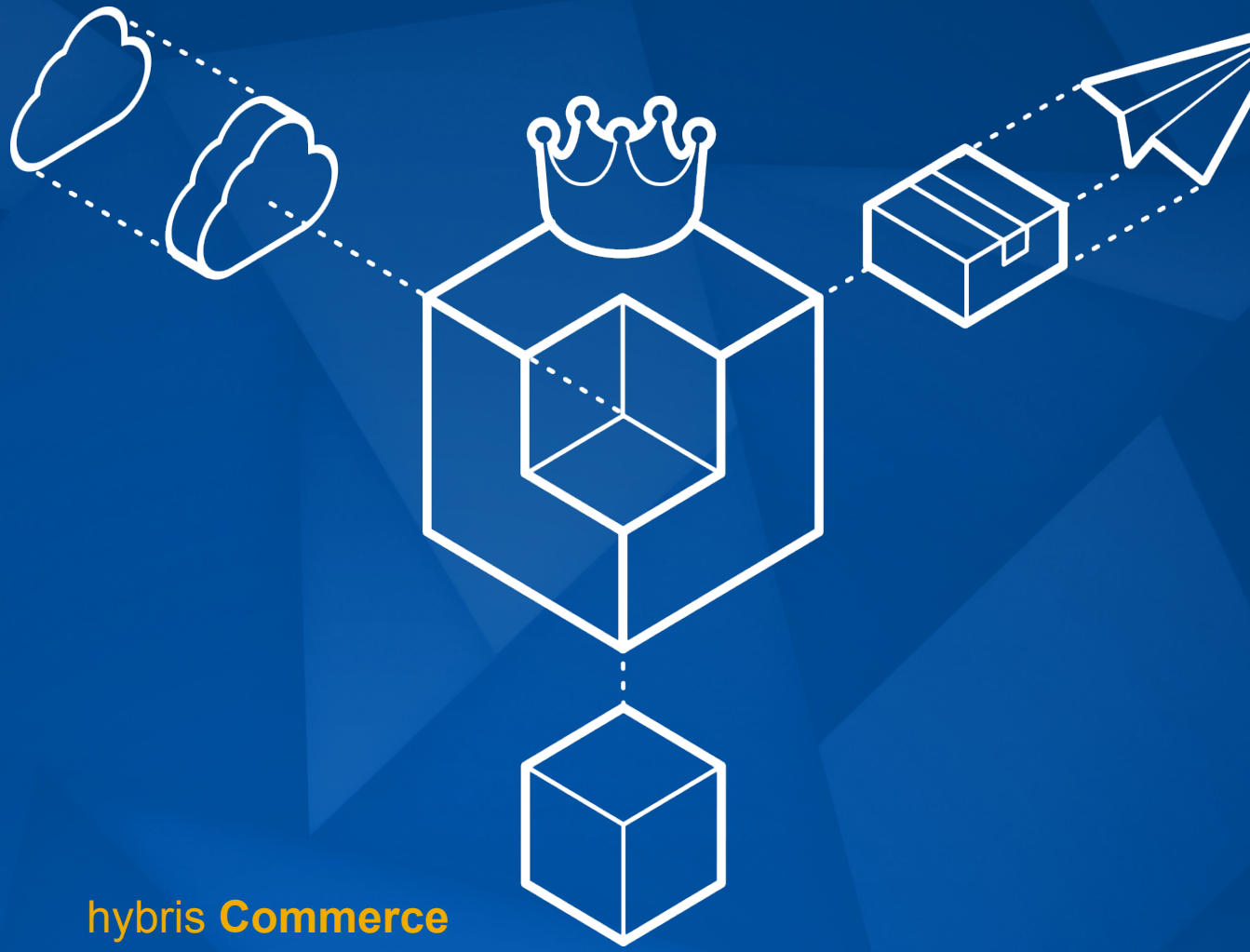
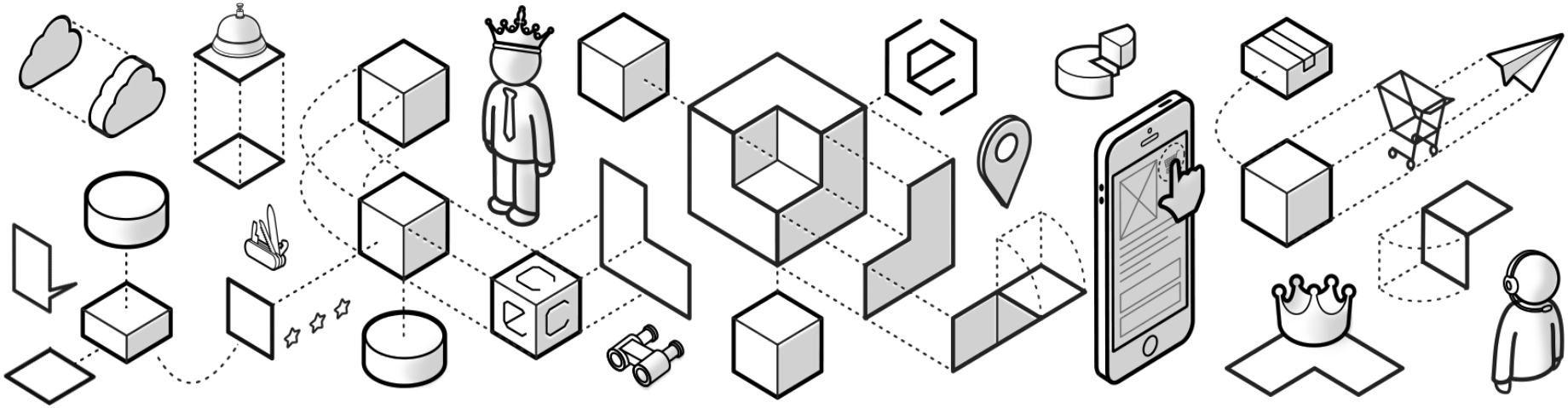
hybris software

