Below is a structured example of how you might document and explain a multi-level process (root-level activities with sub-activities) for a credit card company. This example will help illustrate:

1. **How events can be organized** (root vs. sub-activities).
2. **How to capture parent-child relationships** in your data model.
3. **What kind of filtering or “level of detail” control** you can enable for stakeholders.

**1. Business Context (Credit Card Application)**

Imagine you have a credit card company processing customer applications. The *root-level* activities of the overall process might be:

1. **Application Submission**
2. **Initial Check**
3. **Risk Assessment**
4. **Approval Decision**
5. **Card Issuance**

These represent the **high-level** “milestones” in the process.

However, each of these root-level activities can also break down into multiple **sub-activities**. For example:

* **Application Submission**
  + A1: Gather Customer Details (basic personal info, contact info)
  + A2: Collect Supporting Documents (ID proof, address proof, etc.)
* **Initial Check**
  + B1: Preliminary Credit Score Check
  + B2: Validate Application Completeness
* **Risk Assessment**
  + C1: Fraud Check
  + C2: Income Verification
  + C3: Manual Review (if needed)
* **Approval Decision**
  + D1: Automated Approval
  + D2: Manual Approval Review
  + D3: Rejection Notice
* **Card Issuance**
  + E1: Card Printing/Personalization
  + E2: Dispatch/Shipping

Each of those sub-activities can have further *nested* sub-activities if needed. For instance, under A1 (Gather Customer Details), there might be deeper steps like verifying email address, capturing consent, etc.

**2. Event Log Structure**

You can think of each *activity* (whether root-level or sub-level) as a node in your process. Every time an activity is *started* or *completed*, or any relevant event happens, you have an entry in your event log.

**2.1. Essential Fields**

A minimal set of fields might look like this:

| **Field** | **Description** |
| --- | --- |
| **EventID** | Unique identifier for each event (e.g., EVT12345). |
| **ActivityID** | Unique identifier for the activity (e.g., ACTA1 for "Gather Customer Details"). |
| **ActivityName** | Name/label of the activity (e.g., "Gather Customer Details"). |
| **ParentActivityID** | If this activity is a sub-activity, the ID of its immediate parent (e.g., ACTA if this is a child of "Application Submission"). If it has no parent, it can be NULL or a special marker. |
| **Timestamp** | When this event occurred (start, end, or any relevant point in time). |
| **CaseID** | Identifier to tie events to a single process instance (e.g., the application number). |
| **EventType** | Type of event (e.g., “start,” “complete,” “update,” etc.), if needed. |
| **OtherDetails** | Additional metadata (e.g., user ID, channel, outcome, reason codes). |

*Note*: In real process mining setups, you might have many more fields or even separate tables for event attributes. But the above is the minimum to capture the hierarchy and the flow.

**2.2. Parent-Child Relationship**

* **ParentActivityID** ties sub-activities to their respective parents.
* For example, the main root-level activities (A, B, C, D, E) might have no parent (ParentActivityID = NULL).
* Sub-activities (A1, A2, etc.) would reference the root-level activity’s ID as their parent (ParentActivityID = ACTA if ACTA is “Application Submission”).
* If there are deeper nested levels, the sub-sub-activity references the sub-activity’s ID.

This structure ensures that at any given time, the process mining tool or data analysis query can:

1. Show all **root-level** events (e.g., A, B, C, D, E) for a high-level overview.
2. Drill down into sub-activities by filtering on ParentActivityID = <desired-activity>.
3. Continue drilling down recursively as needed.

**3. Example Data Rows**

Below is a simplified sample (in a table form) of how 10 events might appear in an event log for a single credit card application (CaseID = APP001):

