modelowanie agregatu opiera się na wyznaczanie granicy spójności transakcji bizneswoej, natychmiastowej

agregat zawiera informacje tylko na cele swoich inwariantów (pilnowaczy spójnej zmiany
jego stanu, warunki "commita" transakcji biznesowej), plus id.

agregat zwiera informacje zmieniające się razem

informacje: **identyfikatora klienta**, **wybrana kategoria,** które niesie rozkaz utwórz problem to informacje, które nie są inwariantami, nie są na cele bronienia spójnej zmiany stanu agregata, jednak służą na cele późniejszego readmodela - pokaż zakończone rozwiązania dla id klienta posortowane po kategorii. W związku z tym należy te informacje zapisać, używamy anemicznej encji o tożsamości takiej samej jak agregat, ale bez relacji w orm i na bazie

Rule: Model True Invariants in Consistency Boundaries

When trying to discover the Aggregates in a **Bounded Context (2)**, we must understand the model's true invariants. Only with that knowledge can we determine which objects should be clustered into a given Aggregate.

An invariant is a business rule that must always be consistent. There are different kinds of consistency. One is *transactional consistency*, which is considered immediate and atomic. There is also *eventual consistency*. *When discussing invariants, we are referring to transactional consistency.* We might have the invariant

```
c = a * b
```

Therefore, when `a` is 2 and `b` is 3, `c` must be 5. According to that rule and conditions, if `c` is anything but 5, a system invariant is violated. To ensure that `c` is consistent, we design a boundary around these specific attributes of the model:

```
AggregateType1 {
    int a;
    int b;
    int c;
    operations ...
}
```

The consistency boundary logically asserts that everything inside adheres to a specific set of business invariant rules no matter what operations are performed. The consistency of everything outside this boundary is irrelevant to the Aggregate. Thus, *Aggregate* is synonymous with *transactional consistency boundary*. (In this limited example, `AggregateType1` has three attributes of type `int`, but any given Aggregate could hold attributes of various types.)

When employing a typical persistence mechanism, we use a single transaction[2] to manage consistency. When the transaction commits, everything inside one boundary must be consistent. *A properly designed Aggregate is one that can be modified in any way required by the business with its invariants completely consistent within a single transaction.* And a properly designed Bounded Context modifies only one Aggregate instance per transaction in all cases. What is more, *we cannot correctly reason on Aggregate design without applying transactional analysis.*

Limiting modification to one Aggregate instance per transaction may sound overly strict. However, it is a rule of thumb and should be the goal in most cases. It addresses the very reason to use Aggregates.

Whiteboard Time

- List on your whiteboard all large-cluster Aggregates in your system.
- Make a note next to each of those Aggregates why it is a large cluster and any potential problems caused by its size.
- Next to that list, name any Aggregates that are modified in the same transaction with others.
- Make a note next to each of those Aggregates whether true or false invariants caused the formation of poorly designed Aggregate boundaries.

The fact that Aggregates must be designed with a consistency focus implies that the user interface should concentrate each request to execute a single command on just one Aggregate instance. If user requests try to accomplish too much, the application will be forced to modify multiple instances at once.

Therefore, Aggregates are chiefly about consistency boundaries and not driven by a desire to design object graphs. Some real-world invariants will be more complex than this. Even so, typically invariants will be less demanding on our modeling efforts, making it possible to *design small Aggregates.*

Granica agregatu - odkryta by wiedzieć jaki zestaw zmian informacji ma się spersystować w bazie danych sposób spójny natychmiastowo i transakcyjnie. Tak by system nie był w stanie niespójnym.

By bronić tej spójnej zmiany danych, w agregacie enkapsulujemy dane których zestaw symbolizuje jakiś domenowy stan i które zawsze zmieniają się razem ich spójnej zmiany bronią inwarianty