An SAP Company

Flexible Search



hybris **Commerce Developer** Training – Part I

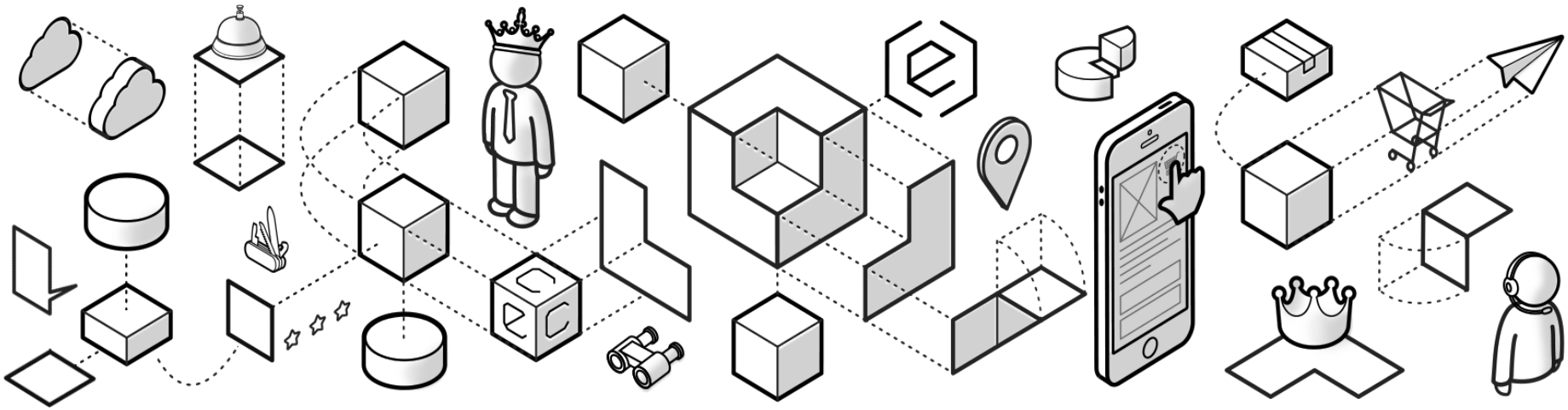


Overview

Overview
Syntax
API Examples
Flexible Search Alternatives

Overview

- SQL-like syntax
- Abstracts a database query into a hybris Item query
- Returns a list of objects (hybris items)
- Makes properties easily queryable
- Is translated into native SQL statements on execution
- Allows nearly every feature of SQL SELECT statements
- Queries go through cache



