Database Table Analysis Report

archive_translations, sample, State

1. Overview

This report provides a procedure-grounded, evidence-based analysis and recommendations for three schema objects used in localization and geographic filtering: archive_translations, sample, and State. Each table section summarizes the schema, indexes, and foreign-key status, followed by observations and recommendations. A stored procedures section explains how the procedures operate on these tables (or related logic), followed by joint findings, tests, summary of findings, and conclusion, mirroring the format used in the previous Sprint 4 report.

The goal for Sprint 5 is to understand how translation archives are stored and accessed, how geographic filters are implemented, and how the small sample table fits into the overall system. The focus is on structural correctness, discoverability of translations, normalization of reference data, and readiness for scaling.

2. archive_translations

2.1 Table Structure

Column	Data type	Notes
id	INT	Primary key, auto increment, unique row id.
channel_id	INT	Channel identifier from source system.
mp4_file_name	VARCHAR(100)	Full file path or file name for the clip.
original_id	INT	Links to original content row in an upstream table.
timecode	VARCHAR(32)	Time range text, for example start and end time.
translated_text	TEXT	Translated caption or transcript.

language	VARCHAR(10)	Language code, default value 'es'.
created_at	TIMESTAMP	Row creation time, default CURRENT_TIMESTAMP.

2.2 Indexes

Index name	Type	Notes
PRIMARY	PRIMARY	Primary key index currently only on HostName.

2.3 Foreign Keys

- None. channel_id and original_id store integer values without enforced links.

2.4 Observations

- Design supports an archive of translated segments for media files.
- channel_id and original_id behave like foreign key columns, but no constraints guard data quality.
- mp4_file_name keeps full paths, so long directory prefixes repeat for many rows.
- timecode stays in free text, so time based filters require string handling.
- Only id has an index, so filters on mp4_file_name, channel_id, original_id, or language scan the table.
- language holds a default value but no whitelist of allowed codes.
- No column tracks last update or user who changed a row.

2.5 Recommendations

- Add composite indexes that match common query patterns, for example:(mp4_file_name, language), (channel_id, original_id, language).
- Introduce foreign keys once upstream tables are stable, for example: channel_id to a Channels table; original_id to an STT or Clips table
- Move file paths into a central media table, and keep only a media id in archive_translations.

- Replace timecode text with numeric start_time and end_time columns, keep a formatted field only for display.
- Add updated_at and updated_by columns so you can audit edits.
- If one translation per original segment and language is a rule, add a unique index on (original_id, language, timecode or start_time, end_time).

3. sample

3.1 Table Structure

Column	Data type	Notes
email	VARCHAR(255)	Primary key, email address.
resolution	VARCHAR(255)	Free text resolution or preference value.

3.2 Indexes

Index name	Type	Notes
PRIMARY	PRIMARY	Single column index on email.

3.3 Foreign Keys

- None. email has no link to a central users table.

3.4 Observations

- This table holds one row per email address.
- No timestamps show when a preference started or changed.
- resolution accepts any string, which leads to multiple spellings such as 720p and 1280x720.
- No stored procedures reference this table, so usage stays manual or test focused.

3.5 Recommendations

- If you keep this table for production, rename to a clearer name such as EmailResolutionPreference.

- Add created_at and updated_at for traceability.
- Define a small lookup table for allowed resolution values and store a key instead of free text.
- Consider a Users table with a numeric primary key, then store user_id here instead of email.
- If usage stays experimental, move ‘sample’ to a separate schema or prefix with tmp_ in order to avoid confusion.

4. State

4.1 Table Structure

Column	Data type	Notes
State	CHAR(2) (PK)	Two character state or province code, primary key.

4.2 Indexes

Index name	Type	Notes
PRIMARY	PRIMARY	Primary key on State.
State_UNIQUE	UNIQUE	Duplicate unique index on State.

4.3 Foreign Keys

- None. No tables reference this list of codes.

4.4 Observations

- This Table behaves as a reference list of state or province codes.
- State_UNIQUE repeats the primary key constraint and adds overhead.
- Column name equals table name, which reduces clarity in queries.

