



# *Osnove web programiranja*

DAL

Termin 9

# Sadržaj

1. *Spring Boot* aplikacija i baza podataka
2. Podešavanje
3. Implementacija
  - I. DAL (*JdbcTemplate*)
  - II. DAL (SELECT upiti)
  - III. DAL (INSERT, UPDATE i DELETE upiti)
  - IV. DAL (SELECT upiti, povezani entiteti)
  - V. DAL (*many-to-many* veze, transakcije)
  - VI. servisi
4. Pretrage
5. Case study – CRUD bioskop veb aplikacija

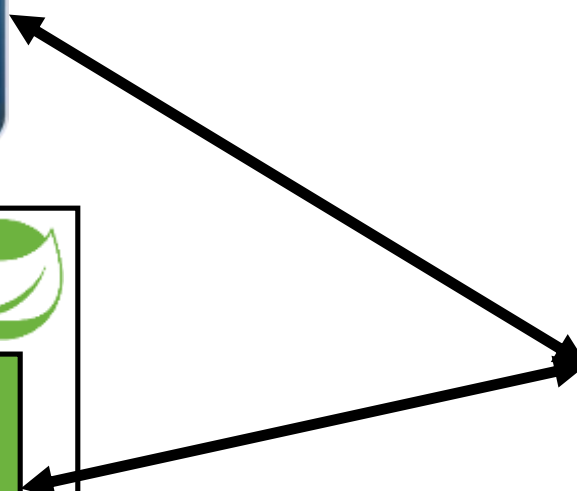
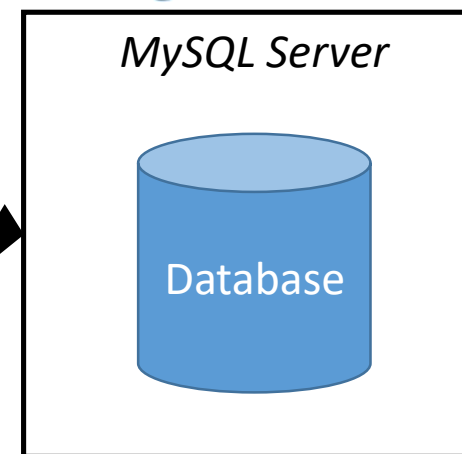
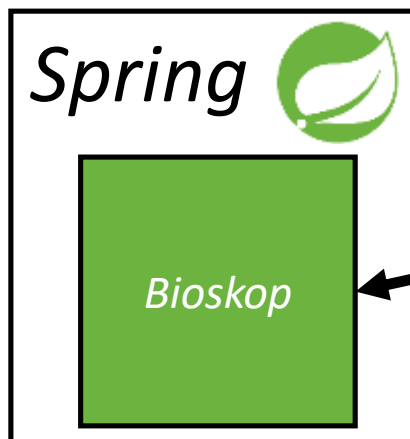
Dodatno:

1. Zašto baza podataka ???
2. Tipovi aplikacija – Arhitekture
3. Java i baze podataka
4. Tipovi drajvera DBMS
5. Korišćenje baze u Javi

# Spring Boot aplikacija i baza podataka

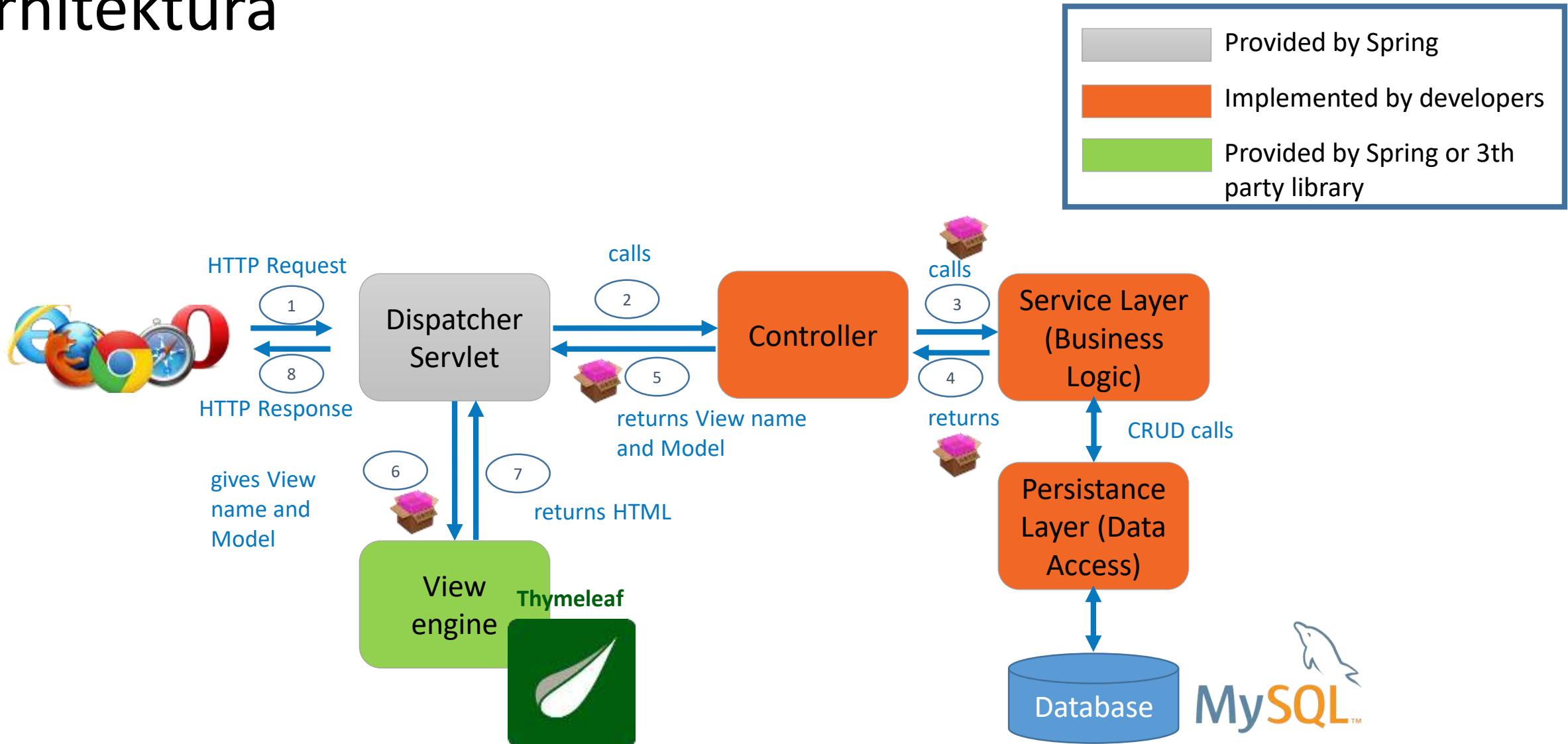
- poput konzolne, i *web* aplikacija može da komunicira sa RDMBS upotrebom **JDBC API-a**
- jedina **razlika** je u tome što se konzolna aplikacija izvršava samostalno, dok se *web* aplikacija izvršava u *framework-u*

MySQL Workbench



# Spring Boot aplikacija i baza podataka

## Arhitektura



# Spring Boot aplikacija i baza podataka

## 1. *controller*:

- a) traži od servisnog sloja podatke
- b) šalje pristigle parametre servisnom sloju radi upisa
- c) vrši kontrolu toka (redirekcija i sl.)

## 2. servisni sloj:

- a) šalje zahtev za čitanje DAL sloju
- b) po potrebi vrši pripremu podataka i šalje ih DAL sloju radi upisa

## 3. DAL sloj:

- a) vrši čitanje podataka iz baze i vraća ih servisnom sloju
- b) vrši upis podataka u bazu i vraća rezultat upisa

## 4. servisni sloj:

- a) po potrebu vrši obradu podataka i vraća ih *controller*-u
- b) prosleđuje rezultat upisa podataka *controller*-u

## 5. *controller*:

- a) prosleđuje podatke *view engine*-u radi prikaza
- b) vrši kontrolu toka (redirekcija i sl.)

# Podešavanje

- da bi se rad sa bazom podataka uključio u *Spring Boot* projekat, sledeće međuzavisnosti se moraju dodati u *pom.xml* datoteku:

```
<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId> ← JDBC driver
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jdbc</artifactId> ← API
</dependency>
```

- da bi se rad sa bazom podataka uključio u *Spring Boot* projekat, sledeći unos se mora dodati u *application.properties* datoteku:

```
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/bioskop?useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=root
```

# Podešavanje

src/main/java

- com.ftn.PrviMavenVebProjekat
- com.ftn.PrviMavenVebProjekat.bean
- com.ftn.PrviMavenVebProjekat.controller
- com.ftn.PrviMavenVebProjekat.dao
  - FilmDAO.java
  - KorisnikDAO.java
  - ProjekcijaDAO.java
  - ZanrDAO.java
- com.ftn.PrviMavenVebProjekat.dao.impl
  - FilmDAOImpl.java
  - KorisnikDAOImpl.java
  - ProjekcijaDAOImpl.java
  - ZanrDAOImpl.java
- com.ftn.PrviMavenVebProjekat.listeners
- com.ftn.PrviMavenVebProjekat.model
- com.ftn.PrviMavenVebProjekat.service
  - FilmService.java
  - KorisnikService.java
  - ProjekcijaService.java
  - ZanrService.java
- com.ftn.PrviMavenVebProjekat.service.impl
  - DatabaseFilmService.java
  - DatabaseKorisnikService.java
  - DatabaseProjekcijaService.java
  - DatabaseZanrService.java

src/main/resources

src/test/java

src/main/db

bioskop.sql

The screenshot shows a Java IDE with a project structure. The 'src/main/java' directory contains several packages. The 'com.ftn.PrviMavenVebProjekat' package is expanded, showing sub-packages like 'dao' and 'dao.impl'. The 'dao' package contains interfaces like 'FilmDAO.java', 'KorisnikDAO.java', 'ProjekcijaDAO.java', and 'ZanrDAO.java'. The 'dao.impl' package contains their implementations like 'FilmDAOImpl.java', 'KorisnikDAOImpl.java', 'ProjekcijaDAOImpl.java', and 'ZanrDAOImpl.java'. There are also 'listeners', 'model', 'service', and 'service.impl' packages. The 'service' package contains 'FilmService.java', 'KorisnikService.java', 'ProjekcijaService.java', and 'ZanrService.java'. The 'service.impl' package contains 'DatabaseFilmService.java', 'DatabaseKorisnikService.java', 'DatabaseProjekcijaService.java', and 'DatabaseZanrService.java'. The 'src/main/resources' directory is also visible. The 'src/test/java' directory is visible. The 'src/main/db' directory contains a file named 'bioskop.sql'. Orange arrows point from text annotations to specific parts of the structure: one to the 'dao' package, one to the 'dao.impl' package, one to the 'service.impl' package, and one to the 'src/main/db' directory.

klase DAL sloja po konvenciji  
treba da stoje u *dao* ili *db* paketu

potrebne su i posebne implementacije  
DAL sloja

potrebne su i posebne implementacije  
servisa za rad sa bazom podataka

SQL skripta za kreiranje/reinicijalizaciju šeme baze po  
konvenciji treba da stoji u *src/main/db* direktorijumu

# Implementacija

## DAL (*JdbcTemplate*)

- umesto interfejsa *sql.Connection*, za komunikaciju sa bazom se koristi klasa *JdbcTemplate*
- ovu klasu *framework inject-uje* za vreme izvršavanja i pri tom vrši **povezivanje sa bazom** na osnovu unosa iz *application.properties* datoteke

```
@Repository
public class FilmDAOImpl implements FilmDAO{

    @Autowired
    private JdbcTemplate jdbcTemplate;

    @Override
    public Film findOne(Long id) {...}
    public List<Film> findAll() {...}
    public void save(Film film) {...}
    public void update(Film film) {...}
    public void delete(Long id) {...}

}
```