| **EventID** | **ActivityID** | **ActivityName** | **ParentActivityID** | **Timestamp** | **CaseID** | **EventType** | **OtherDetails** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| EVT0001 | ACTA | Application Submission | NULL | 2025-01-20 08:05:00 | APP001 | start | Channel = "Online" |
| EVT0002 | ACTA1 | Gather Customer Details | ACTA | 2025-01-20 08:06:00 | APP001 | start | N/A |
| EVT0003 | ACTA1 | Gather Customer Details | ACTA | 2025-01-20 08:08:00 | APP001 | complete | Name="John Doe", Email="..." |
| EVT0004 | ACTA2 | Collect Supporting Documents | ACTA | 2025-01-20 08:10:00 | APP001 | start | Documents="ID Proof, Utility Bill" |
| EVT0005 | ACTA2 | Collect Supporting Documents | ACTA | 2025-01-20 08:15:00 | APP001 | complete | Verification="Passed" |
| EVT0006 | ACTB | Initial Check | NULL | 2025-01-20 08:20:00 | APP001 | start | AssignedTo="System Checker" |
| EVT0007 | ACTB1 | Preliminary Credit Score Check | ACTB | 2025-01-20 08:21:00 | APP001 | complete | FICO="720" |
| EVT0008 | ACTB2 | Validate Application | ACTB | 2025-01-20 08:23:00 | APP001 | complete | Status="Valid" |
| EVT0009 | ACTC | Risk Assessment | NULL | 2025-01-20 08:25:00 | APP001 | start | ManualReviewNeeded="No" |
| EVT0010 | ACTC1 | Fraud Check | ACTC | 2025-01-20 08:28:00 | APP001 | complete | FraudAlert="None" |

*(Note that this is just an illustrative snippet; in real scenarios you’d have many more rows, more timestamps, more detailed data, etc.)*

**4. Filtering and Level of Detail**

When creating a process model (e.g., a BPMN diagram or a variant analysis in your process mining tool), you often **don’t want to show** every single micro-step—because 400 or 500 events per case can become visually overwhelming. Instead, you might:

1. **Start at the Root-Level**: Show the main sequence A -> B -> C -> D -> E (e.g., “Application Submission,” “Initial Check,” “Risk Assessment,” “Approval Decision,” “Card Issuance”).
2. **Drill Down as Needed**: If the stakeholder is interested in how “Application Submission” is performed, you expand or filter to see A1, A2 inside it.
3. **Hide Irrelevant Activities**: Some activities might be purely “system logs” or “trivial steps” that do not change the high-level process or the outcome. You can choose to filter them out. For example, in a real system you may get events like “Price calculation check,” “Logo rendering on page,” “Autosave triggered,” etc., which are not relevant to the actual business *workflow*.

**4.1. Possible Filters**

* **Time-based filters**: “Show me only events that occurred in the last month.”
* **Activity-based filters**: “Show me only root-level activities.” Or “Show me only sub-activities of ACTC.”
* **Outcome-based filters**: “Show me only application journeys that resulted in a rejection.”
* **Custom attribute-based filters**: “Show me only events triggered by a specific department or with a certain risk level.”

Using these filters, you can build multiple **views** of the same underlying event data:

1. **Executive Summary View**: Only the high-level, root activities (A, B, C, D, E).
2. **Operations View**: Potentially expand sub-activities of interest, such as “Application Submission -> (Gather Customer Details, Collect Supporting Documents).”
3. **Exception Handling View**: Focus on risk or error events for specialized analysis.

**5. Why the Parent-Child Model is Important**

1. **Scalability**: If every sub-activity is just a random label, you lose the ability to group them properly under their parent. With the parent-child model, you can easily aggregate or collapse child activities into a single parent.
2. **Drill-Down Capability**: Process mining or any business-intelligence tool can show you top-level processes first, and only when you need more detail, you expand to see children (and children of children, etc.).
3. **Flexibility**: Different lines of business might define sub-activities differently, or they might not exist at all for certain cases. By referencing a ParentActivityID, you can handle these hierarchical differences gracefully.

**6. Summary / Recommendation**

* **Event Data Structure**: Each event has a unique EventID, an ActivityID, a ParentActivityID (if applicable), and a Timestamp.
* **Root vs. Sub-Activities**: Root-level activities have no ParentActivityID (or NULL). Sub-activities reference the ID of their immediate parent. You can nest further levels similarly.
* **Filtering**: Provide the ability to hide or show entire branches of the activity hierarchy. This prevents clutter in your process visualization.
* **Documenting the Data**: Keep a clear data dictionary or reference table that explains each ActivityID, its parent, and its general purpose. This helps new team members quickly understand the structure.

With this approach, your process mining initiative can manage large numbers of events from multiple applications without losing sight of the **big picture** or drowning in unnecessary detail. You’ll have the flexibility to present stakeholder-friendly overviews while still drilling down into the detailed sub-activities whenever deeper analysis is required.

