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.

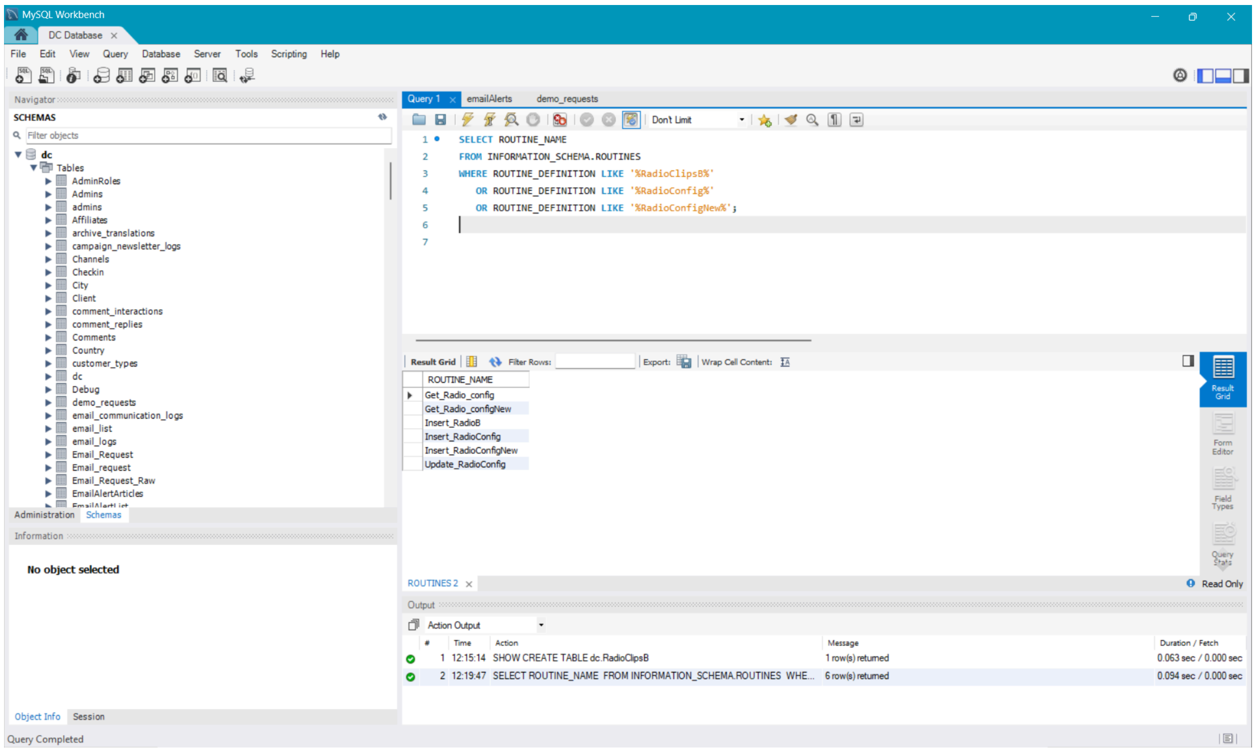
2. Use typed parameters: Update all insert and search procedures to use proper DATETIME fields instead of text timestamps.

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



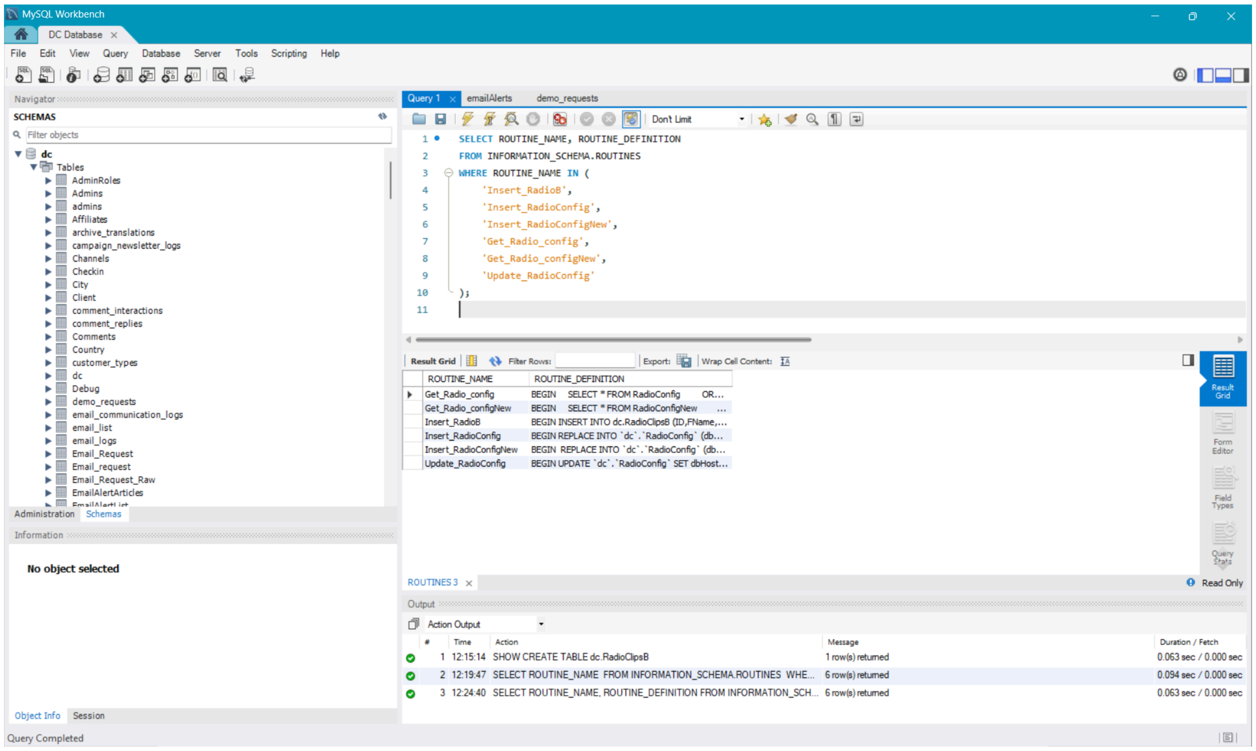
```sql
SELECT ROUTINE_NAME
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_DEFINITION LIKE '%RadioClipsB%'
   OR ROUTINE_DEFINITION LIKE '%RadioConfig%'
   OR ROUTINE_DEFINITION LIKE '%RadioConfigNew%';
```