- No full state names, country links, or region metadata exist for user interfaces.
- Procedures that serve state lists read from Locations.StateProvince, not from this table.

4.5 Recommendations

- Drop State_UNIQUE and keep the primary key on State.
- Add foreign keys from Locations.StateProvince to State.State once data is clean.
- Rename the column to state_code for clearer SQL.
- Add state_name, country_code, and region columns so user interfaces and reports use friendly labels.
- Over time, adjust lookup procedures so they draw allowed state codes from this table.

5. Stored Procedures Analysis

5.1 Overview

The Stored procedures of interest fall into two groups.
Group one uses archive_translations for translation work.
Group two reads Locations for geography filters related to states, cities, and countries.

Group one, translation archive:

- InsertTranslation
- GetTranslations
- DeleteTranslation

Group two, geography lookup based on Locations:

- Get_States
- Get_Cities
- Get_Countries
- Update_Cities_With_State
- Update_Countries_With_State
- Update_States_With_City
- Update_States_With_Country
- Update_Cities_With_Country
- Update_Countries_With_City

The sample table has no stored procedures.

5.2 Key Findings

1. InsertTranslation
Logic summary

- Accepts channel_id, mp4_file_name, original_id, timecode, translated_text, language.
- Inserts one row into archive_translations with a direct mapping from parameters to columns.

Issues / risks

- No validation for missing translated_text or language.
- No validation for channel_id or original_id against source tables.
- No duplicate check for a given original segment and language.

Actions

- Add basic input checks in code or in the procedure.
- Enforce a uniqueness rule where business needs require a single translation per segment and language.

2. GetTranslations
Logic summary

- Accepts mp4_file_name as input.
- Returns rows from archive_translations where mp4_file_name equals that input.

Issues / risks

- Uses SELECT *, which ties callers to the current column list.
- Filters only on mp4_file_name, so callers cannot narrow on language or channel_id.
- No secondary index supports mp4_file_name filters, so queries slow down as the archive grows.

Actions

- Replace SELECT * with an explicit list of columns your front end needs.
- Add optional parameters for language and channel_id.
- Add an index on mp4_file_name or on (mp4_file_name, language).

3. DeleteTranslation
Logic summary

- Accepts p_translation_id.
- • Runs DELETE on archive_translations where id equals p_translation_id.

Issues / risks

- No audit trail for removed translations.
- Caller receives no clear confirmation when no row matches.

Actions

- Return affected row count or a custom status so you see if a delete succeeded.
- For higher safety, add a soft delete flag and keep rows for history.

4. Get_States, Get_Cities, Get_Countries

Logic summary

- Each procedure selects distinct values from Locations.
- Get_States returns StateProvince sorted alphabetically.
- Get_Cities returns City.
- Get_Countries returns Country.

Issues / risks

- All procedures read raw text values from Locations, not from the State table.
- Any spelling or spacing issue in Locations flows into dropdowns and filters.
- DISTINCT queries over large sets cost CPU time without strong indexes.

Actions

- Ensure indexes exist on Locations.StateProvince, City, and Country.
- Plan a shift where the State table holds the canonical list, and Locations references those codes.

5. Update_Cities_With_State and Update_Countries_With_StateLogic summary

- Accept State_Selection as input.
- Return distinct City or Country values from Locations with matching StateProvince.

Issues / risks

- Parameter type TEXT does not align with StateProvince type.
- No validation step that checks State_Selection against the State table.

Actions

- Change parameter type to match StateProvince type, likely CHAR(2) or VARCHAR.
- Add reference checks so only valid state codes feed these procedures.

6. Update_States_With_City and Update_States_With_Country
Logic summary

- Accept City_Selection or Country_Selection.
- Return distinct StateProvince values filtered by those fields in Locations.

Issues / risks

- Same parameter typing concerns as above.
- Ambiguous city names can return multiple state rows, which front ends must handle.

Actions

- Align parameter types with Locations columns.
- Document behavior for city names present in multiple regions, so front ends handle lists correctly.

7. Update_Cities_With_Country and Update_Countries_With_City
Logic summary

- Provide cross filters between city and country.
- Given a country, return related cities.
- Given a city, return related countries.

