/\*\*

 \* related-equipment.js - V7.7.7 FIX8

 \* 連携設備 / 関連する金型・抜型 / Thiết bị liên quan (Cột 4)

 \*

 \* - Giữ nguyên danh sách cột 2 theo từ khóa hiện tại (không thay kết quả tìm kiếm).

 \* - Khi chọn từ cột 4: cập nhật chi tiết (cột 3) + highlight ở cột 2:

 \*   · Nếu có trong danh sách: thẻ đó active (đậm), các thẻ còn lại inactive (xám).

 \*   · Nếu không có trong danh sách: toàn bộ thẻ cột 2 inactive để phân biệt rõ.

 \* - Cột 2: cho phép bấm vào thẻ đang inactive để active lại thẻ đó và inactive thẻ khác.

 \* - Dòng ở cột 4 luôn có nền theo loại (mold/cutter); Badge Rack/Layer co giãn theo nội dung.

 \*/

(function () {

  'use strict';

  const SEL = {

    hostCandidates: ['.actions-lower', '.panel.actions-panel', '[data-col="4"]', '#col-4', '.col-4'],

    wrapId: 'related-list',

    headerId: 'related-list-header',

    listClass: 'related-eq-list',

    // Bắt nhiều biến thể grid và thẻ của cột 2

    quickGrid: '#quick-results-grid, #quick-results, .quick-results-grid',

    quickCard: '.result-card, .quick-result-card'

  };

  const TEXT = {

    titleForMoldJP: '抜き型情報',

    titleForMoldVN: 'Thông tin dao cắt',

    titleForCutterJP: '関連金型',

    titleForCutterVN: 'Khuôn liên quan',

    empty: '関連金型・抜型情報なし',

    badgeSeparateYes: '別抜き あり',

    badgeSeparateNo:  '別抜き なし'

  };

  const THEME = { mold: 'theme-mold', cutter: 'theme-cutter', all: 'theme-all' };

  const State = {

    lastKey: null,

    indexesBuilt: false,

    maps: {

      designByMoldId: new Map(),

      moldsByDesign: new Map(),

      cutterIdsByDesign: new Map(),

      designIdsByCutter: new Map(),

      designSeparate: new Map()

    }

  };

  if (document.readyState === 'loading') document.addEventListener('DOMContentLoaded', init); else init();

  async function init() {

    injectStyles();

    ensureWrapAndChildren(true);

    await waitForData();

    buildIndexes();

    bindEvents();

    refreshFromDetail();

    setInterval(refreshFromDetail, 800);

    console.log('[RelatedEquipment] FIX8 ready');

  }

  function waitForData() {

    return new Promise(resolve => {

      const tick = () => {

        const d = window.DataManager?.data || window.moldAllData || {};

        if ((d.molds?.length||0) && (d.cutters?.length||0) && (d.molddesign?.length||0) && (d.moldcutter?.length||0)) return resolve();

        setTimeout(tick, 120);

      };

      tick();

    });

  }

  function ensureHost() {

    for (const s of SEL.hostCandidates) {

      const el = document.querySelector(s);

      if (el) return el;

    }

    return document.body;

  }

  function ensureWrapAndChildren(adaptHeader = false) {

    const host = ensureHost();

    let wrap = document.getElementById(SEL.wrapId);

    if (!wrap) { wrap = document.createElement('section'); wrap.id = SEL.wrapId; wrap.className = 'related-eq-wrap'; host.appendChild(wrap); }

    let header = document.getElementById(SEL.headerId);

    if (!header) { header = document.createElement('div'); header.id = SEL.headerId; header.className = 'related-eq-header'; wrap.appendChild(header); }

    let list = wrap.querySelector(`.${SEL.listClass}`);

    if (!list) { list = document.createElement('div'); list.className = SEL.listClass; wrap.appendChild(list); }

    if (adaptHeader) {

      const hasPanelTitle = host.querySelector('.card-header, .panel-title, .panel .header, [data-role="panel-title"]');

      if (hasPanelTitle) wrap.classList.add('no-local-title');

    }

    return { wrap, header, list };

