A Lightweight SQL-Native Framework for Detecting and Preventing Exact Data Redundancy in Relational Databases

Noor Azilah Muda[1]*, Azah Kamilah Muda[2], Nunik Destria Arianti[3]

[1]Department of Applied Data Engineering, Faculty of Information and Communication Technology, Universiti Teknikal Malaysia Melaka, Durian Tunggal, Melaka, Malaysia

[2]Department of Intelligent Computing and Analytics, Faculty of Artificial Intelligence and Cyber Security, Universiti Teknikal Malaysia Melaka, Durian Tunggal, Melaka, Malaysia

[3] Department of Information System, Nusa Putra University, Sukabumi, Indonesia

***Corresponding Author**

DOI: https://doi.org/10.47772/IJRISS.2026.101000076

Received: 08 October 2026; Accepted: 13 October 2026; Published: 29 October 2026

ABSTRACT

Although relational databases are a crucial element of an organization's data storage, the occurrence of precise data duplication diminishes data integrity when primary keys are absent, uniqueness constraints are not upheld, or records are redundantly inserted during routine operations and migrations. Unnecessary rows occupy store capacity, distort aggregate reports, and if present in parent tables, they disseminate to related tables, resulting in cascading anomalies during insertion, updates, and deletions. Despite their efficacy, modern solutions such as entity resolution, machine-learning matchers, probabilistic filters, and secure cloud deduplication frequently necessitate external tools, specialized knowledge, or computational capabilities that exceed the requirements for schema auditing in educational and administrative environments. This research intends to create and assess a streamlined, transparent framework that identifies, mitigates, and prevents exact duplicate records with solely standard SQL constructs. The framework was organized into three phases: data preparation, SQL-driven detection and reduction, and evaluation with prevention, and it was applied to a Course Attendance System (CAS) database developed in MySQL. The reduction was accomplished through the use of SELECT DISTINCT and window functions, while single-table duplicates were detected via GROUP BY combined with HAVING COUNT (*), and duplicate relationships were established by multi-table JOIN aggregation. The framework identified redundant email addresses, repetitive staff records resulting from absent uniqueness constraints, duplicate agent entities within test tables, and cross-table redundancy chains in staff-equipment and attendance mappings. Every detection query took less than 0.01 seconds, in contrast to the 7.27 seconds required for a query-application hybrid method documented in previous studies, however the variation in datasets renders this comparison suggestive rather than rigorous. The findings indicate that normalization by itself fails to avert repeated insertions, but native SQL auditing, along with enforced constraints, serves as an effective initial safeguard for institutional data governance practices.

Keywords: Data redundancy, Duplicate record detection, SQL, Relational database integrity, Data governance

INTRODUCTION

Relational databases continue to be the dominant platform for managing structured records in universities, government organizations, and commercial companies due to the combination of a formal data model with declarative querying and integrity enforcement (Silberschatz et al., 2020). Despite decades of normalization theory maturation, one of the most persistent design defects remains, namely, data redundancy (Eessaar, 2016). Redundancy is when the same value or group of values is stored again and over again in the same table or in other tables. Consequences of these include extra storage usage, additional processing overhead, reduced query

performance, and most significantly, inconsistent information when one copy of a fact is updated and another is not (Lin, 2015; Talend, 2021).

Redundancy has become even more important as firms have become more reliant on cloud-hosted databases, multi-source data pipelines and high-volume transactional systems. In such situations, even a normalized schema on paper can have uncontrolled duplication if logical constraints like primary keys or unique indexes are not present, or are nullable or poorly enforced (Adil et al., 2019; Thor & Rahm, 2018). When duplicated parent records propagate into relationship tables, as often happens in attendance logs, inventory tracking and equipment allocation, the result is a chain of cascading anomalies that pollutes aggregate figures and yields misleading joins (Vighio et. al., 2018; Christophides et al., 2020).

More than technology is involved. Administrative data are increasingly used to support institutional decisions, performance reporting, and accountability, and the social sciences rely on them for official statistics and policy analysis (Binette & Steorts, 2022). Ehrlinger and Wöß (2022) state that data quality has moved from technical hygiene to a necessity for reliable analytics and data-driven decision making. One staff member may be doubly counted in a training or course attendance environment, which may overstate participation, skew compliance reporting on mandatory professional development, and misallocate resources. This makes preventing duplicate records as much a matter of organizational governance as database engineering.

