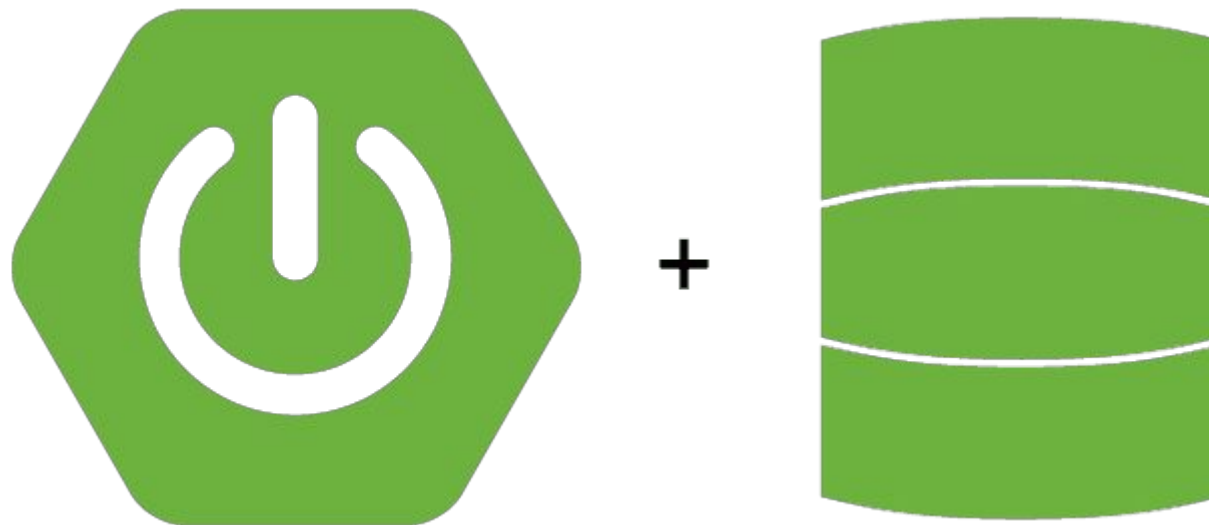


# SPRING DATA JPA – JPQL



**UP ASI  
Bureau E204**

# Plan du Cours

- JPQL
- Requêtes **SELECT, UPDATE, DELETE, INSERT** avec JPQL et Native Query
- **@Query**
- **@Modifying**
- **@Param**

# JPQL

- **JPQL** = Java Persistence Query Language
- JPQL peut être considéré comme une version orientée objet de SQL.
- En JPQL, on sélectionne des objets d'une entité (une classe annotée par @Entity) en utilisant les noms des attributs et le nom de la classe **de l'entité** en question et non plus ceux de la table de base de données et ses colonnes.

# SELECT

- Ces méthodes permettent de récupérer les entreprises avec une adresse donnée :
- **JPQL :**

```
@Query("SELECT e FROM Entreprise e WHERE e.adresse =:adresse")  
List<Entreprise> retrieveEntreprisesByAdresse(@Param("adresse") String adresse);
```

C'est équivalent à :

```
@Query("SELECT e FROM Entreprise e WHERE e.adresse = ?1")  
List<Entreprise> retrieveEntreprisesByAdresse(String adresse);
```

Supposons que nous avons mapper l'entité Entreprise avec **la table associé T\_Entreprise**

- **Native Query (SQL et non JPQL) :**

```
@Query(value = "SELECT * FROM T_Entreprise e WHERE e.entreprise_adresse = :adresse",  
nativeQuery = true)  
List<Entreprise> retrieveEntreprisesByAdresse(@Param("adresse") String adresse);
```

C'est équivalent à :

```
@Query(value = "SELECT * FROM T_Entreprise e WHERE e.entreprise_adresse = ?1" ,  
nativeQuery = true)  
List<Entreprise> retrieveEntreprisesByAdresse( String adresse);
```

# SELECT

- Ces méthodes permettent de récupérer les entreprises qui ont une équipe avec une spécialité donnée :