  }

  function applyTheme(viewType) {

    const { wrap } = ensureWrapAndChildren();

    wrap.classList.remove(THEME.mold, THEME.cutter, THEME.all);

    if (viewType === 'mold') wrap.classList.add(THEME.cutter);

    else if (viewType === 'cutter') wrap.classList.add(THEME.mold);

    else wrap.classList.add(THEME.all);

  }

  function injectStyles() {

    const css = document.createElement('style');

    css.textContent = `

      .related-eq-wrap { display:block; padding:10px; border:1px solid #e5e7ef; border-radius:8px; background:#fff; margin-top:8px; }

      .related-eq-wrap.no-local-title .related-eq-header { display:none; }

      .related-eq-header { margin-bottom:10px; }

      .related-eq-header .title-row { display:flex; align-items:center; justify-content:space-between; margin-bottom:2px; }

      .related-eq-header .title-jp { font-weight:700; color:#111827; font-size:14px; }

      .related-eq-header .title-vn { font-size:11px; color:#6b7280; font-style:italic; opacity: 0.7; }

      .related-eq-header .badge-separate { padding:3px 8px; border-radius:10px; font-size:11px; font-weight:700; }

      .related-eq-header .badge-separate.yes { background:#DC2626; color:#fff; }

      .related-eq-header .badge-separate.no  { background:#e5e7eb; color:#374151; }

      .related-eq-list { max-height: 38vh; overflow:auto; }

      .rel-row { position:relative; display:flex; align-items:center; gap:6px; padding:8px 10px; border:1px solid #e5e7ef; border-radius:8px; margin-bottom:8px; cursor:pointer; transition: background .15s, border-color .15s, filter .15s; }

      .rel-row.row-mold   { background:#F0FDFF; border-color:#A5F3FC; }

      .rel-row.row-cutter { background:#FFF9F0; border-color:#FED7AA; }

      .related-eq-wrap.theme-mold   .rel-row.row-mold:hover   { filter:brightness(0.98); }

      .related-eq-wrap.theme-cutter .rel-row.row-cutter:hover { filter:brightness(0.98); }

      .rel-id { font-family: ui-monospace, Menlo, Consolas, monospace; font-size:13px; font-weight:700; color:#0B5CAB; flex-shrink:0; }

      .rel-name { flex:1; min-width:0; font-size:13px; color:#111827; white-space:nowrap; overflow:hidden; text-overflow:ellipsis; }

      .rel-location { display:flex; align-items:center; gap:3px; flex-shrink:0; color:#059669; font-size:13px; font-weight:600; white-space:nowrap; }

      /\* Badge Rack/Layer co giãn \*/

      .circle-num, .square-num { display:inline-flex; align-items:center; justify-content:center; min-width:20px; height:20px; padding:0 6px; border:1.5px solid #059669; color:#059669; background:#EAFBF6; font-size:12px; line-height:1; }

      .circle-num { border-radius:999px; }

      .square-num { border-radius:6px; }

      /\* Cột 2: active/inactive cho cả hai loại thẻ \*/

      .result-card, .quick-result-card { position:relative; transition: filter .15s, opacity .15s, box-shadow .15s; }

      .result-card.active, .quick-result-card.active   { box-shadow: 0 0 0 2px #2563eb inset; filter:none; opacity:1; }

      .result-card.inactive, .quick-result-card.inactive { filter: grayscale(0.7) brightness(1); opacity:0.8; }

      /\* Nhãn loại để không nhầm (mold/cutter) khi xám \*/

      .result-card.inactive::after, .quick-result-card.inactive::after { content: attr(data-type); position:absolute; top:6px; right:6px; font-size:10px; font-weight:700; color:#374151; background:#F3F4F6; padding:2px 6px; border-radius:10px; }

      /\* 2 dòng cho item liên quan – không sửa JS, chỉ CSS \*/

      #related-list .rel-row{

        display:flex; align-items:center; gap:6px; flex-wrap: wrap;

        padding: 6px 8px;                       /\* giảm đệm để gọn hơn \*/

      }

      #related-list .rel-row .rel-id{ flex:0 0 auto; }

      #related-list .rel-row .rel-name{

        flex:1 1 auto; min-width:0;

        white-space: nowrap; overflow: hidden; text-overflow: ellipsis;