Probabilistic filters, fuzzy clustering, blocking and filtering pipelines for entity resolution, deep-learning matchers based on pre-trained language models, and secure deduplication protocols for encrypted cloud storage have been developed by the research community. Recently, big foundation models have been shown to match entities and clean data without task-specific training (Narayan et al., 2022). These approaches can match near-duplicates at scale, but most are computationally intensive, opaque to developers, or require additional modules, training data, and expertise. In many educational, administrative, and auditing circumstances where exact duplication due to insufficient restrictions is the fundamental problem, these remedies are unnecessary. Thus, database designers, developers, and administrators require a lightweight, explainable, hardware-independent method.

That gap is addressed in this study. It builds on a student-led investigation to propose a SQL-native framework that uses only standard constructs (DISTINCT, GROUP BY, HAVING COUNT (*), JOIN) to detect and reduce exact duplicate records in a single table and across related tables and enforce constraints to prevent their recurrence. The framework prioritizes simplicity, transparency, and speed. Redundancy can be checked during schema design, data migration, and data-governance audits without additional software or compute engines. A MySQL-based Course Attendance System (CAS) database is used to showcase it.

1.1 Problem Statement

The relational model, while theoretically robust, does not inherently prevent exact replication without integrity constraints. This inquiry was prompted by five interrelated issues. Initially, the presence of redundant fields generates inconsistent data, which produces erroneous or nonsensical responses to the user's inquiries (Lithmee, 2018; Talend, 2021). Secondly, superfluous rows squander storage capacity and lead to anomalies in insertion, updating, and deletion, particularly in the absence of primary keys or when they are nullable (Palani et al., 2018; Vighio et al., 2018). Thirdly, the existence of redundant data across several schema locations and associations renders it unfeasible to obtain a singular authorized version of a record (Christophides et al., 2020). Fourth, redundancy in a primary table results in undesirable replication in tables having dependent relationships, such as attendance or equipment distributions, leading to cascading relational degradation (Lin, 2015). Fifth, the retention and reprocessing of duplicates squander CPU, memory, and I/O bandwidth that may be utilized for productive endeavors (Prajapati & Shah, 2022; Talend, 2021). Contemporary solutions primarily tackle these problems with near-duplicates or at the storage level, so creating a need for a clear, engine-integrated method to handle exact duplicates and their related relational consequences.

1.2 Research Objectives and Questions

The study pursues three objectives:

1. To analyze the origins of exact data redundancy in relational databases and to contrast the role of normalization with that of SQL-based detection.

2. To propose and implement a SQL-native framework for identifying, reducing and preventing exact duplicates within and across related tables.

3. To evaluate the framework in terms of detection correctness, execution time and its ability to expose cascading redundancy chains, thereby improving the reliability of aggregate query results.

Correspondingly, the study addresses the following research questions:

RQ1: What schema conditions allow exact duplicates to persist in a normalized relational database?

RQ2: To what extent can standard SQL constructs detect exact duplicates within single tables and across related tables?

RQ3: How efficient is SQL-native detection relative to a previously reported query–application hybrid approach?

1.3 Contributions and Organization

This study makes three contributions. First, it consolidates single-table detection, multi-table relationship auditing, safe reduction and constraint-based prevention into one reproducible SQL workflow. Second, it provides portable query templates that avoid a common downside in multi-table duplicate queries, namely selecting non-aggregated columns, which is rejected under the ONLY_FULL_GROUP_BY mode enabled by default in current MySQL releases (Oracle Corporation, 2024). Third, it offers empirical evidence from a real administrative database that exact duplication propagates across relationship tables, reinforcing the argument that normalization must be complemented by enforced uniqueness. The remainder of the paper is organized as follows. Section 2 reviews related literature, Section 3 describes the methodology and the framework, Section 4 presents and discusses the results, and Section 5 concludes with directions for future work.

