Of course. It's great that you have access to real logs—that's the best way to understand how a system truly behaves. Based on the MySQL output you've provided, here are 15 detailed test cases covering **CREATE**, **UPDATE**, **DELETE**, and **data validation** scenarios.

I've used actual ticket\_ids, conversation\_ids, and timestamps from your logs to make these test cases as realistic as possible for your training.

### Test Cases Based on Database Logs

| Test Case ID | Operation | Ticket ID | Test Case Title | Description | Test Steps | Expected Result |
| --- | --- | --- | --- | --- | --- | --- |
| **TC\_LOG\_01** | **READ** | 50 | **Verify data integrity for a ticket with multiple updates** | Validate that ticket 50, which was updated after creation, has correct timestamps and related conversation and attachment data. | 1. SELECT \* FROM ticket\_details WHERE ticket\_id = 50;<br>2. SELECT \* FROM conversation\_details WHERE ticket\_id = 50;<br>3. SELECT \* FROM attachment\_redact\_info WHERE ticket\_id = 50; | 1. ticket\_details shows created\_at (14:56:09) is earlier than updated\_at (15:03:37).<br>2. conversation\_details for ticket 50 has conversation\_id 14000020106785.<br>3. attachment\_redact\_info shows two entries: one for TICKET\_UPDATE and one for CONVERSATION\_CREATE, both with state = 'DOCUMENT\_CLEANUP\_DONE'. |
| **TC\_LOG\_02** | **READ** | 70 | **Verify ticket without attachment** | Check that ticket 70, marked with is\_attachment\_present = 0, has no corresponding entries in attachment-related tables. | 1. SELECT is\_attachment\_present FROM ticket\_details WHERE ticket\_id = 70;<br>2. SELECT COUNT(\*) FROM attachment\_redact\_info WHERE ticket\_id = 70;<br>3. SELECT COUNT(\*) FROM redaction\_request\_log WHERE ticket\_id = 70; | 1. The query on ticket\_details returns is\_attachment\_present = 0.<br>2. Both COUNT(\*) queries return 0, confirming no attachment or redaction logs exist for this ticket. |
| **TC\_LOG\_03** | **READ** | 100 | **Verify ticket with invalid source data** | Investigate ticket 100, which has source = 100. This appears to be an invalid or placeholder value. | 1. SELECT \* FROM ticket\_details WHERE ticket\_id = 100;<br>2. Check if the application handles or flags tickets with a source ID of 100. | The database correctly stores source = 100. The application's UI should either display this value as 'Unknown' or raise a data integrity alert. The system should not crash. |
| **TC\_LOG\_04** | **CREATE** | 745 | **Create a new ticket and verify log trail** | Simulate the creation of a new ticket (ticket\_id = 745) and ensure it follows the same pattern as ticket\_id = 744. | 1. INSERT a record into ticket\_details for ticket\_id = 745.<br>2. INSERT a related record into conversation\_details.<br>3. INSERT a record into attachment\_redact\_info with state = 'DOCUMENT\_CLEANUP\_PENDING'.<br>4. INSERT a record into redaction\_request\_log with response\_code = 200. | Each table should contain a new, correctly linked entry for ticket\_id = 745. All timestamps should be recent. The redaction\_request\_log should contain a valid JSON payload. |
| **TC\_LOG\_05** | **UPDATE** | 90 | **Verify optimistic locking with version increment** | Ticket 90 has version = 1, indicating it has been updated. This test validates the versioning mechanism. | 1. Note the current updated\_at (16:23:59) and version (1) for ticket\_id = 90.<br>2. UPDATE ticket\_details SET priority = 3 WHERE ticket\_id = 90; | The UPDATE is successful. The version column increments to 2, and the updated\_at timestamp is updated to the current time. |
| **TC\_LOG\_06** | **UPDATE** | 650 | **Verify redaction state transition** | Simulate the redaction process completing for ticket 650's attachment, which is currently in a DONE state. | 1. First, UPDATE attachment\_redact\_info SET state = 'DOCUMENT\_CLEANUP\_PENDING' WHERE ticket\_id = 650;<br>2. Then, UPDATE attachment\_redact\_info SET state = 'REDACTION\_SUCCESSFUL' WHERE ticket\_id = 650; | The state column in attachment\_redact\_info correctly transitions from PENDING to SUCCESSFUL. The updated\_at timestamp should change with each update. |
| **TC\_LOG\_07** | **UPDATE** | 650 | **Mark a conversation as private** | Test updating an existing conversation to be private, simulating a user changing its visibility. The conversation for ticket 650 is already private. We will toggle it. | 1. Note that for conversation\_id = 14000020125000, is\_private is 1.<br>2. UPDATE conversation\_details SET is\_private = 0 WHERE conversation\_id = 14000020125000; | The is\_private flag for the conversation is successfully updated to 0. The updated\_at timestamp for that row is now the current time. |
