



## Native queries - Spring Data JPA

Usar **recursos prontos do SDJ** como *named queries* e *métodos comuns já implementados* é muito útil, mas tem momentos onde é **necessário que uma query seja escrita por nós mesmos**. Seja porque a *named query* é gigantesca e não vale a pena mantê-la, seja para garantir uma melhor performance, legibilidade e segurança e etc...

Sempre aparece motivos para que essa situação aconteça, e é aí que as **native queries** ajudam a gente. É muito simples implementar um comportamento **nativo de sql ou JPQL** em um método do nosso repositório, basta usar a anotação **@Query** e descrever o comportamento, quando o método for chamado a JPQL será executada:

```
@Repository
public interface GoalRepository extends JpaRepository<Goal, Long> {

    Page<Goal> findAllByUserId(Long userId, Pageable pageable);

    @Query("FROM Goal g WHERE g.dateHistory.expectedFinalizationDate BETWEEN :startOfDay AND :endOfDay " +
        "AND g.status <> 'COMPLETED' ")
    List<Goal> findGoalsInDeadline(LocalDateTime startOfDay, LocalDateTime endOfDay);
}
```

**JPQL** é um jeito de escrever queries de sql com uma sintaxe mais voltada pra orientação a objetos, então basta usar o **nome das Entidades do projeto** para se referenciar as entidades do banco de dados, **navegar entre seus atributos(colunas)** também é mais simples, enquanto **palavras chaves se mantém quase que inalteradas** nas queries nativas e em jpql:

```
@Repository
interface PaymentRepository : JpaRepository<PaymentModel, Long> {

    @Query("SELECT SUM(dp.amount) " +
        "FROM PaymentModel dp " +
        "WHERE dp.walletId = :walletId " +
        "AND dp.period LIKE :period " +
        "AND cast(dp.paymentDateTime as LocalDate) = :todayDate")
    fun totalPaymentsUntilNowByPeriod(walletId: UUID, todayDate: LocalDate?, period: Period) : BigDecimal?
}

@Entity
@Table(name = "payment")
data class PaymentModel(

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    val id: Long?,

    @Enumerated(EnumType.STRING)
    @Column
    val period: Period,

    @Column
    val amount: BigDecimal,

    @Column
    val paymentDateTime: LocalDateTime,

    @ManyToOne
    var wallet: WalletModel? = null
)
```

JPQLs garantem um pouco mais de flexibilidade na query (**como nessa de cima onde eu pude fazer um cast de um LocalDateTime para LocalDate**), uma legibilidade maior em relação a named queries em consultas que são mais complexas.