LITERATURE REVIEW

2.1 Data Redundancy and Its Consequences

Data typically arrives in large and disorganized formats, and if not systematically arranged, it becomes challenging to use, oversee, and understand (Mueller & Massaron 2017). A database system offers organized assistance for the storage, retrieval, alteration, and administration of data as interrelated entities inside a logical framework (Sharma, 2019; Silberschatz et al., 2020). Data redundancy refers to the replication of values within rows or associated tables, typically occurring in designs that do not adhere rigorously to functional relationships (Eessaar, 2016; Naeem, 2021). Redundancy directly leads to traditional anomalies and inconsistencies in insertion, updating, and deletion, as a change made to one instance of a fact may not be mirrored in others (Talend, 2021; Vighio et al., 2018). In multi-system settings such as customer relationship management and order fulfillment, redundancy is amplified by the storage of duplicate entities. This is accomplished in autonomously managed databases (Dearmer, 2021). Recurrent insertion devoid of rational constraints results in inaccurate analytical outcomes, squandered I/O cycles, and an incessantly escalating storage expense (IBM Research, 2022; Lin, 2015).

2.2 Normalization Versus Enforced Uniqueness

Normalization disaggregates relations to eliminate redundancy caused by functional and multivalued interdependence. Eessaar (2016) posits that the concepts of normalization share similarities with the division of interests promoted in normalized systems theory. Normalization pertains to the configuration of the schema itself, rather than the individual entries it contains. A relation in third normal form may still contain two tuples exhibiting identical business characteristics if the sole key defined is a surrogate auto-increment identifier, as

the database engine recognizes them as distinct key values (Silberschatz et al., 2020). The prevention of duplicates is contingent upon the designation of the natural or business key as a PRIMARY KEY or UNIQUE constraint. This difference is frequently overlooked in both practical applications and educational settings: Miedema et al. (2021) demonstrated that novice SQL users harbor enduring misunderstandings about grouping and aggregation, the processes essential for revealing duplication. The current framework operates within the realm of structural accuracy and individual uniqueness.

2.3 Duplicate Detection and Entity Resolution

Duplicate detection represents a particular case of entity resolution (ER) focused on recognizing several data associated with the same real-world entity in the absence of unique identifiers (Christophides et al., 2020). Basic entity resolution evaluates every combination of data pairing, resulting in a quadratic increase in cost, while blocking and filtering techniques limit comparisons to pairs with a higher probability of matching (B.-H. Li et al., 2020; Papadakis et al., 2020). Binette and Steorts (2022) delineate the evolution of the discipline from first probabilistic record linkage to contemporary Bayesian and clustering techniques, emphasizing its significance for official statistics, human rights casualty assessment, and social scientific inquiry. Clustering or one-to-one matching algorithms ascertain which records to consolidate following the evaluation of candidate pairs. Papadakis et al. (2023) illustrate that the selection of algorithm significantly impacts precision. Previous comparative research conducted by Adil et al. (2019) examined the efficacy of several duplicate record detection techniques, while Selvi and Shanmuga Priyaa (2019) utilized fuzzy expectation-maximization clustering for near-duplicate values. These are potent, although typically operate externally to the database engine and strive for approximate rather than precise matches.

2.4 Learning-Based and AI-Assisted Data Cleaning

Entity matching was revolutionized by deep learning in 2020. Ditto proposed by Y. Li et al. (2020) significantly enhanced the matching quality on the benchmark datasets by fine-tuning the pre-trained Transformer language models. Peeters and Bizer (2021) improved the BERT-based matching by utilizing the dual-objective fine-tuning. In the work of Thirumuruganathan et al. (2021), deep learning was employed on the blocking stage itself. For more general data cleaning, Mahdavi and Abedjan (2020) suggested Baran, which integrates numerous error-correction algorithms by sharing a context representation and transfer learning. Narayan et al. (2022) further showed that big foundation models can do entity matching, error detection and imputation via prompting alone. Together these systems are good at semantic and fuzzy matching, but they need labeled data, specialist hardware or external services, and the decisions are difficult to audit or explain to database managers.