Syntax

Overview
Syntax
API Examples
Flexible Search Alternatives

Syntax

- Basic Syntax:

```
SELECT <selects> FROM {types} (where <conditions>)
   ?(ORDER BY <order>)?
```

- Mandatory:

```
SELECT  <selects>
FROM {types}
```

- Optional:

```
where <conditions>
ORDER BY <order>
```

- SQL Command / Keywords:

  ASC, DESC, DISTINCT, AND, OR, LIKE, LEFT JOIN, CONCAT, ..

Query examples

- Basic query
  - Special case: returns Car object instead of PK value

```
SELECT {pk} FROM {Car}
```

- Simple queries

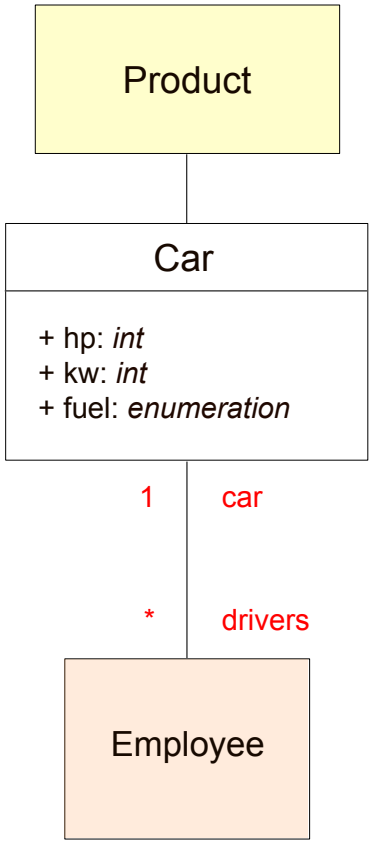
```
SELECT {code},{hp} FROM {Car}
```

- Single type queries
  - Returns only Product items, not subtypes

```
SELECT {code} FROM {Product!}
```

- Joins

```
SELECT {c.code},{e.uid} FROM {
  Car as c JOIN Employee as e
                        ON {c.pk} = {e.car}
          } WHERE {e.uid} LIKE '%Columbo'
```