Issues / risks

- Procedures still rely on free text in Locations.
- No link to a normalized country table.

Actions

- Add indexes on relevant columns in Locations.
- Move toward country and city reference tables as data grows.

5.3 Recommendations

1. **Parameter validation and normalization**
    - Add basic checks in InsertTranslation so required inputs are not empty.
    - Example, require non empty translated_text, language, mp4_file_name, and timecode.
    - Add checks in geo lookup procedures for valid state, city, and country values.
    - Normalize station, country, and city names or move to numeric IDs from reference tables.
2. **Stabilize timestamp and key handling**
    - Move time based logic in archive_translations to numeric start_time and end_time columns instead of free text timecode.

    - Use consistent types for geography parameters.
      State_Selection should match State.State, for example CHAR(2).
      City_Selection and Country_Selection should match related columns in Locations.
3. **Reduce SELECT * usage**
    - Update GetTranslations to return a fixed column list that front end code needs.
    - Update geography procedures to return named columns only, not entire rows with unused data.
4. **Strengthen indexing and integrity**
    - Add composite indexes that match real queries.
      For example, an index on (mp4_file_name, language) in archive_translations.
      Add indexes on Locations.StateProvince, City, and Country to support DISTINCT filters.
    - Add foreign keys where upstream tables are stable.
      Link archive_translations.channel_id to the channel table.
      Link archive_translations.original_id to the STT or clip table.
      Link Locations.StateProvince to State.State.
5. **Definer and permissions cleanup**
    - Review 'definers' for translation and geo procedures.
      Use a shared application user or DEFINER that exists in each environment.
    - Confirm that users who call these procedures have EXECUTE privilege only, not direct table write access when not needed.

6. Joint Analysis and Recommendations

6.1 Joint Findings

- All three tables use simple schemas with one primary key each.
- Relationships stay implicit. archive_translations holds channel_id and original_id with no foreign keys. State holds codes with no links from Locations. sample stands alone.
- Procedures focus on basic CRUD and lookup logic and leave validation to the caller.
- Geography logic reads from Locations only and ignores the State table.
- Several procedures use SELECT * and TEXT parameters, which weakens stability and performance.
- Index coverage does not match typical filters such as mp4_file_name or StateProvince.