      }

      /\* Đẩy Giá・Tầng xuống dòng 2 \*/

      #related-list .rel-row .rel-location{

        flex: 1 1 100%; order: 2; margin-top: 2px;

        white-space: nowrap;                     /\* badge nằm cùng một hàng \*/

      }

      /\* Giảm khoảng trống tiêu đề và giữa các dòng \*/

      #related-list .related-eq-header{

        padding: 4px 8px;                        /\* trước 8–10px \*/

        margin-bottom: 6px;                      /\* trước 10–12px \*/

      }

      #related-list .related-eq-list{ gap: 4px; }/\* trước 6–8px \*/

      /\* Thu khoảng trống giữa header cột 4 và khu liên quan \*/

      .actions-panel .actions-lower{ padding-top: 6px; } /\* trước 10–12px \*/

      /\* Đậm màu mặc định theo loại (không cần hover) \*/

    #related-list .rel-row.row-mold{

      /\* fallback rgba + ưu tiên color-mix khi trình duyệt hỗ trợ \*/

      background: rgba(59,130,246,0.16);

      background: color-mix(in srgb, var(--kiosk-blue) 16%, #fff);

      border: 1.5px solid rgba(59,130,246,0.55);

    }

    #related-list .rel-row.row-cutter{

      background: rgba(245,158,11,0.16);

      background: color-mix(in srgb, var(--kiosk-orange) 16%, #fff);

      border: 1.5px solid rgba(245,158,11,0.55);

    `;

    document.head.appendChild(css);

  }

  /\* ===== Indexes (MoldDesignID-based) ===== \*/

  function buildIndexes() {

    const d = window.DataManager?.data || window.moldAllData || {};

    const molds = d.molds || [];

    const molddesign = d.molddesign || [];

    const moldcutter = d.moldcutter || [];

    const designByMoldId = new Map();

    const moldsByDesign = new Map();

    const designSeparate = new Map();

    for (const dj of molddesign) {

      const moldId = str(dj.MoldID ?? dj.MoldId ?? dj.mold\_id);

      const designId = str(dj.MoldDesignID ?? dj.MoldDesignId);

      if (!moldId || !designId) continue;

      designByMoldId.set(moldId, designId);

      const sepRaw = dj.SeparateCutter ?? dj.separate\_cutter ?? dj.Separate ?? dj.separate ?? '';

      const sep = str(sepRaw).toUpperCase() === 'YES';

      designSeparate.set(designId, (designSeparate.get(designId) || false) || sep);

    }

    for (const m of molds) {

      const moldId = str(m.MoldID ?? m.mold\_id);

      let designId = str(m.MoldDesignID ?? m.MoldDesignId);

      if (!designId) designId = designByMoldId.get(moldId) || '';

      if (!designId) continue;

      const arr = moldsByDesign.get(designId) || [];

      arr.push(m);

      moldsByDesign.set(designId, arr);

    }

    const cutterIdsByDesign = new Map();

    const designIdsByCutter = new Map();

    for (const mc of moldcutter) {

      const designId = str(mc.MoldDesignID ?? mc.MoldDesignId);

      const cutterId = str(mc.CutterID ?? mc.CutterId);

      if (!designId || !cutterId) continue;

      if (!cutterIdsByDesign.has(designId)) cutterIdsByDesign.set(designId, new Set());

      cutterIdsByDesign.get(designId).add(cutterId);

      if (!designIdsByCutter.has(cutterId)) designIdsByCutter.set(cutterId, new Set());

      designIdsByCutter.get(cutterId).add(designId);

    }

    State.maps.designByMoldId = designByMoldId;

    State.maps.moldsByDesign = moldsByDesign;

    State.maps.cutterIdsByDesign = cutterIdsByDesign;

    State.maps.designIdsByCutter = designIdsByCutter;

    State.maps.designSeparate = designSeparate;

    State.indexesBuilt = true;