2.5 Storage-Level and Secure Deduplication

Another line of research looks at deduplication at the storage layer, where identical blocks or files are saved just once and accessed numerous times. This has to be harmonized with encryption in the cloud settings, resulting in convergent encryption, proof-of-ownership and associated techniques (Palani et al., 2018; Prajapati & Shah, 2022). Bloom-filter-based deduplication performs well in high-velocity insertion pipelines, but its probabilistic, hash-based detection logic is separated from the relational schema and hence provides little aid in validating design integrity (Adil et al., 2019; Palani et al., 2018). Storage deduplication also works below the logical level, two rows describing the same staff person under different surrogate keys will not be byte-identical and will not be merged.

2.6 Data Quality Tooling and the Research Gap

Ehrlinger and Wöß (2022) systematically identified 667 data quality tools and examined 13 in depth, finding that although data profiling and rule-based checks are widely available, generally applicable data quality metrics and automated monitoring remain unevenly implemented. Commercial tools also introduce licensing costs and a learning curve that may not be justified for small and medium institutional databases. Table 1 synthesizes the approaches reviewed. Two gaps emerge. First, few studies focus on exact duplicates arising from missing constraints, even though this is a common source of redundancy in operational and educational databases.

Second, little attention has been paid to how duplicates in parent tables cascade into relationship tables and how such chains can be detected using the database engine alone. The present study addresses both gaps.

Table 1. Comparison of redundancy mitigation approaches

Approach	Duplicate type addressed	Where it runs	Strengths	Limitations for schema auditing	Representative sources
Normalization and integrity constraints	Structural redundancy; exact duplicates only when keys are enforced	Schema / DBMS	Theoretically grounded; no runtime cost	Does not stop repeated inserts unless UNIQUE or PK constraints exist	Eessaar (2016); Silberschatz et al. (2020)
Blocking, filtering and classical ER	Exact and near-duplicates	External pipeline	Scales to large, heterogeneous data	Requires configuration and similarity thresholds	Christophides et al. (2020); Papadakis et al. (2020)
Fuzzy clustering	Near-duplicates	External model	Tolerates typographical variation	Additional computational model and tuning	Selvi & Shanmuga Priyaa (2019)
Deep learning and foundation models	Semantic and near-duplicates	GPU or external service	State-of-the-art matching accuracy	Needs labelled data or costly inference; limited explainability	Y. Li et al. (2020); Narayan et al. (2022)
Storage-level and secure deduplication	Identical blocks or files	Storage / cloud layer	Saves storage and bandwidth	Invisible to the logical schema; security trade-offs	Palani et al. (2018); Prajapati & Shah (2022)
Proposed SQL-native framework	Exact duplicates and duplicate relationships	Inside the DBMS	Transparent, fast, no additional software	Not designed for near-duplicates	This study

METHODOLOGY

3.1 Research Design

The study adopts a design-oriented, single-case approach in which an artefact, the SQL-native framework, is constructed and then evaluated in a real organizational database. The framework comprises three stages (Figure 1). Stage 1, data preparation, involves collecting the target records and profiling the schema to identify tables whose business keys are not protected by PRIMARY KEY or UNIQUE constraints. Stage 2, SQL detection and reduction, applies standard queries to reveal exact duplicates within tables and duplicate relationships across tables, and then removes redundant rows safely. Stage 3, evaluation and prevention, measures correctness and execution time, examines anomaly chains and introduces constraints that prevent recurrence. A feedback loop returns the process to Stage 1 whenever the schema changes or data are migrated.