Below is a more comprehensive explanation, building on our previous discussion. We will address:

1. **Why a “parent event ID”** can help disambiguate sub-activities that have the same activity name but different parents.
2. **How grouping** (or bundling) sub-activities can reduce clutter at the top-level view.
3. **Data requirements** and an **illustrative example** with multiple applications (and ~20 events each).
4. **Filtering strategies** to ensure only relevant data appears in the final process visualization.

**1. Updated Requirements**

1. **Parent Event ID**:
   * In addition to a ParentActivityID, each event can have a ParentEventID pointing to the **specific event** that triggered or spawned it.
   * This is useful when multiple root activities or different branches share a sub-activity name (e.g., “B1” and “E1” are both “Document Review”). By using ParentEventID, you can distinguish which specific event gave rise to the new sub-event.
2. **Grouping (Bundling) of Sub-Activities**:
   * Some sub-activities might be too granular for a business-level view. For instance, if sub-activity A1 can contain 20 smaller events (start, processing, notification, etc.), you might “group” them into 4 summary steps so that the top-level process map only shows these 4 steps.
   * This grouping allows different levels of detail, depending on the stakeholder.
3. **Filtering**:
   * The system must allow flexible filtering based on different criteria (e.g., by root activity, sub-activity, grouping, outcome, or time).
   * Only the top 10 activities (out of 20 total) might be truly relevant for business stakeholders, so we need a way to hide or show certain events.

**2. Data Model Proposal**

Below is an updated structure that includes **ParentEventID** alongside **ParentActivityID**.

| **Field** | **Description** |
| --- | --- |
| **EventID** | Unique identifier for the event (e.g., EVT1001). |
| **ActivityID** | Unique identifier for the activity type (e.g., ACTB1, “DocumentReview”). |
| **ActivityName** | A human-readable label for the activity (e.g., “Document Review”). |
| **ParentActivityID** | The activity ID of the immediate parent (if this event is a child in the activity hierarchy). If NULL, it’s a root-level activity. |
| **ParentEventID** | The specific event that spawned the current event (e.g., EVT0006). Useful when multiple events share the same ActivityID but are triggered by different parents. |
| **CaseID** | Links the event to a specific process instance or journey (e.g., “APP001” for application #1). |
| **Timestamp** | When this event occurred. |
| **EventType** | Type of event (start, complete, update, etc.). |
| **Grouping** | Optional field to identify a “bundle” or “summary group” (e.g., “Notification Steps” or “Processing Steps”). This helps in collapsing or expanding multiple micro-steps. |
| **OtherDetails** | Additional attributes (e.g., user, channel, outcome, reason code). |

**2.1. Why ParentEventID Helps**

* **Scenario**: Activity B and Activity E both have a sub-activity “Review.” If you only used ParentActivityID = ACTB or ParentActivityID = ACTE, you could tell which parent activity “Review” belongs to, but you might still have multiple parallel instances. By adding a **ParentEventID**, you know exactly which event triggered which “Review” event.
* This is especially helpful in **concurrent** or **parallel** branches, or if you have the same sub-activity repeating multiple times within one parent activity.

**3. Example With Three Applications**

Let’s consider **3 credit card applications**:

* **APP001** (20 events)
* **APP002** (20 events)
* **APP003** (20 events)

Each has some subset of root activities: A, B, C, D, E (like in the previous example). But in total, you might have **40 or 50** different “raw” event types across sub-activities. Out of these, maybe **10** are truly business-critical (the rest are technical or very detailed).

**3.1. Illustrative Data Snippet**

Below is a **condensed** sample to show you how the new fields (ParentEventID, Grouping) might be used. (We won’t list all 60 events across 3 applications, but let’s show a representative set.)