  }

  /\* ===== Helpers ===== \*/

  function findMoldByKey(k) {

    const d = window.DataManager?.data || window.moldAllData || {};

    const molds = d.molds || [];

    const key = str(k);

    let mold = molds.find(m => str(m.MoldID ?? m.mold\_id) === key);

    if (!mold) mold = molds.find(m => str(m.MoldCode ?? m.displayCode ?? m.Code).toUpperCase() === key.toUpperCase());

    return mold || null;

  }

  function findCutterByKey(k) {

    const d = window.DataManager?.data || window.moldAllData || {};

    const cutters = d.cutters || [];

    const key = str(k);

    let c = cutters.find(x => str(x.CutterID ?? x.cutter\_id) === key);

    if (!c) c = cutters.find(x => str(x.CutterNo ?? x.displayCode ?? x.Code).toUpperCase() === key.toUpperCase());

    return c || null;

  }

  function getDesignIdOfMold(mold) {

    const direct = str(mold.MoldDesignID ?? mold.MoldDesignId);

    if (direct) return direct;

    const moldId = str(mold.MoldID ?? mold.mold\_id);

    return State.maps.designByMoldId.get(moldId) || '';

  }

  function separateByDesign(designId) { return !!State.maps.designSeparate.get(str(designId)); }

  function getRelatedCuttersForMoldKey(moldKey) {

    const d = window.DataManager?.data || window.moldAllData || {};

    const cutters = d.cutters || [];

    const mold = findMoldByKey(moldKey);

    if (!mold) return [];

    const designId = getDesignIdOfMold(mold);

    if (!designId) return [];

    const ids = Array.from(State.maps.cutterIdsByDesign.get(designId) || []);

    const sep = separateByDesign(designId);

    return ids.map(cid => {

      const c = cutters.find(x => str(x.CutterID ?? x.CutterId) === str(cid));

      return c ? { item: c, separate: sep } : null;

    }).filter(Boolean);

  }

  function getRelatedMoldsForCutterKey(cutterKey) {

    const cutter = findCutterByKey(cutterKey);

    if (!cutter) return [];

    const cutterId = str(cutter.CutterID ?? cutter.CutterId ?? cutter.cutter\_id);

    const designIds = Array.from(State.maps.designIdsByCutter.get(cutterId) || []);

    const out = [];

    for (const did of designIds) {

      const molds = State.maps.moldsByDesign.get(did) || [];

      const sep = separateByDesign(did);

      for (const m of molds) out.push({ item: m, separate: sep });

    }

    return out;

  }

  function getRackId(it) {

    return it?.rackLayerInfo?.RackID ?? it?.rackInfo?.RackID ?? it?.rackInfo?.RackNo ?? it?.rackInfo?.RackName ?? '-';

  }

  function getLayerNo(it) {

    return it?.rackLayerInfo?.RackLayerNumber ?? it?.rackLayerInfo?.LayerNo ?? it?.rackLayerInfo?.LayerNumber ?? it?.RackLayerNumber ?? it?.LayerNo ?? '-';

  }

  function formatDisplay(it, isMold) {

    const id = isMold ? (it.MoldID ?? '-') : (it.CutterNo ?? it.CutterCode ?? it.CutterID ?? '-');

    const name = isMold ? (it.MoldCode ?? it.displayCode ?? it.Code ?? '-') : (it.CutterName ?? it.displayName ?? it.Name ?? (it.CutterCode ?? '-'));

    const rackId = getRackId(it);

    const layerNo = getLayerNo(it);

    return { id, name, rackId, layerNo };

  }

  function circledHTML(text) { return `<span class="circle-num">${escapeHtml(String(text))}</span>`; }

  function squaredHTML(text) { return `<span class="square-num">${escapeHtml(String(text))}</span>`; }

  function renderRow(targetType, rec, index) {

    const it = rec.item;

    const isMold = (targetType === 'mold');

    const { id, name, rackId, layerNo } = formatDisplay(it, isMold);

    const rowTypeClass = isMold ? 'row-mold' : 'row-cutter';

    return `

      <div class="rel-row ${rowTypeClass}" data-idx="${index}" data-type="${targetType}" data-id="${escapeHtml(id)}" title="${escapeHtml(String(id))} ${escapeHtml(String(name))}">

