

HF O² Hands-on session Solutions

V. Kučera (Inha University),
L. Dello Stritto (CERN)



Exercise 1

```
+++ b/Tutorials/PWGHE/taskMini.cxx
@@ -138,6 +138,7 @@ struct HfCandidateSelectorD0 {
    Configurable<double> nSigmaTpc{"nSigmaTpc", 3., "Nsigma cut on TPC only"};
    // topological cuts
    Configurable<double> cpaMin{"cpaMin", 0.98, "Min. cosine of pointing angle"};
+   Configurable<double> dlenMin{"dlenMin", 0.01, "Min. decay length"};
    Configurable<double> massWindow{"massWindow", 0.4, "Half-width of the invariant-mass window"};

    HfHelper hfHelper;
@@ -169,6 +170,10 @@ struct HfCandidateSelectorD0 {
    if (candidate.cpa() < cpaMin) {
        return false;
    }
+   // decay length
+   if (candidate.decayLength() < dlenMin) {
+       return false;
+   }
    return true;
}
```

In the **taskMini.cxx**,
in the **HfCandidateSelectorD0** struct

- Add a configurable with the default decay length cut.

```
+++ b/Tutorials/PWGHE/dpl-config_task.json
@@ -170,6 +170,7 @@
    "ptPidTpcMax": "10",
    "nSigmaTpc": "3",
    "cpaMin": "0.98",
+   "dlenMin": "0.02",
    "massWindow": "0.4"
},
"hf-task-d0": {
```

In the **dpl-config_task.json**

- Add the decay length cut setter.

Exercise 1

```
@@ -289,9 +294,11 @@ struct HfTaskD0 {
    const TString strTitle = "D^{0} candidates";
    const TString strPt = "#it{p}_{T} (GeV/#it{c})";
    const TString strEntries = "entries";
+   AxisSpec ptAxis = {100, 0., 10.};
    registry.add("hPtCand", strTitle + ";" + strPt + ";" + strEntries, {HistType::kTH1F, {{100, 0., 10.}}});
    registry.add("hMass", strTitle + ";" + "inv. mass (#pi K) (GeV/#it{c}^{2})" + ";" + strEntries, {HistType::kTH1F, {{500, 0., 5.}}});
    registry.add("hCpaVsPtCand", strTitle + ";" + "cosine of pointing angle" + ";" + strPt + ";" + strEntries, {HistType::kTH2F, {{110, -1.1, 1.1}, {100, 0., 10.}}});
+   registry.add("hDlenVsPtCand", strTitle + ";" + "decay length" + ";" + strPt + ";" + strEntries, {HistType::kTH2F, {{150, 0, 0.1}, {ptAxis}}});
}

void process(soa::Join<aod::HfCandProng2, aod::HfSelCandidateD0> const& candidates)
@@ -305,6 +312,7 @@ struct HfTaskD0 {
}
    registry.fill(HIST("hPtCand"), candidate.pt());
    registry.fill(HIST("hCpaVsPtCand"), candidate.cpa(), candidate.pt());
+   registry.fill(HIST("hDlenVsPtCand"), candidate.decayLength(), candidate.pt());
}
}
};
```

In the **taskMini.cxx**,
in the **HfTaskD0** struct

- Add the decay length histogram.
- Fill the decay length histogram.