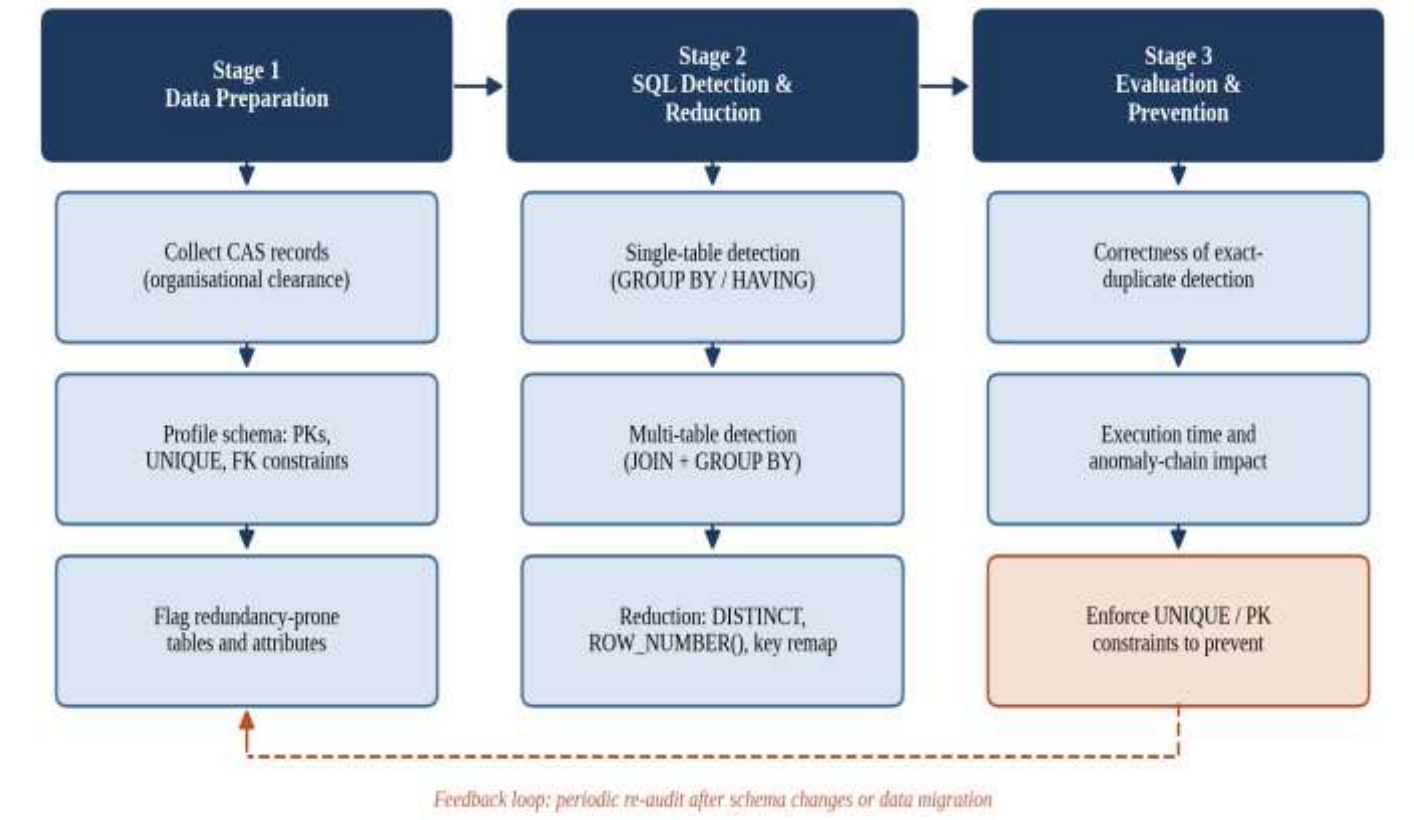

Figure 1. Three-stage SQL-native framework for detecting, reducing and preventing exact data redundancy

3.2 Case Study: Course Attendance System

The CAS database records staff participation in training courses organized by the host organization, together with related data on staff, courses, attendance and equipment allocated to staff. It was selected because attendance and allocation records are typical bridge (many-to-many) tables in which duplicated parent records propagate readily, and because redundancy auditing was a practical requirement for the organization. Unstructured staff attendance and course input records were obtained with organizational clearance; owing to confidentiality constraints, personally identifiable values are not reproduced in this paper. Figure 2 presents a simplified view of the relationships relevant to this study, using illustrative rather than actual values.