        <span class="rel-id">${escapeHtml(id)}</span>

        <span class="rel-name">${escapeHtml(name)}</span>

        <span class="rel-location">${circledHTML(rackId)}<span class="sep-dash">-</span>${squaredHTML(layerNo)}</span>

      </div>

    `;

  }

  function renderList(viewType, related) {

    const { header, list } = ensureWrapAndChildren();

    const groupSeparate = related?.some(r => r.separate) === true;

    const titleJP = (viewType === 'mold') ? TEXT.titleForMoldJP : TEXT.titleForCutterJP;

    const titleVN = (viewType === 'mold') ? TEXT.titleForMoldVN : TEXT.titleForCutterVN;

    const badgeClass = groupSeparate ? 'yes' : 'no';

    const badgeText = groupSeparate ? TEXT.badgeSeparateYes : TEXT.badgeSeparateNo;

    header.innerHTML = `

      <div class="title-row">

        <span class="title-jp">${titleJP}</span>

        <span class="badge-separate ${badgeClass}">${badgeText}</span>

      </div>

      <div class="title-vn"> ${titleVN}</div>

    `;

    applyTheme(viewType);

    if (!related || related.length === 0) {

      list.innerHTML = `<div class="related-empty">${TEXT.empty}</div>`;

      return;

    }

    const targetType = (viewType === 'mold') ? 'cutter' : 'mold';

    const rows = related.map((r, i) => renderRow(targetType, r, i)).join('');

    list.innerHTML = rows;

    list.querySelectorAll('.rel-row').forEach(row => {

      row.addEventListener('click', () => {

        const idx = Number(row.getAttribute('data-idx'));

        const rec = related[idx];

        chooseItem(rec.item); // cập nhật detail + highlight cột 2

      });

    });

  }

  /\* ===== Chọn item: cập nhật detail + highlight cột 2 ===== \*/

  function chooseItem(item) {

    const type = item.MoldID != null ? 'mold' : 'cutter';

    const id   = String(type === 'mold' ? (item.MoldID ?? item.MoldCode) : (item.CutterID ?? item.CutterNo));

    const code = String(type === 'mold' ? (item.MoldCode ?? '') : (item.CutterNo ?? item.CutterCode ?? ''));

    // Cập nhật chi tiết (cột 3)

    if (window.UIRenderer) {

      if (typeof window.UIRenderer.renderDetailInfo === 'function') window.UIRenderer.renderDetailInfo(item, type);

      else if (typeof window.UIRenderer.renderDetail === 'function') window.UIRenderer.renderDetail(item, type);

      if (window.UIRenderer.state) window.UIRenderer.state.currentDetailItem = item;

    }

    document.dispatchEvent(new CustomEvent('detail:changed', { detail: { item, type } }));

    // Highlight theo id/type + dự phòng theo code/text

    highlightQuickResult(id, type, code);

    // Làm mới danh sách liên quan theo item vừa chọn

    State.lastKey = null;

    setTimeout(refreshFromDetail, 50);

  }

  function highlightQuickResult(selectedId, selectedType, selectedCode) {

    const grid = document.querySelector(SEL.quickGrid);

    if (!grid) return;

    const cards = Array.from(grid.querySelectorAll(SEL.quickCard));

    if (!cards.length) return;

    let matched = false;

    const idStr = String(selectedId);

    const codeStr = String(selectedCode || '');

    for (const card of cards) {

      const cid   = String(card.dataset.id   || card.getAttribute('data-id')   || '').trim();

      const ctype = String(card.dataset.type || card.getAttribute('data-type') || '').trim().toLowerCase();

      const ccode = String(card.dataset.code || card.getAttribute('data-code') || card.dataset.key || '').trim();

      const text  = (card.textContent || '').toUpperCase();

      const typeOk = !ctype || ctype === String(selectedType);

      const idOk   = cid && (cid === idStr || cid === codeStr);

      const codeOk = ccode && (ccode === idStr || ccode === codeStr);

      const textOk = codeStr && text.includes(codeStr.toUpperCase());

      const isMatch = typeOk && (idOk || codeOk || textOk);

      if (isMatch) {

        card.classList.add('active');

        card.classList.remove('inactive');

        matched = true;

