

eRA Follow-Up Samples

RX26 Presentation Takeaways

ERA: Tracking Who's Leaving Before They're Gone · Ann Morricks & Paul Loots · Grace Family Church, Tampa Bay

Description

This document collects the queries, Lava patterns and configuration notes behind our session, so you can build the same kind of eRA follow-up system on your own Rock server. The names of the sections match the slide titles where we presented each idea.

Everything here was built and tested on **Rock 18.2** (the core facts were checked against the 18.2.4 source). It uses **core Rock features only**: no plugins and no C#. Each component is labelled:

- **CORE** ships with Rock. Nothing to create; you may need to switch it on.
- **CONFIGURED** a core feature where you create your own record (a Step Type, a workflow type, a note type).
- **CUSTOM** Lava, SQL or JavaScript we wrote.

Every person, family, email address, phone number and figure in this document and in the slides is fictional. Ids that differ between Rock installs have been replaced by attribute keys or core GUIDs; anything marked `<like this>` is yours to fill in.

Before You Start: Notes

- **Enable SQL in the block.** An HTML Content block only runs `{% sql %}` when its **Enabled Lava Commands** setting includes `sql`. Without it the block renders nothing and shows no error. Persisted datasets have the same setting.
- **Use keys and GUIDs, never Ids.** Attribute, defined value, page and workflow Ids differ in every Rock install. All samples look things up by key or GUID.
- **Build in a sandbox page first.** Copy a block to a hidden page, test there, then move it.
- **The SQL is T-SQL** for Rock's SQL Server or Azure SQL database. `GETDATE()` returns server time, which is UTC on hosted Azure SQL while Rock stores local times; allow a few hours of slack on date edges.
- **Put a build stamp in every block** (for example `<!-- build 2026-09-30a -->` or a JavaScript variable) and read it back from the rendered page. What Rock stores and what it serves are not always the same.

How Rock Decides Who's Regular

CORE The **Family Analytics** job (`Rock.Jobs.CalculateFamilyAnalytics`) runs the stored procedure `spCrm_FamilyAnalyticsEraDataset` and sets eRA on every member of each family. The thresholds are fixed in the procedure; you do not configure them.

Event	Rule (adults only; per family for attendance, per giving unit for giving)
Enters eRA (either)	4 or more contributions in the last 52 weeks and at least 1 in the last 6 weeks; or attended 8 or more distinct Sundays in the last 16 weeks
Leaves eRA (all)	No contributions in the last 8 weeks; no attendance in the last 4 weeks; fewer than 8 Sundays attended in the last 16 weeks

The job maintains these core person attributes (reference them by key):

Key	Meaning	Note
<code>core_CurrentlyAnEra</code>	True while the family is regular	On exit the value is deleted , so test for empty, not False
<code>core_EraStartDate</code>	Date the family entered eRA	
<code>core_EraEndDate</code>	Date the family left eRA	Deleted again if they re-enter
<code>core_EraFirstCheckin</code> , <code>core_EraLastCheckin</code> , <code>core_EraFirstGave</code> , <code>core_EraLastGave</code> , <code>core_EraTimesGiven52Wks</code> , <code>core_EraTimesGiven6Wks</code> , <code>core_EraTimesCheckedIn16Wks</code>	Giving and attendance history	Handy for drill-downs

Rock 18 also records each entry into eRA as a Step of the core **eRA** Step Type in the core **Core Steps** program, and completes it on exit, and it can write "eRA entered/exited" History records.

Check Three Settings First

1. **Worship attendance counts.** Admin Tools, then General Settings, then Group Types: open each weekend service or check-in group type and tick **Attendance counts as weekend service**. Only those group types feed eRA.
2. **Giving counts.** Every contribution account that should count must be marked **Tax Deductible**, and gifts must use the core **Contribution** transaction type.
3. **The job runs.** Admin Tools, then System Settings, then Jobs Administration: the **Family Analytics** job is active and scheduled (weekly is typical). After you fix settings 1 or 2, run it once and spot-check a few families you know.

Two Signals, for Free

CORE The Family Analytics job has two settings, **eRA Entry Workflow** and **eRA Exit Workflow**. Pick a workflow type and Rock launches it for every family that enters or leaves eRA. It names the workflow after the head of household and fills these workflow attributes **if your workflow type defines them with these keys**:

Attribute key	Suggested field type	Value Rock sets
Family	Group	The family group's GUID
HeadOfHouse	Person	Head of household (the first adult, ordered by gender)
Spouse	Person	The other adult, if there is one
Campus	Campus	The family's campus

This is the fastest way to start: an exit workflow that adds a follow-up Step and a reminder for a campus pastor gives you the whole loop without any dashboard. We assign by hand from the dashboard instead, which is why the rest of this document exists.