← kao *Servisni sloj*, i klase DAL sloja se anotiraju *@Repository* anotacijom



# Implementacija

## DAL (SELECT upiti)

- *RowMapper* je interfejs čija implementacija služi za mapiranje svakog pojedinačnog reda *ResultSet*-a na objekat klase modela koji je potrebno kreirati
- metoda *mapRow* se poziva za svaki red *ResultSet*-a, a njena implementacija čita kolonu po kolonu za dati red i na osnovu pročitanih vrednosti **kreira jedan objekat i vraća jedan objekat klase modela**

```
@Repository
public class ZanrDAOImpl implements ZanrDAO{

    @Autowired
    private JdbcTemplate jdbcTemplate;

    private class ZanrRowMapper implements RowMapper<Zanr> {
        @Override
        public Zanr mapRow(ResultSet rs, int rowNum) throws SQLException {
            int index = 1;
            Long id = rs.getLong(index++);
            String naziv = rs.getString(index++);

            Zanr zanr = new Zanr(id, naziv);
            return zanr;
        }
    }

    public Zanr findOne(Long id) {
        String sql = "SELECT id, naziv FROM zanrovi WHERE id = ?";
        return jdbcTemplate.queryForObject(sql, new ZanrRowMapper(), id);
    }

    public List<Zanr> findAll() {
        String sql = "SELECT id, naziv FROM zanrovi";
        return jdbcTemplate.query(sql, new ZanrRowMapper());
    }
}
```

*PreparedStatement*

čita i vraća jedan objekat

*Statement*

čita i vraća listu objekata

# Implementacija

## DAL (INSERT, UPDATE i DELETE upiti, nezavisni entiteti)

```
@Repository
public class ZanrDAOImpl implements ZanrDAO{

    @Autowired
    private JdbcTemplate jdbcTemplate;

    public int save(Zanr zanr) {
        String sql = "INSERT INTO zanrovi (naziv) VALUES (?)";
        return jdbcTemplate.update(sql, zanr.getNaziv());
    }

    public int update(Zanr zanr) {
        String sql = "UPDATE zanrovi SET naziv = ? WHERE id = ?";
        return jdbcTemplate.update(sql, zanr.getNaziv(), zanr.getId());
    }

    public int delete(Long id) {
        String sql = "DELETE FROM zanrovi WHERE id = ?";
        return jdbcTemplate.update(sql, id);
    }
}
```

vrši  
upis/izmenu/brisanje  
u bazi

*PreparedStatement*

# Implementacija

## DAL (SELECT upiti, povezani entiteti)

- *RowCallbackHandler* je interfejs čija implementacija služi za mapiranje jednog reda *ResultSet*-a na objekat klase modela koji je potrebno kreirati, ali tako da je **programmer u kontroli kreiranja rezultata**
- metoda *processRow* se poziva za svaki red *ResultSet*-a, a njena implementacija čita kolonu po kolonu za dati red i na osnovu pročitanih vrednosti **kreira/pruzima jedan objekat i ne vraća nijedan objekat**

```
private static class FilmZanrRowCallBackHandler implements RowCallbackHandler {
    private Map<Long, Film> filmovi = new LinkedHashMap<>();

    @Override
    public void processRow(ResultSet resultSet) throws SQLException {
        int index = 1;
        Long filmId = resultSet.getLong(index++);
        String filmNaziv = resultSet.getString(index++);
        Integer filmTrajanje = resultSet.getInt(index++);

        Film film = filmovi.get(filmId);
        if (film == null) {
            film = new Film(filmId, filmNaziv, filmTrajanje);
            filmovi.put(film.getId(), film);
        }

        Long zanrId = resultSet.getLong(index++);
        String zanrNaziv = resultSet.getString(index++);
        Zanr zanr = new Zanr(zanrId, zanrNaziv);
        film.getZanrovi().add(zanr);
    }

    public List<Film> getFilmovi() {return new ArrayList<>(filmovi.values());}
}

public List<Film> findAll() {
    String sql =
        "SELECT f.id, f.naziv, f.trajanje, z.id, z.naziv FROM filmovi f " +
        "LEFT JOIN filmZanr fz ON fz.filmId = f.id " +
        "LEFT JOIN zanrovi z ON fz.zanrId = z.id " +
        "ORDER BY f.id";

    FilmZanrRowCallBackHandler rowCallbackHandler = new FilmZanrRowCallBackHandler();
    jdbcTemplate.query(sql, rowCallbackHandler);

    return rowCallbackHandler.getFilmovi(); ← čitanje rezultata
}
```

# Implementacija

## DAL (SELECT upiti, povezani entiteti)

- Za čuvanje pojedinačnog kreiranog objekta i preuzimanje svih kreiranih objekata odgovoran je programer.
- Primer je prikazan za **FilmDAOImpl** u kome je neophodno da se za svaki očitani film očitaju i njegovi žanrovi što zahteva povezivanje podataka iz tabela **film**, **filmZanr** i **zanr**.

```
private static class FilmZanrRowCallBackHandler implements RowCallbackHandler {  
    private Map<Long, Film> filmovi = new LinkedHashMap<>();  
  
    @Override  
    public void processRow(ResultSet resultSet) throws SQLException {  
        int index = 1;  
        Long filmId = resultSet.getLong(index++);  
        String filmNaziv = resultSet.getString(index++);  
        Integer filmTrajanje = resultSet.getInt(index++);  
  
        Film film = filmovi.get(filmId);  
        if (film == null) {  
            film = new Film(filmId, filmNaziv, filmTrajanje);  
            filmovi.put(film.getId(), film);  
        }  
  
        Long zanrId = resultSet.getLong(index++);  
        String zanrNaziv = resultSet.getString(index++);  
        Zanr zanr = new Zanr(zanrId, zanrNaziv);  
        film.getZanrovi().add(zanr);  
    }  
  
    public List<Film> getFilmovi() {return new ArrayList<>(filmovi.values());}  
}  
  
public List<Film> findAll() {  
    String sql =  
        "SELECT f.id, f.naziv, f.trajanje, z.id, z.naziv FROM filmovi f " +  
        "LEFT JOIN filmZanr fz ON fz.filmId = f.id " +  
        "LEFT JOIN zanrovi z ON fz.zanrId = z.id " +  
        "ORDER BY f.id";  
  
    FilmZanrRowCallBackHandler rowCallbackHandler = new FilmZanrRowCallBackHandler();  
    jdbcTemplate.query(sql, rowCallbackHandler);  
  
    return rowCallbackHandler.getFilmovi(); ← čitanje rezultata  
}
```

# Implementacija

## DAL (SELECT upiti, povezani entiteti)

- Drugi način za dobijanje podataka o povezanim entitetima bio bi korišćenje *RowMapper*.
- U ovom slučaju potrebno je kreirati onoliko *RowMapper* implementacija za koliko se tabela pristupa (**film**, **filmZanr** i **zanr**).
- Kreiraće se *FilmRowMapper* za tabelu **film**, *FilmZanrRowMapper* za tabelu **filmZanr**.
- Za tabelu **zanr** već je kreiran rowmapper u *ZanrDAOImpl* kalsi.

@Override

```
public List<Film> find(String naziv, Long zanrId, Integer trajanjeOd, Integer trajanjeDo) {
```

```
    ArrayList<Object> listaArgumenata = new ArrayList<Object>();  
    String sql = "SELECT f.id, f.naziv, f.trajanje FROM filmovi f ";
```

```
    StringBuffer whereSql = new StringBuffer(" WHERE ");  
    boolean imaArgumenata = false;
```

```
    if(naziv!=null) {  
        naziv = "%" + naziv + "%";  
        if(imaArgumenata)  
            whereSql.append(" AND ");  
        whereSql.append("f.naziv LIKE ?");  
        imaArgumenata = true;  
        listaArgumenata.add(naziv);  
    }
```

```
    if(trajanjeOd!=null) {  
        if(imaArgumenata)  
            whereSql.append(" AND ");  
        whereSql.append("f.trajanje >= ?");  
        imaArgumenata = true;  
        listaArgumenata.add(trajanjeOd);  
    }
```

```
    if(trajanjeDo!=null) {  
        if(imaArgumenata)  
            whereSql.append(" AND ");  
        whereSql.append("f.trajanje <= ?");  
        imaArgumenata = true;  
        listaArgumenata.add(trajanjeDo);  
    }
```

```
    if(imaArgumenata)  
        sql=sql + whereSql.toString()+" ORDER BY f.id";
```

```
    else
```

```
        sql=sql + " ORDER BY f.id";
```



# Implementacija

## DAL (SELECT upiti, povezani entiteti)

- Drugi način za dobijanje podataka o povezanim entitetima bio bi korišćenje *RowMapper*.
- U ovom slučaju potrebno je kreirati onliko *RowMapper* implementacija za koliko se tabela pristupa (*film*, *filmZanr* i *zanr*).
- Kreiraće se *FilmRowMapper* za tabelu *film*, *FilmZanrRowMapper* za tabelu *filmZanr*.
- Za tabelu *zanr* već je kreiran rowmapper u *ZanrDAOImpl* kalsi.

```
List<Film> filmovi = jdbcTemplate.query(sql, listaArgumenata.toArray(),
new FilmRowMapper());
for (Film film : filmovi) {
    film.setZanrovi(findFilmZanr(film.getId(), null));
}
//ako se treži film sa određenim žanrom
// tada se taj žanr mora nalaziti u listi žanrova od filma
if(zanrId!=null)
    for (Iterator iterator = filmovi.iterator();
        iterator.hasNext();) {
        Film film = (Film) iterator.next();
        boolean zaBrisanje = true;
        for (Zanr zanr : film.getZanrovi()) {
            if(zanr.getId() == zanrId) {
                zaBrisanje = false;
                break;
            }
        }
        if(zaBrisanje)
            iterator.remove();
    }
}
return filmovi;
}

private class FilmRowMapper implements RowMapper<Film> {

    @Override
    public Film mapRow(ResultSet rs, int rowNum) throws SQLException {
        int index = 1;
        Long filmId = rs.getLong(index++);
        String filmNaziv = rs.getString(index++);
        Integer filmTrajanje = rs.getInt(index++);

        Film film = new Film(filmId, filmNaziv, filmTrajanje);
        return film;
    }
}
```

# Implementacija

## DAL (SELECT upiti, povezani entiteti)

- Drugi način za dobijanje podataka o povezanim entitetima bio bi korišćenje *RowMapper*.
- U ovom slučaju potrebno je kreirati onoliko *RowMapper* implementacija za koliko se tabela pristupa (**film**, **filmZanr** i **zanr**).
- Kreiraće se *FilmRowMapper* za tabelu **film**, *FilmZanrRowMapper* za tabelu **filmZanr**.
- Za tabelu **zanr** već je kreiran rowmapper u *ZanrDAOImpl* kalsi.

```
private List<Zanr> findFilmZanr(Long filmId, Long zanrId) {  
  
    List<Zanr> znanoviFilma = new ArrayList<Zanr>();  
  
    ArrayList<Object> listaArgumenata = new ArrayList<Object>();  
  
    String sql =  
        "SELECT fz.filmId, fz.zanrId FROM filmZanr fz ";  
  
    StringBuffer whereSql = new StringBuffer(" WHERE ");  
    boolean imaArgumenata = false;  
  
    if(filmId!=null) {  
        if(imaArgumenata)  
            whereSql.append(" AND ");  
        whereSql.append("fz.filmId = ?");  
        imaArgumenata = true;  
        listaArgumenata.add(filmId);  
    }  
  
    if(zanrId!=null) {  
        if(imaArgumenata)  
            whereSql.append(" AND ");  
        whereSql.append("fz.zanrId = ?");  
        imaArgumenata = true;  
        listaArgumenata.add(zanrId);  
    }  
  
    if(imaArgumenata)  
        sql=sql + whereSql.toString()+" ORDER BY fz.filmId";  
    else  
        sql=sql + " ORDER BY fz.filmId";  
}
```

...