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'
);
```

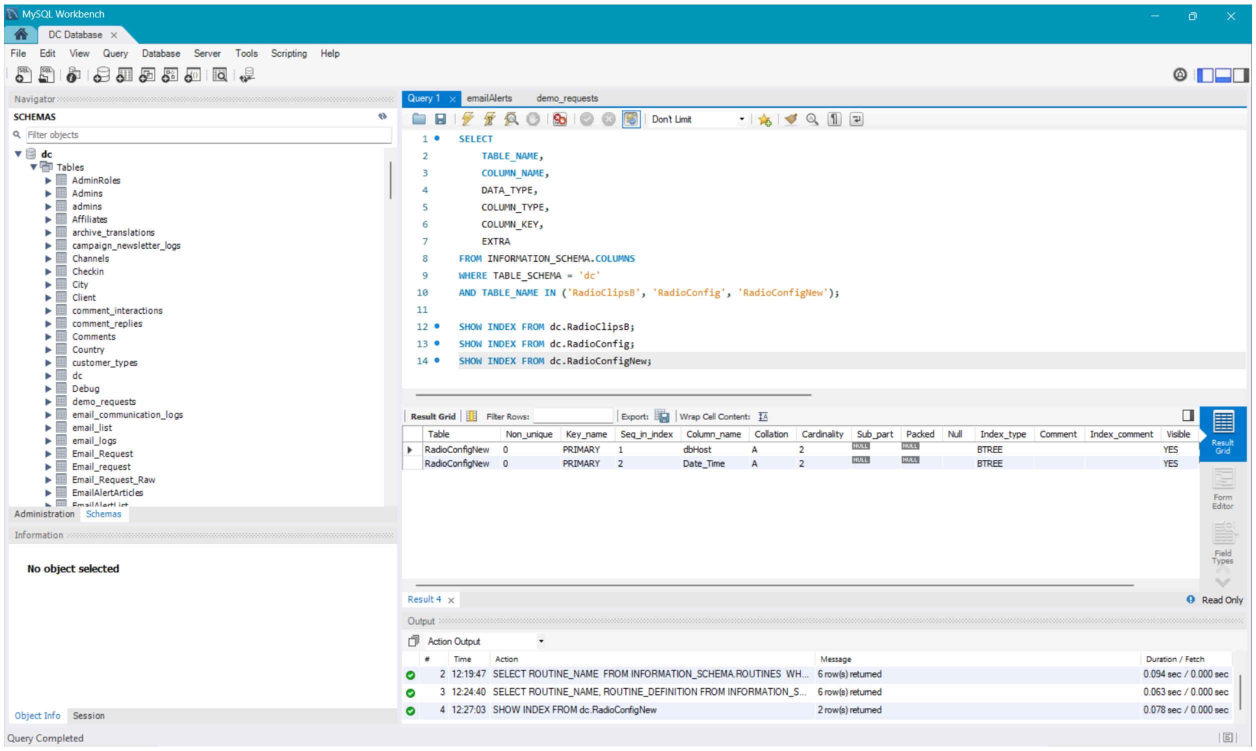
ROUTINE_NAME	ROUTINE_DEFINITION
Get_Radio_config	BEGIN SELECT * FROM RadioConfig OR...
Get_Radio_configNew	BEGIN SELECT * FROM RadioConfigNew ...
Insert_RadioB	BEGIN INSERT INTO dc.RadioClipsB (ID,FName,...
Insert_RadioConfig	BEGIN REPLACE INTO `dc`.`RadioConfig` (db...
Insert_RadioConfigNew	BEGIN REPLACE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

Query Completed

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

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.