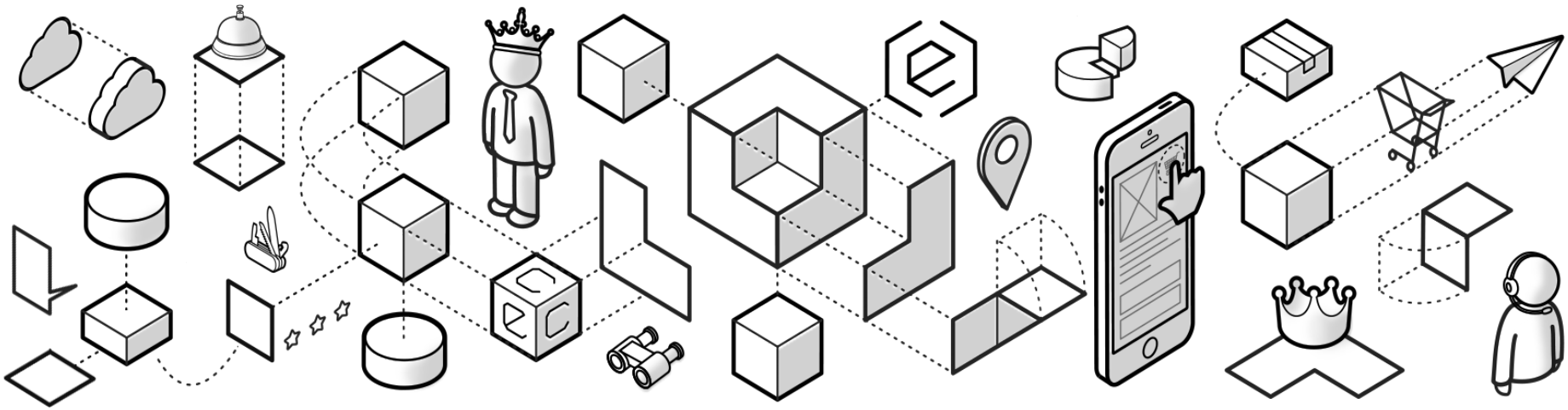
More query examples

- Inner queries:

```
SELECT {c.code} FROM {Car as c}
   WHERE {c.mechanic} IN
   ({{
      SELECT {pk} FROM {Employee}
         WHERE {uid} LIKE ‘%Tesla’
    }})
```

- Parametrized queries:

```
SELECT count(*) FROM {Car}
   WHERE {hp} > ?hpMin
   AND   {hp} < ?hpMax
```



API Examples

Overview
Syntax
API Examples
Flexible Search Alternatives

Querying with parameters

- Parameter substitution references map keys as *?key*

```
String fsq = "SELECT {PK} FROM {Car} WHERE {mechanic} = ?mechanic”;
FlexibleSearchQuery query = new FlexibleSearchQuery(fsq);
query.addQueryParameter("mechanic", "Nikola Tesla");
SearchResult<CarModel> result =
   getFlexibleSearchService().search( query );
List<CarModel> cars = result.getResult();
```

Querying for other types than Models

- The data type returned by a query must be set if not a hybris item

```
String fsq = "SELECT COUNT( {PK} ) FROM {Car}”
           + "WHERE {Car.warrantyExpiry} < '2015-12-01 0:00:00.0'";
FlexibleSearchQuery query = new FlexibleSearchQuery( fsq );
query.setResultClassList( Arrays.asList( Integer.class ) );
SearchResult<Integer> result =
   getFlexibleSearchService().search( query );
List<Integer> carsCount = result.getResult();
```

Querying against today's date • Caching Considerations

- When comparing with today's date, truncate date value
  - Every Flexible Search query is cached, but using the current date — which changes every millisecond — prevents use of the query cache
  - Truncate date to as needed — for instance to nearest day:

```
String fsq = "SELECT {PK} FROM {Car} WHERE {Car.warrantyExpiry} < ?today";
final Calendar cal = Calendar.getInstance();
   cal.setTime(new Date());
   cal.set(Calendar.HOUR_OF_DAY, 0);
   cal.set(Calendar.MINUTE, 0);
   cal.set(Calendar.SECOND, 0);
   cal.set(Calendar.MILLISECOND, 0);
FlexibleSearchQuery query = new FlexibleSearchQuery( fsq );
query.addQueryParameter("today", cal.getTime());
List<CarModel> cars =
   getFlexibleSearchService().<carModel>search(query).getResult();
```

Pagination

- Paginate to reduce bandwidth
  - However, performance improved only if underlying DB supports pagination
  - Note use of generics to return `List<CarModel>` rather than `List<Object>`

```
public List<CarModel> getCars(int start, int range)
{
   String fsq= "SELECT {PK} FROM {Car}";
   FlexibleSearchQuery query = new FlexibleSearchQuery( fsq );
   query.setNeedTotal( true );
   query.setCount( range );
   query.setStart( start );
   return
      getFlexibleSearchService().<CarModel>search( query ).getResult();
}
```