Reading eRA in SQL

CUSTOM Families whose eRA ended in the last 30 days, one row per family, with campus. Paste it into an HTML Content block with Sql enabled, or run it in Admin Tools, then Power Tools, then SQL Command.

```
-- Families who left eRA in the last @Days days
DECLARE @Days int = 30;
DECLARE @PersonEntityTypeId int = (SELECT Id FROM EntityType WHERE [Name] =
  'Rock.Model.Person');
DECLARE @EraEndAttributeId int = (SELECT Id FROM Attribute
  WHERE [Key] = 'core_EraEndDate' AND EntityTypeId = @PersonEntityTypeId);
DECLARE @FamilyGroupTypeId int = (SELECT Id FROM GroupType
  WHERE [Guid] = '790E3215-3B10-442B-AF69-616C0DCB998E'); -- core Family
DECLARE @ActiveStatusId int = (SELECT Id FROM DefinedValue
  WHERE [Guid] = '618F906C-C33D-4FA3-8AEF-E58CB7B63F1E'); -- core Active record status

SELECT
  g.Id AS FamilyGroupId,
  g.[Name] AS FamilyName,
  ISNULL(c.[Name], '') AS Campus,
  MAX(av.ValueAsDateTime) AS EraEndDate,
  COUNT(DISTINCT p.Id) AS People
FROM AttributeValue av
JOIN Person p ON p.Id = av.EntityId
JOIN GroupMember gm ON gm.PersonId = p.Id
JOIN [Group] g ON g.Id = gm.GroupId AND g.GroupTypeId = @FamilyGroupTypeId
LEFT JOIN Campus c ON c.Id = g.CampusId
WHERE av.AttributeId = @EraEndAttributeId
  AND av.ValueAsDateTime ≥ DATEADD(day, -@Days, GETDATE())
  AND p.RecordStatusValueId = @ActiveStatusId
```

```
GROUP BY g.Id, g.[Name], c.[Name]
ORDER BY EraEndDate DESC;
```

Entering is the same query with `core_EraStartDate`, plus a check that `core_CurrentlyAnEra` is `'True'`.

A minimal Lava wrapper for the block:

```
{% sql return:'families' %}
-- paste the query above here
{% endsql %}
<table class="table table-condensed">
  <tr><th>Family</th><th>Campus</th><th>eRA ended</th></tr>
  {% for f in families %}
  <tr>
    <td><a href="/Group/{{ f.FamilyGroupId }}">{{ f.FamilyName }}</a></td>
    <td>{{ f.Campus }}</td>
    <td>{{ f.EraEndDate | HumanizeDateTime }}</td>
  </tr>
  {% endfor %}
</table>
```

Adjust the family link to whatever route your Rock uses for a family or person profile.

What Were They Connected To?

CUSTOM For the same families, were any members in a small group or on a serving team? This version uses current active memberships, which is a good first approximation. "Serving" uses the core **Serving Area** group type purpose; small groups vary between churches, so list your own small group type Ids.

```
DECLARE @ServingPurposeId int = (SELECT Id FROM DefinedValue
  WHERE [Guid] = '36A554CE-7815-41B9-A435-93F3D52A2828'); -- core Serving Area purpose
DECLARE @SmallGroupTypes TABLE (Id int);
INSERT @SmallGroupTypes VALUES (<your small group type Id>); -- add more rows if needed

-- #Leaving holds FamilyGroupId from the "Reading eRA in SQL" query (SELECT ... INTO #Leaving)
WITH Conn AS (
  SELECT l.FamilyGroupId,
    MAX(CASE WHEN g.GroupTypeId IN (SELECT Id FROM @SmallGroupTypes) THEN 1 ELSE 0 END) AS
InGroup,
    MAX(CASE WHEN gt.GroupTypePurposeValueId = @ServingPurposeId THEN 1 ELSE 0 END) AS
Serving
  FROM #Leaving l
  JOIN GroupMember fm ON fm.GroupId = l.FamilyGroupId
  LEFT JOIN GroupMember gm ON gm.PersonId = fm.PersonId
    AND gm.GroupMemberStatus = 1 AND gm.IsArchived = 0
  LEFT JOIN [Group] g ON g.Id = gm.GroupId AND g.IsActive = 1 AND g.IsArchived = 0
  LEFT JOIN GroupType gt ON gt.Id = g.GroupTypeId
  GROUP BY l.FamilyGroupId
)
SELECT CASE WHEN InGroup = 1 AND Serving = 1 THEN 'Both'
  WHEN InGroup = 1 THEN 'Small group'
  WHEN Serving = 1 THEN 'Serving'
```

```

        ELSE 'Neither' END AS Connection,
        COUNT(*) AS Families
FROM Conn
GROUP BY CASE WHEN InGroup = 1 AND Serving = 1 THEN 'Both'
             WHEN InGroup = 1 THEN 'Small group'
             WHEN Serving = 1 THEN 'Serving'
             ELSE 'Neither' END;

```

Every Number Is a Door

CUSTOM Make every count on the dashboard clickable, opening the list behind it. The rule that keeps pastors trusting the page: **the list always adds up to the number that was clicked**. To guarantee it:

- Emit the list rows from the **same** query, date window and filters as the count. Never re-derive them separately.
- Match rows to what was clicked by natural keys you select in SQL (a campus Id, a date offset), not by loop counters. Loop counters can shift by one between Lava engines.
- After every change, open each clickable number and compare the list length to the number.

