**1 Sử dụng cấu trúc đặc tả property để query**. Ex A:

List<Person> findByAddressZipCode(ZipCode zipCode);

-Cấu trúc trên được chia thành 2 phần: 1: findBy, findFistby, … định nghĩa mục đích(subject) của query

Phần 2 còn lại, là các predicate(điều kiện) của query , ngăn cách bởi by.

* Quy tắc tìm cấu trúc đúng của câu truy vấn trên với thuộc tính thích hợp **x.address.zipCode()**)

Là sẽ từ đầu cuối (tail) của phương thức trên, đầu quét quét về bên trái sẽ dừng tại chữ cái camelcase đầu tiên để tìm ra property.

1: tìm thấy : AddressZip.code() -> fail -> chuyển đầu quét đến chữ cái camel kế tiếp.

2: tìm thấy : Address.ZipCode() -> đúng -> tiếp tục phân tích method name tìm ra cây thuộc tính của đối tượng cho đến khi gặp **delimiter**: **BY**.

* Vẫn có trường hợp hệ thống ra KQ cuối sai khi mà trong đối tuọng cũng có thuộc tính: private Type AddressZip;
* Để tránh trường hợp này ta tên phương thức nên là:

**findByAddress\_ZipCode(ZipCode zipCode);** để Jpa biết chắc chắn chỗ nào cần tách ra.

* Nhưng hầu hết đều không có lỗi nếu code đúng quy tắc camel case cho tên phương thức. **không khuyến khích dùng underScore**

**. 2. Using**Pageable**,**Slice**, and**Sort**in query methods**

Những param này không cần phải chỉ rõ trong tên phương thức vẫn có thể hoạt động khi là đối số của phương thức.

Ví dụ:

Page<User> findByLastname(String lastname, Pageable pageable);

**Slice<User>** findByLastname(String lastname, Pageable pageable);

List<User> findByLastname(String lastname, **Sort sort**);

Khi khi báo mannual các phương thức trên nhưng không muốn sort hay pageable nữa, hãy để: Sort.unsorted() / Pageable.unpaged()

-Khi trả ra Page<User> findByLastname, có thể tốn tài nguyên và hiệu năng **khi list lớn** vì KQ sẽ biết tổng số bn phần tử + các trang có sẵn(có phần tử)

=> thay vì dùng như vậy ta sử dụng Slice<Type> để trả về KQ, slice chỉ trẢ KQ của **page index** được truyền vào.

=> nếu không cần phân trang, ta chỉ cần pass đối số **sort** vào method, thay vì a Page:

Sort sort = Sort.by("firstname").ascending()

.and(Sort.by("lastname").descending())

3. **Limiting the result size of a query with**Top**and**First

Có thể giới hạn số lượng KQ bằng việc sd **top(**X row cuối**) or fist (**X row đầu**)**

User findFirstByOrderByLastnameAsc(); // 1 row đầu

User findTopByOrderByAgeDesc(); // 1 row cuối

Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);

Slice<User> find**Distinct**Top3ByLastname(String lastname, Pageable pageable);

List<User> findFirst10ByLastname(String lastname, Sort sort);

List<User> findTop10ByLastname(String lastname, Pageable pageable);

* Có thể dùng feature **distinct** trong các query trên
* Có thể sd Java - Optional để cover các KQ trên.
* Có thể sd Java – streamerable, java Stream để cover các KQ trên. Để filter, map, concatenate:
* interface PersonRepository extends Repository<Person, Long> {
* Streamable<Person> findByFirstnameContaining(String firstname);
* Streamable<Person> findByLastnameContaining(String lastname);
* }
* Streamable<Person> result = repository.findByFirstnameContaining("av")
* .and(repository.findByLastnameContaining("ea"));
* Có thể sd page, slices

-Nếu sử dụng wraper type(Optiona, Collection, Page) cho các phương thức, chúng **sẽ không bao giờ null chỉ empty nếy k có data,**  nếu k có wrapper, có thể null.