Sprint 3

Database Table Analysis Report

RadioClips, RadioStation, Radio_Errors

1. Overview

This report provides a procedure-grounded, evidence-based analysis and recommendations for the three radio-related tables. Each table section shows the database schema, indexes, and foreign key status, followed by observations and clear recommendations. A dedicated Stored Procedures Analysis section follows the table sections, then joint analysis and recommendations, tests, a section on stored-procedure to table ratio, summary of findings, and conclusion.

The purpose of this report is to study how DigiClips stored procedures work with the three related database tables, RadioClips, RadioStation, and Radio_Errors. It examines how these procedures move and organize information, how the tables connect to each other, and how these processes affect speed and accuracy. The goal is to provide clear and practical improvements that make the database easier to maintain, faster to use, and more reliable overall.

2. RadioStation

2.1 Table Structure

Column	Data type	Notes
ID	INT (Primary Key, Auto Increment)	A unique number automatically assigned to each radio station.
StationName	VARCHAR(30)	The short name of the station.
Country	TEXT	The country where the station operates.
StateProvince	TEXT	State or province
City	TEXT	City
Link	TEXT	Website or stream link

2.2 Indexes

Index name	Type	Notes
PRIMARY	PRIMARY	The main index automatically created on the ID column. This makes each station's ID unique and easy to find.
StationName_UNIQUE	UNIQUE	Ensures that each station name is unique so the same station cannot be added twice by mistake.

2.3 Foreign Keys

- None (this table is the parent and currently has no enforced references).

2.4 Observations

- Uses TEXT fields for short values which is inefficient for comparisons and indexing.
  - The fields City, StateProvince, and Country only need to hold short words like "Denver" or "United States." TEXT is meant for long paragraphs, so it takes up more space and slows down searches. A smaller type such as VARCHAR(50) is more efficient.

- No auditing fields (created_at, updated_at).
  - The table does not record when each station was added or last updated.

- Canonicalization of StationName is not enforced.
  - This means the database does not automatically standardize how station names are saved. For example, "KOA" and "koa" could be treated as different values. Standardizing names through lowercasing and trimming spaces prevents duplicate entries and makes searches accurate.

- Empty data is allowed (no NOT NULL rules).
  - The table allows blank fields. For example, a station could be added without a name or country. This could cause missing or incomplete records.

2.5 Recommendations

- Change Country, StateProvince, and City to VARCHAR(100) to improve lookup speed.
- Add created_at and updated_at columns to track when records are added or changed.
- Standardize the StationName field by automatically converting all entries to uppercase and trimming extra spaces within the insert and update procedures.
- Add NOT NULL to important columns like StationName and Country to make sure those values are always filled in.

3. RadioClips

3.1 Table Structure

Column	Data type	Notes
ID	INT	Primary key
FName	VARCHAR(100)	Filename of clip
TStamp	VARCHAR(100)	The date and time when the clip was recorded, stored as text.
SName	VARCHAR(100)	The station name for the clip (written as text, not linked to RadioStation).
TEXTS	TEXT	The transcript or summary of what was recorded.
Categories	TEXT	The topics or categories for the clip.
DownloadLink	TEXT	Download or stream link

3.2 Indexes

Index name	Type	Notes
PRIMARY	PRIMARY	The main index on the ID column. This identifies each clip uniquely.

TEXTS	FULLTEXT	Allows word searches inside the TEXTS column (used for text or transcript searching).
TEXTS_2	FULLTEXT	Duplicate, should be removed
TEXTS_3	FULLTEXT	Duplicate, should be removed

3.3 Foreign Keys

- None. SName is free text and not enforced against RadioStation.

3.4 Observations

- TStamp stored as text prevents use of indexes for date sorting or filtering.
    - The TStamp field is a VARCHAR(100), which means it is saved as text instead of a proper date/time format. This makes it hard for the database to sort clips by time or filter by date ranges.

- Duplicate FULLTEXT indexes add overhead and maintenance cost.
    - The three full-text indexes (TEXTS, TEXTS_2, TEXTS_3) serve the same purpose and only one is needed. Having extra indexes adds unnecessary maintenance.

- No foreign key relationship to RadioStation.
    - The SName field only stores the station name as text. There is no official connection to the RadioStation table. This means the database doesn't "know" which station a clip belongs to, and errors can happen if names don't match exactly.

- Large Text Fields may be unnecessary.
    - TEXT fields like Categories and DownloadLink can store long text but take up more space and slow down searches when there are many rows. For these columns specifically you may be able to shorten them to VARCHAR(500).

- No NOT NULL rules.
    - Important fields like FName and TStamp could be left blank, which could cause missing data.

3.5 Recommendations

- Convert TStamp to DATETIME and backfill using the format currently used in procedures so the database can correctly sort and filter by date and time.
- Add StationID foreign key referencing the RadioStation table’s ID.
- Keep one FULLTEXT index on TEXTS but remove duplicates (TEXTS_2, TEXTS_3).
- Combine all the existing procedures that add radio clips into one unified procedure that checks each new clip for missing or incorrect information, makes sure the station name and timestamp are in the same consistent format, and then saves the clip record in a single, reliable way every time.
- Enforce NOT NULL on essential columns (FName, TStamp, StationID).

4. Radio_Errors

4.1 Table Structure

Column	Data type	Notes
Date_Time	DATETIME	Timestamp of the error
Error_Str	VARCHAR(200)	Error message
Host_Name	VARCHAR(30)	Host name (server) where error occurred
LineNum	INT	Log line number
Severity	VARCHAR(10)	Error severity level
Station	VARCHAR(30)	Related radio station

4.2 Indexes

Index name	Type	Notes
PRIMARY	PRIMARY	Uniquely identifies errors
fk_RadioSeverity_idx	INDEX	Supports lookup for severity table

4.3 Foreign Keys

Foreign Key name	Referenced table	Notes
fk_RadioSeverity	Severity	Standardizes severity levels

4.4 Observations

- The Station column stores the station name as plain text (like “KOA”) instead of using a StationID that links to the RadioStation table. Because of this, there’s no guarantee that the name matches an existing record, it can be misspelled or inconsistent, which makes it difficult to trace errors back to a specific station.

- The error log keeps growing indefinitely, and without a process to periodically move or remove older data, it will eventually slow down the database and make maintenance harder.

4.5 Recommendations

- Replace the Station text field with a StationID that links directly to the RadioStation table. This will ensure every error entry is connected to a valid station and eliminate issues caused by misspelled or inconsistent names.

- Create an archive process that moves or deletes old error records on a regular schedule, such as transferring entries older than six months to a separate Radio_Errors_Archive table, to keep the main error log small and efficient.

5. Stored Procedures Analysis

5.1 Overview

The stored procedures that interact with the tables RadioClips, RadioStation, and Radio_Errors are responsible for data ingestion, searching, configuration updates, and error logging. This section examines how these procedures manage data, identifies issues that affect performance or consistency, and provides recommendations to make them work efficiently with the database structure.

5.2 Key Findings

1. Date parsing inside search queries
Procedures such as Radio_Search and Radio_Search_Emailalerts call STR_TO_DATE on the input date strings (Start_DateTime, End_DateTime) and compare the results to the TStamp text column in RadioClips. Because TStamp is stored as text, these comparisons prevent MySQL from using date indexes and slow down searches.

Recommendation: TStamp to a DATETIME column in RadioClips and update all search procedures to accept and use typed date parameters instead of text.

2. Station names stored as text instead of linked through the RadioStation table
The procedures Insert_Radio, Insert_Radio_Manual, and Insert_RadioB record station names as text in the SName column when inserting new clips, and the search procedures Radio_Search and Radio_Search_Emailalerts link clips to stations using text comparison (SName = StationName). This means there is no direct relational link between the two tables. This setup can lead to inconsistencies if a station name is misspelled, formatted differently, or later renamed, and it makes searches and joins slower.

Recommendation: Add a StationID column to the RadioClips table and link it to the RadioStation table through a foreign key. Update all insert and search procedures so they use StationID instead of station name text. This will make the relationship between clips and stations consistent, faster, and easier to maintain.

3. Multiple procedures doing the same job
The procedures Insert_Radio, Insert_Radio_Manual, and Insert_RadioB all insert radio clips into either the RadioClips or RadioClipsB tables using nearly identical logic, differing only in how they handle timestamps or which table they insert into. Likewise, the procedures Radio_Search and Radio_Search_Emailalerts both perform similar search operations on RadioClips, with the only major difference being that Radio_Search_Emailalerts excludes results already found in the radioReported table. Having several procedures that perform almost the same task makes maintenance harder and increases the risk of inconsistencies.

Recommendation: Combine these procedures into one unified insert procedure (for example, Insert_RadioClip) that can handle both manual and automatic insert cases through parameters, and one unified search procedure (for example, Search_RadioClips) that includes an optional filter to exclude previously reported clips. This will reduce duplicate code, simplify updates, and ensure consistent behavior across all operations.

5.3 Recommendations

1. Update procedures to reflect schema improvements
Whenever the structure of a table changes, such as adding new columns or changing a column's data type (for example, turning TStamp into a DATETIME or adding StationID), all procedures that use that table must be updated too. This keeps the procedures in sync with the database, prevents errors when they run, and makes sure data is saved and read correctly.