A Family, Not a Row

CUSTOM Build the household from `GroupMember` on the family group (not from `Person.PrimaryFamilyId` alone), and pick exactly one mobile number per person so nobody appears twice:

```

DECLARE @FamilyGroupId int = <family group Id>;
SELECT p.Id, p.NickName, p.LastName, r.[Name] AS FamilyRole, p.Email, ph.NumberFormatted AS
Mobile
FROM GroupMember gm
JOIN Person p      ON p.Id = gm.PersonId
JOIN GroupTypeRole r ON r.Id = gm.GroupRoleId
OUTER APPLY (
    SELECT TOP 1 pn.NumberFormatted
    FROM PhoneNumber pn
    JOIN DefinedValue dv ON dv.Id = pn.NumberTypeValueId
    WHERE pn.PersonId = p.Id
          AND dv.[Guid] = '407E7E45-7B2E-4FCD-9605-ECB1339F2453' -- core Mobile
    ORDER BY pn.Id
) ph
WHERE gm.GroupId = @FamilyGroupId
ORDER BY r.[Order], p.BirthDate;

```

Why OUTER APPLY: a plain `LEFT JOIN PhoneNumber` duplicates every row for a person with two mobile numbers. Some people do have two.

The Right List for the Right Person

We split the audience: campus pastors see everyone except the top giving segment; the stewardship team follows up with that segment on a separate Donor Development dashboard.

CORE Rock's **Giving Automation** job already writes a **Giving Bin** person attribute (GUID 7FBB63CC-F4FC-4F7E-A8C5-44DC3D0F0720) using the 95th, 80th and 60th percentile cut-offs per giving unit. **Start here.**

```
SELECT p.Id, p.NickName, p.LastName, av.[Value] AS GivingBin
FROM Person p
JOIN AttributeValue av ON av.EntityId = p.Id
JOIN Attribute a      ON a.Id = av.AttributeId
WHERE a.[Guid] = '7FBB63CC-F4FC-4F7E-A8C5-44DC3D0F0720';
```

CUSTOM We calculate our own bins live, per person, from the last 12 months of contribution gifts: bin 1 the top 5%, bin 2 the next 15%, bin 3 the next 20%, bin 4 the rest, bin 5 no gift. Only do this if you need live, per-person bins; the core attribute is refreshed by a job and handles refunds and giving units for you.

The 30-Second Cliff

Page SQL has a default command timeout of 30 seconds. Our first dashboards ran their heavy queries on every page view: 3 to 7 seconds on a quiet day, and a hard timeout the first time the database was busy. Moving the heavy work into a persisted dataset brought the same page to about 200 ms.

- Reach for a persisted dataset whenever page SQL reads six figures of rows, especially from `Interaction`, `FinancialTransaction` or `AttributeValue`.
- **Count passes, not just scope.** Seven small, well-indexed queries per page view still add up; one of our pages timed out that way.
- `{% sql timeout:'180' %}` is fine inside a dataset build, where waiting is free. On a page it only moves the cliff.

The Persisted Dataset Pattern

CORE feature, **CONFIGURED** record, **CUSTOM** build script. In Rock 17 and later: CMS Configuration, then Content Platform, then **Persisted Datasets**. Add a dataset with:

- **Access Key:** for example `eraLeaving`. Pages use this to read it.
- **Enabled Lava Commands:** `Sql`.
- **Refresh Interval:** in minutes. We use 60 to 360 depending on the page.
- **Build Script:** Lava whose output is **pure JSON**.

Build script

```
{% sql return:'rows' timeout:'180' %}
-- run the "Reading eRA in SQL" query into #Leaving first, then:
SELECT Campus, COUNT(*) AS Families
FROM #Leaving
GROUP BY Campus
{% endsql %}{{ rows | ToJSON }}
```

Dashboard block

```
{% assign data = 'eraLeaving' | PersistedDataset %}
{% for row in data %}
  <div>{{ row.Campus }}: {{ row.Families }}</div>
{% endfor %}
```

Gotchas. One stray HTML comment or blank banner in the build output makes it invalid JSON. Temp tables work inside the build. Rock's **Update Persisted Datasets** job must be running. For anything that changes minute to minute (a new assignment), overlay a small live query on top of the cached data.