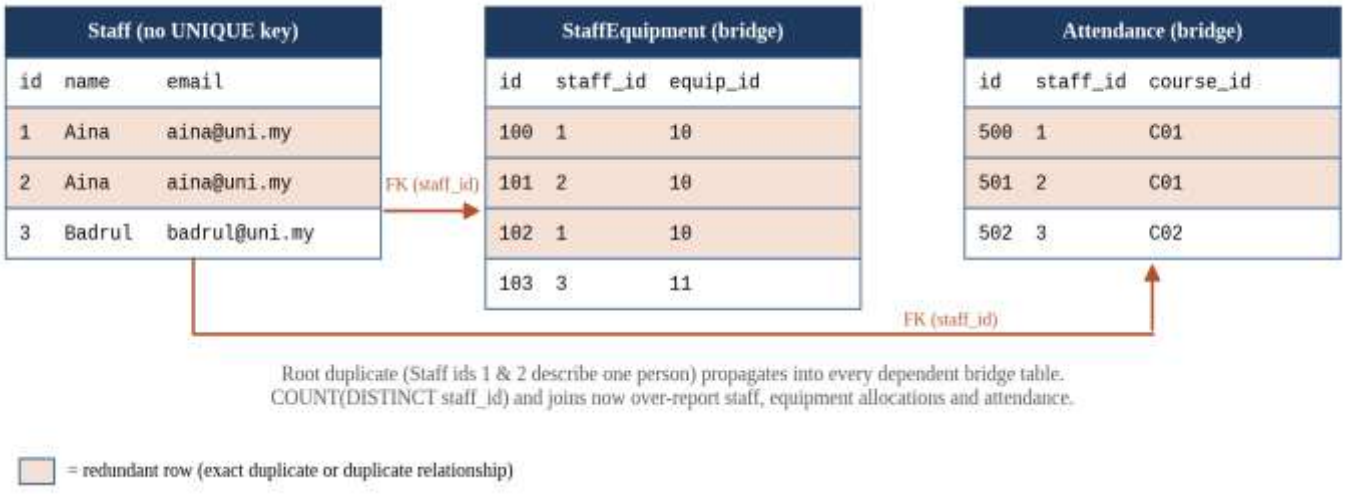

Figure 2. Simplified CAS relationships showing how a duplicated Staff record cascades into dependent bridge tables (illustrative values)

3.3 SQL-Based Detection, Reduction and Prevention Queries

The framework uses five query patterns. Patterns A to C correspond to the detection and reduction queries applied in the CAS experiments; Patterns D and E complete the framework with safe reduction in the presence of foreign keys and with constraint-based prevention. Table and column names follow the simplified schema in Figure 2 and can be substituted for any relational schema. The choice of business key is a design decision: in many institutions a staff number or e-mail address alone is the appropriate natural key, and the combination of name and e-mail used below is illustrative.

A. Single-table duplicate detection

Pattern A groups rows by the attributes that should identify an entity uniquely (the business key) and retains groups that occur more than once. Joining the result back to the base table returns every member of each duplicate group, so that an auditor can inspect the surrogate keys involved (Adil et al., 2019; Thor & Rahm, 2018).

```
SELECT s.*
FROM Staff s
JOIN (
    SELECT staff_name, email
    FROM Staff
    GROUP BY staff_name, email
    HAVING COUNT(*) > 1
) dup
  ON s.staff_name = dup.staff_name
 AND s.email      = dup.email
ORDER BY s.staff_name, s.staff_id;
```

B. Single-table reduction using Distinct

Pattern B creates a clean copy that contains one instance of each distinct combination of attribute values (Lithmee, 2018; Vighio et al., 2018). An important caveat is that `SELECT DISTINCT *` treats rows as duplicates only when every column, including any surrogate key, is identical. Where each row carries a unique auto-increment identifier, the key must be excluded and the business columns listed explicitly; otherwise no rows are removed.

```
-- Only when rows are identical in every column:

CREATE TABLE Staff_clean AS

SELECT DISTINCT * FROM Staff;



-- When a surrogate key exists, keep one identifier per business key:

CREATE TABLE Staff_clean AS

SELECT MIN(staff_id) AS staff_id, staff_name, email

FROM Staff

GROUP BY staff_name, email;
```