| **TC\_LOG\_08** | **DELETE** | 135 | **Simulate soft-deleting a ticket** | Test the soft-delete functionality by updating the is\_deleted flag for a recent ticket. | 1. SELECT is\_deleted FROM ticket\_details WHERE ticket\_id = 135; (Should be 0).<br>2. UPDATE ticket\_details SET is\_deleted = 1 WHERE ticket\_id = 135; | The is\_deleted column for ticket 135 is now 1. The row remains in the table. The related conversation (14000020121230) is **not** deleted from its table (cascading deletes are not typical for soft deletes). |
| **TC\_LOG\_09** | **DELETE** | 130 | **Simulate soft-deleting a conversation** | Test deleting a single conversation from a ticket without deleting the ticket itself. | 1. SELECT is\_deleted FROM conversation\_details WHERE conversation\_id = 14000020120251; (Should be 0).<br>2. UPDATE conversation\_details SET is\_deleted = 1 WHERE conversation\_id = 14000020120251; | The is\_deleted column for conversation 14000020120251 is now 1. The parent ticket (ticket\_id = 130) remains unaffected. |
| **TC\_LOG\_10** | **FAILURE** | 49 | **Analyze a failed attachment redaction** | attachment\_redact\_info for ticket\_id = 49 shows a DB\_INSERTION\_ERROR and a 403 Forbidden message. This test case documents this failure. | 1. SELECT \* FROM attachment\_redact\_info WHERE ticket\_id = 49;<br>2. SELECT \* FROM redaction\_request\_log WHERE ticket\_id = 49; | The attachment\_redact\_info shows state = 'PENDING', response\_code = 'DB\_INSERTION\_ERROR', a high retry\_attempt\_count (637), and a message indicating a 403 Forbidden error on the URL. This confirms the system logs failures correctly after multiple retries. |
| **TC\_LOG\_11** | **FAILURE** | 25 | **Investigate data type mismatch** | Ticket 25 has status = 'open' (a string) while all others use integer codes. This is a data integrity issue. | 1. SELECT \* FROM ticket\_details WHERE ticket\_id = 25;<br>2. Attempt to UPDATE ticket\_details SET status = 'pending' WHERE ticket\_id = 25; | The database stores the string 'open'. An update with another string might succeed, but this indicates a lack of data type enforcement on the status column, which could cause application errors. The expected behavior should be a database-level rejection of non-integer types for this column. |
| **TC\_LOG\_12** | **FAILURE** | 50 | **Investigate NULL foreign key** | Conversation for ticket 50 has conversation\_source = NULL. This is another data integrity issue. | 1. SELECT \* FROM conversation\_details WHERE ticket\_id = 50;<br>2. Attempt to INSERT a new conversation for ticket 50 with a NULL source. | The database allows NULL for conversation\_source, which may be unintentional. The system should enforce a NOT NULL constraint on this column if a source is always required for a conversation. |
| **TC\_LOG\_13** | **CREATE** | 800 | **Create ticket with attachment\_on = 'CONVERSATION'** | All logs show attachment\_on = 'TICKET'. Test the system's ability to handle attachments linked directly to a conversation. | 1. Create a full ticket flow for ticket\_id = 800.<br>2. In the attachment\_redact\_info table, set attachment\_on = 'CONVERSATION'. | The data is inserted successfully. The system logic should correctly process the attachment, understanding its context is a conversation reply rather than the initial ticket body. |
| **TC\_LOG\_14** | **UPDATE** | 123 | **Update a closed ticket** | Ticket 123 has status = 4, which likely means 'Closed' or 'Resolved'. Test if the system allows updates to a closed ticket. | 1. Note the current status = 4 and version = 2 for ticket\_id = 123.<br>2. UPDATE ticket\_details SET priority = 1 WHERE ticket\_id = 123; | The UPDATE should succeed, incrementing the version to 3 and updating the timestamp. This validates that tickets can be modified even after being closed. |
| **TC\_LOG\_15** | **READ** | 131 | **Verify data relationships with multiple conversations** | Ticket 131 has multiple conversations (14000020120250, 14000020121103, etc.). Verify that a JOIN query correctly retrieves all related data. | 1. Execute: SELECT t.ticket\_id, c.conversation\_id, a.raw\_attachment\_name FROM ticket\_details t JOIN conversation\_details c ON t.ticket\_id = c.ticket\_id LEFT JOIN attachment\_redact\_info a ON c.conversation\_id = a.conversation\_id WHERE t.ticket\_id = 131; | The query should return all conversations linked to ticket 131 and correctly show which conversation has an associated attachment and which does not. |