# Implementacija

## DAL (SELECT upiti, povezani entiteti)

- Drugi način za dobijanje podataka o povezanim entitetima bio bi korišćenje *RowMapper*.
- U ovom slučaju potrebno je kreirati onliko *RowMapper* implementacija za koliko se tabela pristupa (**film**, **filmZanr** i **zanr**).
- Kreiraće se *FilmRowMapper* za tabelu **film**, *FilmZanrRowMapper* za tabelu **filmZanr**.
- Za tabelu **zanr** već je kreiran rowmapper u *ZanrDAOImpl* kalsi.

```
List<Long[]> filmZanrovi = jdbcTemplate.query(sql,
listaArgumenata.toArray(), new FilmZanrRowMapper());

for (Long[] fz : filmZanrovi) {
    zanroviFilma.add(zanrDAO.findOne(fz[1]));
}
return zanroviFilma;
}

private class FilmZanrRowMapper implements RowMapper<Long []> {

    @Override
    public Long [] mapRow(ResultSet rs, int rowNum) throws SQLException {
        int index = 1;
        Long filmId = rs.getLong(index++);
        Long zanrId = rs.getLong(index++);

        Long [] filmZanr = {filmId, zanrId};
        return filmZanr;
    }
}
```



# Implementacija

DAL (*many-to-many* veze, ažuriranje podataka, transakcije)

- `@Transactional` anotacija služi da grupiše sve upite iz metode u transakciju
- *PreparedStatementCreator* je interfejs čija implementacija služi da se dodatno upravlja kreiranjem *PreparedStatement*-a
- *GeneratedKeyHolder* je klasa čiji objekti čuvaju ključeve, generisane od strane baze, koji su nastali pri prethodnom INSERT upitu

```
@Transactional
public int save(Film film) {
    PreparedStatementCreator preparedStatementCreator =
        new PreparedStatementCreator() {

        @Override
        public PreparedStatement createPreparedStatement(
            Connection connection) throws SQLException {
            String sql = "INSERT INTO filmovi (naziv, trajanje) VALUES (?, ?)";

            PreparedStatement preparedStatement =
                connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
            int index = 1;
            preparedStatement.setString(index++, film.getNaziv());
            preparedStatement.setInt(index++, film.getTrajanje());

            return preparedStatement;
        }
    };

    GeneratedKeyHolder keyHolder = new GeneratedKeyHolder();
    boolean uspeh = jdbcTemplate.update(preparedStatementCreator, keyHolder) == 1;
    if (uspeh) {
        String sql = "INSERT INTO filmZanr (filmId, zanrId) VALUES (?, ?)";
        for (Zanr itZanr: film.getZanrovi()) {
            uspeh = uspeh && jdbcTemplate.update(sql, keyHolder.getKey(),
itZanr.getId()) == 1;
        }
    }
    return uspeh?1:0;
}
```

koncept vraćanja generisanog ključa je potreban samo za *many-to-many* veze!

# Implementacija

## Servisi

naznaka *framework*-u da  
prioritetno koristi ovaj servis

```
@Service
@Primary
public class DatabaseZanrService implements ZanrService {

    @Autowired
    private ZanrDAO zanrDAO;

    @Override
    public List<Zanr> findAll() {
        return zanrDAO.findAll();
    }
}
```

```
@Repository
public class ZanrDAOImpl implements ZanrDAO{

    @Autowired
    private JdbcTemplate jdbcTemplate;

    public List<Zanr> findAll() {
        String sql = "SELECT id, naziv FROM zanrovi";
        return jdbcTemplate.query(sql, new ZanrRowMapper());
    }
}
```

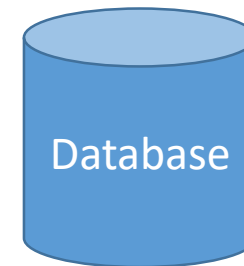
```
@Controller
@RequestMapping(value="/Zanrovi")
public class ZanroviController {

    @Autowired
    private ZanrService zanrService;

    @GetMapping
    public ModelAndView index() {
        List<Zanr> zanrovi = zanrService.findAll();

        ModelAndView rezultat = new ModelAndView("zanrovi");
        rezultat.addObject("zanrovi", zanrovi);

        return rezultat;
    }
}
```



# Pretrage

forma bez *action* atributa  
pravi zahtev na tekući URL




```
<form method="get">  
...  
</form>
```

**Pretraga FILMOVA, U  
servisnom sloju se očitaju  
svi fimovi pa se pristupi  
filtriranju rezultata**



Filmovi				
r. br.	naziv	žanr	trajanje	
	<input type="text" value="er"/>	<input type="text" value="odaberi"/>	od: 300 do: 400	<input type="button" value="Pretraži"/>
1	<a href="#">Avengers: Endgame</a>	<ul style="list-style-type: none"><li><a href="#">akcija</a></li><li><a href="#">avantura</a></li><li><a href="#">naučna fantastika</a></li></ul>	182	<a href="#">projekcije</a>
2	<a href="#">Life</a>	<ul style="list-style-type: none"><li><a href="#">naučna fantastika</a></li><li><a href="#">horor</a></li></ul>	110	<a href="#">projekcije</a>
3	<a href="#">It: Chapter 2</a>	<ul style="list-style-type: none"><li><a href="#">horor</a></li></ul>	170	<a href="#">projekcije</a>
4	<a href="#">Pirates of the Caribbean: Dead Men Tell No Tales</a>	<ul style="list-style-type: none"><li><a href="#">komediya</a></li><li><a href="#">avantura</a></li><li><a href="#">akcija</a></li></ul>	153	<a href="#">projekcije</a>

# Pretrage



localhost:8080/Bioskop/Filmovi?naziv=er&zanrId=0&trajanjeOd=100&trajanjeDo=300

```
@Controller
@RequestMapping(value="/Filmovi")
public class FilmoviController implements ServletContextAware {
```

```
@GetMapping
```

```
public ModelAndView index(
```

```
    @RequestParam(required=false) String naziv,
    @RequestParam(required=false) Long zanrId,
    @RequestParam(required=false) Integer trajanjeOd,
    @RequestParam(required=false) Integer trajanjeDo,
```

```
    HttpSession session) throws IOException {
```

```
//ako je input tipa text i ništa se ne unese
```

```
//a parametar metode Sting onda će vrednost parametra handler metode biti "" što nije null
```

```
    if(naziv!=null && naziv.trim().equals(""))
```

```
        naziv=null;
```

```
    List<Film> filmovi = filmService.find(naziv, zanrId, trajanjeOd, trajanjeDo);
```

```
    List<Zanr> zanrovi = zanrService.findAll();
```

```
    ModelAndView rezultat = new ModelAndView("filmovi");
```


```
    rezultat.addObject("filmovi", filmovi);
```

```
    rezultat.addObject("zanrovi", zanrovi);
```

```
    return rezultat;
```

```
}
```

podrazumevane vrednosti  
parametara za slučaj da se  
stranici pristupilo putem *link-a*  
su null



# Pretrage

```
//U ovom metodi pretragu radimo manuelno tako što programski filtriramo kompletnu listu rezultata
dobijenu sa findAll()
//čitaju se svi filmovi iz baze pa se filtrira po uslovu
//nije praktično imajući u vidu da u bazi može biti i 500.000 filmova
//prekličnije je urađeno u DatabaseProjekcijeService kada se poziva find(parametri) od ProjekcijeDAO
//ideja je koristiti select sa where delom sa se smanji ResultSet
@Override
public List<Film> find(String naziv, Long zanrId, Integer trajanjeOd, Integer trajanjeDo) {
    List<Film> filmovi = filmDAO.findAll();

    // maksimalno inkluzivne vrednosti parametara ako su izostavljeni
    // filtriranje radi u Servisnom sloju - izbegavati
    if (naziv == null) {
        naziv = "";
    }
```

maskimalno inkluzivne vrednosti parametara za  
slučaj da su izostavljeni jer se radi filtriranje u  
kodu

# Pretrage

```
@Override
public List<Film> find(String naziv, Long zanrId, Integer trajanjeOd, Integer trajanjeDo) {
    List<Film> filmovi = filmDAO.findAll();

    if (naziv == null) {naziv = "";}
    if (zanrId == null) {zanrId = 0L;}
    if (trajanjeOd == null) {trajanjeOd = 0;}
    if (trajanjeDo == null) {trajanjeDo = Integer.MAX_VALUE;}

    //odabran je da se filtriranje radi u Servisnom sloju - izbegavati
    List<Film> rezultat = new ArrayList<>();
    for (Film itFilm: filmovi) {
        if (!itFilm.getNaziv().toLowerCase().contains(naziv.toLowerCase())) {
            continue;
        }
        if (zanrId > 0) {
            boolean pronadjeno = false;
            for (Zanr itZanr: itFilm.getZanrovi()) {
                if (itZanr.getId() == zanrId) {
                    pronadjeno = true;
                    break;
                }
            }
            if (!pronadjeno) {
                continue;
            }
        }
        if (!(itFilm.getTrajanje() >= trajanjeOd && itFilm.getTrajanje() <= trajanjeDo)) {
            continue;
        }

        rezultat.add(itFilm);
    }

    return rezultat;
}
```

maskimalno inkluzivne vrednosti  
parametara za slučaj da su izostavljeni  
jer se radi filtriranje u kodu

ako je odabran žanr

kriterijum pretrage

# Pretrage

localhost:8080/Bioskop/Filmovi?naziv=er&zanrId=0&trajanjeOd=100&trajanjeDo=300

```
@Controller
@RequestMapping(value="/Filmovi")
public class FilmoviController implements ServletContextAware {

    @GetMapping
    public ModelAndView index(
        @RequestParam(required=false) String naziv,
        @RequestParam(required=false) Long zanrId,
        @RequestParam(required=false) Integer trajanjeOd,
        @RequestParam(required=false) Integer trajanjeDo,
        HttpSession session) throws IOException {
        //ako je input tipa text i ništa se ne unese
        //a parametar metode Sting onda će vrednost parametra handler metode biti "" što nije null
        if(naziv!=null && naziv.trim().equals(""))
            naziv=null;
        List<Film> filmovi = filmService.find(naziv, zanrId, trajanjeOd, trajanjeDo);
        List<Zanr> zanrovi = zanrService.findAll();

        ModelAndView rezultat = new ModelAndView("filmovi");
        rezultat.addObject("filmovi", filmovi);
        rezultat.addObject("zanrovi", zanrovi);

        return rezultat;
    }
}
```

# Pretrage

```
<tr>
<th></th>
<th><input type="search" name="naziv" th:value="${param.naziv}?: null"/></th>
<th>
<select name="zanrId">
<option value="">odaberi</option>
<option th:each="itZanr: ${zanrovi}" th:value="${itZanr.id}"
th:text="${itZanr.naziv}" th:selected="${#strings.equals(itZanr.id, param.zanrId)}"></option>
</select>
</th>
<th>
od:<input type="number" min="0" th:value="${param.trajanjeOd}?: '0'"
name="trajanjeOd"/><br/>
do:<input type="number" min="0" th:value="${param.trajanjeDo}?: '240'"
name="trajanjeDo"/>
</th>
<th><input type="submit" value="Pretraži"/></th>
</tr>
```

localhost:8080/Bioskop/Filmovi?naziv=er&zanrId=0&trajanjeOd=100&trajanjeDo=300

preslikavanje parametara  
na generisani prikaz da bi  
forma za pretragu ostala  
popunjena vrednostima