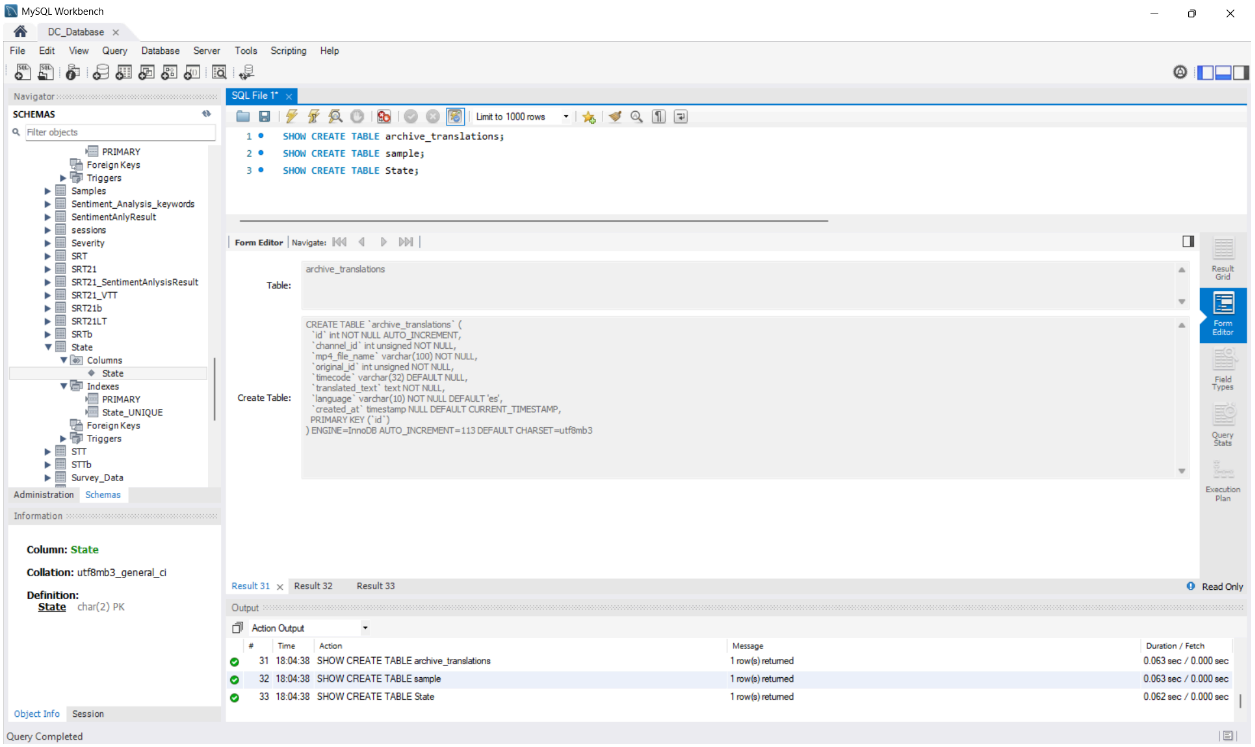
6.2 Recommendations

- **Move toward reference driven design.**
  Use State as the master list of state codes and link Locations.StateProvince to those codes. Plan similar reference tables for country and city when data size grows.

- **Improve integrity for translation data.**
  Add foreign keys from archive_translations to channel and source STT or clip tables.
  Add unique constraints if one translation per segment and language is required.
- **Standardize free text fields.**
  Normalize state codes, country names, and city names.
  Clean up mp4_file_name values or move paths into a separate media table.
- **Refresh stored procedures.**
  Replace SELECT * with explicit column lists.
  Align procedure parameter types with table column types.
  Add minimal logging and checks for delete operations such as DeleteTranslation.