| **EventID** | **ActivityID** | **ActivityName** | **ParentActivityID** | **ParentEventID** | **Timestamp** | **CaseID** | **EventType** | **Grouping** | **OtherDetails** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EVT0001 | ACTA | Application Submission | NULL | NULL | 2025-01-20 08:00:00 | APP001 | start | Root-Level | Channel="Online" |
| EVT0002 | ACTA1 | Gather Customer Details | ACTA | EVT0001 | 2025-01-20 08:02:00 | APP001 | start | Data Collection | - |
| EVT0003 | ACTA1 | Gather Customer Details | ACTA | EVT0002 | 2025-01-20 08:05:00 | APP001 | complete | Data Collection | CustomerName="John Doe" |
| EVT0004 | ACTA2 | Collect Supporting Documents | ACTA | EVT0001 | 2025-01-20 08:06:00 | APP001 | start | Data Collection | DocumentsNeeded="ID, UtilityBill" |
| EVT0005 | ACTA2 | Collect Supporting Documents | ACTA | EVT0004 | 2025-01-20 08:10:00 | APP001 | complete | Data Collection | Verification="All Good" |
| EVT0006 | ACTB | Initial Check | NULL | NULL | 2025-01-20 08:12:00 | APP001 | start | Root-Level | AssignedTo="SystemChecker" |
| EVT0007 | ACTB1 | Preliminary Credit Score Check | ACTB | EVT0006 | 2025-01-20 08:13:00 | APP001 | start | Processing Steps | FICOCheck="InProgress" |
| EVT0008 | ACTB1 | Preliminary Credit Score Check | ACTB | EVT0007 | 2025-01-20 08:14:00 | APP001 | complete | Processing Steps | FICO=720 |
| EVT0009 | ACTB2 | Validate Application | ACTB | EVT0006 | 2025-01-20 08:15:00 | APP001 | complete | Processing Steps | Status="Valid" |
| EVT0010 | ACTB2N | Notification Email | ACTB2 | EVT0009 | 2025-01-20 08:16:00 | APP001 | complete | Notification Steps | EmailSentTo="[john.doe@example.com](mailto:john.doe@example.com)" |

*(Imagine this pattern repeats similarly for APP002 and APP003, each with ~20 events.)*

**Observations:**

* **EventID**: Each event is uniquely identified (EVT0001, EVT0002, etc.).
* **ActivityID** vs. **ActivityName**: ACTA1 might be “Gather Customer Details,” ACTB2N might be “Notification Email,” etc.
* **ParentActivityID**: For ACTA2, it’s ACTA because “Collect Supporting Documents” belongs under “Application Submission.”
* **ParentEventID**: For EVT0004, it’s EVT0001 because that’s the event that started the root-level “Application Submission.” By referencing EVT0001, we know which instance it belongs to.
* **Grouping**:
  + Root-Level for the big five milestones (A, B, C, D, E).
  + Data Collection, Processing Steps, Notification Steps for bundling sub-activities.
  + This grouping helps us filter out minor steps if we only care about the larger categories.

**4. Grouping (Bundling) Logic**

**4.1. Why Group?**

* A sub-activity like ACTB2 might spawn a dozen “micro-steps” (start, check, notify, logging, etc.). For **business** analysis, we might only want to see one “Validate Application” step.
* By assigning all these micro-steps the same **Grouping** value (e.g., “Processing Steps”), the process mining or visualization tool can collapse them into one node if needed.

**4.2. Configuration**

You could store grouping rules in a **reference table** like:

| **ActivityID** | **GroupName** | **IsBusinessCritical** |
| --- | --- | --- |
| ACTA | Root-Level | TRUE |
| ACTA1 | Data Collection | TRUE |
| ACTA2 | Data Collection | TRUE |
| ACTB | Root-Level | TRUE |
| ACTB1 | Processing Steps | TRUE |
| ACTB2 | Processing Steps | TRUE |
| ACTB2N | Notification Steps | FALSE |
| ACTC | Root-Level | TRUE |
| ... | ... | ... |

* **GroupName**: Indicates the higher-level grouping (“Root-Level,” “Processing Steps,” “Notification Steps,” etc.).
* **IsBusinessCritical**: Flags which events to display in the “business view.” For example, ACTB2N might not be that critical from a business perspective and can be hidden unless someone explicitly wants to see notifications.

With these rules:

* A **high-level** process map can filter only on IsBusinessCritical = TRUE -> you end up with fewer nodes and edges.
* A **detailed** view can include IsBusinessCritical = FALSE -> you see all the micro-steps for advanced analysis.