Built on Core Rock Steps

The follow-up record is one **Step** per family, placed on the head of household. Create these in your Rock (People, then Steps):

Item	Class	Setup
Step Program "eRA Follow-Up"	CONFIGURED	Create your own program. Do not use the core "Core Steps" program; statuses added there show up on core Step Types.
Step Type "eRA Follow-Up"	CONFIGURED	Allow multiple = yes; has end date = no.
Statuses	CONFIGURED	Open: In Progress, Assigned, Escalated . Closed: Connected, Closed, Complete . Tip: flag the closed ones as complete statuses, so "open" is simply <code>IsCompleteStatus = 0</code> .
Step attributes	CONFIGURED	See the table below.
Note Type "eRA Follow-Up Note"	CONFIGURED	Entity: Person. Security: allow your staff roles, deny All Users, so people don't see these notes on their own profile.
Reminder Type "eRA Follow-Up"	CONFIGURED	Entity: Person. Notification type Communication with no notification workflow (bell only, no email). Show Note on.
Link from Step to family	CUSTOM	We store the family group Id in <code>Step.ForeignKey</code> using the Entity Property Set action (Step Add has no input for it).

Attribute key	Field type	Purpose
---------------	------------	---------

AssignedTo	Person	Who owns the follow-up. Stored as a PersonAlias GUID .
AssignedAt	Date Time	Starts the age clock
LastTouchAt	Date Time	Last logged contact
LastTouchBy	Person	Who logged it
LastOutcome	Text	Reached, Voicemail, WrongInfo, Closed
Outcome	Text	Final outcome with reason, for example "Closed (Moved)"
EscalatedAt	Date Time	When it was escalated
EscalatedTo	Person	Usually the campus pastor
OriginEvent	Text	Entering or Leaving
SourceCampus	Campus	The head of household's campus (stored as the campus GUID)

The Follow-Up Loop: Workflows W1 to W5

CONFIGURED workflow types built only from core actions (**Lava Run**, **Step Add**, **Reminder Add**, **Person Note Add**, **Entity Attribute Set**, **Entity Property Set**, **Attribute Set from Entity**, **Activate Activity**). The Lava and SQL inside them is **CUSTOM**. Pages launch them with the core REST endpoint `POST /api/Workflows/WorkflowEntry/{workflowTypeId}`, passing attribute values on the query string.

W1 Assign

Inputs: FamilyGroupId, AssigneePersonId, AssignerPersonId, OriginEvent.

1. Look for the family's existing **open** follow-up Step. If found, reuse it (this makes re-assigning safe).
2. Find the head of household and resolve people into **Person-type** workflow attributes (Step Add and Reminder Add need Person attributes, not Ids).
3. **Step Add**, only when no open Step exists. Then set `ForeignKey` to the family.
4. **Entity Attribute Set:** AssignedTo, AssignedAt, OriginEvent, SourceCampus.
5. Clear any earlier open reminders for this family, then **Reminder Add** for the assignee (skip when unassigning).

Unassign by passing `0`, never an empty string: an empty value interpolated into SQL is a syntax error and leaves the workflow stuck in Active.

W2 Log Outcome

Inputs: FamilyGroupId, Outcome, SubReason, NoteText, StaffPersonId. Maps the outcome to a status (see "One Tap to Log What Happened"), sets LastTouchAt, LastTouchBy and LastOutcome, adds a **Person Note Add** note on the head of household, and completes or clears the family's reminder.

W3 Stale Escalation

Scheduled daily with the core **Launch Workflow** job. Finds Assigned Steps with no touch for N days, then escalates them one at a time: status Escalated, EscalatedAt, EscalatedTo the campus pastor (via `Campus.TeamGroupId` and the core Pastor role, matched by GUID), and a reminder for the pastor. Include a DryRun attribute.

Do not loop with `{% workflowactivate workflowtype:'...' activitytype:'...' %}`. Without a `workflowid` it starts a brand-new workflow each time, which runs its "find" activity again, which starts more workflows. Loop instead with an activity that pops one item from a queue attribute, does its work, and re-activates itself with **Activate Activity**, with a counter that stops it after a safe maximum.

W4 Bulk Transfer

Inputs: FromPersonId, ToPersonId, AdminPersonId, ResetAgeClock. Builds a queue of the person's open Steps and moves AssignedTo (and optionally AssignedAt) one at a time, then logs a summary. Move or recreate the reminders too.

W5 Reminder Completed

Triggered by a core **Workflow Trigger** on `Rock.Model.Reminder`, **Post Save**, qualifier `IsComplete` changing from False to True. If the person who ticked it owns the family's open Step, record the touch (LastOutcome "Reminder completed, outcome not logged") and add one reminder asking them to log the outcome.

A workflow trigger created through the REST API may not fire until it has been saved once in the UI or the cache is cleared.