Filmovi				
r. br.	naziv	žanr	trajanje	
	er	odaberi	od: 100 do: 300	Pretraži
1	<a href="#">Avengers: Endgame</a>	<ul style="list-style-type: none"><li><a href="#">akcija</a></li><li><a href="#">avantura</a></li><li><a href="#">naučna fantastika</a></li></ul>	182	<a href="#">projekcije</a>
2	<a href="#">It: Chapter 2</a>	<ul style="list-style-type: none"><li><a href="#">horor</a></li></ul>	170	<a href="#">projekcije</a>



# Pretrage

Pretraga PROJEKCIJA, U servisnom sloju se očitaju samo oni projekcije koje zadovoljavaju zadate kriterijume, DAO izvršava pretragu putem formiranog SQL

```
@Override
public List<Projekcija> find(LocalDateTime datumIVremeOd, LocalDateTime datumIVremeDo, Long filmId,
String tip,
Integer sala, Double cenaKarteOd, Double cenaKarteDo) {
    // minimalne inkluzivne vrednosti parametara ako su izostavljeni
    //1. način bi bilo pozivanje ogovarajuće DAO metode u odnosu na broj parametara
    //gde bi trebalo implementirati više dao metoda tako da pokriju različite situacije
    //2. način reši sve u DAO sloju

    //odabran 2.

    return projekcijaDAO.find(datumIVremeOd, datumIVremeDo, filmId, tip, sala, cenaKarteOd,
cenaKarteDo);
}
```

minimalne inkluzivne vrednosti  
parametara za slučaj da su izostavljeni  
jer se radi pretraga u DAO sloju

return projekcijaDAO.find(datumIVremeOd, datumIVremeDo, filmId, tip, sala, cenaKarteOd, cenaKarteDo);

# Pretrage

@Override

```
public List<Projekcija> find(LocalDateTime datumIVremeOd, LocalDateTime datumIVremeDo, Long filmId, String tip, Integer sala, Double cenaKarteOd, Double cenaKarteDo) {
```

```
    ArrayList<Object> listaArgumenata = new ArrayList<Object>();
```

```
    String sql = "SELECT p.id, p.datumIVreme, p.tip, p.sala, p.cenaKarte, f.id, f.naziv, f.trajanje FROM projekcije p " +  
        "LEFT JOIN filmovi f ON p.filmId = f.id ";
```

```
    StringBuffer whereSql = new StringBuffer(" WHERE ");  
    boolean imaArgumenata = false;
```

```
    if(datumIVremeOd!=null) {  
        if(imaArgumenata)  
            whereSql.append(" AND ");  
        whereSql.append("p.datumIVreme >= ?");  
        imaArgumenata = true;  
        listaArgumenata.add(datumIVremeOd);  
    }
```

minimalne inkluzivne vrednosti  
parametara za slučaj da su izostavljeni  
jer se radi pretraga u DAO sloju



# Pretrage

```
if(datumIVremeDo!=null) {  
    if(imaArgumenata)  
        whereSql.append(" AND ");  
    whereSql.append("p.datumIVreme <= ?");  
    imaArgumenata = true;  
    listaArgumenata.add(datumIVremeDo);  
}
```

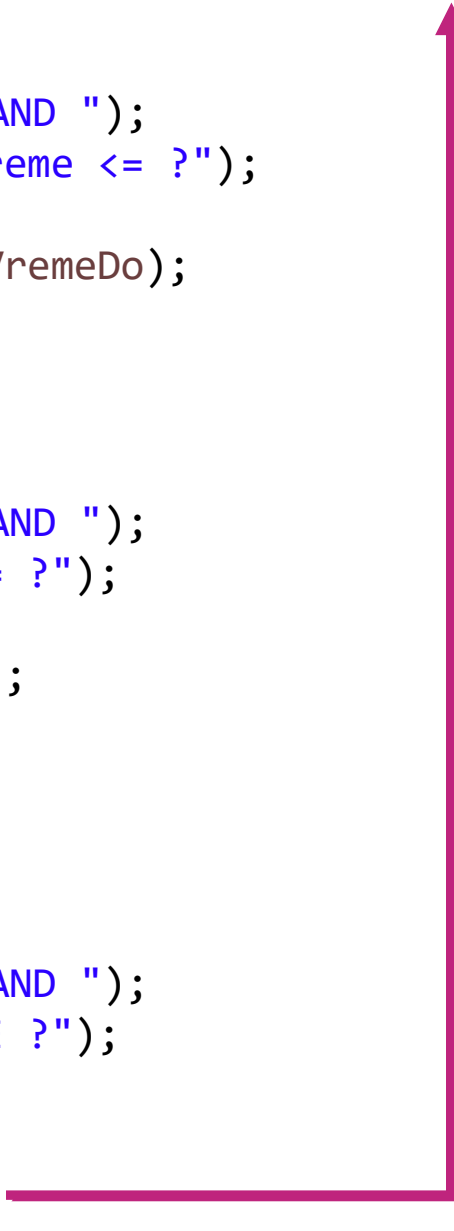
```
if(filmId!=null) {  
    if(imaArgumenata)  
        whereSql.append(" AND ");  
    whereSql.append("p.filmId = ?");  
    imaArgumenata = true;  
    listaArgumenata.add(filmId);  
}
```

```
if(tip!=null) {  
    tip = "%" + tip + "%";  
    if(imaArgumenata)  
        whereSql.append(" AND ");  
    whereSql.append("p.tip LIKE ?");  
    imaArgumenata = true;  
    listaArgumenata.add(tip);  
}
```

```
if(sala!=null) {  
    if(imaArgumenata)  
        whereSql.append(" AND ");  
    whereSql.append("p.sala = ?");  
    imaArgumenata = true;  
    listaArgumenata.add(sala);  
}
```

```
if(cenaKarteOd!=null) {  
    if(imaArgumenata)  
        whereSql.append(" AND ");  
    whereSql.append("p.cenaKarte >= ?");  
    imaArgumenata = true;  
    listaArgumenata.add(cenaKarteOd);  
}
```

```
if(cenaKarteDo!=null) {  
    if(imaArgumenata)  
        whereSql.append(" AND ");  
    whereSql.append("p.cenaKarte <= ?");  
    imaArgumenata = true;  
    listaArgumenata.add(cenaKarteDo);  
}
```



# Pretrage

```
if(imaArgumenata)
    sql=sql + whereSql.toString()+" ORDER BY p.id";
else
    sql=sql + " ORDER BY p.id";

System.out.println(sql);

← return jdbcTemplate.query(sql, listaArgumenata.toArray(), new ProjekcijaRowMapper());
}
```

SELECT p.id, p.datumIVreme, p.tip, p.sala, p.cenaKarte, f.id, f.naziv, f.trajanje FROM projekcije p LEFT JOIN filmovi f ON p.filmId = f.id WHERE p.datumIVreme >= ? AND p.datumIVreme <= ? ORDER BY p.id

SELECT p.id, p.datumIVreme, p.tip, p.sala, p.cenaKarte, f.id, f.naziv, f.trajanje FROM projekcije p LEFT JOIN filmovi f ON p.filmId = f.id WHERE p.datumIVreme >= ? AND p.datumIVreme <= ? AND p.tip LIKE ? ORDER BY p.id

SELECT p.id, p.datumIVreme, p.tip, p.sala, p.cenaKarte, f.id, f.naziv, f.trajanje FROM projekcije p LEFT JOIN filmovi f ON p.filmId = f.id WHERE p.datumIVreme >= ? AND p.datumIVreme <= ? AND p.tip LIKE ? AND p.cenaKarte <= ? ORDER BY p.id

# Case study – CRUD bioskop veb aplikacija

- USE CASE korišćenje DAL za Bioskop web aplikaciju
- *com.ftn.PrviMavenVebProjekat:*
  - *DatabaseZanrService.java, ZanrDAOImpl.java, zanrovi.html*
  - *DatabaseFilmService.java, FilmDAOImpl.java, filmovi.html*
  - *DatabaseProjekcijeService.java, projekcijeDAOImpl.java, projekcije.html*
  - *DatabaseKorisnikService.java, KorisnikDAOImpl.java, korisnici.html*

# Dodatni materijali

- <https://spring.io/guides/gs/relational-data-access/>
- <https://mkyong.com/spring-boot/spring-boot-jdbc-examples/>
- <https://www.petrikainulainen.net/programming/spring-framework/spring-data-jpa-tutorial-part-eight-adding-functionality-to-a-repository/>
- <https://docs.spring.io/spring-data/jpa/docs/1.5.0.RELEASE/reference/html/repositories.html>
- <https://www.baeldung.com/spring-data-repositories>
- <https://www.baeldung.com/java-connection-pooling>

Dodatno

# Zašto baza podataka ???

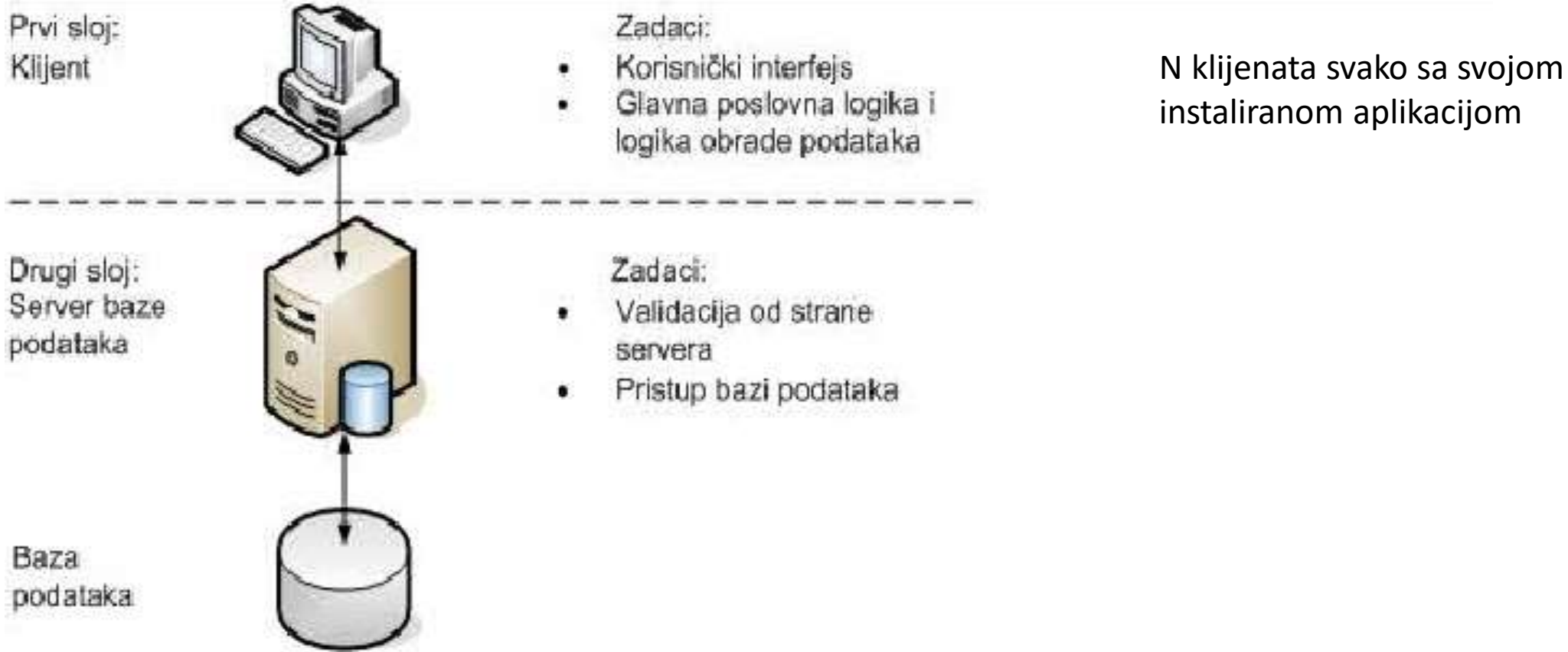
1. “Primary role of a database is to store and display updated information in a web application”.
2. Function of database – “Database applications are used to search, sort, filter and present information based upon web requests from users”
3. Features – “Databases grant and limit access to data based upon criteria such as user name, password, region or account number. Databases also enforce data integrity by ensuring that data is collected and presented using a consistent format”.
4. Effects - A dynamic website displays updated information on web pages when the database is updated by the host or when users submit information using web forms.