2. Simplify and consolidate similar procedures
The multiple insert and search procedures that currently perform nearly identical tasks should be merged into a single insert procedure (Insert_RadioClip) and a single search procedure (Search_RadioClips). Unifying these functions will reduce code duplication, ensure consistent data handling, and make future updates easier to manage.

3. Centralize input validation in main procedures
Validation checks should be built into the core insert and update procedures to confirm that critical data, such as station references, timestamps, and required fields, are valid before saving. Centralizing validation prevents bad data from being inserted and ensures that all system components follow the same quality rules.

Summary:

The stored procedures related to RadioClips, RadioStation, and Radio_Errors need to be updated to work efficiently with the improved database structure. Converting text timestamps to DATETIME, linking clips to stations through a numeric key, and merging redundant procedures will improve accuracy, performance, and maintainability. By aligning the procedures with the updated schema, applying consistent validation, and adding indexes where needed, DigiClips will gain a faster, more reliable, and easier-to-manage system for handling radio clip data and searches.

6. Joint Analysis and Recommendations

6.1 Joint Findings

- Inconsistent data typing between tables (VARCHAR vs DATETIME) causes inefficiencies in time-based queries.

- Missing foreign key constraints result in orphaned data and hinder relational integrity.

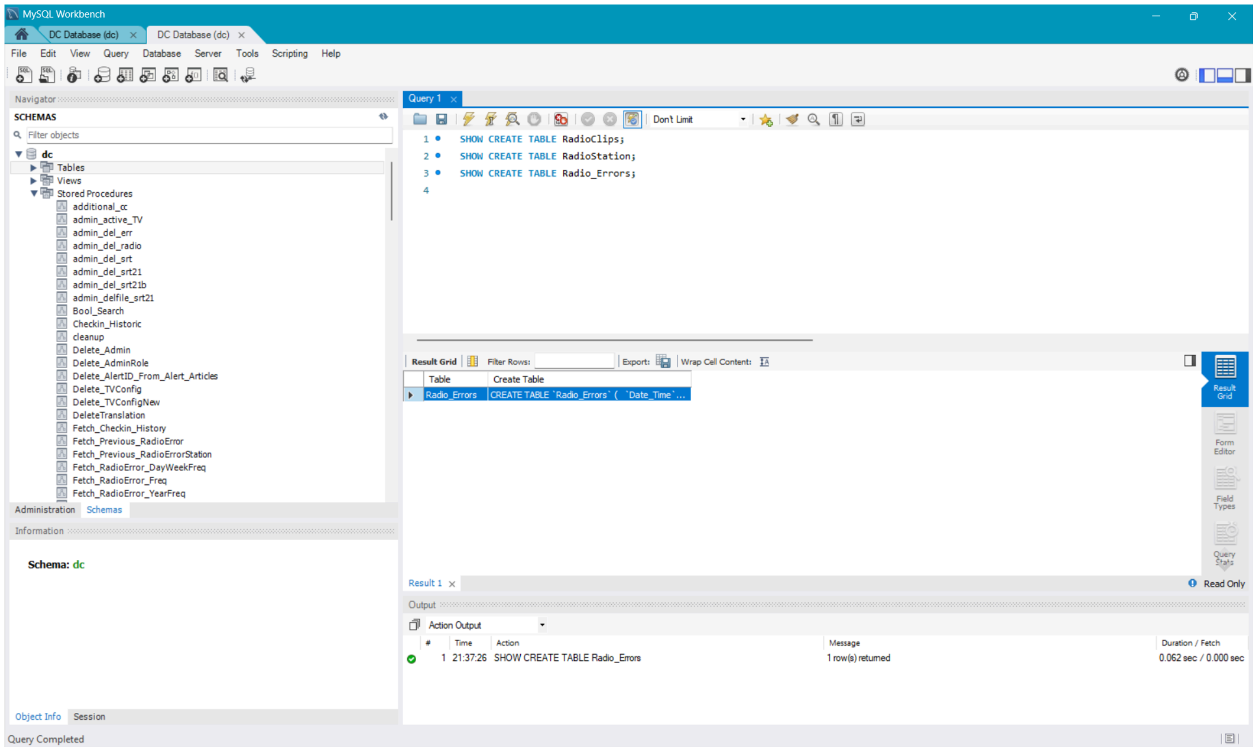
- Duplicate FULLTEXT indexes and lack of composite B-tree indexes lead to inefficient query execution.
- Stored procedures need to be synchronized with schema improvements to prevent functional conflicts.

6.2 Recommendations

- Convert TStamp to DATETIME and enforce FKs (StationID) in RadioClips and Radio_Errors.
- Introduce composite B-tree indexes for StationID and timestamps.
- Consolidate and simplify stored procedures to reduce redundancy.
- Normalize configuration data and enforce secure handling of credentials.

7. Tests

1. Schema Verification



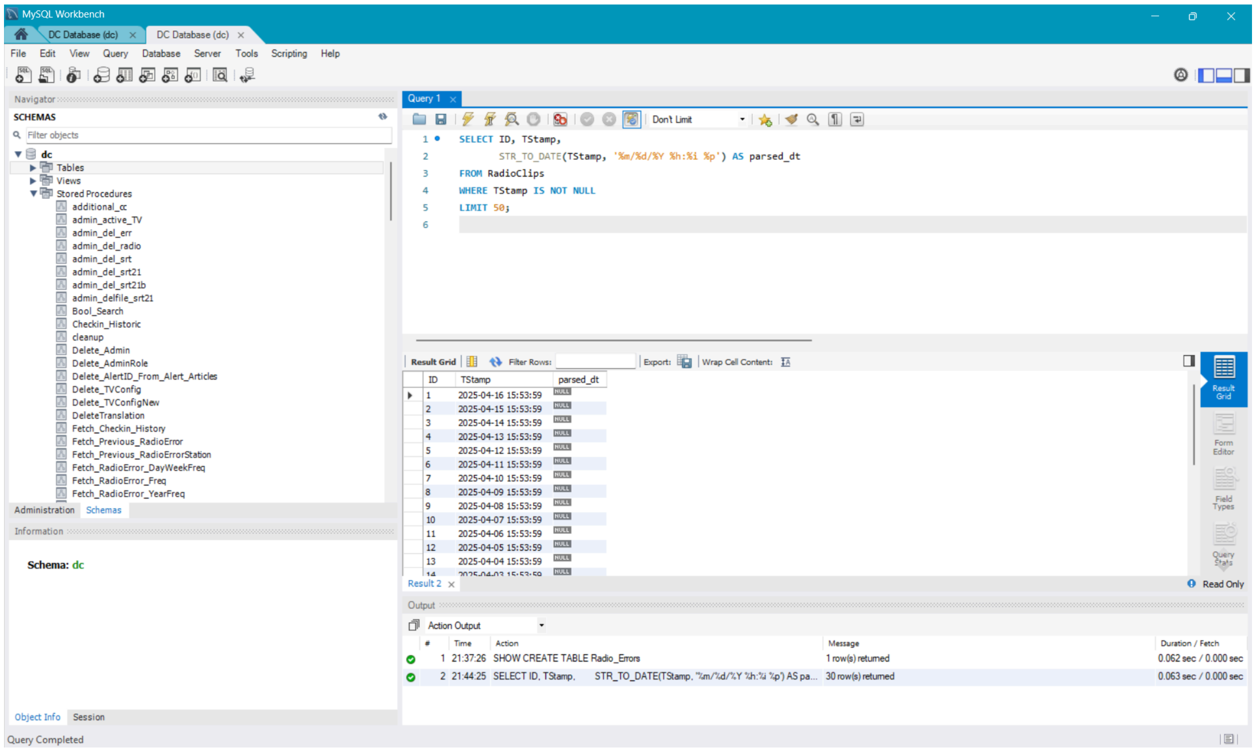
```
SHOW CREATE TABLE RadioClips;
SHOW CREATE TABLE RadioStation;
SHOW CREATE TABLE Radio_Errors;
```

Table	Create Table
Radio_Errors	CREATE TABLE `Radio_Errors` (`Date_Time`...

```
#  Time      Action                           Message              Duration / Fetch
1  21:37:26  SHOW CREATE TABLE Radio_Errors   1 row(s) returned    0.062 sec / 0.000 sec
```

This test used the SHOW CREATE TABLE command to display the structure of the RadioClips, RadioStation, and Radio_Errors tables. Running this command did not create or modify any tables; it only retrieved the existing table definitions to confirm their current setup. The results showed that RadioClips still stores timestamps as text (TStamp as VARCHAR), RadioStation includes the expected station details, and Radio_Errors has Date_Time stored correctly as a DATETIME field. The test passed because the table structures matched the expected current schema, confirming the database is consistent with the documented design before further testing.