One Tap to Log What Happened

Outcome	Step status	Also
Reached	Connected	Note saved; reminder completed
Voicemail	stays In Progress	Touch recorded; reminder kept
Wrong info	stays In Progress	Touch recorded; flag the record for a data fix
Closed	Closed	Outcome with reason (moved, another church, asked not to be contacted)

Notes Belong to the Whole Family

CUSTOM W2 writes the note on the head of household. The page then reads that note back and copies it to the family's other eRA members with the core REST endpoint `POST /api/Notes` (same note type, same text), so whoever reaches out next sees the whole conversation. Treat the copy as best-effort: the Step has already been updated, so a failed copy should never show as a failed outcome.

Nudges Without More Email

- **Assigning** adds a Rock reminder for the owner. With the bell-only reminder type there's no email; the count shows on the Reminders bell.
- **Logging** an outcome completes or clears it.
- **Ticking it off without logging** fires W5, which records the touch and asks how it went.
- Keep the core **Process Reminders** job running. Reminder notes must be plain text; the edit form rejects `<` and `>`.

Seeing the Whole Team's Load

CUSTOM Age an open follow-up from its last touch, or from assignment when there has been no touch yet. Bands: under 14 days healthy, 14 to 30 stale, over 30 critical.

```
DECLARE @StepTypeId int = (SELECT Id FROM StepType WHERE [Guid] = '<your eRA Follow-Up step type GUID>');
DECLARE @StepEntityTypeId int = (SELECT Id FROM EntityType WHERE [Name] = 'Rock.Model.Step');

SELECT s.Id AS StepId,
       ISNULL(assignee.NickName + ' ' + assignee.LastName, 'Unassigned') AS AssignedTo,
       DATEDIFF(day,
                COALESCE(TRY_CAST(NULLIF(lt.[Value], '') AS datetime2),
                        TRY_CAST(NULLIF(aa.[Value], '') AS datetime2)),
                GETDATE()) AS DaysSinceTouch
FROM Step s
JOIN StepStatus ss ON ss.Id = s.StepStatusId AND ss.IsCompleteStatus = 0
OUTER APPLY (SELECT TOP 1 av.[Value] FROM AttributeValue av JOIN Attribute a ON a.Id =
av.AttributeId
              WHERE a.EntityTypeId = @StepEntityTypeId AND a.[Key] = 'AssignedTo' AND av.EntityId
= s.Id) at
OUTER APPLY (SELECT TOP 1 av.[Value] FROM AttributeValue av JOIN Attribute a ON a.Id =
av.AttributeId
              WHERE a.EntityTypeId = @StepEntityTypeId AND a.[Key] = 'AssignedAt' AND av.EntityId
= s.Id) aa
OUTER APPLY (SELECT TOP 1 av.[Value] FROM AttributeValue av JOIN Attribute a ON a.Id =
av.AttributeId
              WHERE a.EntityTypeId = @StepEntityTypeId AND a.[Key] = 'LastTouchAt' AND
av.EntityId = s.Id) lt
LEFT JOIN PersonAlias pa    ON pa.[Guid] = TRY_CAST(at.[Value] AS uniqueidentifier)
LEFT JOIN Person assignee  ON assignee.Id = pa.PersonId
WHERE s.StepTypeId = @StepTypeId;
```

Two traps in one query. A Person attribute stores a **PersonAlias GUID**, so join `PersonAlias.[Guid]`, never cast it to an int. And `TRY_CAST('' AS datetime2)` returns 1900-01-01, not NULL, so wrap values in `NULLIF(..., '')` or every blank reads as critical.

Did Anyone Use It?

CORE Every page view is an `Interaction`. Always filter by `InteractionComponentId` (indexed); an unscoped scan of `Interaction` will time out.

```
-- 1. Find the interaction components for your eRA pages (run once)
SELECT ic.Id, ic.[Name], ch.[Name] AS Channel
FROM InteractionComponent ic
JOIN InteractionChannel ch ON ch.Id = ic.InteractionChannelId
WHERE ic.EntityId IN (<your eRA page Ids>)
      AND ch.ComponentEntityType = (SELECT Id FROM EntityType WHERE [Name] = 'Rock.Model.Page');

-- 2. Views per staff member, last 30 days
SELECT p.NickName + ' ' + p.LastName AS Staff, COUNT(*) AS Views, MAX(i.InteractionDate) AS
LastView
FROM Interaction i
JOIN PersonAlias pa ON pa.Id = i.PersonAliasId
JOIN Person p      ON p.Id = pa.PersonId
WHERE i.InteractionComponentId IN (<component Ids from step 1>)
      AND i.Operation = 'View'
      AND i.InteractionDate ≥ DATEADD(day, -30, GETDATE())
GROUP BY p.NickName, p.LastName
ORDER BY Views DESC;
```