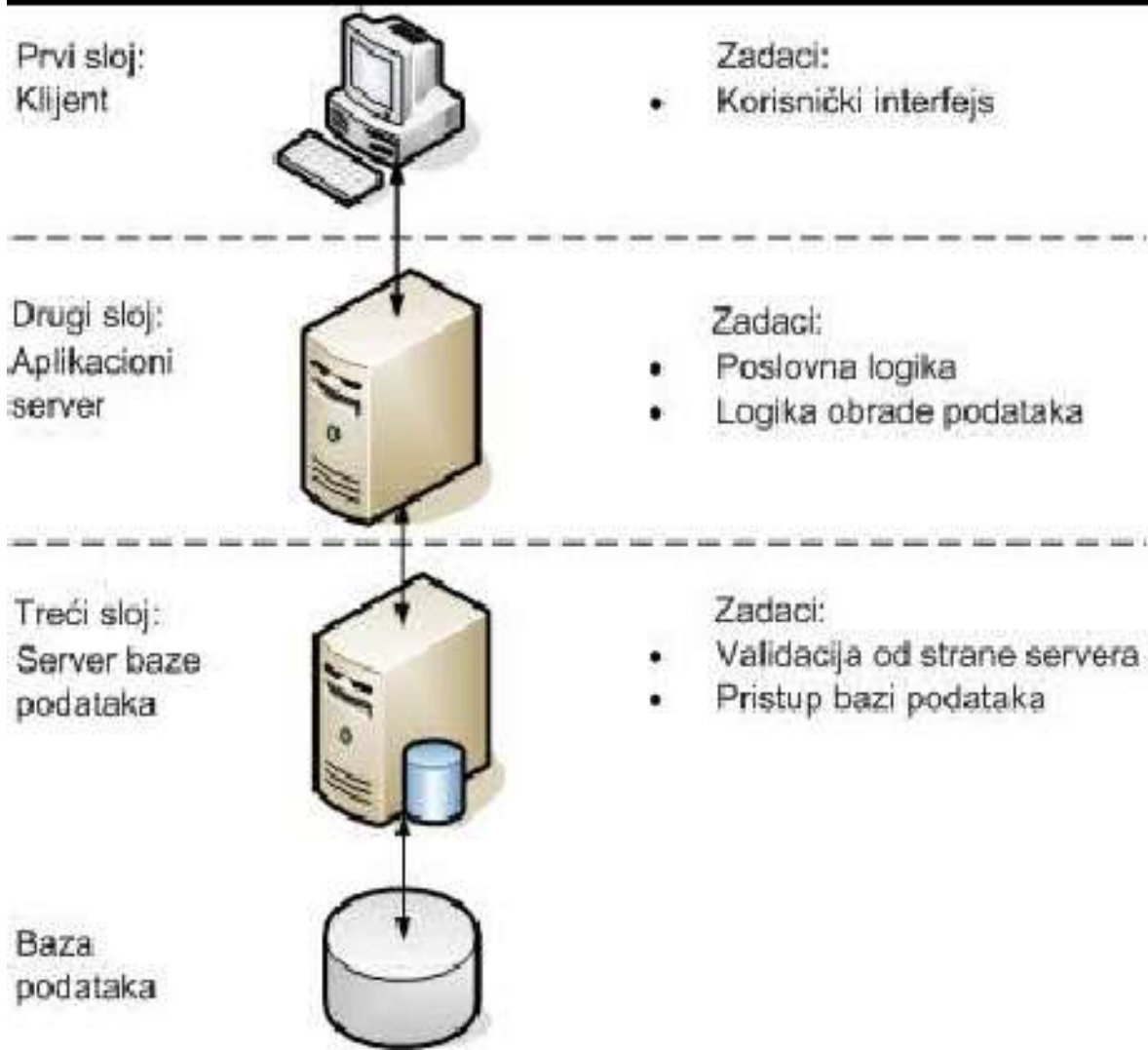
# Zašto baza podataka ???

1. Danas, svi moderni sajtovi čuvaju podatke u nekoj vrsti baze podataka
2. Za početak relacionala
3. Kako se povezati na bazu ???
4. Pišemo program koji nam ovo omogućava ???
5. Baze podataka prave različiti proizvođači i postoje različite verzije istih ???

# Tipovi aplikacija – Arhitekture dvoslojna



# Tipovi aplikacija – Arhitekture troslojna



N klijenata svako ima samo korisnički interfejs preko kojeg se pristupa jednoj aplikaciji

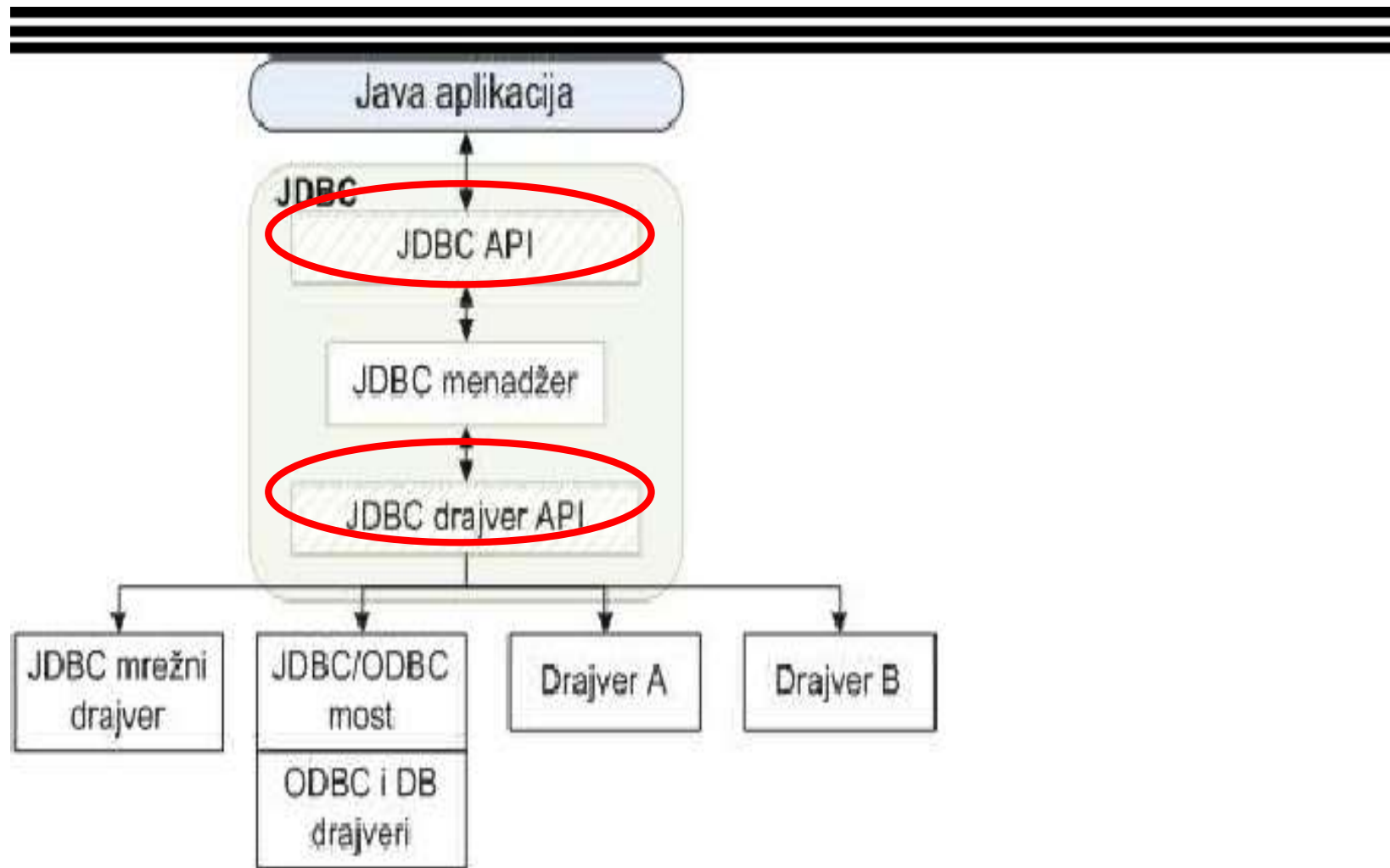
# Tipovi aplikacija - Arhitekture

- Prednosti troslojne :
  - Smanjenje troškova vezano za hardver klijentskih mašina
  - Modularnost, “lako” izmenjivi delovi
  - Sa izdvajanjem poslovne logike koja se **odražava na brojne krajnje korisnike** u zaseban sloj u vidu **aplikativnog servera, ažuriranje i održavanje aplikacije je centralizovano**. Ovim se eliminiše problem distribucije softvera, koji je bio prisutan u dvoslojnom klijent-server modelu
  - Sa dobijenom modularnošću moguće je lako izmeniti ili zameniti neki od slojeva bez uticaja na ostale

# Java i baze podataka

- Java ima definisan standardni interfejs za komunikaciju sa bazama podataka nazvan JDBC. (Java DataBase Connectivity)
- JDBC definiše skup klasa i interfejsa koji se koriste za pristup bazama podataka. Za komunikaciju sa serverima najčešće se koristi TCP/IP mrežna konekcija.
- Razvio Sun Microsystems kao uniforman skup interfejsa za pristup heterogenim relacionim bazama podataka.