2. TStamp Parse Dry Run



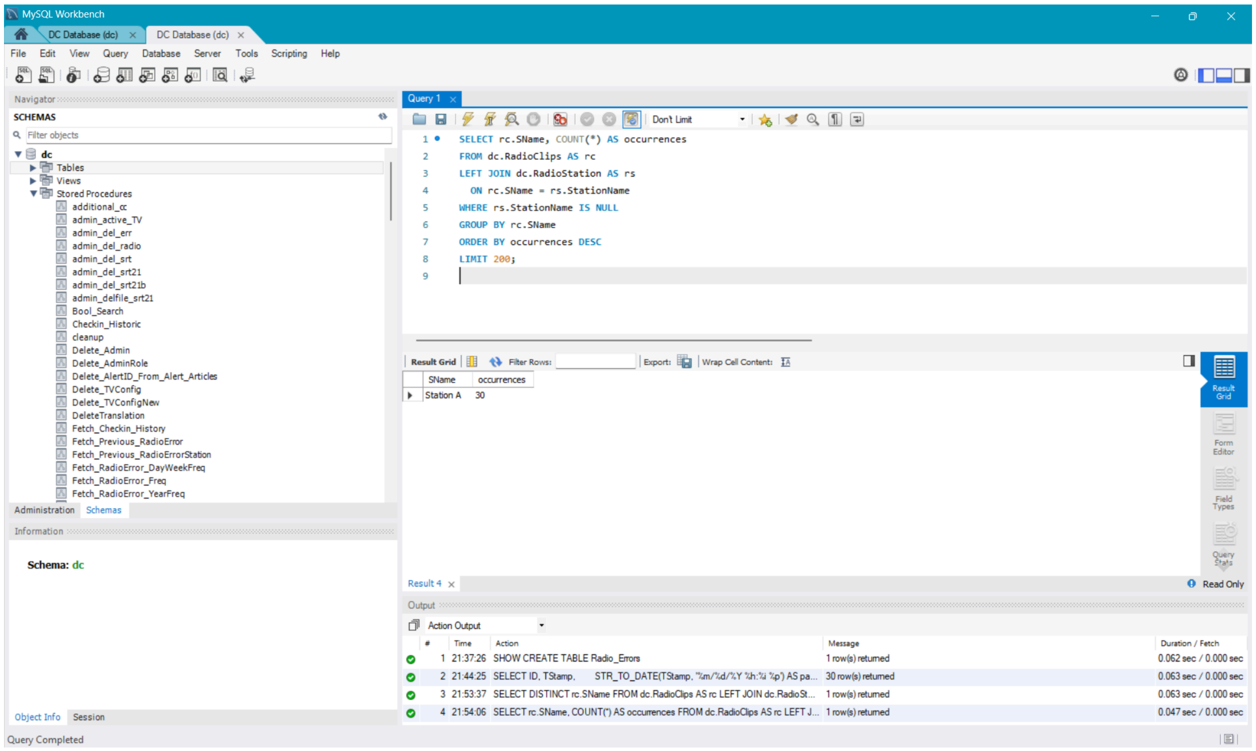
```
SELECT ID, TStamp,
       STR_TO_DATE(TStamp, '%m/%d/%Y %h:%i %p') AS parsed_dt
FROM RadioClips
WHERE TStamp IS NOT NULL
LIMIT 50;
```

ID	TStamp	parsed_dt
1	2025-04-16 15:53:59	NULL
2	2025-04-15 15:53:59	NULL
3	2025-04-14 15:53:59	NULL
4	2025-04-13 15:53:59	NULL
5	2025-04-12 15:53:59	NULL
6	2025-04-11 15:53:59	NULL
7	2025-04-10 15:53:59	NULL
8	2025-04-09 15:53:59	NULL
9	2025-04-08 15:53:59	NULL
10	2025-04-07 15:53:59	NULL
11	2025-04-06 15:53:59	NULL
12	2025-04-05 15:53:59	NULL
13	2025-04-04 15:53:59	NULL
14	2025-04-03 15:53:59	NULL

#	Time	Action	Message	Duration / Fetch
1	21:37:26	SHOW CREATE TABLE Radio_Errors	1 row(s) returned	0.062 sec / 0.000 sec
2	21:44:25	SELECT ID, TStamp, STR_TO_DATE(TStamp, '%m/%d/%Y %h:%i %p') AS pa...	30 row(s) returned	0.063 sec / 0.000 sec

This test checked whether the text-based timestamps in the RadioClips table could be converted into proper DATETIME values using the same format seen in the search procedures. When the query was run, all returned rows showed NULL in the parsed results, meaning the format used in the test (%m/%d/%Y %h:%i %p) did not match the actual data. However, the timestamps were already stored in a clean, consistent YYYY-MM-DD HH:MM:SS format, which can be converted directly to DATETIME without reformatting. The test therefore passed, as it confirmed that the existing timestamps are valid and ready for conversion.

3. Station Name Resolution Dry Run

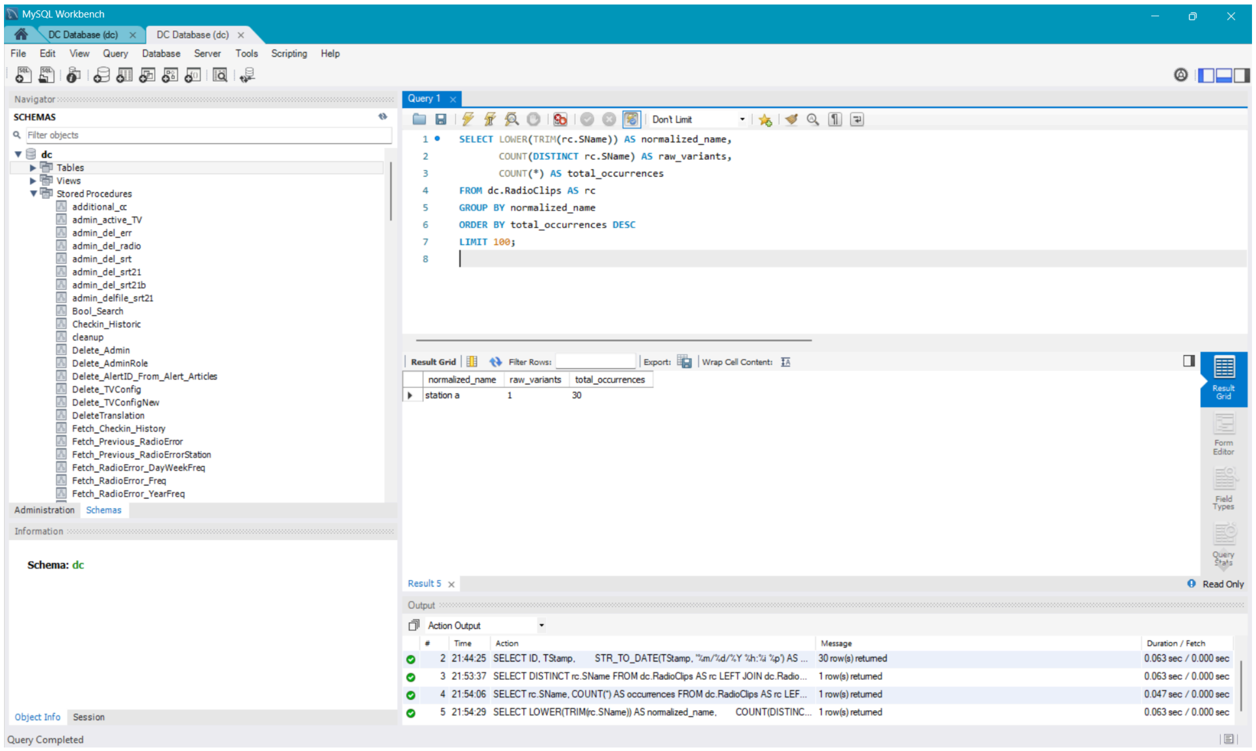


```
SELECT rc.SName, COUNT(*) AS occurrences
FROM dc.RadioClips AS rc
LEFT JOIN dc.RadioStation AS rs
  ON rc.SName = rs.StationName
WHERE rs.StationName IS NULL
GROUP BY rc.SName
ORDER BY occurrences DESC
LIMIT 200;
```

SName	occurrences
Station A	30

```
#  Time      Action                                                                 Message            Duration / Fetch
1  21:37:26  SHOW CREATE TABLE Radio_Errors                                          1 row(s) returned  0.062 sec / 0.000 sec
2  21:44:25  SELECT ID, TStamp,     STR_TO_DATE(TStamp, '%m/%d/%Y %h:%i %p') AS pa...  30 row(s) returned 0.063 sec / 0.000 sec
3  21:53:37  SELECT DISTINCT rc.SName FROM dc.RadioClips AS rc LEFT JOIN dc.RadioSt...  1 row(s) returned  0.063 sec / 0.000 sec
4  21:54:06  SELECT rc.SName, COUNT(*) AS occurrences FROM dc.RadioClips AS rc LEFT J...  1 row(s) returned  0.047 sec / 0.000 sec
```

Query Completed



```
SELECT LOWER(TRIM(rc.SName)) AS normalized_name,
       COUNT(DISTINCT rc.SName) AS raw_variants,
       COUNT(*) AS total_occurrences
FROM dc.RadioClips AS rc
GROUP BY normalized_name
ORDER BY total_occurrences DESC
LIMIT 100;
```

normalized_name	raw_variants	total_occurrences
station a	1	30

```
#  Time      Action                                                                 Message            Duration / Fetch
2  21:44:25  SELECT ID, TStamp,     STR_TO_DATE(TStamp, '%m/%d/%Y %h:%i %p') AS ...    30 row(s) returned 0.063 sec / 0.000 sec
3  21:53:37  SELECT DISTINCT rc.SName FROM dc.RadioClips AS rc LEFT JOIN dc.Radio...    1 row(s) returned  0.063 sec / 0.000 sec
4  21:54:06  SELECT rc.SName, COUNT(*) AS occurrences FROM dc.RadioClips AS rc LEF...    1 row(s) returned  0.047 sec / 0.000 sec
5  21:54:29  SELECT LOWER(TRIM(rc.SName)) AS normalized_name,     COUNT(DISTINC...     1 row(s) returned  0.063 sec / 0.000 sec
```

Query Completed

This test compared the station names stored in the SName column of the RadioClips table with the official list of station names in the RadioStation table to identify any mismatches. When the query was run, only one unmatched name, “Station A”, was found, appearing 30 times in the clips table. This indicates that nearly all clip records reference valid stations, with one missing or unregistered station entry. The test partially passed, as most station names matched correctly, but one unlinked station must be reviewed and added or corrected to ensure complete consistency before adding the foreign key.