- **Plan for growth.**
  Review query patterns from logs.
  Adjust indexes on archive_translations and Locations to match those patterns.

7. Tests

1. Schema Verification



```
SHOW CREATE TABLE archive_translations;
SHOW CREATE TABLE sample;
SHOW CREATE TABLE State;
```

Table: archive_translations

Create Table:

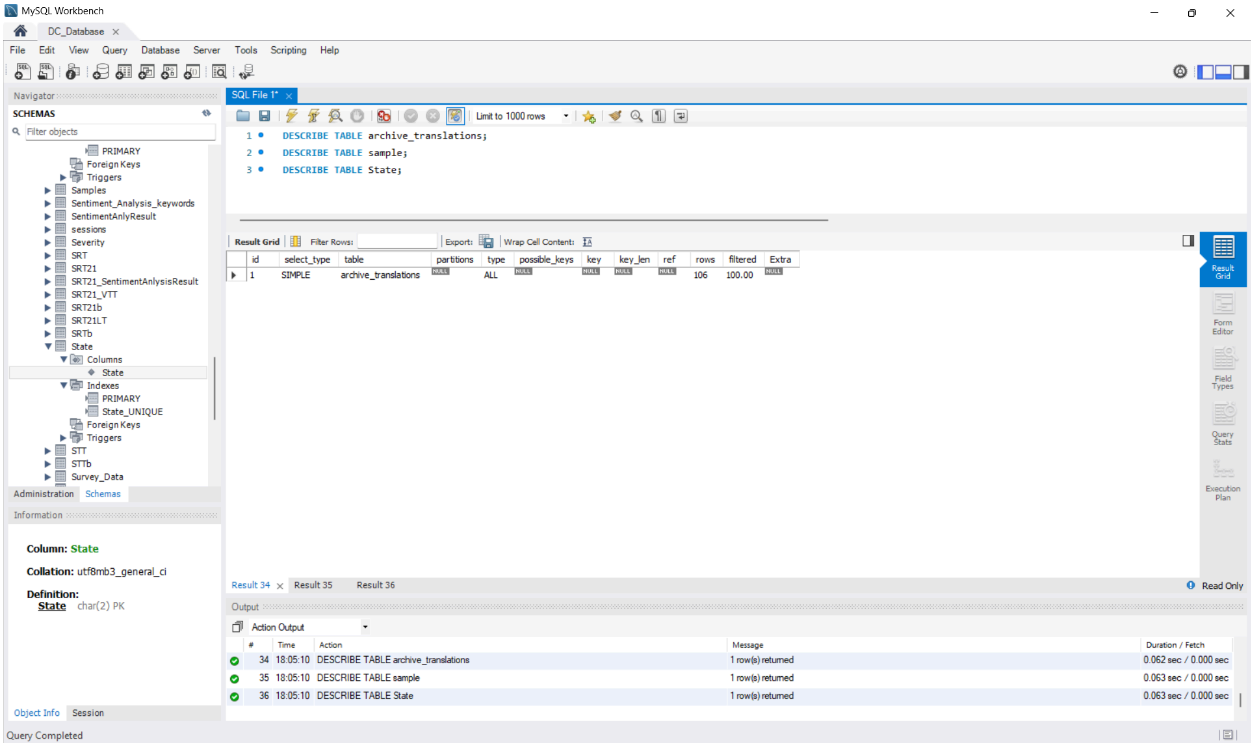
```
CREATE TABLE `archive_translations` (
  `id` int NOT NULL AUTO_INCREMENT,
  `channel_id` int unsigned NOT NULL,
  `mp4_file_name` varchar(100) NOT NULL,
  `original_id` int unsigned NOT NULL,
  `timecode` varchar(32) DEFAULT NULL,
  `translated_text` text NOT NULL,
  `language` varchar(10) NOT NULL DEFAULT 'es',
  `created_at` timestamp NULL DEFAULT CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=113 DEFAULT CHARSET=utf8mb3
```