What's Core, and What We Built

Component	Class	How to recreate
eRA calculation and person attributes	CORE	Check the three settings; keep Family Analytics running
eRA Entry / Exit Workflow settings	CORE	Optional: point them at a workflow type
Core eRA Step Type and person badge	CORE	Nothing to do (Rock 18+ for the Step Type)
Giving Bin attribute	CORE	Keep the Giving Automation job running
Steps, Reminders, Notes, Workflows, Workflow Triggers	CORE	Available on every install
Persisted datasets and the Update Persisted Datasets job	CORE	Make sure the job is scheduled
Interactions (page views)	CORE	Recorded automatically for internal pages
HTML Content block with Lava <code>{% sql %}</code>	CORE	Enable the Sql Lava command per block
Core staff roles as the assignee pool	CORE	RSR - Staff Workers 2C112948-FF4C-46E7-981A-0257681EADF4 , RSR - Staff Like Workers 300BA2C8-49A3-44BA-A82A-82E3FD8C3745

Campus pastor lookup	CORE	Campus Team group (<code>Campus.TeamGroupId</code>) with the core Pastor role
eRA Follow-Up Step Program, Step Type, six statuses	CONFIGURED	People, then Steps
Ten Step attributes	CONFIGURED	On the Step Type
Five workflow types (W1 to W5) and one workflow trigger	CONFIGURED	Admin Tools, then Workflow configuration
Note type and reminder type	CONFIGURED	As described above
One persisted dataset per heavy dashboard	CONFIGURED	CMS Configuration, then Content Platform
Launch Workflow job for W3	CONFIGURED	Jobs Administration
Dashboard, queue, manager and transfer pages	CUSTOM	Lava, SQL and a little JavaScript in HTML Content blocks; Chart.js for charts
Lava and SQL inside the workflows	CUSTOM	Patterns above
Copying notes to the whole family	CUSTOM	Page JavaScript calling <code>POST /api/Notes</code>
Our own giving bins	CUSTOM	Optional; prefer the core Giving Bin

Lessons We Learned the Hard Way

1. **Make workflows safe to run twice.** Re-assigning created a duplicate Step every time until W1 looked for the open Step first.
2. **Open wins.** When a family has several Steps, choose the row's Step with open statuses first, then newest: `ORDER BY CASE WHEN ss.IsCompleteStatus = 0 THEN 0 ELSE 1 END, s.Id DESC`. Newest alone shows a closed Step and makes re-assignments look like they "revert".
3. **A Person attribute stores a PersonAlias GUID.** Casting it to an int silently broke three pages and turned bulk transfer into a no-op.
4. **Don't loop with workflowactivate** (see W3).
5. **Stamp every build.** If a block has several content versions, make sure you edit the one Rock is actually serving, then confirm the stamp on the live page.
6. **Write attributes with Entity Attribute Set or the REST API, never with a raw** `UPDATE AttributeValue`. Rock keeps denormalized copies that raw SQL leaves stale.
7. **A workflow action that errors leaves the workflow stuck in Active.** Gate risky actions on `IsNotBlank` and never interpolate an empty value into SQL.
8. **Sort on something unique.** `ORDER BY` a non-unique column under `TOP` or `ROW_NUMBER` returns a different set between page loads; add the `Id` as a tiebreaker.

What We'd Do Differently

- **Create our own Step Program.** We put ours in Rock 18's Core Steps program, so our statuses appear on core Step Types.
- **Start with the core Giving Bin** and only write our own if we truly needed live, per-person bins.
- **Use the eRA Exit Workflow** to create the follow-up the moment a family leaves eRA, then assign it.
- **Match by GUID, never by name.** Our campus-pastor lookup still matches the role by name; one rename in the UI and it finds nobody.

AI Project Plan

A phased plan you can follow yourself or hand to an AI coding assistant together with the starter prompt below. Each phase ends with something a pastor can use, and with a check you can run before moving on. Build every phase on a sandbox page first.

Phase 0 · Confirm eRA is right (1 to 2 hours)

Check the three settings. Run Family Analytics. Spot-check ten families you know. **Done when** the people you expect to be regular are, and a family you know stopped attending shows an eRA end date.

Phase 1 · A "Leaving" list (half a day)

One HTML Content block with the "Reading eRA in SQL" query, grouped by family, filtered by campus. Give it to one pastor. **Done when** the pastor has called someone from the list.

Phase 2 · Ownership: Step Type and an assign workflow (1 to 2 days)

Create your Step Program, Step Type, statuses and the ten attributes. Build W1 (or use the eRA Exit Workflow setting). Add an "Assign" button to the list. **Done when** assigning the same family twice leaves exactly one open Step.

Phase 3 · Outcomes and notes (1 to 2 days)

Build W2 and the one-tap outcome form, the note type, and the family note copy. **Done when** each outcome moves the Step to the right status and the note appears on every family member.