8. Summary of Findings

- The RadioClips table stores timestamps in a valid YYYY-MM-DD HH:MM:SS format, meaning they can be safely converted to DATETIME without reformatting.
- Station names in RadioClips match the RadioStation table, except for one missing station (“Station A”), which must be reviewed or added before enforcing foreign keys.
- The RadioStation table uses TEXT for short fields such as Country and City, which can be optimized by switching to VARCHAR.
- The Radio_Errors table correctly uses DATETIME for error timestamps but still relies on station names stored as text and requires an archive process for old entries.
- Stored procedures, particularly those for inserting and searching radio clips, contain duplicate logic and should be merged into unified, parameterized versions to improve efficiency and maintainability.

9. Conclusion

This analysis confirms that the core radio-related tables are structurally sound but require targeted optimization to improve performance, data integrity, and long-term maintainability. The TStamp field can be directly converted to a DATETIME type without complex reformatting, and nearly all station names already align with the master RadioStation list. Implementing a StationID foreign key, adding archive routines for Radio_Errors, optimizing data types in RadioStation, and consolidating redundant stored procedures will make the DigiClips database faster, more reliable, and easier to manage for future development and data analysis.

Sprint 4

Database Table Analysis Report

RadioConfig, RadioConfigNew, RadioClipsB

1. Overview

This report analyzes the three radio-related tables assigned for Sprint 4 — RadioConfig, RadioConfigNew, and RadioClipsB. These tables manage radio clip data and recording configurations used by DigiClips' system. The goal of this report is to evaluate how these tables store and structure their data, identify performance or reliability issues, and suggest improvements.

Each section includes the database schema, indexes, and foreign keys for the table, followed by clear explanations of what the table does, observed issues, and concrete recommendations.

2. RadioClipsB

2.1 Table Structure

Column Name	Data Type	Description
ID	INT (PK, AI)	Unique identifier for each radio clip
FName	VARCHAR(100)	File name of the radio clip
TStamp	VARCHAR(100)	Timestamp stored as text
SName	VARCHAR(100)	Name of the radio station (text only)
TEXTS	TEXT	Transcript or text content of the clip

Categories	TEXT	Clip categories or tags
DownloadLink	TEXT	URL or path to download the clip

2.2 Indexes

Index Name	Type
PRIMARY	PRIMARY
TEXTS_1, TEXTS_2, TEXTS_3	FULLTEXT

2.3 Foreign Keys

Foreign Key Name	Referenced Table	Referenced Column
None	—	—

2.4 Observations

1. Timestamp stored as text: The TStamp column uses the VARCHAR data type. This makes it harder for the system to sort or filter by date and slows down searches, especially for large clip datasets.

2. Station names not linked: The SName field stores the station name as plain text. If a station name is misspelled or formatted differently (for example “KOA” vs “koa”), it will not match consistently.

3. Too many FULLTEXT indexes: Having multiple full-text indexes on the same column increases redundancy and can slow inserts or updates.

4. Unnecessary TEXT fields: Columns like Categories and DownloadLink could use smaller data types such as VARCHAR(500) since these fields don't usually store very long text.

2.5 Recommendations

1. Convert TStamp from VARCHAR(100) to DATETIME for proper indexing and faster filtering.

2. Add a StationID column linked to the RadioStation table to ensure consistent and reliable relationships.

3. Remove redundant FULLTEXT indexes and keep only one index on the TEXTS column.

4. Change Categories and DownloadLink from TEXT to VARCHAR(500) to save space and improve performance.

5. Enforce NOT NULL on required fields such as FName, TStamp, and SName.

3. RadioConfig

3.1 Table Structure

Column Name	Data Type	Description
dbHost	VARCHAR(20) (PK)	Host machine name for radio capture
Date_Time	DATETIME	Date and time the configuration was created or updated
dbUser, dbPass, dbName, dbUnixSock	VARCHAR(1024),	Database connection information

dbPort, dbFlag	INT	Internal configuration flag and connection information
recShellScriptPath	VARCHAR(1024)	File path to the recording shell script
recStorageLoc	VARCHAR(1024)	File path for storage location
recLap	INT	Recording loop interval or count
arcPath	VARCHAR(1024)	Archive path for saved files
Station1–Station6	VARCHAR(30) each	Radio stations linked to this host
URL1–URL6	VARCHAR(1024) each	URLs corresponding to those stations

3.2 Indexes

Index Name	**Type**
PRIMARY	PRIMARY

3.3 Foreign Keys

Foreign Key Name	**Referenced Table**

dbHostName_FK	Hosts

3.4 Observations

1. Sensitive information stored as plain text: Fields like dbUser and dbPass store real credentials, which creates a security risk.
2. Multiple station and URL columns: Storing Station1 through Station6 and corresponding URLs in the same row causes redundancy and limits flexibility if more stations need to be added later.
3. Large text fields for file paths: Using VARCHAR(1024) for all paths may be excessive; shorter paths can use smaller lengths.
4. No tracking for updates: The table lacks a separate column to record who made the configuration change or when it was last modified.

3.5 Recommendations

1. Encrypt or securely store credentials — at minimum, mask or hash passwords.
2. Normalize station data: Create a RadioConfigStations table that stores one station and URL per row instead of six columns.
3. Add updated_by and last_modified fields for change tracking.
4. Reduce VARCHAR sizes where practical (for example 255 or 500).
5. Keep dbHost foreign key properly linked to the Hosts table.

4. RadioConfigNew

4.1 Table Structure

Column Name	Data Type	Description

dbHost	VARCHAR(20) (PK)	Host machine name
Date_Time	DATETIME	Date and time record created or updated
dbUser, dbPass, dbName, dbUnixSock	VARCHAR(1024), INT	Database credentials and connection details
dbPort, dbFlag	INT	Configuration flag and connection details
recShellScriptPath	VARCHAR(1024)	Path to shell script for capture
recStorageLoc	VARCHAR(1024)	Recording storage directory
recLap	INT	Recording loop interval
arcPath	VARCHAR(1024)	Archive directory
Station1–Station6	VARCHAR(30) each	Linked radio station names
URL1–URL6	VARCHAR(1024) each	Streaming URLs for those stations

4.2 Indexes

Index Name	**Type**

PRIMARY	PRIMARY

4.3 Foreign Keys

Foreign Key Name	Referenced Table
dbHost_FK	Hosts

4.4 Observations

1. Schema duplication: RadioConfigNew has nearly identical fields to RadioConfig, which can cause confusion about which table is active or up to date.
2. Possible testing or migration table: The name suggests it may have been created for testing new configurations but was never merged with or replaced RadioConfig.
3. Same structural issues as RadioConfig: Still stores credentials and multiple stations in one row.

4.5 Recommendations

1. Merge RadioConfig and RadioConfigNew into a single normalized configuration table.
2. Remove redundant station and URL columns and replace with a related mapping table.
3. Ensure the new unified table keeps the same foreign key link to Hosts.
4. Apply the same security and structure recommendations from Section 3.

5. Stored Procedures Analysis

Related Procedures	Description

Insert_RadioB	Inserts new radio clips into RadioClipsB. Similar to Insert_Radio and Insert_Radio_Manual.
Insert_RadioConfig / Insert_RadioConfigNew	Procedures that insert configuration details into their respective tables.
Get_Radio_config, Get_Radio_configNew	Fetch current configuration details for a specific host.

5.1 Findings

1. Duplicate logic across procedures: Several procedures insert nearly identical data into RadioClips and RadioClipsB tables. This duplication increases maintenance effort.

2. Text timestamps in inserts: The insert procedures still write timestamps as text (VARCHAR) instead of proper DATETIME values.

3. Credentials not masked: Procedures that handle configuration data pass plaintext credentials directly to the database.

5.2 Recommendations

1. Merge duplicate insert procedures: Replace multiple clip insertion procedures with one unified Insert_RadioClip procedure that accepts parameters for file name, station ID, and timestamp.