Exercise 2



```
+++ b/Tutorials/PWG/HF/DataModelMini.h
@@ -69,6 +69,8 @@ DECLARE_SOA_DYNAMIC_COLUMN(PtProng1, ptProng1, //! pt of prong 1
// candidate properties
DECLARE_SOA_DYNAMIC_COLUMN(DecayLength, decayLength, //! decay length of candidate
    [](float xVtxP, float yVtxP, float zVtxP, float xVtxS, float yVtxS, float zVtxS) -> float { return RecoDecay::distance(std::array{xVtxP, yVtxP, zVtxP}, std::array{xVtxS, yVtxS, zVtxS}); });
+DECLARE_SOA_DYNAMIC_COLUMN(DecayLengthXY, decayLengthXY, //! decay length XY of candidate
+    [](float xVtxP, float yVtxP, float xVtxS, float yVtxS) -> float { return RecoDecay::distance(std::array{xVtxP, yVtxP}, std::array{xVtxS, yVtxS}); });
DECLARE_SOA_DYNAMIC_COLUMN(Pt, pt, //! pt of candidate
    [](float px, float py) -> float { return RecoDecay::pt(px, py); });
DECLARE_SOA_EXPRESSION_COLUMN(Px, px, //! px of candidate
@@ -90,6 +92,7 @@ DECLARE_SOA_TABLE(HfCandProng2Base, "A0D", "HFCANDP2BASE", //! 2-prong candidate
    hf_cand_prong2::XSecondaryVertex, hf_cand_prong2::YSecondaryVertex, hf_cand_prong2::ZSecondaryVertex,
    /* dynamic columns */ hf_cand_prong2::RSecondaryVertex<hf_cand_prong2::XSecondaryVertex, hf_cand_prong2::YSecondaryVertex>,
    hf_cand_prong2::DecayLength<collision::PosX, collision::PosY, collision::PosZ, hf_cand_prong2::XSecondaryVertex, hf_cand_prong2::YSecondaryVertex, hf_cand_prong2::ZSecondaryVertex>,
+    hf_cand_prong2::DecayLengthXY<collision::PosX, collision::PosY, hf_cand_prong2::XSecondaryVertex, hf_cand_prong2::YSecondaryVertex>,
    /* prong 0 */ hf_cand_prong2::PtProng0<hf_cand_prong2::PxProng0, hf_cand_prong2::PyProng0>,
    hf_cand_prong2::PxProng0, hf_cand_prong2::PyProng0, hf_cand_prong2::PzProng0,
    /* prong 1 */ hf_cand_prong2::PtProng1<hf_cand_prong2::PxProng1, hf_cand_prong2::PyProng1>,
```

In the DataModelMini.h

- Add a dynamic column for the decay length XY.
- Add the decay length XY column to the candidate table.

Exercise 2



```
+++ b/Tutorials/PWGHF/taskMini.cxx
@@ -292,6 +292,7 @@ struct HfTaskD0 {
    registry.add("hPtCand", strTitle + ";" + strPt + ";" + strEntries, {HistType::kTH1F, {{100, 0., 10.}}});
    registry.add("hMass", strTitle + ";" + "inv. mass (#pi K) (GeV/#it{c}^2)" + ";" + strEntries, {HistType::kTH1F, {{500, 0., 5.}}});
    registry.add("hCpaVsPtCand", strTitle + ";" + "cosine of pointing angle" + ";" + strPt + ";" + strEntries, {HistType::kTH2F, {{110, -1.1, 1.1}, {100, 0., 10.}}});
+   registry.add("hDlenXYVsPtCand", strTitle + ";" + "decay length XY" + ";" + strPt + ";" + strEntries, {HistType::kTH2F, {{150, 0., 0.1}, {100, 0., 10.}}});
}

void process(soa::Join<aod::HfCandProng2, aod::HfSelCandidateD0> const& candidates)
@@ -305,6 +306,7 @@ struct HfTaskD0 {
}
    registry.fill(HIST("hPtCand"), candidate.pt());
    registry.fill(HIST("hCpaVsPtCand"), candidate.cpa(), candidate.pt());
+   registry.fill(HIST("hDlenXYVsPtCand"), candidate.decayLengthXY(), candidate.pt());
}
}
};
```

In the **taskMini.cxx**,
in the **HfTaskD0** struct

- Add the decay length XY histogram.
- Fill the decay length XY histogram.