Phase 4 · Dashboard on a persisted dataset (2 to 3 days)

Entering, leaving, net and currently-eRA cards; connection at exit; six-month trend; campus filter; every number clickable. Heavy SQL goes in a persisted dataset. **Done when** the page loads in under a second and every list matches its number.

Phase 5 · Reminders, workload and adoption (2 to 3 days)

Reminder type, W5 trigger, My Follow-Ups queue, workload view, W4 transfer, W3 escalation (dry run first), and an Interactions usage view. **Done when** every open follow-up has an owner, an age band and a reminder.

Time estimates assume someone comfortable with Lava and SQL working part-time; adjust for your team.

AI Starter Prompt

This prompt turns a general-purpose AI assistant (Claude, ChatGPT, Copilot or similar) into a Rock-aware build partner that already knows everything in this document: what Rock gives you, the traps we hit, and the order to build in. It won't have access to your Rock server; you stay the person who clicks, pastes and tests.

How to use it

1. **Fill in the "About our Rock" block** before you paste. Anything you don't know, write "don't know" and the assistant will tell you where to look.
2. **Paste the whole prompt as your first message** in a new conversation (or save it as a project or custom instruction so it applies to every chat).
3. **Work one phase at a time.** The prompt tells the assistant to stop after each phase. Test on a sandbox page, then reply "next".
4. **Paste structure, never people.** Share query text, error messages, block settings and column names. Don't paste query results that contain names, emails, phone numbers or giving amounts.
5. **Read before you run.** Run any SQL it writes as a `SELECT` first, in a sandbox, and read what it changes before touching production.

The prompt

```
# ROLE
You are a senior Rock RMS developer and a patient teacher. You are helping
[church name] build an eRA follow-up system: a way to notice families who
are drifting away and make sure a real person reaches out to them.

# ABOUT OUR ROCK (fill in; write "don't know" if unsure)
- Rock version: [e.g. 18.2]
- Hosting: [self-hosted SQL Server / Azure SQL / hosting partner]
- Campuses: [number, or "single campus"]
- Weekend service attendance group types: [names]
- Small group type(s): [names]
- Serving teams use the core "Serving Area" group type purpose: [yes/no]
- Contribution accounts are marked tax-deductible: [yes/no]
- Who will do the follow-up: [campus pastors / care team / role names]
- Who may see giving-related segments: [roles]
- Our builder: [role], comfortable with [Lava / SQL / neither]
- Time available: [hours per week]
- We have a sandbox page or test server: [yes/no]

# WHAT ROCK ALREADY GIVES US (build on this; don't reinvent it)
- The Family Analytics job (Rock.Jobs.CalculateFamilyAnalytics) sets eRA
  for each family weekly via spCrm_FamilyAnalyticsEraDataset.
  Entry: 4+ gifts in 52 weeks and 1+ in 6 weeks, OR 8+ Sundays in 16 weeks.
  Exit: no gifts in 8 weeks, no attendance in 4, fewer than 8 Sundays in 16.
  Only group types flagged "Attendance counts as weekend service" and
```

tax-deductible contribution accounts count.

- Person attributes: core_CurrentlyAnEra (DELETED, not set to False, on exit), core_EraStartDate, core_EraEndDate, plus giving and check-in history (core_EraLastGave, core_EraTimesGiven52Wks, ...).
- The job's "eRA Entry Workflow" and "eRA Exit Workflow" settings launch a workflow per family and fill workflow attributes keyed Family (Group), HeadOfHouse (Person), Spouse (Person) and Campus (Campus) if they exist.
- Giving Automation writes a core Giving Bin person attribute (GUID 7FBB63CC-F4FC-4F7E-A8C5-44DC3D0F0720).
- Steps, Workflows (Step Add, Reminder Add, Person Note Add, Entity Attribute Set, Entity Property Set, Lava Run, Activate Activity), Reminders, Notes, Workflow Triggers, Persisted Datasets, Interactions, and HTML Content blocks with the Sql Lava command enabled.