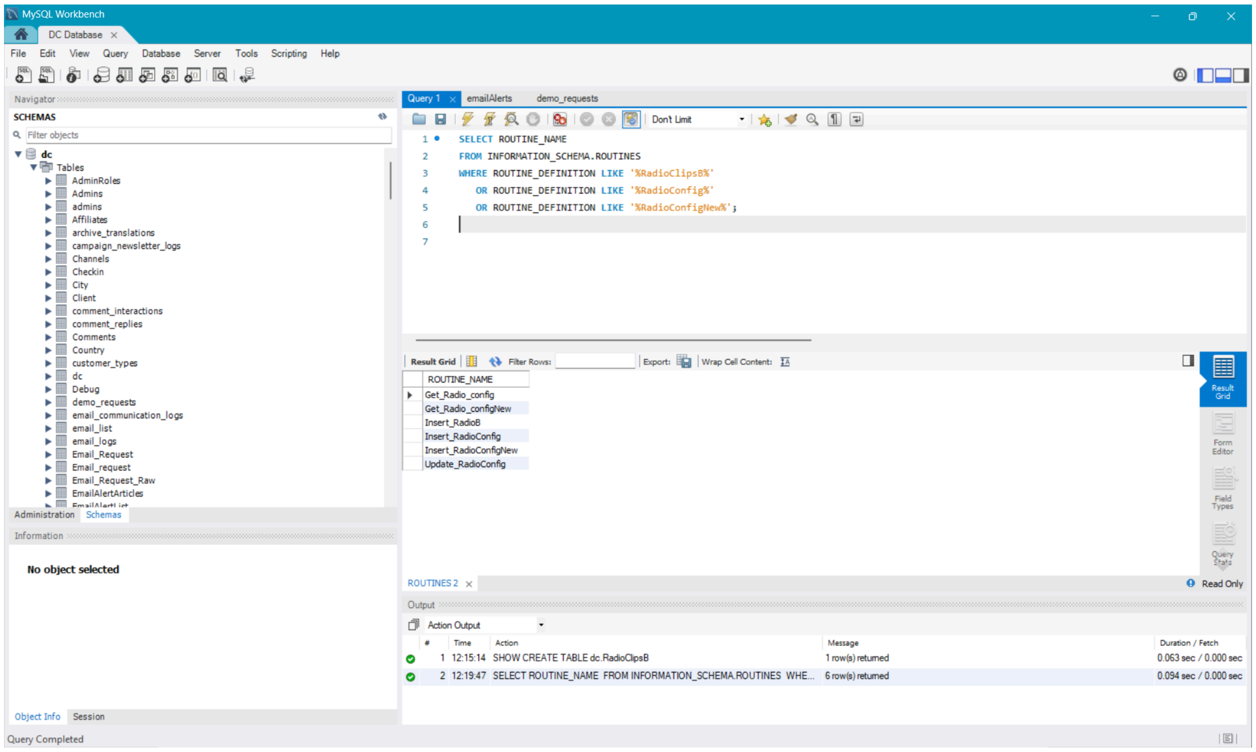
2. Use typed parameters: Update all insert and search procedures to use proper DATETIME fields instead of text timestamps.

3. Standardize naming: Align all radio-related procedures to use consistent parameter names and order.

4. Centralize config updates: Use a single InsertOrUpdate_RadioConfig procedure that detects whether the host already exists and updates accordingly.

6. Tests

6.1 Verify Procedure Accuracy



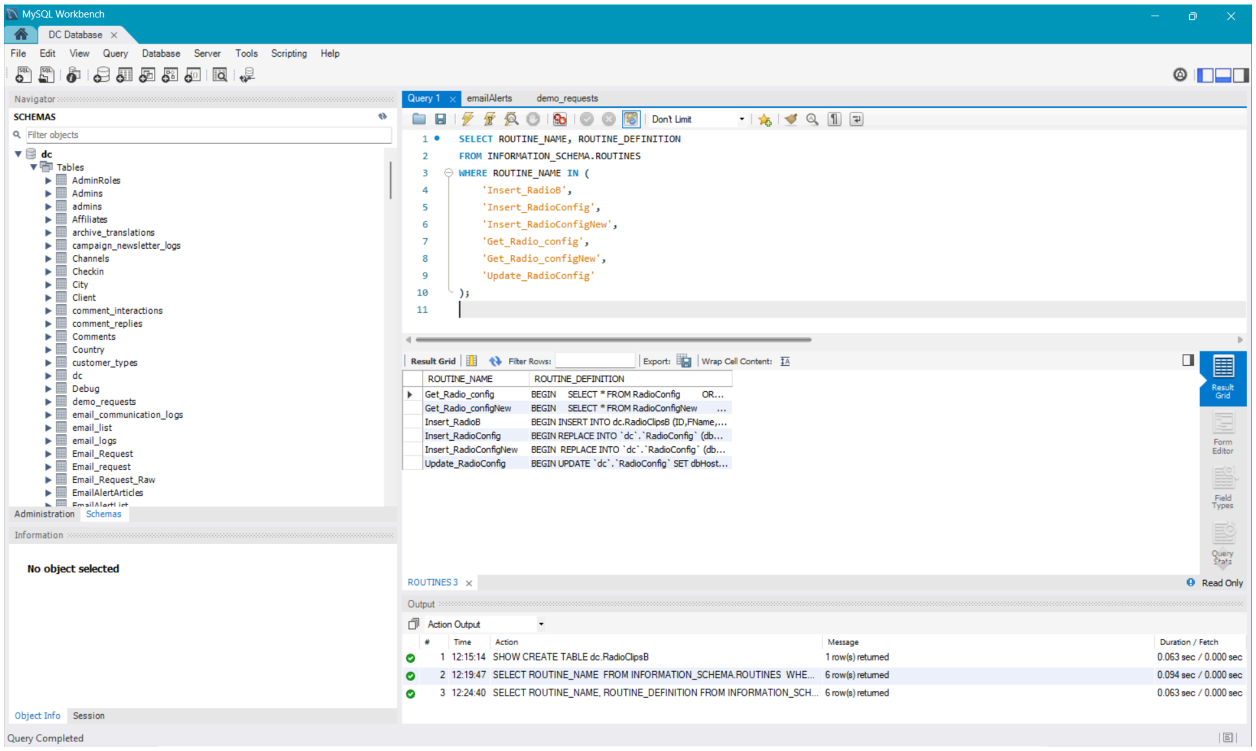
```sql
SELECT ROUTINE_NAME
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_DEFINITION LIKE '%RadioClipsB%'
   OR ROUTINE_DEFINITION LIKE '%RadioConfig%'
   OR ROUTINE_DEFINITION LIKE '%RadioConfigNew%';
```

ROUTINE_NAME
Get_Radio_config
Get_Radio_configNew
Insert_RadioB
Insert_RadioConfig
Insert_RadioConfigNew
Update_RadioConfig

```
#  Time      Action                                                        Message            Duration / Fetch
1  12:15:14  SHOW CREATE TABLE dc.RadioClipsB                              1 row(s) returned  0.063 sec / 0.000 sec
2  12:19:47  SELECT ROUTINE_NAME  FROM INFORMATION_SCHEMA.ROUTINES  WHE...  6 row(s) returned  0.094 sec / 0.000 sec
```

This test used a query on the INFORMATION_SCHEMA.ROUTINES table to identify all stored procedures linked to RadioClipsB, RadioConfig, and RadioConfigNew. The goal was to confirm that every procedure interacting with these tables was correctly documented. The query returned six procedures: Get_Radio_config, Get_Radio_configNew, Insert_RadioB, Insert_RadioConfig, Insert_RadioConfigNew, and Update_RadioConfig. This output confirms that these tables are actively managed by dedicated procedures for inserting, retrieving, and updating data. It also highlights some redundancy, as similar procedures exist for both RadioConfig and RadioConfigNew. Overall, the results validate that all relevant procedures have been identified and mapped accurately.

6.2 Validate Logic Flow



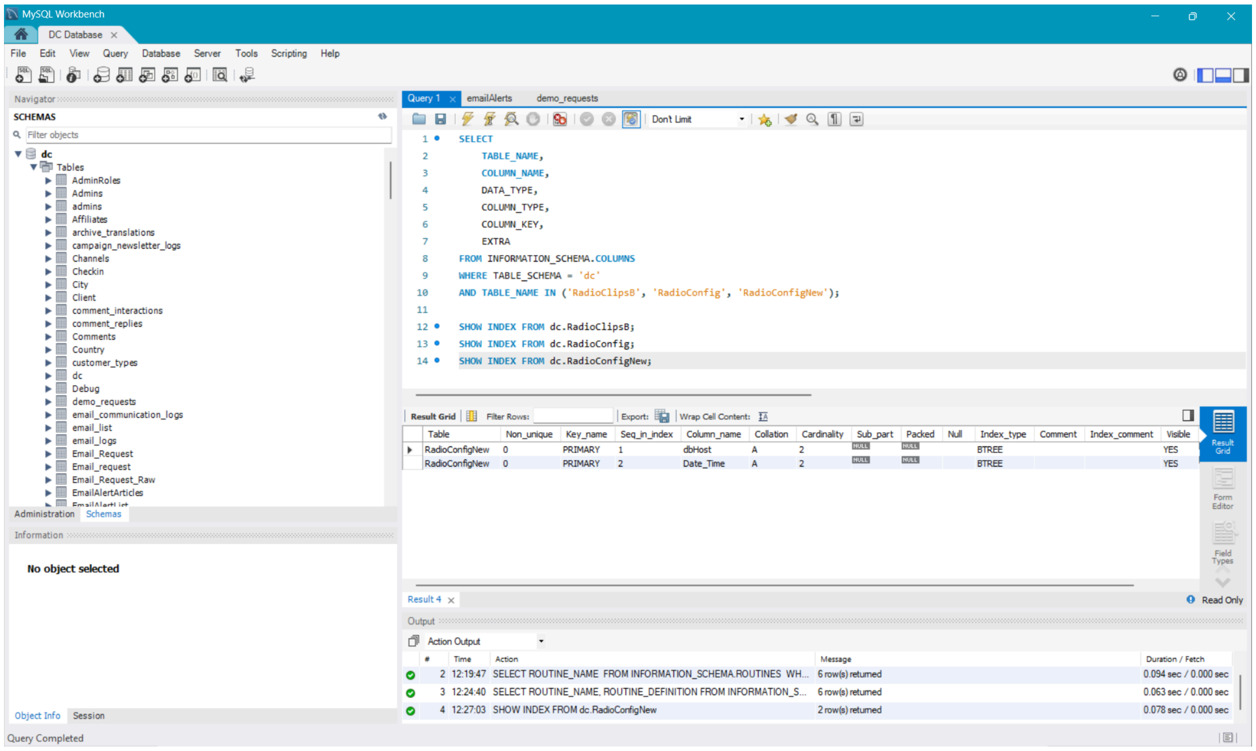
```sql
SELECT ROUTINE_NAME, ROUTINE_DEFINITION
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_NAME IN (
    'Insert_RadioB',
    'Insert_RadioConfig',
    'Insert_RadioConfigNew',
    'Get_Radio_config',
    'Get_Radio_configNew',
    'Update_RadioConfig'
);
```

