVERSITATE
ANNO 1919

VADĪBAS UN UZŅĒMĒJDARBĪBAS