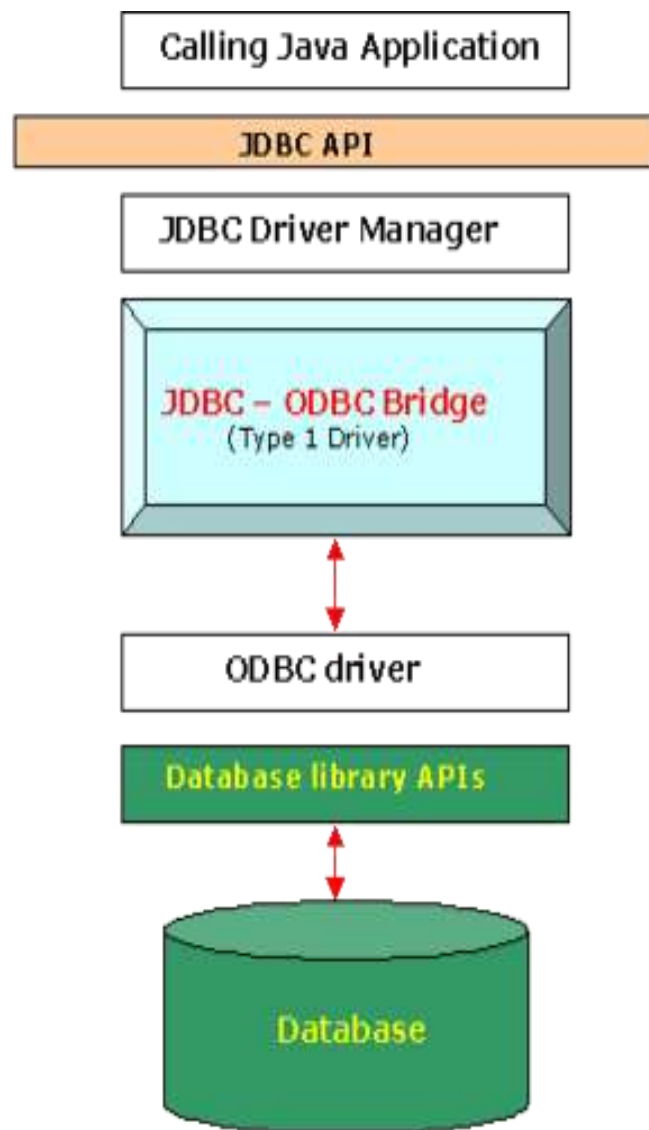
# Java i baze podataka - JDBC



# Java i baze podataka

- **JDBC API** - namenjen aplikativnim programerima, koji definiše komunikaciju na relaciji **Java aplikacija – JDBC menadžer**
- **JDBC drajver API**, interfejs nižeg nivoa namenjen programerima drajvera, koji definiše način komunikacije na relaciji **JDBC menadžer – drajver konkretnog DBMS**

# Tipovi drajvera DBMS – JDBC-ODBC bridge



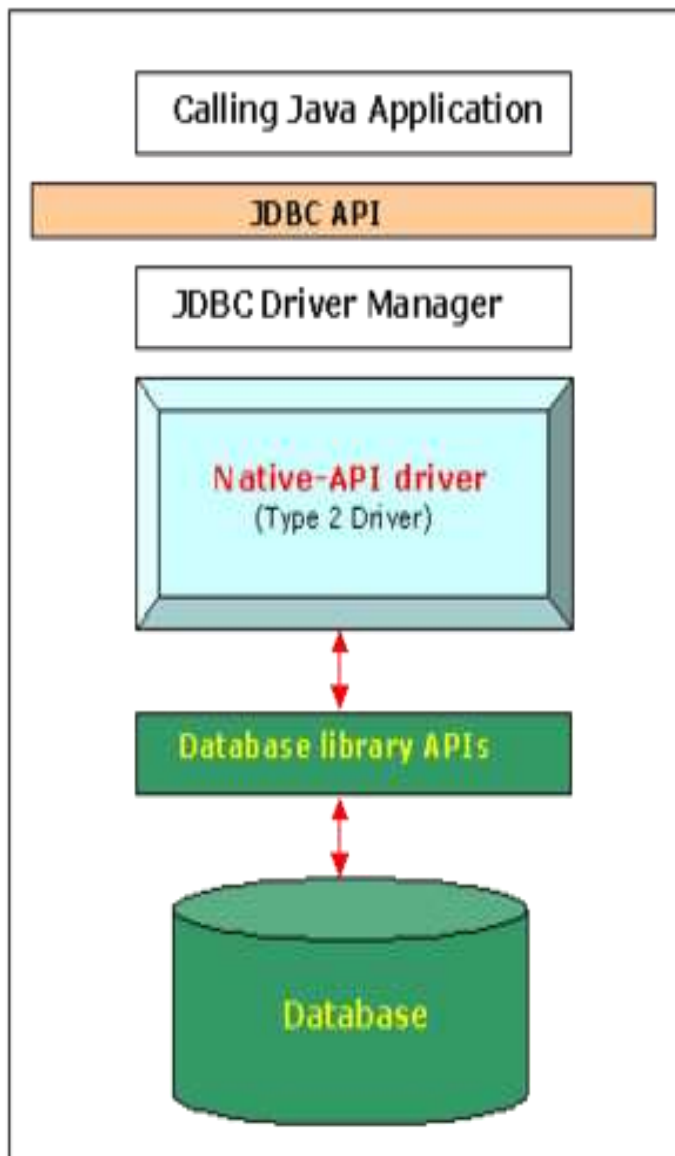
- Type 1 driver – JDBC-ODBC bridge
- Prednosti: Može da se poveže **sa skoro svakom** bazom podataka, zbog generičke prirode odbc driver
- Mane: Performanse - ne radi sa svim bazama podataka **podjednako brzo**. Sam driver mora biti na klijentskoj mašini. Za današenje web aplikacije **neprihvatljivo**



# Tipovi drajvera DBMS – JDBC-ODBC bridge

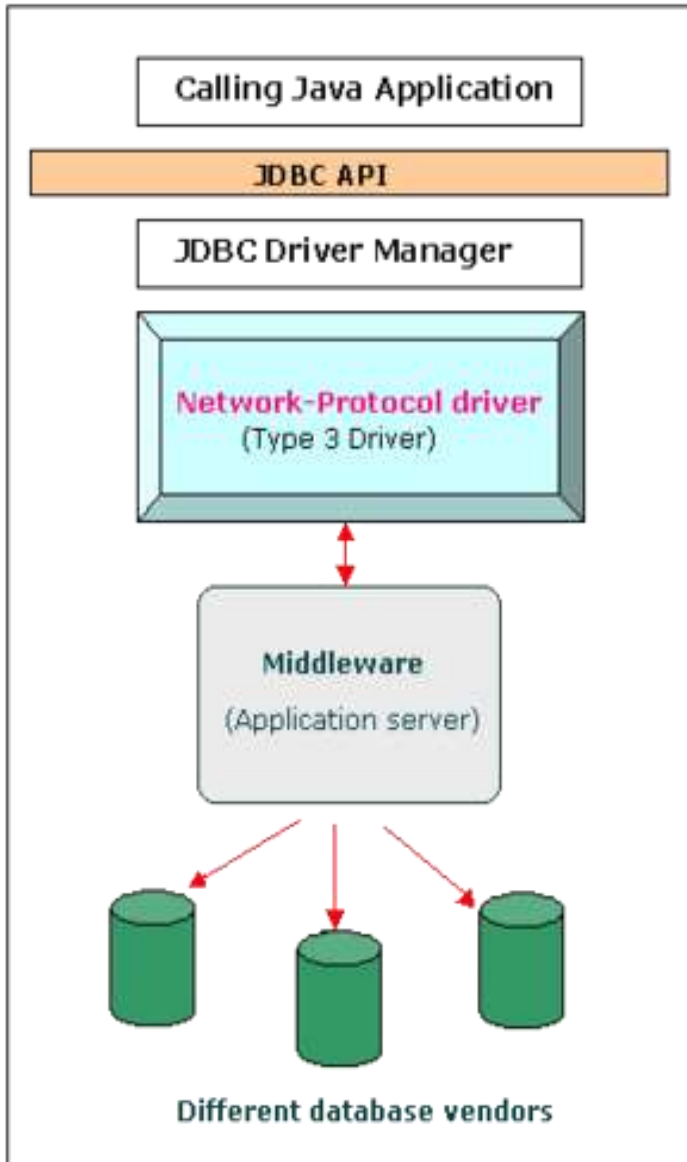
- ODBC Stands for Open Database Connectivity
- Introduced by Microsoft in 1992.

# Tipovi drajvera DBMS – Native-API driver



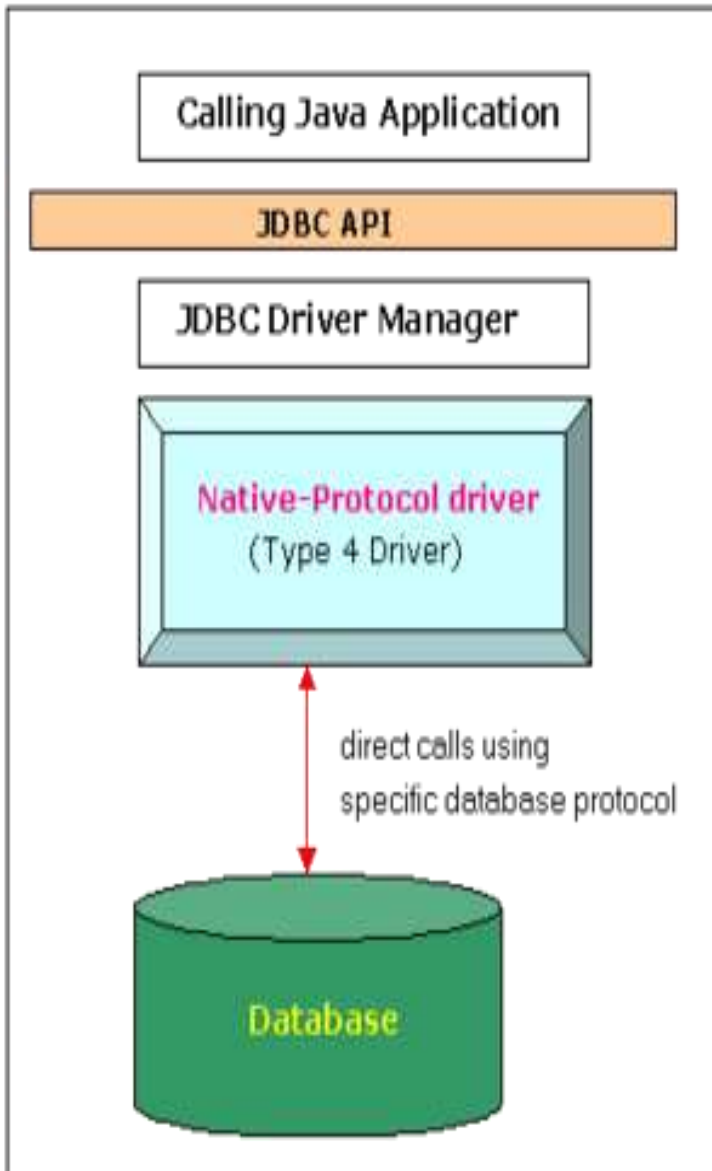
- Type 2 driver – Native-API driver
- Prednosti: Nema JDBC-ODBC bridge što povećava stepen performansi u radu sa bazom
- Mane: Nativna biblioteka **mora biti instalirana na klijentu. Ne postoje** native biblioteke za sve baze podataka. Zavisna je od platforme (operativnog sistema)

# Tipovi drajvera DBMS – Network-Protocol driver



- Type 3 driver – Network-Protocol driver (middleware driver)
- Prednosti: Driver napisan u javi. Jedan driver za sve baze. Middleware prevodi u komade koje razume baza. **Nema biblioteke** na klijentu
- Mane: Nekad je neophodno menjati middleware. Middleware komunikacija unosi dodatni overhead.