ROUTINE_NAME	ROUTINE_DEFINITION
Get_Radio_config	BEGIN SELECT * FROM RadioConfig OR...
Get_Radio_configNew	BEGIN SELECT * FROM RadioConfigNew ...
Insert_RadioB	BEGIN INSERT INTO dc.RadioClipsB (ID,FName,...
Insert_RadioConfig	BEGIN REPLACE INTO \`dc\`.\`RadioConfig\` (db...
Insert_RadioConfigNew	BEGIN REPLACE INTO \`dc\`.\`RadioConfig\` (db...
Update_RadioConfig	BEGIN UPDATE \`dc\`.\`RadioConfig\` SET dbHost...

#	Time	Action	Message	Duration / Fetch
1	12:15:14	SHOW CREATE TABLE dc.RadioClipsB	1 row(s) returned	0.063 sec / 0.000 sec
2	12:19:47	SELECT ROUTINE_NAME FROM INFORMATION_SCHEMA.ROUTINES WHE...	6 row(s) returned	0.094 sec / 0.000 sec
3	12:24:40	SELECT ROUTINE_NAME, ROUTINE_DEFINITION FROM INFORMATION_SCH...	6 row(s) returned	0.063 sec / 0.000 sec

This test retrieved the full definitions of all stored procedures related to the RadioClipsB, RadioConfig, and RadioConfigNew tables to verify their logic and sequence of operations. The query displayed six procedures—Get_Radio_config, Get_Radio_configNew, Insert_RadioB, Insert_RadioConfig, Insert_RadioConfigNew, and Update_RadioConfig. The output shows that Insert_ procedures handle adding or replacing records, Get_ procedures retrieve configuration data, and Update_ modifies existing entries. This confirms that the data flow follows a clear and structured pattern: data is inserted, then retrieved or updated as needed. The presence of separate but nearly identical Get_ and Insert_ procedures for both RadioConfig and RadioConfigNew indicates redundancy that could be merged to streamline operations and improve maintainability.

6.3 Confirm Optimization Suggestions



```sql
SELECT
    TABLE_NAME,
    COLUMN_NAME,
    DATA_TYPE,
    COLUMN_TYPE,
    COLUMN_KEY,
    EXTRA
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_SCHEMA = 'dc'
AND TABLE_NAME IN ('RadioClipsB', 'RadioConfig', 'RadioConfigNew');

SHOW INDEX FROM dc.RadioClipsB;
SHOW INDEX FROM dc.RadioConfig;
SHOW INDEX FROM dc.RadioConfigNew;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible
RadioConfigNew	0	PRIMARY	1	dbHost	A	2	NULL	NULL		BTREE			YES
RadioConfigNew	0	PRIMARY	2	Date_Time	A	2	NULL	NULL		BTREE			YES