Column: State

Collation: utf8mb3_general_ci

Definition:
State char(2) PK

```
 #   Time      Action                                  Message             Duration / Fetch
 31  18:04:38  SHOW CREATE TABLE archive_translations  1 row(s) returned   0.063 sec / 0.000 sec
 32  18:04:38  SHOW CREATE TABLE sample                1 row(s) returned   0.063 sec / 0.000 sec
 33  18:04:38  SHOW CREATE TABLE State                 1 row(s) returned   0.062 sec / 0.000 sec
```



```
DESCRIBE TABLE archive_translations;
DESCRIBE TABLE sample;
DESCRIBE TABLE State;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	archive_translations	NULL	ALL	NULL	NULL	NULL	NULL	106	100.00	NULL

Column: State

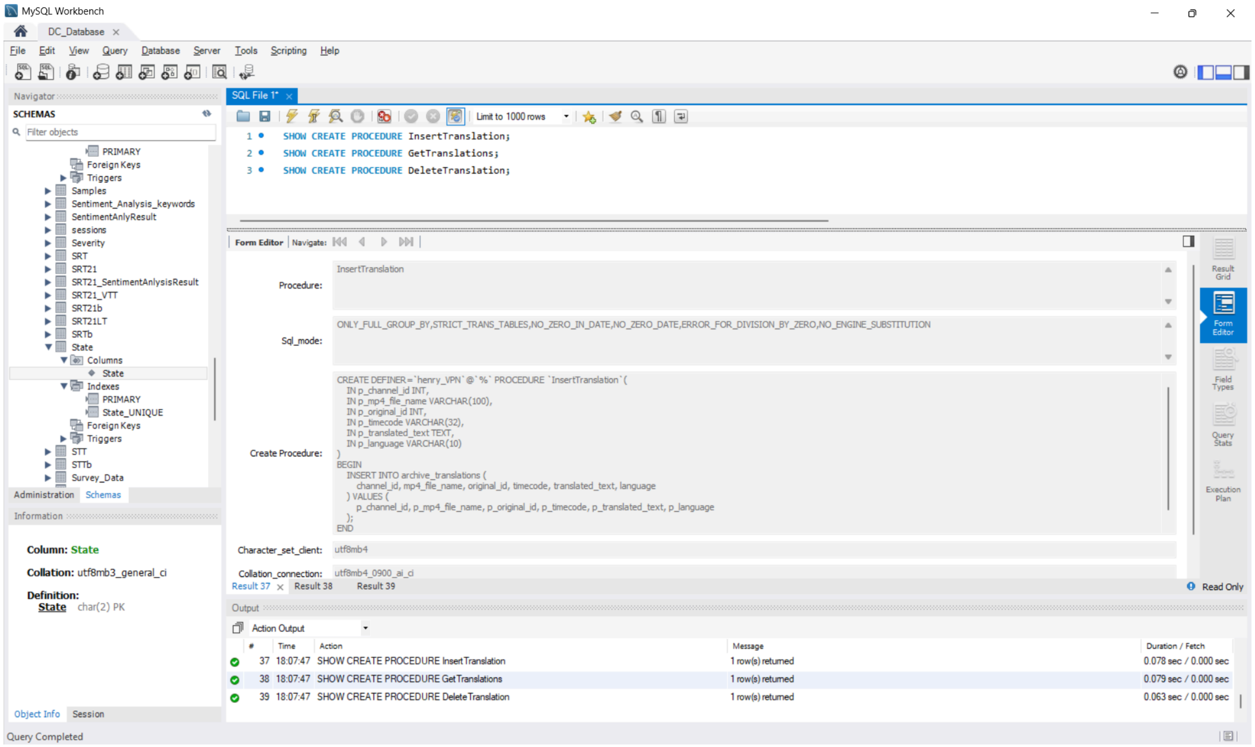
Collation: utf8mb3_general_ci

Definition:
State char(2) PK

	#	Time	Action	Message	Duration / Fetch
✓	34	18:05:10	DESCRIBE TABLE archive_translations	1 row(s) returned	0.062 sec / 0.000 sec
✓	35	18:05:10	DESCRIBE TABLE sample	1 row(s) returned	0.063 sec / 0.000 sec
✓	36	18:05:10	DESCRIBE TABLE State	1 row(s) returned	0.063 sec / 0.000 sec

- **Purpose:**
  Confirm that archive_translations, sample, State, and key related tables such as Locations match the expected schema before deeper analysis.
- **Findings:**
  Structure matches documented design.
- **Outcome:**
  Primary keys and existing indexes appear as described.
  No unexpected columns or types appear.

2. Stored Procedure Extraction Validation



Column: State

Collation: utf8mb3_general_ci

Definition:
State char(2) PK