# Tipovi drajvera DBMS – Database-Protocol driver

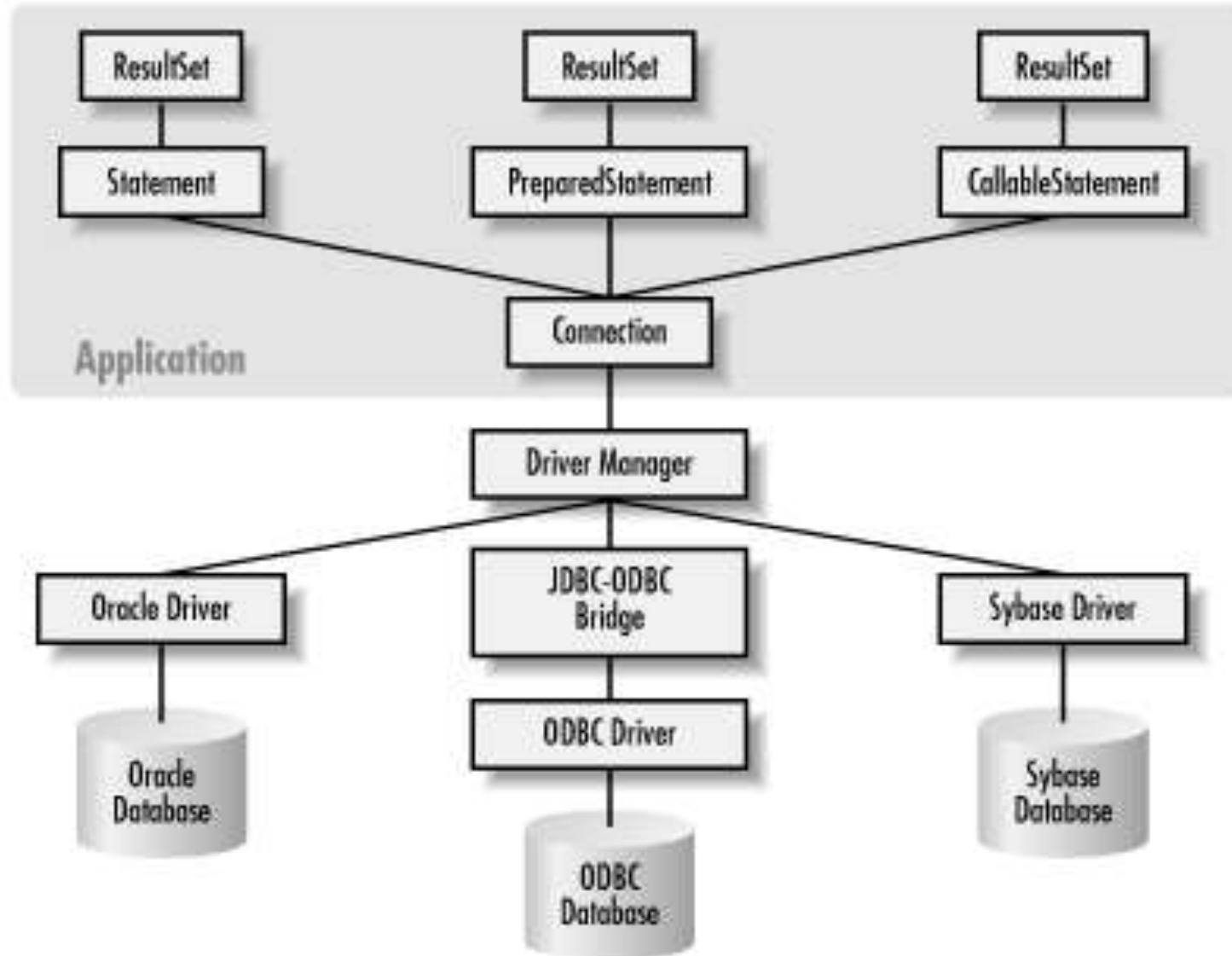


- Type 4 driver – Database-Protocol driver (Pure Java driver)
- Prednosti: Ugrađen u JVM klijenta (moguć debugg 😊). U odnosu na 1,2 nema ODBC. Bolji od 3 jer mu ne treba middleware
- Mane: Drajeveri za baze ponekad imaju zatvorene protokole. Pa ga ne možemo implementirati u Javi ili naći preuzeti implementaciju u Javi.
- Ovo se koristi u našoj aplikaciji

# Korišćenje baze u Javi

- Šta je potrebno kako bi koristili bazu ?
- Drajver za MySQL bazu podataka. Može i za druge baze (više drajvera)
- treba da je dostupan u CLASSPATH-u projekta;obično se stavlja u WEB-INF/lib folder projekta
- Skript (najčešće model + vrednosti) će da kreira bazu podataka i da je napuni inicijalnim podacima.

# Korišćenje baze u Javi



# Korišćenje baze u Javi - Inicijalizacija konekcije

- Inicijalizacija konekcije prema bazi kreiranje objekta klase Connection

```
private Connection conn = null;

public Connection getConnection() {
    if (conn == null) {
        // učitavanje MySQL drajvera
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            // konekcija
            conn = DriverManager.getConnection(url, username, password);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    return conn;
}
```

# Korišćenje baze u Javi - Inicijalizacija konekcije

- `forName` – poziv statičke metode klase `Class`. Statička metoda vraća inicijalizovan objekat klase koji odgovara parametru.
- Klasa `java.sql.DriverManager` koja upravlja učitavanjem drajvera i obezbeđuje podršku za otvaranje konekcija ka bazi podataka. Njegova osnovna funkcija je održavanje liste dostupnih drajvera i učitavanje odgovarajućeg drajvera na osnovu informacija iz URL adrese
- Struktura JDBC URL adrese je sledeća: `jdbc:<podprotokol>:<ime baze>`, gde podprotokol predstavlja konkretan mehanizam povezivanja na određenu bazu podataka, a koji može biti podržan od strane više drajvera. Komponente i sintaksa imena baze zavise od vrste podprotokola u upotrebi.

Baza podataka	JDBC URL
ODBC izvor podataka	<code>jdbc:odbc:DATA_SOURCE_NAME</code>
MySQL	<code>jdbc:mysql://SERVER[:PORT]/DATABASE_NAME</code>
Oracle	<code>jdbc:oracle:thin:@SERVER:PORT:INSTANCE_NAME</code>



# Korišćenje baze u Javi - Inicijalizacija konekcije

- Rezultat uspešnog uspostavljanja veze sa SUBP je inicijalizivani Connection objekat
- Connection interfejs ne može da imati svoje instance, ali se ovde radi o instanci.
- `java.sql.Connection` - predstavlja konkretnu konekciju sa bazom podataka kroz koju se šalju SQL iskazi

# Korišćenje baze u Javi – Statement obj

- Kada korisnik želi da izlista sve žanrove, bazi će biti poslat upit.
- Na osnovu rezultata, generiše se odgovor

@Override

```
public List<Zanr> findAll() {  
    List<Zanr> zanrovi = new ArrayList<Zanr>();  
  
    Statement stmt = null;  
    ResultSet rset = null;  
    try {  
        String sql = "SELECT id, naziv FROM zanrovi";  
        stmt = connectionManager.getConnection().createStatement();  
        rset = stmt.executeQuery(sql);  
        while (rset.next()) {  
            int index = 1;  
            Long id = rset.getLong(index++);  
            String naziv = rset.getString(index++);  
            Zanr zanr = new Zanr(id, naziv);  
            zanrovi.add(zanr);  
        }  
    } catch (Exception ex) {  
        ex.printStackTrace();  
    } finally {  
        try {stmt.close();} catch (Exception ex) {ex.printStackTrace();}  
        try {rset.close();} catch (Exception ex) {ex.printStackTrace();}  
        try {connectionManager.closeConnection();} catch (Exception ex) {ex.printStackTrace();}  
    }  
    return zanrovi;  
}
```

# Korišćenje baze u Javi – Statement obj

- `java.sql.Statement` - se ponaša kao kontejner za izvršavanje SQL upita kroz datu konekciju (kompletan string pretrage)
  - Ako bi SQL upit trebao da sadrži **where** klauzulu tada bi se on formirao na  
`String sql = "SELECT id, naziv FROM zanrovi WHERE id = " + idZanr + ";"`  
nije baš praktično ako where deo ima više parametara pretrage
- Sve operacije nad bazom podataka, pa tako i postavljanje upita, definišu se odgovarajućim SQL naredbama koje se šalju serveru. SQL naredba je, u okviru JDBC interfejsa, definisana Statement objektom.
- Za slanje upita serveru koristi se metoda `executeQuery`
- Rezultat ove metode je inicijalizovani objekat klase `ResultSet`, koji je namenjen za skladištenje rezultata upita. Rezultat upita se može pročitati pomoću ovog objekta.

# Korišćenje baze u Javi – Statement obj

- Objekat klase ResultSet

- Čitanje rezultata je operacija koja se odvija red-po-red u okviru tabele koja predstavlja rezultat. Za kretanje kroz tabelu rezultata koristi se koncept tekućeg reda. Tekući red se može pomerati isključivo od početka ka kraju tabele, bez preskakanja i bez više prolaza.
- Prvim pozivom metode next klase ResultSet tekući red će biti prvi red tabele rezultata. Metoda next vraća boolean vrednost koja označava da li novi tekući red postoji ili ne.
- Za tekući red rezultata pojedine vrednosti polja očitavaju se metodama getString, getInt, getDate, itd. (za svaki tip podatka u bazi postoji odgovarajuća metoda). Konverziju između tipova podataka baze i jezika Java obavlja JDBC drajver. Parametar getXXX metoda je redni broj kolone koja se očitava; redni brojevi počinju od 1, a ne od nula kako bi se očekivalo.
- Nakon prestanka korišćenja ResultSet objekta potrebno je pozvati njegovu metodu close radi oslobađanja resursa koje je taj rezultat upita zauzimao. Slično važi i za Statement objekat.

# Korišćenje baze u Javi – PreparedStatement obj

- INSERT, UPDATE ili DELETE mogu da koriste Statement objekat ili PreparedStatement objekat, koji se inicijalizuje na isti način kao i u prethodnom slučaju, osim što se poziva executeUpdate.
- Za razliku od metode executeQuery koja vraća ResultSet, metoda executeUpdate vraća samo jednu int vrednost koja predstavlja broj redova na koje je operacija uticala

@Override

```
public int save (Zanr zanr) {
    PreparedStatement stmt = null;
    int rezultat = 0;
    try {
        String sql = "INSERT INTO zanrovi (naziv) VALUES (?)";
        stmt = connectionManager.getConnection().prepareStatement(sql);
        int index = 1;
        stmt.setString(index++, zanr.getNaziv());
        rezultat = stmt.executeUpdate();
    } catch (Exception ex) {
        ex.printStackTrace();
    } finally {
        try {stmt.close();} catch (Exception ex) {ex.printStackTrace();}
        try {connectionManager.closeConnection();} catch (Exception ex) {ex.printStackTrace();}
    }
    return rezultat;
}
```

# Korišćenje baze u Javi – PreparedStatement obj

- `java.sql.PreparedStatement` - je objekat koji predstavlja prekompajliran Statement koji sadrži SQL
- Server samo jednom izvrši parsiranje i analizu SQL naredbe i formira plan izvršavanja
- U okviru naredbe njen promenljivi deo predstavljen upitnicima. Pre slanja definisati vrednosti upitnika.
- Praktično je ako where deo ima više parametara pretrage i ako su oni tipa Date ili Sting (metoda `setString` automatski eskejpuje karaktere koji se nalaze u Stringu a koji se mogu tumači kao specijalni karateri za sintaksu SQL naredbe )

```
@Override
public int save (Zanr zanr) {
    PreparedStatement stmt = null;
    int rezultat = 0;
    try {
        String sql = "INSERT INTO zanrovi (naziv) VALUES (?)";
        stmt = connectionManager.getConnection().prepareStatement(sql);
        int index = 1;
        stmt.setString(index++, zanr.getNaziv());
        rezultat = stmt.executeUpdate();
    } catch (Exception ex) {
        ex.printStackTrace();
    } finally {
        try {stmt.close();} catch (Exception ex) {ex.printStackTrace();}
        try {connectionManager.closeConnection();} catch (Exception ex) {ex.printStackTrace();}
    }
    return rezultat;
}
```