      } else {

        card.classList.remove('active');

      }

    }

    if (!matched) {

      // Không có trong kết quả hiện tại → toàn bộ thành inactive

      for (const card of cards) { card.classList.remove('active'); card.classList.add('inactive'); }

    } else {

      // Có item khớp → các item khác inactive

      for (const card of cards) { if (!card.classList.contains('active')) card.classList.add('inactive'); }

    }

  }

  /\* ===== Lắng nghe click ở cột 2 để bật active khi đang inactive ===== \*/

  function quickGridClickHandler(e) {

    const card = e.target.closest(SEL.quickCard);

    if (!card) return;

    const grid = card.closest(SEL.quickGrid) || document.querySelector(SEL.quickGrid);

    if (!grid) return;

    // Active card được bấm, inactive các card còn lại

    const cards = Array.from(grid.querySelectorAll(SEL.quickCard));

    for (const c of cards) {

      if (c === card) { c.classList.add('active'); c.classList.remove('inactive'); }

      else { c.classList.remove('active'); c.classList.add('inactive'); }

    }

    // Gọi chọn chi tiết tương ứng

    const id = card.getAttribute('data-id') || '';

    const type = (card.getAttribute('data-type') || '').toLowerCase();

    if (!id || !type) return;

    if (window.AppController && typeof window.AppController.selectResult === 'function') {

      window.AppController.selectResult(id, type);

    } else {

      const item = findItemByIdType(id, type);

      if (item) chooseItem(item);

    }

  }

  function findItemByIdType(id, type) {

    const d = window.DataManager?.data || window.moldAllData || {};

    if (type === 'mold') {

      const molds = d.molds || [];

      // Khớp theo ID hoặc Code

      return molds.find(m => String(m.MoldID ?? m.mold\_id) === String(id) || String(m.MoldCode ?? m.Code) === String(id)) || null;

    } else {

      const cutters = d.cutters || [];

      return cutters.find(c => String(c.CutterID ?? c.cutter\_id) === String(id) || String(c.CutterNo ?? c.CutterCode) === String(id)) || null;

    }

  }

  function refreshFromDetail() {

    const item = window.UIRenderer?.state?.currentDetailItem || null;

    if (!item) return;

    const type = item.MoldID != null ? 'mold' : 'cutter';

    const k = type === 'mold' ? `m:${item.MoldID ?? item.MoldCode}` : `c:${item.CutterID ?? item.CutterNo}`;

    if (k === State.lastKey) return;

    State.lastKey = k;

    if (type === 'mold') renderList('mold', getRelatedCuttersForMoldKey(item.MoldID ?? item.MoldCode));

    else renderList('cutter', getRelatedMoldsForCutterKey(item.CutterID ?? item.CutterNo));

  }

  function onSearchUpdated() {

    if (window.UIRenderer?.state?.currentDetailItem) return;

    const results = window.SearchModule?.getResults?.() || [];

    if (!results.length) return;

    const first = results[0];

    if (first.MoldID != null) renderList('mold', getRelatedCuttersForMoldKey(first.MoldID ?? first.MoldCode));

    else if (first.CutterID != null) renderList('cutter', getRelatedMoldsForCutterKey(first.CutterID ?? first.CutterNo));

  }

  function bindEvents() {

    document.addEventListener('detail:changed', refreshFromDetail);

    document.addEventListener('search:updated', onSearchUpdated);

    // Bắt sự kiện click toàn cục để không phụ thuộc đúng ID container

    document.addEventListener('click', quickGridClickHandler, true);

    window.RelatedEquipment = { refresh: () => { State.lastKey = null; refreshFromDetail(); } };

  }

  function str(v) { return v == null ? '' : String(v).trim(); }

  function escapeHtml(s) { return String(s||'').replace(/&/g,'&amp;').replace(/</g,'&lt;').replace(/>/g,'&gt;'); }

})();