- JPQL :**

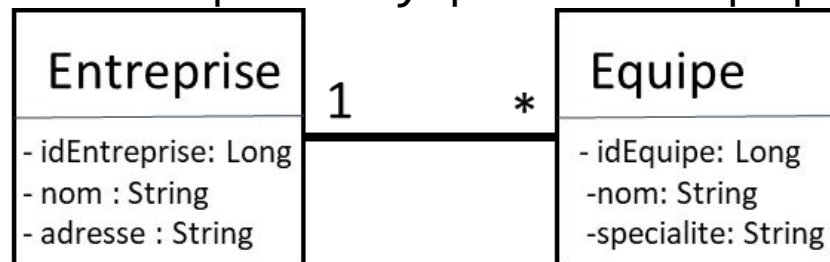
```
@Query("SELECT entreprise FROM Entreprise entreprise , Equipe equipe where  
entreprise.id = equipe.entreprise.id and equipe.specialite =:specialite")
```

```
List<Entreprise> retrieveEntreprisesBySpecialiteEquipe(@Param("specialite")  
String specialite);
```

- Native Query (SQL et non JPQL) :**

```
@Query(value = "SELECT * FROM T_ENTREPRISE entreprise INNER JOIN T_EQUIPE  
equipe ON entreprise.ENTREPRISE_ID = equipe.ENTREPRISE_ENTREPRISE_ID where  
equipe.EQUIPE_SPECIALITE =:specialite", nativeQuery = true)
```

```
List<Entreprise> retrieveEntreprisesBySpecialiteEquipe(@Param("specialite")  
String specialite);
```



# SELECT

- Cette méthode permet d'afficher les projets qui ont un coût supérieur à coût donné et une technologie donnée.
- **JPQL :**

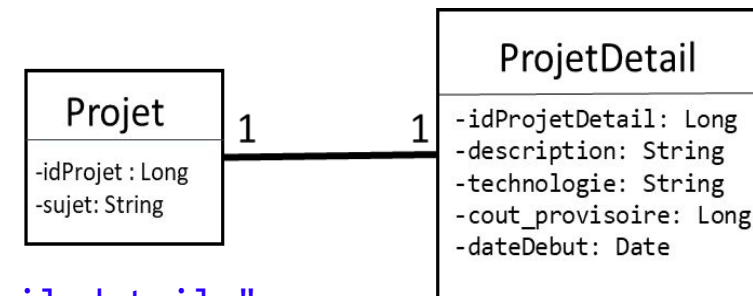
```
@Query("SELECT projet FROM Projet projet INNER JOIN ProjetDetail detail "  
      + "ON detail.projet = projet.idProjet WHERE detail.technologie =:technologie "  
      + "and detail.cout_provisoire >:cout_provisoire")
```

```
List<Projet> retrieveProjetsByCoutAndTechnologie(@Param("technologie") String technologie,  
      @Param("cout_provisoire") Long cout_provisoire);
```

C'est équivalent à :

```
@Query("SELECT projet FROM Projet projet INNER JOIN ProjetDetail detail "  
      + "ON detail.projet = projet.idProjet WHERE detail.technologie = ?1 "  
      + "and detail.cout_provisoire > ?2")
```

```
List<Projet> retrieveProjetsByCoutAndTechnologie(String technologie, Long cout_provisoire);
```