RULES YOU MUST FOLLOW

1. Core first. Propose custom code only when no core feature fits, and say why.
2. Label every piece you propose CORE, CONFIGURED or CUSTOM.
3. Never hard-code Ids. Look up attributes by key and defined values, group types and system groups by GUID. For records we create (our Step Type, workflow types), ask me to paste the GUID.
4. A Person-type attribute value is a PersonAlias GUID: join PersonAlias.[Guid], never cast it to an int.
5. Wrap string-to-date casts in NULLIF(value, ''), because TRY_CAST('' AS datetime2) returns 1900-01-01, not NULL.
6. Build households from GroupMember on the family group. Pick one phone per person with OUTER APPLY (SELECT TOP 1 ...) so nobody appears twice.
7. Sort on something unique: add the Id as a tiebreaker under TOP or ROW_NUMBER.
8. Page performance budget: under 2 seconds. Anything that reads large tables (Interaction, FinancialTransaction, AttributeValue) on page load goes into a persisted dataset whose build output is pure JSON. Always filter Interaction by InteractionComponentId.
9. Workflows must be safe to run twice: the same family must never get a second open Step or a second reminder. Never interpolate an empty string into SQL (pass 0). Gate risky actions on IsNotBlank. Never loop with {% workflowactivate %} without a workflowid; loop with an activity that re-activates itself and a maximum counter.
10. Write attribute values with Entity Attribute Set or the REST API, never with raw UPDATE statements against AttributeValue.
11. Every count on a dashboard must equal the list it opens. Emit both from the same query and filters.
12. Security: follow-up pages restricted to [roles]; the follow-up note type visible to staff roles only (deny All Users); giving segments visible only to [roles].
13. Privacy: never ask me for real names, emails, phone numbers or giving amounts. Use obviously fake sample data (example.org, (555) 01xx). If I paste output containing personal data, tell me to remove it.
14. If you are not sure a table, column, Lava filter, block setting or workflow action exists in my Rock version, say so and give me a query or a place in the UI to check. Do not invent it.

HOW TO WORK WITH ME (for every phase)

- a. Restate the goal in one paragraph a pastor would understand.
- b. List what to create, each labelled CORE / CONFIGURED / CUSTOM.
- c. Give exact click paths in Rock for configuration.
- d. Give complete code (SQL, Lava, workflow action settings). No "...".

- e. Give a sandbox test plan: what the numbers should look like and how to prove each list matches its count.
- f. List the risks and how to undo the change.
- g. STOP and wait for me to reply "next".

THE PHASES

- Phase 0 Confirm eRA is right: the settings checklist, a query of eRA counts by campus, and ten sample families for me to spot-check.
- Phase 1 A "Leaving in the last 30 days" list: one row per family, campus filter, a count at the top that equals the rows.
- Phase 2 Ownership: our OWN Step Program (not the core "Core Steps" program) with an "eRA Follow-Up" Step Type; statuses In Progress, Assigned, Escalated (open) and Connected, Closed, Complete (closed, flagged complete); attributes AssignedTo, AssignedAt, LastTouchAt, LastTouchBy, LastOutcome, Outcome, OriginEvent, SourceCampus, EscalatedAt, EscalatedTo. An idempotent Assign workflow launched from the list with POST /api/Workflows/WorkflowEntry/{id}, or from the eRA Exit Workflow setting.
- Phase 3 Log outcome: Reached → Connected; Voicemail and Wrong info stay open; Closed with a reason. A staff-only note type, and the note copied to every family member.
- Phase 4 Dashboard on a persisted dataset: Entering, Leaving, Net and Currently eRA cards; connection at exit; six-month trend; campus filter; every number clickable.
- Phase 5 A bell-only reminder type; a Workflow Trigger when a reminder is completed without an outcome; a My Follow-Ups queue with healthy / stale / critical bands (age = COALESCE>LastTouchAt, AssignedAt)); a workload view; bulk transfer; stale escalation (dry run first); adoption from Interactions.

Before Phase 0, ask me about anything I left blank above.

Useful follow-up prompts

- "Review this SQL against the rules in my first message, especially hard-coded Ids, PersonAlias joins and NULLIF: [paste the query]."
- "The card says 33 but the list shows 31. Here are both queries: [paste]. Find the difference."
- "This page takes 12 seconds. Turn its heavy query into a persisted dataset build script and the matching page Lava: [paste]."
- "This workflow is stuck in Active. Here are the action settings and the workflow log: [paste]. Which action errored and why?"
- "Write a sandbox test plan for Phase 3 using three fictional families."
- "Explain what this block does to a campus pastor in three sentences."
- "List every CONFIGURED and CUSTOM piece we have built so far, so I can document it."

Keep a human in the loop. AI gets you past the blank page, but it can be confidently wrong about a column name or a Rock version difference. Test every phase in a sandbox, run queries read-only first, and keep your Lava and SQL in version control.

Start Monday

- **eRA is already in Rock.** Check the three settings before you build anything.
- **Start with one list and one person.** Families whose eRA ended in the last 30 days, handed to one pastor to call this week.
- **Give every follow-up one owner** and a one-tap way to record what happened.
- **Build on core:** Steps, Workflows, Reminders, Notes, Persisted Datasets. Reach for custom code last.
- **Keep dashboards fast and honest:** heavy SQL in a persisted dataset; every list adds up to its number.
- **Measure adoption** with Interactions, and adjust.

Questions after RX?

Paul Loots, Director of Innovation · ploots@gfcflorida.com

Ann Morricks, Donor Development Director · amorricks@gfcflorida.com