Referring to Model Attributes • Failsafe Approach

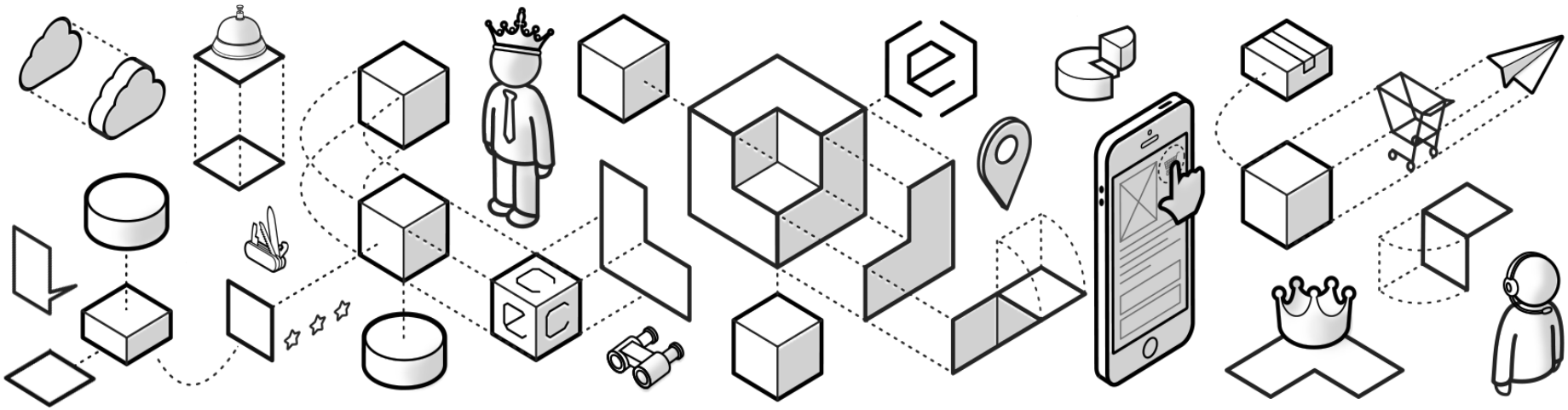
- When building query in Java code, use the static constants defined in each model class to refer to its attributes
  - Constants are the attribute name in uppercase. The itemtype itself is defined as the constant `_TYPECODE`
  - Using constants makes your code more difficult to read, but any changes to the item model (in *extensionName*–items.xml) will cause a compilation error in your Dao classes (alerting you that your queries need to be updated).

- For example, instead of

```
String fsq = "SELECT {Code}, {hp} FROM {Car}";
```

- Use the static constants

```
String fsq = "SELECT {" + CarModel.CODE +"}, {" + CarModel.HP + "}"
           + " FROM {" + CarModel._TYPECODE + "}";
```



Flexible Search Alternatives

Overview
Syntax
API Examples

GenericDao

Use DefaultGenericDao as an alternative to Flexible Search

- Configure target item type in constructor
- Tip: use Spring Expression Language shortcut to refer to Model's static typecode variable

```
<bean name="ProductDao" class="de.hybris.platform.servicelayer.
                               internal.dao.DefaultGenericDao">
   <constructor-arg value="#{T(de.hybris.platform.core.model.
                               product.ProductModel)._TYPECODE}"/>
</bean>
```

- Perform basic search using parameters
  - For example, to return all products weighing 200 kilos:

```
final Map<String,Integer> params = new HashMap<>();
params.put(ProductModel.WEIGHT, 200);
return productDao.find(params);
```

GenericSearch

- Similar to *HibernateCriteriaSearches*
- Search for items as well as raw data fields
- Unlimited number of conditions
- Inner joins and outer joins between item types possible
- Unlimited number of “order by” clauses
- Sub-selects supported

```
GenericQuery query = new GenericQuery(CarModel._TYPECODE);
GenericSerchField carField = new
   GenericSearchField( CarModel.PK, CarModel.Name );
GenericCondition condition =
   GenericCondition.createConditionForValueComparison(carField,
                                                      Operator.LIKE, “BMW”);
query.addCondition( condition );
query.addOrderBy(new GenericSearchOrderBy( carField, true ));

List<CarModel> cars = genericSearchService.search( query );
```

GenericSearch Example

```
GenericQuery query = new GenericQuery(CarModel._TYPECODE);
GenericSerchField carField = new
   GenericSearchField( CarModel.PK, CarModel.Name );
GenericCondition condition =
   GenericCondition.createConditionForValueComparison(carField,
                                                      Operator.LIKE,
                                                      “BMW”);
query.addCondition( condition );
query.addOrderBy(new GenericSearchOrderBy( carField, true ));

List<CarModel> cars = genericSearchService.search( query );
```

Exercise 5