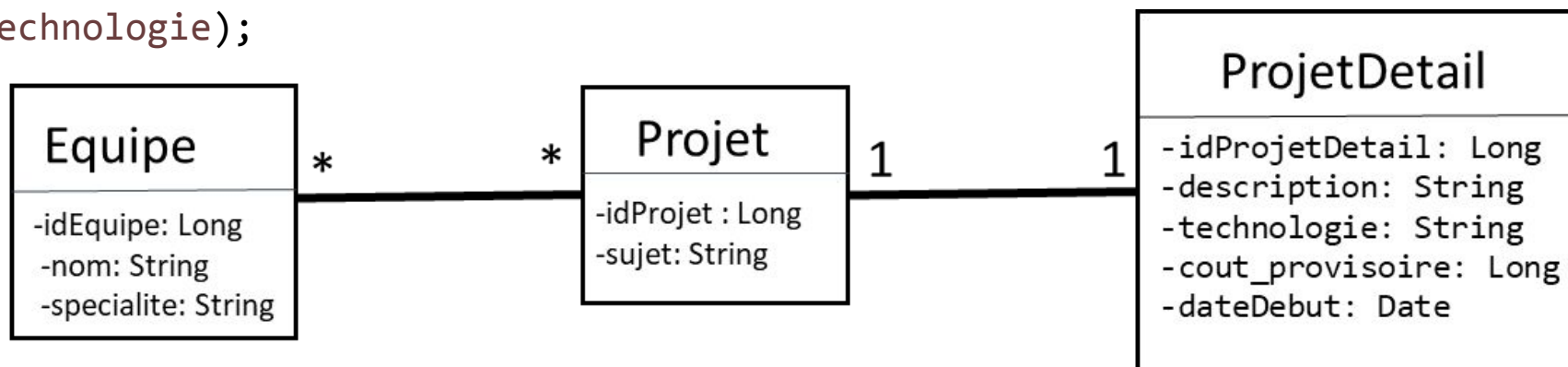
# SELECT

- Cette méthode permet d'afficher les équipes qui travaillent sur une technologie donnée.
- JPQL :

```
@Query("SELECT equipe FROM Equipe equipe"  
      + " INNER JOIN equipe.projets projet"  
      + " INNER JOIN ProjetDetail detail"  
      + " ON detail.projet = projet.idProjet"  
      + " where detail.dateDebut > current_date"  
      + " and detail.technologie =:technologie")
```

```
List<Equipe> retrieveEquipesByProjetTechnologie(@Param("technologie")
```

```
String technologie);
```



# UPDATE

- Si nous souhaitons faire un **UPDATE, DELETE et INSERT**, nous devons ajouter l'annotation **@Modifying** pour activer la modification de la base de données.
- Cette méthode permet de mettre à jour l'adresse de l'entreprise.

- **JPQL :**

**@Modifying**

```
@Query("update Entreprise e set e.adresse = :adresse where e.idEntreprise = :idEntreprise")
```

```
int updateEntrepriseByAdresse(@Param("adresse") String adresse,  
@Param("idEntreprise")
```

```
Long idEntreprise);
```

- **Native Query (SQL et non JPQL) :**
- A compléter ensemble.



# DELETE

- Cette méthode permet de supprimer les entreprises qui ont une adresse donnée :

- **JPQL :**

**@Modifying**

```
@Query("DELETE FROM Entreprise e WHERE e.adresse= :adresse")
```

```
int deleteFournisseurByCategorieFournisseur(@Param("adresse") String adresse);
```

C'est équivalent à :

**@Modifying**

```
@Query("DELETE FROM Entreprise e WHERE e.adresse= ?1")
```

```
int deleteFournisseurByCategorieFournisseur(String adresse);
```

- **Native Query (SQL et non JPQL) :**

- A compléter ensemble.

# INSERT

- Cette méthode permet d'insérer des projets dans la table T\_Projet:
- **JPQL** : Nous utilisons Spring Data JPA. Or INSERT ne fait pas partie des spécifications JPA. Donc, nous sommes obligés d'utiliser les Natives Query pour le INSERT.
- **Pas de JPQL pour les requêtes INSERT.**
- **Native Query (SQL et non JPQL) :**  
@Modifying  
@Query(value = "INSERT INTO T\_Projet(projet\_sujet) VALUES (:projetsujet)",  
nativeQuery = true)  
void insertProjet(@Param("projetsujet") String projetsujet);

# Exercice

- Faites une requête permettant de sélectionner ..... selon l'étude de cas en SQL.
- Faites la même requête en JPQL.

# Conclusion

- Ces requêtes JPQL seront appliquées lors du **TP .... selon l'étude de cas** .
- Ce TP permettra de manipuler :
  - Les Entities,
  - Les Associations,
  - Le CrudRepository,
  - JPQL,
  - Spring MVC REST,
  - Spring Core (IoC : Injection de Dépendances).

# SPRING DATA JPA – JPQL

Si vous avez des questions, n'hésitez pas à nous contacter :

**Département Informatique**  
**UP Architectures des Systèmes d'Information**  
Bureau E204