```
SHOW CREATE PROCEDURE InsertTranslation;
SHOW CREATE PROCEDURE GetTranslations;
SHOW CREATE PROCEDURE DeleteTranslation;
```

Procedure: InsertTranslation

Sql_mode: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION

Create Procedure:

```
CREATE DEFINER=`henry_VPN`@`%` PROCEDURE `InsertTranslation`(
  IN p_channel_id INT,
  IN p_mp4_file_name VARCHAR(100),
  IN p_original_id INT,
  IN p_timecode VARCHAR(32),
  IN p_translated_text TEXT,
  IN p_language VARCHAR(10)
)
BEGIN
  INSERT INTO archive_translations (
    channel_id, mp4_file_name, original_id, timecode, translated_text, language
  ) VALUES (
    p_channel_id, p_mp4_file_name, p_original_id, p_timecode, p_translated_text, p_language
  );
END
```

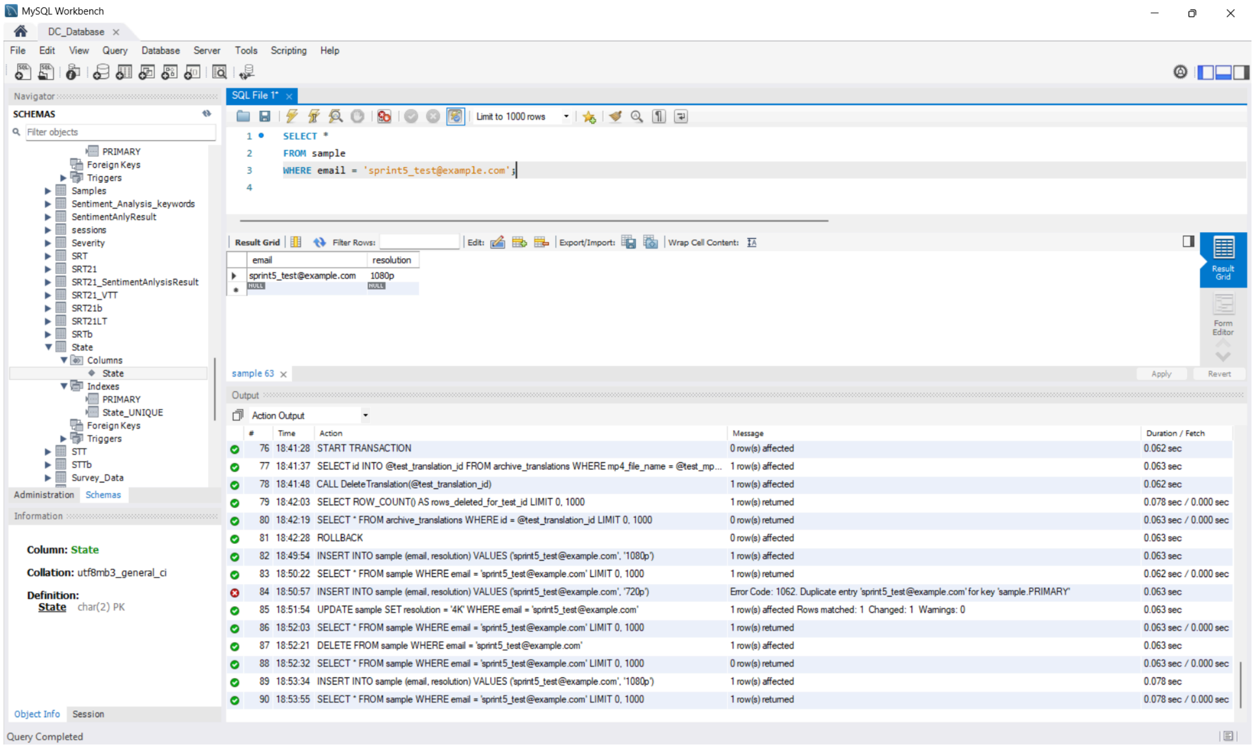
Character_set_client: utf8mb4

Collation_connection: utf8mb4_0900_ai_ci

```
   #   Time      Action                                  Message             Duration / Fetch
   37  18:07:47  SHOW CREATE PROCEDURE InsertTranslation  1 row(s) returned   0.078 sec / 0.000 sec
   38  18:07:47  SHOW CREATE PROCEDURE GetTranslations    1 row(s) returned   0.079 sec / 0.000 sec
   39  18:07:47  SHOW CREATE PROCEDURE DeleteTranslation  1 row(s) returned   0.063 sec / 0.000 sec
```

- **Purpose:**
  Confirm that all the procedures for this sprint compile and return definitions without syntax or definer errors.
- **Findings:**
  - Procedures compile and generate output.
  - Any definer issues are documented so the DBA adjusts users or definers before production use.
- **Outcome:**
  Test Passed — All procedures returned definitions successfully.

3. Sample Table CRUD Validation



```
SELECT *
FROM sample
WHERE email = 'sprint5_test@example.com';
```

email	resolution
sprint5_test@example.com	1080p
NULL	NULL

Column: State

Collation: utf8mb3_general_ci

Definition:
State char(2) PK