# Korišćenje baze u Javi – PreparedStatement obj

- PreparedStatement se može koristiti i za višestruki unos objekata
- Uzastopno izvršavanje istih SQL naredbi

```
@Override
public int[] save(ArrayList<Zanr> zanrovi) {
    PreparedStatement stmt = null;
    int [] result = new int [zanrovi.size()];
    try {
        String sql = "INSERT INTO zanrovi (naziv) VALUES (?)";
        stmt = connectionManager.getConnection().prepareStatement(sql);
        for (int i=0; i<zanrovi.size();i++) {
            int index = 1;
            stmt.setString(index++, zanrovi.get(i).getNaziv());
            result[i] = stmt.executeUpdate();
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    } finally {
        try {stmt.close();} catch (Exception ex) {ex.printStackTrace();}
        try {connectionManager.closeConnection();} catch (Exception ex) {ex.printStackTrace();}
    }
    return result;
}
```

# Korišćenje baze u Javi – PreparedStatement obj

- PreparedStatement se može korsitiiti selekciju sa where delom

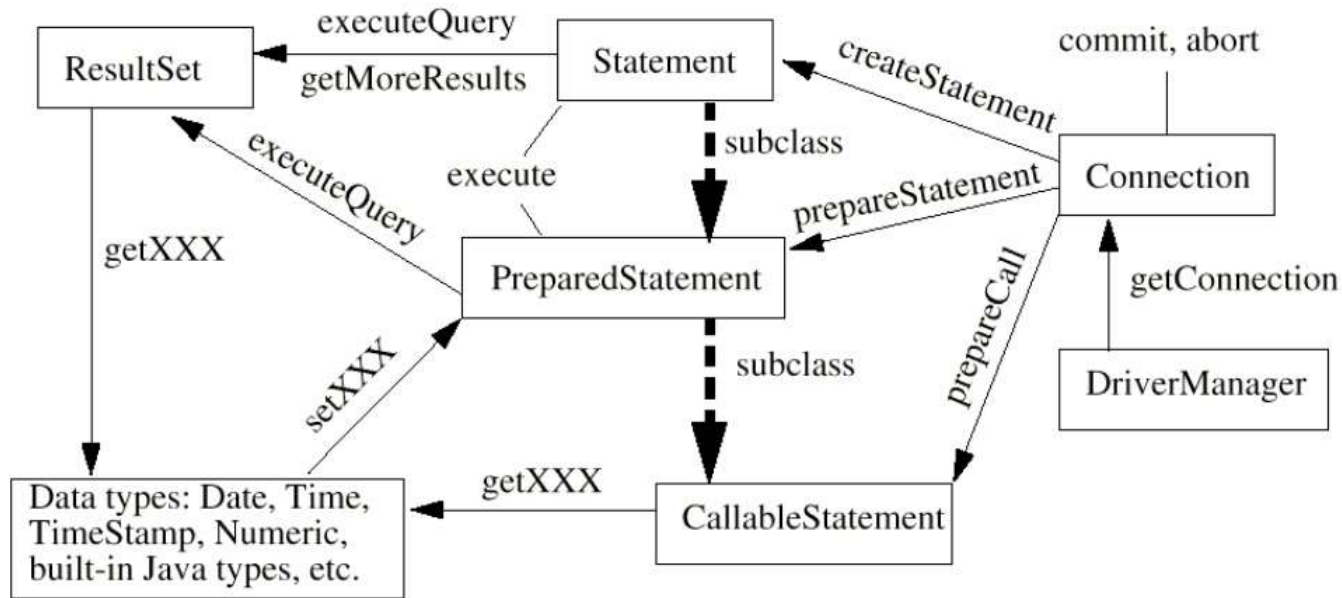
```
@Override
public Zanr findOne(Long id) {
    Zanr zanr = null;
    PreparedStatement stmt = null;
    ResultSet rset = null;
    try {
        String sql = "SELECT id, naziv FROM zanrovi WHERE id = ?";
        stmt = connectionManager.getConnection().prepareStatement(sql);
        int index = 1;
        stmt.setLong(index++, id);
        rset = stmt.executeQuery();

        if (rset.next()) {
            zanr = new Zanr(id, rset.getString(2));
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    } finally {
        try {stmt.close();} catch (Exception ex) {ex.printStackTrace();}
        try {rset.close();} catch (Exception ex) {ex.printStackTrace();}
        try {connectionManager.closeConnection();} catch (Exception ex) {ex.printStackTrace();}
    }
    return zanr;
}
```



# JDBC CLASSES

## JDBC Class Diagram



# JDBC JAVA CONVERSION

SQL data type	Java data type	
	Simply mappable	Object mappable
CHARACTER		String
VARCHAR		String
LONGVARCHAR		String
NUMERIC		java.math.BigDecimal
DECIMAL		java.math.BigDecimal
BIT	boolean	Boolean
TINYINT	byte	Integer
SMALLINT	short	Integer
INTEGER	int	Integer
BIGINT	long	Long
REAL	float	Float
FLOAT	double	Double
DOUBLE PRECISION	double	Double
BINARY		byte[]
VARBINARY		byte[]
LONGVARBINARY		byte[]
DATE		java.sql.Date
TIME		java.sql.Time
TIMESTAMP		java.sql.Timestamp

# Korišćenje baze u Javi – Connection pool

- Data access pattern čija je uloga da smanji broj “skupih” operacija kreiranja veza ka bazi podataka.
- Neka vrsta keša za konekcije. Ne pravi se nova konekcija dok se ne potroše postojeće. Postavlja se maksimalni i minimalni broj konekcija. Postoji poseban proces u biblioteci koji prati konekcije koje su neaktivne i sam ih oslobađa.
- Third-party biblioteka je bolje rešenje nego praviti neku svoju imlementaciju.
- Posebni algoritmi za povećavanje i smanjivanje konekcija.

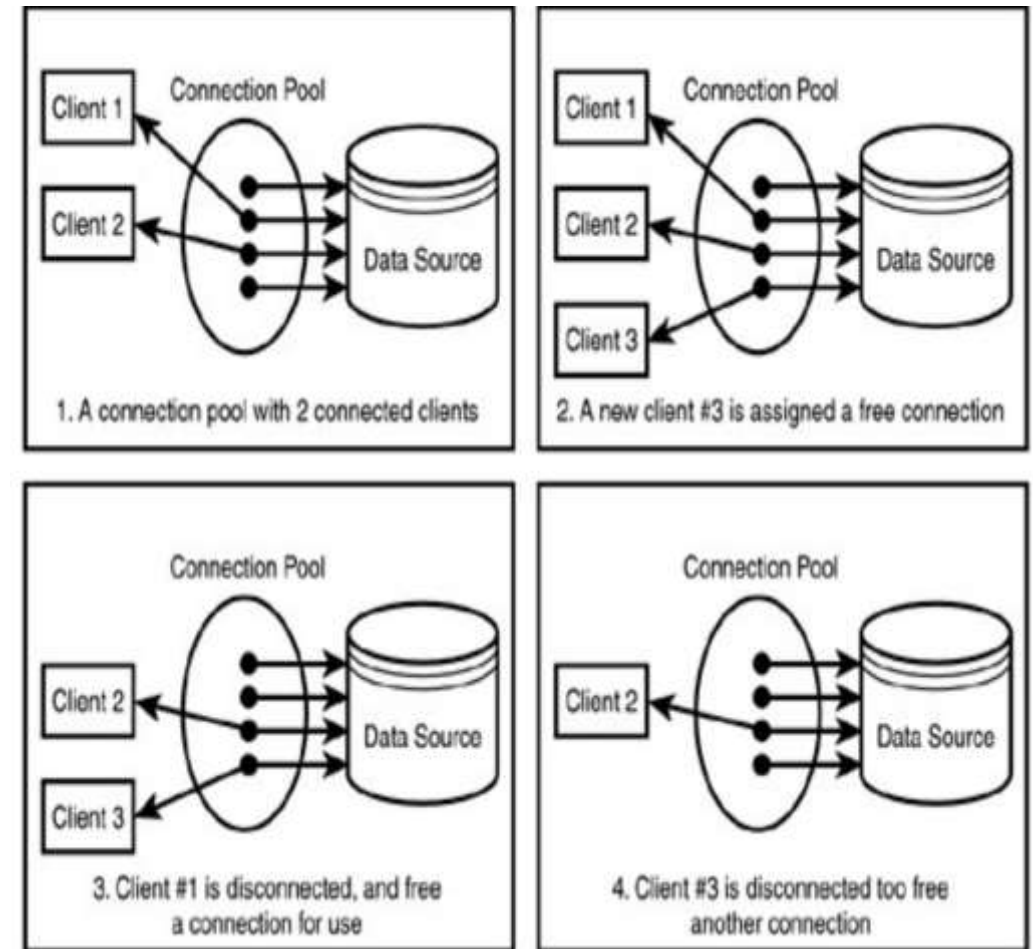
	100 Iterations	100 Iterations	1000 Iterations	3000 Iterations
Pooling	547 ms	<10 ms	47 ms	31 ms
Non-Pooling	4859 ms	4453 ms	43625 ms	134375 ms

# Korišćenje baze u Javi – Connection pool

- Creating a new connection for each user can be time consuming (often requiring multiple seconds of clock time), in order to perform a database transaction that might take milliseconds.
- Opening a connection per user can be unfeasible in a publicly-hosted Internet application where the number of simultaneous users can be very large.
- Accordingly, developers often wish to share a "pool" of open connections between all of the application's current users.
- The number of users actually performing a request at any given time is usually a very small percentage of the total number of active users, and during request processing is the only time that a database connection is required

# Korišćenje baze u Javi – Connection pool

	100 Iterations	100 Iterations	1000 Iterations	3000 Iterations
Pooling	547 ms	<10 ms	47 ms	31 ms
Non-Pooling	4859 ms	4453 ms	43625 ms	134375 ms



# Korišćenje baze u Javi – Connection pool

- Apache DBCP je primer jedne takve biblioteke  
<https://commons.apache.org/proper/commons-dbcp/>

```
/**
 * The default cap on the number of "sleeping" instances in the pool.
 * @see #getMaxIdle
 * @see #setMaxIdle
 */
public static final int DEFAULT_MAX_IDLE = 8;
/**
 * The default minimum number of "sleeping" instances in the pool
 * before before the evictor thread (if active) spawns new objects.
 * @see #getMinIdle
 * @see #setMinIdle
 */
public static final int DEFAULT_MIN_IDLE = 0;
/**
 * The default cap on the total number of active instances from the pool.
 * @see #getMaxActive
 */
public static final int DEFAULT_MAX_ACTIVE = 8;
```

# Korišćenje baze u Javi – Connection pool

- Apache DBCP je primer jedne takve biblioteke  
<https://commons.apache.org/proper/commons-dbcp/>

```
public class DBCPDataSource {  
    private static BasicDataSource ds = new BasicDataSource();  
    static {  
        ds.setUrl("jdbc:h2:mem:test");  
        ds.setUsername("user");  
        ds.setPassword("password");  
        ds.setMinIdle(5);  
        ds.setMaxIdle(10);  
        ds.setMaxOpenPreparedStatements(100);  
    }  
    public static Connection getConnection() throws SQLException {  
        return ds.getConnection();  
    }  
    private DBCPDataSource(){ }  
}
```

# Korišćenje baze u Javi – Connection pool

- HikariCP je primer jedne takve biblioteke  
<https://github.com/brettwooldridge/HikariCP>

```
public class HikariCPDataSource {  
    private static HikariConfig config = new HikariConfig();  
    private static HikariDataSource ds;  
    static {  
        config.setJdbcUrl("jdbc:h2:mem:test");  
        config.setUsername("user");  
        config.setPassword("password");  
        config.addDataSourceProperty("cachePrepStmts", "true");  
        config.addDataSourceProperty("prepStmtCacheSize", "250");  
        config.addDataSourceProperty("prepStmtCacheSqlLimit", "2048");  
        ds = new HikariDataSource(config);  
    }  
    public static Connection getConnection() throws SQLException {  
        return ds.getConnection();  
    }  
    private HikariCPDataSource(){}  
}
```