**5. Filtering: Showing Only Relevant Events**

**5.1. Basic Filter Options**

1. **Activity-Level Filter**: Show/hide entire **ActivityIDs** (e.g., hide ACTB2N if not interested in notifications).
2. **Grouping Filter**: Show only “Root-Level” and “Data Collection” groups. Hide “Notification Steps.”
3. **Parent/Child Expand/Collapse**: Start with root-level activities, then expand a single node (like ACTB) to see its children.
4. **Time/CaseID Filter**: For instance, only show events for CaseID = APP001 that happened in the last month.

**5.2. Advanced Filtering**

* **Filtering by ParentEventID**: If you only want to see sub-activities triggered by a specific event, you can do:  
  ParentEventID = '<some-event-id>'.
* **Filtering by “Business Relevance”**: If an attribute like IsBusinessCritical or a similar dimension indicates the event’s importance, you can toggle a “business view” vs. “technical view.”

**6. End-to-End Flow Example (Condensed)**

Below is a more narrative flow, **especially focusing on how grouping** changes what you see at each level:

1. **High-Level (Root View)**: You might only see 5 big nodes in your process mining tool for each journey:
   * Application Submission (A)
   * Initial Check (B)
   * Risk Assessment (C)
   * Approval Decision (D)
   * Card Issuance (E)

These 5 are set as Root-Level in the **GroupName** reference.

1. **Drill Down on “Application Submission” (A)**:
   * Now you see sub-activities (A1, A2). They might both be in the Data Collection grouping.
   * Within “Gather Customer Details (A1)” there could be multiple smaller events (start, validation, complete). If we show “Data Collection” at a more detailed level, we see them. Otherwise, we can collapse them into a single node.
2. **Drill Down on “Initial Check” (B)**:
   * “Preliminary Credit Score Check (B1)” might be within Processing Steps.
   * “Validate Application (B2)” might also be in Processing Steps.
   * Each of those might have more micro-steps (logging, notifications, etc.) grouped under “Notification Steps.”
3. **Filtering Out Non-Business-Critical Steps**:
   * All those small events about sending emails or internal logs can be hidden if IsBusinessCritical = FALSE.
   * The result is a simpler path that only shows “B1” and “B2” as completed tasks.
4. **Comparing Multiple Applications**:
   * If you have 3 journeys (APP001, APP002, APP003), you can see a combined process map. Some journeys might skip certain sub-activities (e.g., manual reviews).
   * The structure remains consistent because each event is tied to CaseID and either references a ParentEventID or a ParentActivityID.

**7. Summary of the Approach**

1. **Data Requirements**
   * Each event must contain enough identifiers (EventID, ParentEventID, ActivityID, ParentActivityID, CaseID, timestamps).
   * Each ActivityID (and optionally each event) should also have a way to classify **grouping** or **business relevance**.
2. **ParentEventID**
   * Solves ambiguity when different root activities or parallel branches have similarly named sub-activities.
   * Ensures we can trace exactly which event triggered the next event in the sequence.
3. **Grouping / Bundling**
   * Allows us to collapse many micro-steps into a single business activity (e.g., “Notification Steps”).
   * Gives you a user-friendly top-level process flow, which can be **expanded** for deeper analysis if needed.
4. **Filtering**
   * You can filter by root-level, sub-level, grouping, or any custom dimension to tailor the process map to your audience.
   * This ensures that out of hundreds of raw events, only ~10 or so might appear at the top-level (the rest are still in the database for deeper dives).

**Final Takeaway**

By **combining** the hierarchical activity data (ParentActivityID), the **specific event lineage** (ParentEventID), and a robust **grouping/filtering** strategy, you can:

* **Provide a clean, high-level** process map for business stakeholders (show only the main events or group-level nodes).
* **Allow deeper technical or operational** drilling into sub-activities or micro-steps when needed, without cluttering the default view.

This structure is flexible enough to handle **multiple applications** (or journeys), large numbers of events, and complex parent-child relationships while still presenting an intelligible process narrative to every type of end-user.

Below is a **complete, step-by-step example** of a credit card application process, showcasing how you can structure activities, sub-activities, and individual events. The aim is to illustrate how all the concepts (parent/child relationships, parent event IDs, groupings, etc.) come together in a single cohesive scenario.