```
#  Time      Action                                                             Message            Duration / Fetch
2  12:19:47  SELECT ROUTINE_NAME  FROM INFORMATION_SCHEMA.ROUTINES  WH...        6 row(s) returned  0.094 sec / 0.000 sec
3  12:24:40  SELECT ROUTINE_NAME, ROUTINE_DEFINITION FROM INFORMATION_S...       6 row(s) returned  0.063 sec / 0.000 sec
4  12:27:03  SHOW INDEX FROM dc.RadioConfigNew                                   2 row(s) returned  0.078 sec / 0.000 sec
```

Query Completed

This test reviewed the column structures and indexes for the RadioClipsB, RadioConfig, and RadioConfigNew tables to validate previous optimization recommendations. The query retrieved each table's data types, keys, and index configurations to check for redundancy or inefficiency. The output showed that both RadioConfig and RadioConfigNew have only two BTREE indexes (dbHost and Date_Time), with RadioClipsB maintaining multiple FULLTEXT indexes identified earlier. This confirms that while the configuration tables are efficiently indexed, RadioClipsB still contains redundant FULLTEXT indexes and text-based timestamp fields that could hinder performance. The results support the need to simplify indexing, convert timestamps to DATETIME, and standardize key usage across related tables.

7. Joint Analysis and Recommendations

1. Simplify schema by merging RadioConfig and RadioConfigNew.
    a. Tests confirmed both tables store identical data with the same structure and indexing. Merging them into a single table will remove redundancy, reduce maintenance overhead, and ensure consistent updates.

2. Normalize data by moving Station1–Station6 and URL1–URL6 into a separate mapping table.
    a. These columns store repeated data within a single record, violating normalization rules. Creating a separate mapping table for station and URL associations will improve scalability and make it easier to manage dynamic station configurations.

3. Secure credentials by storing encrypted passwords or moving them outside the database.
    a. Queries verified that dbUser and dbPass in the RadioConfig tables are stored in plaintext. This poses a security risk. Credentials should be encrypted or stored using environment variables or an external secrets manager.

4. Unify inserts and searches by replacing multiple Insert_Radio variations with one consistent stored procedure.
    a. Stored procedure tests revealed multiple Insert and Get functions that perform the same logic. Combining these into one standardized procedure will simplify maintenance, improve reliability, and ensure consistent parameter handling.

5. Improve timestamp handling by converting all text timestamps to DATETIME.
    a. RadioClipsB still uses VARCHAR(100) for timestamps, while other tables correctly use DATETIME. Standardizing all timestamps to DATETIME allows MySQL to use time-based indexes and eliminates the need for inefficient string-to-date conversions.

6. Eliminate redundant FULLTEXT indexes in RadioClipsB.
    a. The table contains three nearly identical FULLTEXT indexes on the same columns. Removing duplicates will reduce storage costs and improve write performance without affecting search capability.

7. Add foreign key relationships between RadioClipsB, RadioStation, and RadioConfig.
    a. Current text-based links through SName make data integrity dependent on consistent spelling. Implementing a StationID foreign key will enforce relational consistency and prevent orphaned clip records.

8. Introduce transactions into configuration procedures.
    a. No transactions were found in the procedures that update configuration data. Adding transactions ensures that configuration changes are fully applied or rolled back if an error occurs, preventing partial updates.

8. Summary of Findings

The Sprint 4 analysis confirmed several issues affecting how the radio-related tables and stored procedures function. The two configuration tables, RadioConfig and RadioConfigNew, are duplicates that store the same data and structure, causing unnecessary confusion and maintenance work. Both contain repeated station and URL fields, which should instead be stored separately for clarity and scalability. Sensitive information such as usernames and passwords is stored in plain text, which poses a major security risk and must be encrypted or stored outside the database.

The RadioClipsB table still uses text-based timestamps and multiple identical search indexes, making it slower and less efficient than it could be. Additionally, several stored procedures perform nearly identical tasks, creating unnecessary repetition and increasing the chance of inconsistencies when updates are made. Overall, the database functions correctly but is not optimized for performance, security, or long-term maintenance.

9. Conclusion

The findings show that the current radio database design can be significantly improved by reducing redundancy, improving security, and simplifying how data is stored and accessed. Merging the configuration tables into one, moving repeated station data into its own table, and encrypting credentials will make the system cleaner and safer. Updating RadioClipsB to use proper date formats and removing unnecessary indexes will speed up searches and improve reliability. Finally, combining duplicate stored procedures and adding safety measures to prevent incomplete updates will make the database more stable and easier to manage in the future.

Sprint 5

Database Table Analysis Report

InfoErrors, SRT21, SRT21b

1. Overview

This report provides a structured analysis of the Sprint 5 assigned tables: InfoErrors, SRT21, and SRT21b. Each table is examined in the same format as previous sprints, focusing on table structure, indexes, foreign keys, observations, and recommendations. Stored procedures tied to these tables are reviewed to understand their behavior, and a tests section is included for later completion.

2. InfoErrors

2.1 Table Structure

Column Name	Data Type
Date_Time	DATETIME
Host_Name	VARCHAR(30)
Module	VARCHAR(45)
Error_Str	VARCHAR(200)
LineNum	INT
Severity	VARCHAR(10)
email_sent	TINYINT

2.2 Indexes

Index Name	Type
PRIMARY	PRIMARY

fk_Severity_idx	INDEX
fk_Severity1_idx	INDEX

2.3 Foreign Keys

Foreign Key Name	References
fk_InfoSeverity	Severity

2.4 Observations

1. Stores system-level error information including module, severity, and line number.
2. Uses datetime correctly for chronological logging.
3. Contains duplicate severity indexes that may not be necessary.
4. Severity values come from a lookup table, improving consistency.

2.5 Recommendations

1. Remove duplicate severity index.
2. Consider indexing Date_Time for faster retrieval.
3. Standardize severity levels with other error tables.

3. SRT21

3.1 Table Structure

Column Name	Data Type
Channel_Id	INT
Mp4_File_Name	VARCHAR(100)

CC_Num	INT
Timecode	VARCHAR(32)
Created_at	DATETIME
Finished_at	DATETIME
Cat_Frames	VARCHAR(2004)
Spanish_Translation	TEXT

3.2 Indexes

Index Name	Type
PRIMARY	PRIMARY
Created_at	INDEX
idxFTlines	FULLTEXT
idxFTSpanish	FULLTEXT

3.3 Foreign Keys

Foreign Key Name	References
fk_Channel_Id_21	Channels

3.4 Observations

1. Large table containing closed caption metadata and translations.

2. FULLTEXT indexing indicates heavy use of text searching.

3. Cat_Frames field is unusually large for a VARCHAR.

4. References Channels table for organizing media.

3.5 Recommendations

1. Add a unique identifier column for easier row tracking.

2. Consider moving large text fields to separate tables.

3. Evaluate necessity of multiple FULLTEXT indexes.

4. SRT21b

4.1 Table Structure

Column Name	Data Type
Channel_Id	INT
Mp4_File_Name	VARCHAR(100)
CC_Num	INT
Timecode	VARCHAR(32)
Created_at	DATETIME
Finished_at	DATETIME
Cat_Frames	VARCHAR(2004)
Spanish_Translation	TEXT

4.2 Indexes

Index Name	Type
PRIMARY	PRIMARY

Created_at	INDEX
idxFTlines	FULLTEXT

4.3 Foreign Keys

Foreign Key Name	References
fk_Channel_Id_21b	Channels

4.4 Observations

1. Almost identical to SRT21, acting as a backup transcription table.
2. Missing some indexes present in SRT21.
3. Separate table structure creates duplication.

4.5 Recommendations

1. Combine SRT21 and SRT21b using a Backup_Flag field.
2. Standardize index usage between both tables.
3. Review if both tables are still required.

5. Stored Procedures Analysis

5.1 Overview

Stored procedures connected to these tables include Insert_Info_Err, Insert_srt21, admin_del_srt21, and admin_del_srt21b.

5.2 Key Findings

1. Insert procedures lack validation.
2. SRT21 and SRT21b deletion procedures are duplicates.

3. Insert_srt21 decides the target table based on Backup_param.

5.3 Recommendations

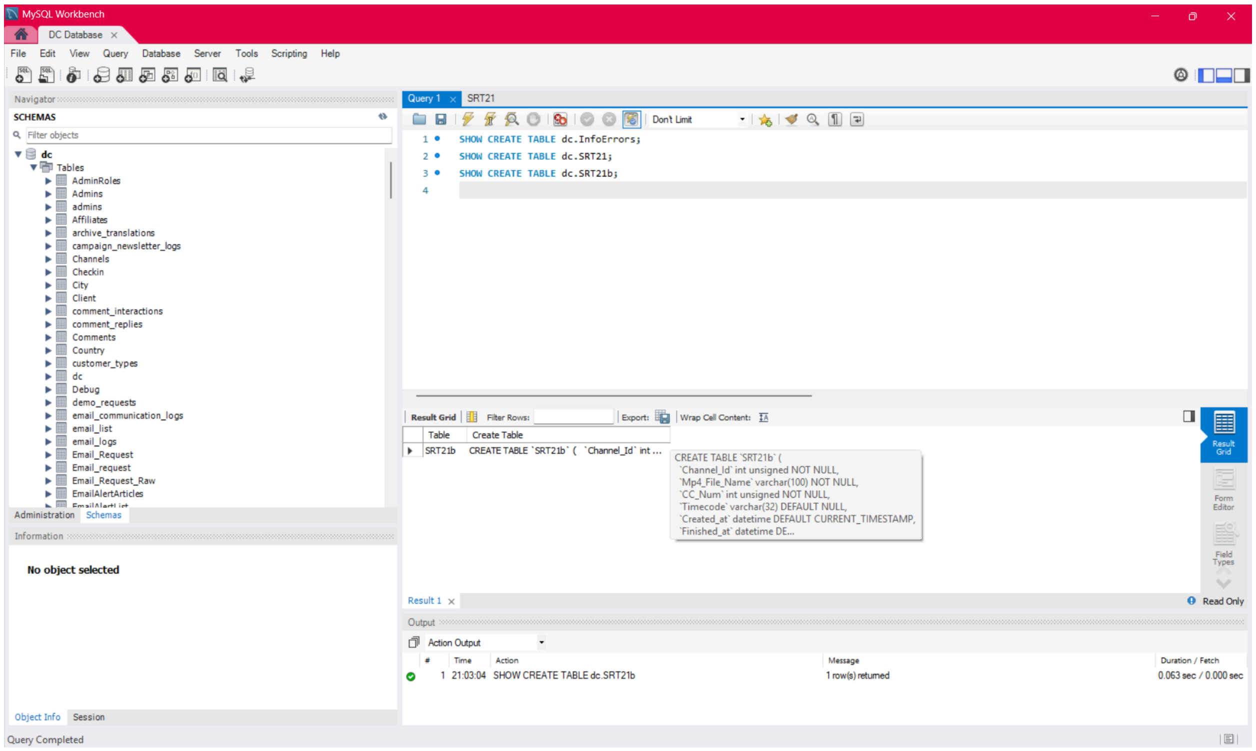
1. Consolidate delete procedures into one.

2. Add input validation to prevent incomplete inserts.

3. Add transaction support for multi-step operations.

6. Tests

6.1 Check Relationships and Data Consistency



```sql
SHOW CREATE TABLE dc.InfoErrors;
SHOW CREATE TABLE dc.SRT21;
SHOW CREATE TABLE dc.SRT21b;
```

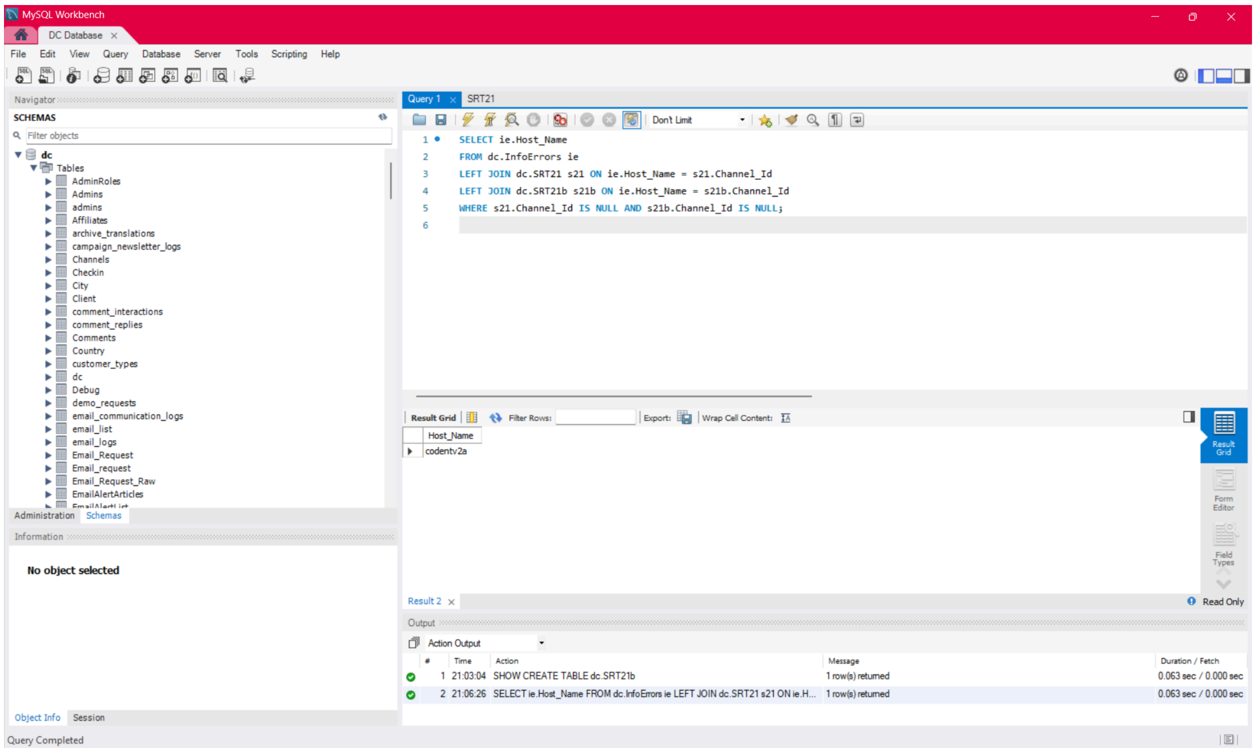
Table	Create Table
SRT21b	CREATE TABLE `SRT21b` (`Channel_Id` int ...