C. Multi-table redundant relationship detection

Pattern C audits bridge tables by counting how many times the same pair of related entities is linked. A common formulation selects all columns of the joined tables while grouping by a single identifier; under the ONLY_FULL_GROUP_BY SQL mode, enabled by default in current MySQL releases, such a query is rejected, and in permissive modes it returns arbitrary values from each group (Oracle Corporation, 2024). The form below instead groups by the relationship itself and reports its multiplicity (Christophides et al., 2020).

```
SELECT se.staff_id,     s.staff_name,

       se.equipment_id, e.equipment_name,

       COUNT (*) AS link_count

FROM StaffEquipment se

JOIN Staff s     ON s.staff_id     = se.staff_id

JOIN Equipment e ON e.equipment_id = se.equipment_id

GROUP BY se.staff_id, s.staff_name,

         se.equipment_id, e.equipment_name

HAVING COUNT(*) > 1;
```

Links that are distinct by key but identical in meaning, such as rows 100 and 101 in Figure 2, originate from duplicate parents. They become visible to Pattern C once duplicate parent identifiers are mapped to a single survivor, as in Pattern D.

D. Safe reduction with foreign-key remapping

Deleting a duplicate parent row that is still referenced by child tables either fails because of a foreign-key constraint or, if cascading deletes are configured, silently removes legitimate attendance history. Pattern D therefore first builds a mapping from each redundant identifier to a surviving identifier using window functions (available from MySQL 8.0), repoints child rows to the survivor and only then deletes the redundant parents. The procedure should be preceded by a backup and executed within a transaction. After remapping, formerly

hidden duplicate relationships become exact duplicates that Pattern C detects and that can be removed in the same manner.

```
START TRANSACTION;



CREATE TEMPORARY TABLE staff_map AS

SELECT staff_id AS dup_id, keep_id

FROM (

    SELECT staff_id,

           MIN (staff_id) OVER (PARTITION BY staff_name, email) AS keep_id,

           ROW_NUMBER () OVER (PARTITION BY staff_name, email

                               ORDER BY staff_id) AS rn

    FROM Staff

) t

WHERE rn > 1;



UPDATE StaffEquipment se

JOIN staff_map m ON se.staff_id = m.dup_id

SET se.staff_id = m.keep_id;

-- Repeat the UPDATE for Attendance and any other child table



DELETE s FROM Staff s

JOIN staff_map m ON s.staff_id = m.dup_id;



COMMIT;
```

E. Prevention through enforced uniqueness

Detection and reduction treat the symptom; prevention treats the cause. Once a table has been cleaned, declaring its business key as UNIQUE causes the engine to reject any subsequent duplicate insert, and a composite UNIQUE constraint on each bridge table prevents the same relationship from being recorded twice (Silberschatz et al., 2020). Where a course runs over several sessions, the session date should be added to the attendance key.

```
ALTER TABLE Staff
  ADD CONSTRAINT uq_staff_identity UNIQUE (staff_name, email);


ALTER TABLE StaffEquipment
  ADD CONSTRAINT uq_staff_equipment UNIQUE (staff_id, equipment_id);


ALTER TABLE Attendance
  ADD CONSTRAINT uq_attendance UNIQUE (staff_id, course_id);
```

Figure 3 summarizes how the five patterns combine into a decision flow for each table under audit.