**1. Imagined Use Case: Credit Card Application**

**1.1. Main (Root-Level) Activities**

1. **Application Submission** (ACTA)
2. **Initial Check** (ACTB)
3. **Risk Assessment** (ACTC)
4. **Approval Decision** (ACTD)
5. **Card Issuance** (ACTE)

**1.2. Sub-Activities (Example)**

* **Application Submission (ACTA)**
  + ACTA1: Gather Customer Details
  + ACTA2: Collect Supporting Documents
* **Initial Check (ACTB)**
  + ACTB1: Preliminary Credit Score Check
  + ACTB2: Validate Application
  + ACTB2N: Notification Email (sub-step of Validate Application)
* **Risk Assessment (ACTC)**
  + ACTC1: Fraud Check
  + ACTC2: Income Verification
* **Approval Decision (ACTD)**
  + ACTD1: Automated Approval
  + ACTD2: Manual Review (if automated approval fails)
* **Card Issuance (ACTE)**
  + ACTE1: Card Printing & Personalization
  + ACTE2: Dispatch/Shipping
  + ACTE3: Send Tracking Info (sub-step of Dispatch)

*(These are just examples; in a real system, you might have many more or fewer sub-activities.)*

**2. Sample Event Log (For One Application)**

Below is a hypothetical sequence of **12 events** for **CaseID = “APP001”**. We show how each event references its **ParentActivityID** and **ParentEventID**, demonstrating the hierarchical and triggering relationships. We also include a **Grouping** field to illustrate bundling of similar steps.

| **EventID** | **CaseID** | **ActivityID** | **ActivityName** | **ParentActivityID** | **ParentEventID** | **Timestamp** | **EventType** | **Grouping** | **OtherDetails** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EVT0001 | APP001 | ACTA | Application Submission | NULL | NULL | 2025-01-20 08:00:00 | start | Root-Level | Channel="Online" |
| EVT0002 | APP001 | ACTA1 | Gather Customer Details | ACTA | EVT0001 | 2025-01-20 08:02:00 | start | Data Collection | N/A |
| EVT0003 | APP001 | ACTA1 | Gather Customer Details | ACTA | EVT0002 | 2025-01-20 08:05:00 | complete | Data Collection | CustomerName="John Doe" |
| EVT0004 | APP001 | ACTA2 | Collect Supporting Documents | ACTA | EVT0001 | 2025-01-20 08:06:00 | start | Data Collection | Documents="ID Proof, Utility Bill" |
| EVT0005 | APP001 | ACTA2 | Collect Supporting Documents | ACTA | EVT0004 | 2025-01-20 08:10:00 | complete | Data Collection | Verification="All Good" |
| EVT0006 | APP001 | ACTB | Initial Check | NULL | NULL | 2025-01-20 08:12:00 | start | Root-Level | AssignedTo="SystemChecker" |
| EVT0007 | APP001 | ACTB1 | Preliminary Credit Score Check | ACTB | EVT0006 | 2025-01-20 08:13:00 | complete | Processing Steps | FICO=720 |
| EVT0008 | APP001 | ACTB2 | Validate Application | ACTB | EVT0006 | 2025-01-20 08:14:00 | complete | Processing Steps | Status="Valid" |
| EVT0009 | APP001 | ACTB2N | Notification Email | ACTB2 | EVT0008 | 2025-01-20 08:15:00 | complete | Notification Steps | EmailSentTo="[john.doe@example.com](mailto:john.doe@example.com)" |
| EVT0010 | APP001 | ACTC | Risk Assessment | NULL | NULL | 2025-01-20 08:20:00 | start | Root-Level | AutoCheck="Passed" |
| EVT0011 | APP001 | ACTC1 | Fraud Check | ACTC | EVT0010 | 2025-01-20 08:22:00 | complete | Risk Steps | FraudAlert="None" |
| EVT0012 | APP001 | ACTD | Approval Decision | NULL | NULL | 2025-01-20 08:25:00 | complete | Root-Level | Decision="Approved" |

*(In a real scenario, there might be events for ACTD1 or ACTD2, and also for ACTE if the application is fully approved and the card is issued. But this table gives a clear, concise illustration.)*