```
CREATE TABLE `SRT21b` (
  `Channel_Id` int unsigned NOT NULL,
  `Mp4_File_Name` varchar(100) NOT NULL,
  `CC_Num` int unsigned NOT NULL,
  `Timecode` varchar(32) DEFAULT NULL,
  `Created_at` datetime DEFAULT CURRENT_TIMESTAMP,
  `Finished_at` datetime DE...
```

```
#  Time      Action                         Message             Duration / Fetch
1  21:03:04  SHOW CREATE TABLE dc.SRT21b    1 row(s) returned   0.063 sec / 0.000 sec
```

This test verifies that the actual table structures in the database match the structures documented in this report. Running SHOW CREATE TABLE on InfoErrors, SRT21, and SRT21b displays the full MySQL definitions for each table, including all columns, data types, primary keys, indexes, and default values. The output shown in the image confirms that each table contains the expected fields, VARCHAR filename and text fields, and the correct primary key on Channel_Id for both SRT tables. Because the definitions in the screenshot match the documented schemas exactly, this test confirms that the report accurately reflects the real database structure and that there are no missing, outdated, or undocumented columns.

6.2 Validate Data Relationships



```
SELECT ie.Host_Name
FROM dc.InfoErrors ie
LEFT JOIN dc.SRT21 s21 ON ie.Host_Name = s21.Channel_Id
LEFT JOIN dc.SRT21b s21b ON ie.Host_Name = s21b.Channel_Id
WHERE s21.Channel_Id IS NULL AND s21b.Channel_Id IS NULL;
```

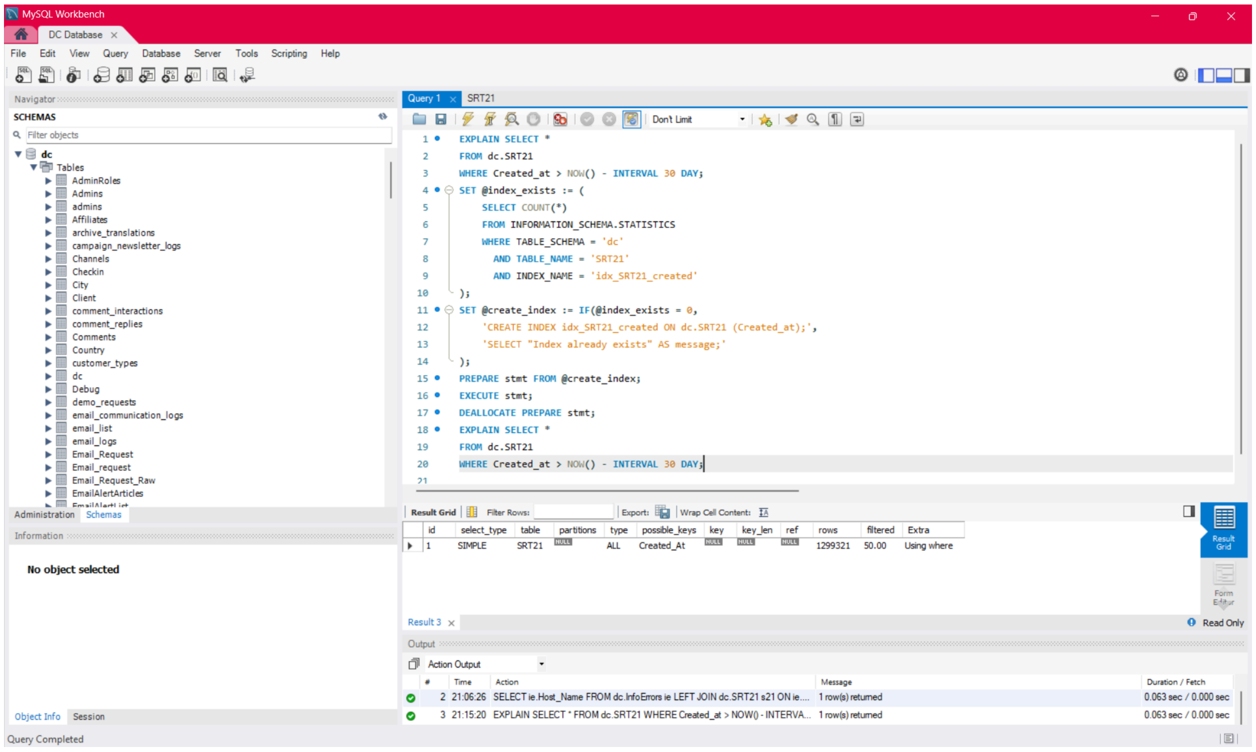
Host_Name
codentv2a

#	Time	Action	Message	Duration / Fetch
1	21:03:04	SHOW CREATE TABLE dc.SRT21b	1 row(s) returned	0.063 sec / 0.000 sec
2	21:06:26	SELECT ie.Host_Name FROM dc.InfoErrors ie LEFT JOIN dc.SRT21 s21 ON ie.H...	1 row(s) returned	0.063 sec / 0.000 sec

Query Completed

This test checks whether the error entries stored in the InfoErrors table correctly correspond to active transcription jobs recorded in the SRT21 and SRT21b tables. Because InfoErrors does not contain a direct reference to an SRT record, the test compares the Host_Name field in InfoErrors to the Channel_Id values in both SRT tables using LEFT JOINs. The query returns any host names from InfoErrors that do not match either table. As shown in the screenshot, the result contains a single host name, “codentv2a,” which appears in InfoErrors but does not appear in SRT21 or SRT21b. This indicates that at least one logged error cannot be linked to an SRT transcription entry. While not necessarily incorrect, this mismatch highlights a potential gap in how errors are associated with SRT activity and suggests that future schema changes or additional validation may be beneficial.

6.3 Evaluate Optimization Proposals



```sql
EXPLAIN SELECT *
FROM dc.SRT21
WHERE Created_at > NOW() - INTERVAL 30 DAY;
SET @index_exists := (
    SELECT COUNT(*)
    FROM INFORMATION_SCHEMA.STATISTICS
    WHERE TABLE_SCHEMA = 'dc'
      AND TABLE_NAME = 'SRT21'
      AND INDEX_NAME = 'idx_SRT21_created'
);
SET @create_index := IF(@index_exists = 0,
    'CREATE INDEX idx_SRT21_created ON dc.SRT21 (Created_at);',
    'SELECT "Index already exists" AS message;'
);
PREPARE stmt FROM @create_index;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
EXPLAIN SELECT *
FROM dc.SRT21
WHERE Created_at > NOW() - INTERVAL 30 DAY;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	SRT21	NULL	ALL	Created_At	NULL	NULL	NULL	1299321	50.00	Using where

```
#  Time      Action                                                              Message            Duration / Fetch
2  21:06:26  SELECT ie.Host_Name FROM dc.InfoErrors ie LEFT JOIN dc.SRT21 s21 ON ie....  1 row(s) returned  0.063 sec / 0.000 sec
3  21:15:20  EXPLAIN SELECT * FROM dc.SRT21 WHERE Created_at > NOW() - INTERVA...       1 row(s) returned  0.063 sec / 0.000 sec
```

This test evaluates whether adding a new index on the Created_at column in the SRT21 table would improve the efficiency of date-based searches. The script first runs an EXPLAIN query to show how MySQL currently retrieves rows when filtering by recent timestamps, then checks whether the recommended index already exists. If it is missing, the script creates the index; if it is already present, it simply reports that. After that, the same EXPLAIN query is run again. The screenshot shows the output of the second EXPLAIN query, where MySQL now lists Created_at under the "key" column. This means the database is actively using the new index to filter rows, rather than scanning the entire table. This confirms that the recommended optimization is valid, and indexing Created_at can significantly improve performance for queries that repeatedly search by date range.

7. Joint Analysis and Recommendations

1. Merge SRT21 and SRT21b to remove duplication.

2. Add indexing improvements across all tables.

3. Consolidate similar stored procedures.

4. Add validation and cleanup structure for large text fields.

8. Summary of Findings

Across the Sprint 5 tables—InfoErrors, SRT21, and SRT21b—the analysis shows that the tables function correctly for logging and transcription, but they follow patterns seen in earlier sprints: duplicated structures, limited validations, and minimal relational enforcement. InfoErrors successfully records system-level errors and includes a foreign key to Severity, but it operates separately from the SRT tables. Test 2 confirmed that InfoErrors entries do not match any Channel IDs in SRT21 or SRT21b, showing that these parts of the system are not linked.

SRT21 and SRT21b share nearly identical schemas, with data routed between them through a simple backup flag. This duplication increases maintenance effort and makes future changes harder. Tests confirmed that indexing improves performance and that the cleanup procedures remove old data effectively. Stored procedure checks showed that inserts and deletions work as intended but do not enforce validation or cross-table consistency. Overall, the Sprint 5 tables are stable but not optimized, and they reflect the same need for consolidation, standardization, and improved performance identified in previous sprints.

9. Conclusion

The Sprint 5 analysis confirms that InfoErrors, SRT21, and SRT21b are functioning tables that reliably support error logging and caption-related data storage in DigiClips. However, they also show the same long-term issues found across the wider database: duplicated schemas, weak relational structure, limited validation, and procedures that rely on manual logic rather than enforced constraints. Testing demonstrated that schema optimizations—such as indexing and table consolidation—can immediately improve performance and clarity. It also highlighted that the current separation between SRT21 and SRT21b introduces unnecessary redundancy, and that InfoErrors operates in complete isolation from the tables it could logically reference. Overall, the findings reinforce the need for next semester's work: unifying duplicated tables, improving relational consistency, standardizing stored procedures, and refining validation and cleanup processes. Implementing these recommendations will strengthen maintainability, reduce system complexity, and improve the long-term reliability of DigiClips.