Exercise 3



```
+++ b/Tutorials/PWGHF/taskMini.cxx
@@ -278,7 +278,7 @@ struct HfTaskD0 {

    HfHelper hfHelper;

-   Partition<soa::Join<aod::HfCandProng2, aod::HfSelCandidateD0>> selectedD0Candidates = aod::hf_selcandidate_d0::isSelected >= selectionFlagD0 || aod::hf_selcandidate_d0::isSelectedbar >= selectionFlagD0bar;
+   Filter filterD0Candidates = aod::hf_selcandidate_d0::isSelected >= selectionFlagD0 || aod::hf_selcandidate_d0::isSelectedbar >= selectionFlagD0bar;

    HistogramRegistry registry{
        "registry",
@@ -294,9 +294,9 @@ struct HfTaskD0 {
        registry.add("hCpaVsPtCand", strTitle + ";" + "cosine of pointing angle" + ";" + strPt + ";" + strEntries, {HistType::kTH2F, {{110, -1.1, 1.1}, {100, 0., 10.}}});
    }

-   void process(soa::Join<aod::HfCandProng2, aod::HfSelCandidateD0> const& candidates)
+   void process(soa::Filtered<soa::Join<aod::HfCandProng2, aod::HfSelCandidateD0>> const& candidates)
    {
        for (const auto& candidate : selectedD0Candidates) {
+       for (const auto& candidate : candidates) {
            if (candidate.isSelected() >= selectionFlagD0) {
                registry.fill(HIST("hMass"), hfHelper.invMassD0ToPiK(candidate));
            }
        }
    }
}
```

In the **taskMini.cxx**,
in the **HfTaskD0** struct

- Remove the partition and define a Filter.
- Subscribe the filtered candidates to the process function.

Exercise 4



```
@@ -279,7 +279,7 @@ struct HfTaskMiniD0 {
    HfHelper hfHelper;

- Partition<soa::Join<aod::HfTCand2Prong, aod::HfTSEL0>> selectedD0Candidates = aod::hf_selcandidate_d0::isSelD0 >= selectionFlagD0 || aod::hf_selcandidate_d0::isSelD0bar >= selectionFlagD0bar;
+ Filter filterD0Candidates = aod::hf_selcandidate_d0::isSelD0 >= selectionFlagD0 || aod::hf_selcandidate_d0::isSelD0bar >= selectionFlagD0bar;

    HistogramRegistry registry{
        "registry",
@@ -293,11 +293,17 @@ struct HfTaskMiniD0 {
        registry.add("hPtCand", strTitle + ";" + strPt + ";" + strEntries, {HistType::kTH1F, {{100, 0., 10.}}});
        registry.add("hMass", strTitle + ";" + "inv. mass (#pi K) (GeV/#it{c}^{{2}})" + ";" + strEntries, {HistType::kTH1F, {{500, 0., 5.}}});
        registry.add("hCpaVsPtCand", strTitle + ";" + "cosine of pointing angle" + ";" + strPt + ";" + strEntries, {HistType::kTH2F, {{110, -1.1, 1.1}, {100, 0., 10.}}});
+       registry.add("hTracks", strTitle + ";" + strPt + ";" + strEntries, {HistType::kTH1F, {{100, 0., 300.}}});
+       registry.add("hCands", strTitle + ";" + strPt + ";" + strEntries, {HistType::kTH1F, {{10, 0., 10.}}});
+       registry.add("hColl", strTitle + ";" + strPt + ";" + strEntries, {HistType::kTH1F, {{1, 0., 2.}}});
    }

- void process(soa::Join<aod::HfTCand2Prong, aod::HfTSEL0> const& /*candidates*/)
+ void process(aod::Collision const& collision, aod::Tracks const& tracks, soa::Filtered<soa::Join<aod::HfTCand2Prong, aod::HfTSEL0>> const& candidates)
    {
-     for (const auto& candidate : selectedD0Candidates) {
+     for (const auto& candidate : candidates) {
+         if (candidate.isSelD0() >= selectionFlagD0) {
+             registry.fill(HIST("hMass"), hfHelper.invMassD0ToPiK(candidate));
+         }
    }
}
```

- Change the Partition into a Filter.
- Add the *number of tracks* and *number of candidates* per collision histograms.
- Subscribe the collisions and tracks table to the process function.
- Fill the histograms

In the **taskMini.cxx**,
in the **HfTaskD0** struct