**2.1. Observations & Structure**

1. **Root-Level Activities**:
   * ACTA, ACTB, ACTC, ACTD all have ParentActivityID = NULL.
   * They each **start** or **complete** at different times.
   * ParentEventID is also NULL if they are initiated independently or triggered by an external system.
2. **Sub-Activities**:
   * ACTA1, ACTA2 belong to ACTA. So ParentActivityID = ACTA.
   * They also reference the **specific event** (EVT0001) that created them in their ParentEventID or the subsequent step that triggered their completion.
   * ACTB1, ACTB2, ACTB2N belong to ACTB.
     + For ACTB1, ParentActivityID = ACTB and ParentEventID = EVT0006.
     + Notice that ACTB2N is a sub-sub-activity under ACTB2 (i.e., parent is ACTB2) to handle the email notification.
3. **Timestamp**:
   * Each event has a **clear time** so the order of events is unambiguous.
4. **Grouping**:
   * “Root-Level,” “Data Collection,” “Processing Steps,” “Notification Steps,” “Risk Steps.”
   * Stakeholders can filter out (for example) “Notification Steps” if those are too detailed.
5. **EventType**:
   * “start” or “complete” in this example, though you can have others (“update,” “cancel,” etc.).
6. **OtherDetails**:
   * Illustrative metadata: who handled the step, the result of a check, etc.
   * This field is **flexible** and can store additional info like user IDs, device info, or reason codes for rejections.

**3. Possible Scenarios / Views**

**3.1. High-Level Business View**

* **Hide** all sub-activities except root-level (ACTA, ACTB, ACTC, ACTD, ACTE).
* The resulting map or timeline might look like:
  + **Application Submission** -> **Initial Check** -> **Risk Assessment** -> **Approval Decision** -> **Card Issuance**

*(In the example data, we only made it to ACTD, but typically you’d see ACTE if the approval goes through.)*

**3.2. Operational View (Partial Drill-Down)**

* **Show** “Data Collection” and “Processing Steps” but **hide** “Notification Steps” or “Risk Steps.”
* You might see:
  + **Application Submission**
    - Gather Customer Details (A1)
    - Collect Supporting Documents (A2)
  + **Initial Check**
    - Preliminary Credit Score Check (B1)
    - Validate Application (B2)
  + **Risk Assessment** (still collapsed if you want)
  + **Approval Decision**

**3.3. Technical/Full Detail View**

* **Show all** groupings, including sub-sub-activities like the “Notification Email” (B2N).
* Useful for diagnosing issues or auditing micro-steps.

**4. Why This Matters**

1. **Clear Parent-Child Structure**:
   * ParentActivityID ensures sub-activities are correctly grouped.
   * ParentEventID ensures we know which **specific event** triggered the next step, vital in parallel scenarios.
2. **Flexible Grouping**:
   * Allows managers to see a simpler, high-level flow.
   * Allows analysts/technical users to expand complex branching or repeated sub-steps.
3. **Consistent Data**:
   * Having these fields stored reliably (with no missing references) means the process mining tool can **accurately reconstruct** the process flow.
4. **Scalability**:
   * Even if there are 500+ events in a single application (due to system logs, notifications, etc.), **grouping** and hierarchical structure prevent visual clutter.

**5. Summary & Recommendations**

1. **Adopt the Log Schema**:
   * **EventID**, **CaseID**, **ActivityID**, **ActivityName**, **ParentActivityID**, **ParentEventID**, **Timestamp**, **EventType**, **Grouping**, **OtherDetails**.
2. **Define Activity Hierarchies**:
   * Clearly list out root-level activities (ACTA, ACTB, ACTC, ACTD, ACTE) and their possible sub-activities (A1, A2, etc.).
3. **Use ParentEventID**:
   * Ensure each sub-event records which **event** caused it. This is crucial for process paths that fork or repeat.
4. **Maintain Grouping**:
   * Label each event with a grouping category that allows easy filtering/collapsing (e.g., “Root-Level,” “Notification Steps,” etc.).
5. **Stay Consistent**:
   * Use a reference table or dictionary to keep track of all valid ActivityIDs, their ParentActivityID, and their default grouping.

By following this approach, your credit card application process (or any other process) will be **well-documented**, **hierarchically organized**, and **ready for robust process mining** and reporting. Stakeholders at every level—from executives wanting a bird’s-eye view to technical teams requiring detailed traces—will benefit from this structured event log design.