```
#   Time      Action                                                                                   Message                                                                          Duration / Fetch
76  18:41:28  START TRANSACTION                                                                        0 row(s) affected                                                                0.062 sec
77  18:41:37  SELECT id INTO @test_translation_id FROM archive_translations WHERE mp4_file_name = @test_mp...  1 row(s) affected                                                       0.063 sec
78  18:41:48  CALL DeleteTranslation(@test_translation_id)                                             1 row(s) affected                                                                0.062 sec
79  18:42:03  SELECT ROW_COUNT() AS rows_deleted_for_test_id LIMIT 0, 1000                             1 row(s) returned                                                                0.078 sec / 0.000 sec
80  18:42:19  SELECT * FROM archive_translations WHERE id = @test_translation_id LIMIT 0, 1000          0 row(s) returned                                                                0.063 sec / 0.000 sec
81  18:42:28  ROLLBACK                                                                                 0 row(s) affected                                                                0.063 sec
82  18:49:54  INSERT INTO sample (email, resolution) VALUES ('sprint5_test@example.com', '1080p')       1 row(s) affected                                                                0.063 sec
83  18:50:22  SELECT * FROM sample WHERE email = 'sprint5_test@example.com' LIMIT 0, 1000             1 row(s) returned                                                                0.062 sec / 0.000 sec
84  18:50:57  INSERT INTO sample (email, resolution) VALUES ('sprint5_test@example.com', '720p')        Error Code: 1062. Duplicate entry 'sprint5_test@example.com' for key 'sample.PRIMARY'  0.063 sec
85  18:51:54  UPDATE sample SET resolution = '4K' WHERE email = 'sprint5_test@example.com'             1 row(s) affected Rows matched: 1  Changed: 1  Warnings: 0                        0.063 sec
86  18:52:03  SELECT * FROM sample WHERE email = 'sprint5_test@example.com' LIMIT 0, 1000             1 row(s) returned                                                                0.063 sec / 0.000 sec
87  18:52:21  DELETE FROM sample WHERE email = 'sprint5_test@example.com'                              1 row(s) affected                                                                0.063 sec
88  18:52:32  SELECT * FROM sample WHERE email = 'sprint5_test@example.com' LIMIT 0, 1000             0 row(s) returned                                                                0.063 sec / 0.000 sec
89  18:53:34  INSERT INTO sample (email, resolution) VALUES ('sprint5_test@example.com', '1080p')       1 row(s) affected                                                                0.078 sec
90  18:53:55  SELECT * FROM sample WHERE email = 'sprint5_test@example.com' LIMIT 0, 1000             1 row(s) returned                                                                0.078 sec / 0.000 sec
```

Query Completed

- **Purpose:**
  Confirm that the sample table supports basic insert, read, update, and delete operations and that the primary key on email enforces uniqueness.
- **Findings:**
  - Inserted a temporary test row into the sample table.
  - Queried the row by email and confirmed the stored resolution value.
  - Attempted a second insert with the same email and observed a duplicate key error, which shows the primary key works.
  - Updated the resolution for the test row and verified the change with a SELECT.
  - Deleted the test row and confirmed that no rows with the test email remain.
- **Outcome:**
  Test Passed. The sample table handled basic CRUD operations correctly and enforced primary key uniqueness on email.

8. Summary of Findings

- archive_translations, sample, and State are structurally correct but use minimal indexing and no foreign keys.
- archive_translations keeps channel_id and original_id as plain integers with no enforced links, so referential integrity depends on application logic.
- archive_translations stores timecode as free text and has only a primary key index, which limits efficient search by file, time, or language.
- sample uses email as the primary key and supports simple preference storage, but has no timestamps, no validation of resolution values, and no link to a central users table.
- State holds two character state codes and a redundant unique index on the same column as the primary key, and no other tables reference it for validation.
- Translation procedures (InsertTranslation, GetTranslations, DeleteTranslation) match the archive_translations table and run after definer cleanup, but they use SELECT *, perform no input checks, and do not record delete history.
- Geography procedures read distinct values from Locations for states, cities, and countries and provide cross filtering, but they rely on free text columns, parameter types that do not always match the table, and the State table is not part of this workflow.
- Schema verification and procedure extraction tests confirmed stable definitions, corrected definers, and successful execution for the procedures in scope.
- Sample table CRUD validation showed that insert, read, update, delete, and duplicate key handling work as expected for the primary key on email.

9. Conclusion

The Sprint 5 analysis shows that the translation and geography tables support current use but depend heavily on application code for data quality and performance. archive_translations records translations reliably, yet missing foreign keys, free text time fields, and limited indexing will slow common queries as data grows. sample and State act as simple support tables, but they do not anchor a shared reference model for users, resolutions, or state codes.

Stored procedures for translations and geography run successfully and return correct data for normal inputs, although they use broad SELECT queries and unvalidated parameters. Addressing indexing gaps, enforcing relationships with foreign keys, normalizing reference data, and tightening procedure interfaces will improve reliability, simplify maintenance, and prepare the database layer for future automation and feature work in later sprints.