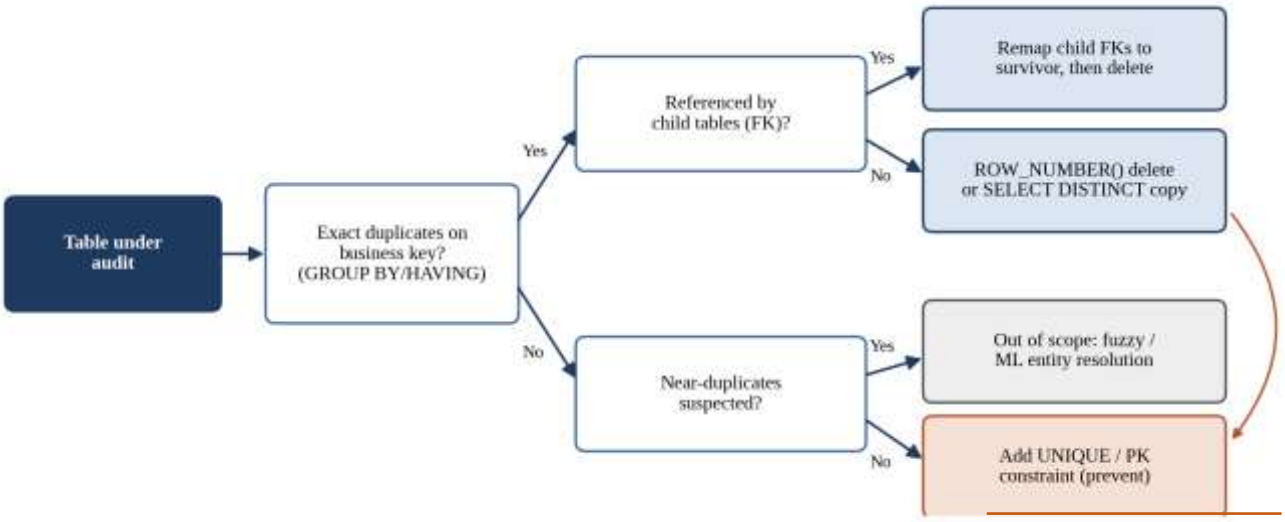

Figure 3. Decision flow for resolving and preventing detected exact duplicates

3.4 Evaluation Criteria

The framework was evaluated against three criteria aligned with the research objectives: (i) detection correctness, defined as returning all and only the exact duplicates present, verified by manual inspection; (ii) efficiency, measured as query execution time; and (iii) relational impact, assessed by whether the queries exposed duplicate chains across relationship tables.

RESULTS AND DISCUSSION

4.1 Detection Outcomes

Applying the pattern A to C to the CAS database revealed four categories of exact redundancy, summarized in Table 2.

Table 2. Categories of exact redundancy detected in the CAS database

Redundancy category	Location	Root cause	Consequence	Detected by
Duplicate e-mail addresses	Contact and attendance personnel records	No UNIQUE constraint on e-mail; repeated registration	Inflated head-counts; ambiguous identity	Pattern A
Duplicate staff records	Staff table	Business key not declared; surrogate key only	Update and deletion anomalies	Pattern A
Repeated agent entities	Marketing / client test tables	Repeated test inserts	Risk of test data propagating into production	Patterns A and B
Cross-table cascades	StaffEquipment and attendance mapping tables	Duplicate parent rows referenced by child rows	Double-counted allocations and attendance; misleading joins	Pattern C

Duplicate email addresses were detected in contact and attendance personnel records. An email address is usually unique to a person hence repetition suggested repeated registration or manual re-entry without validation. Duplicate staff records were present in the Staff table as no uniqueness constraint had been specified on the staff identity or the combination of staff and course. Thus, each resubmission of the identical form resulted in a new row with a new surrogate key. The marketing and client tables used for insert testing had repeated agent entities, demonstrating how redundancy generated in test data might seep into production if it is not verified.

Most importantly, cascades over tables were seen. Staff entries had been duplicated and copied into connection tables such as StaffEquipment and the attendance mapping table, so that one person seemed to have the same equipment twice or to have attended the same course twice. These chains are the relational analog of the update anomaly: fixing the staff record in one place leads to duplicate or conflicting references elsewhere, and any report counting allocations or attendances through a join is quietly bloated.

4.2 Performance Evaluation

All detection queries returned their results in under 0.01 seconds, which is effectively instantaneous for interactive auditing. Table 3 and Figure 4 contrast this with the 7.27 seconds reported for a query–application hybrid detection method in prior work (Koushki et al., 2025). A difference of more than two orders of magnitude is consistent with the expectation that grouping and index- or hash-based aggregation executed inside the DBMS avoids the cost of transferring rows to an application layer and comparing them there (Silberschatz et al., 2020). However, the two figures were obtained on different datasets, hardware and software versions, so the comparison should be interpreted as indicative rather than as a controlled benchmark.

Table 3. Execution time of duplicate detection approaches

Method	Execution environment	Time taken
Query–application hybrid detection (prior work)	Application layer with DBMS	7.27 seconds
Proposed SQL-native framework (CAS, this study)	Inside MySQL	< 0.01 seconds per query

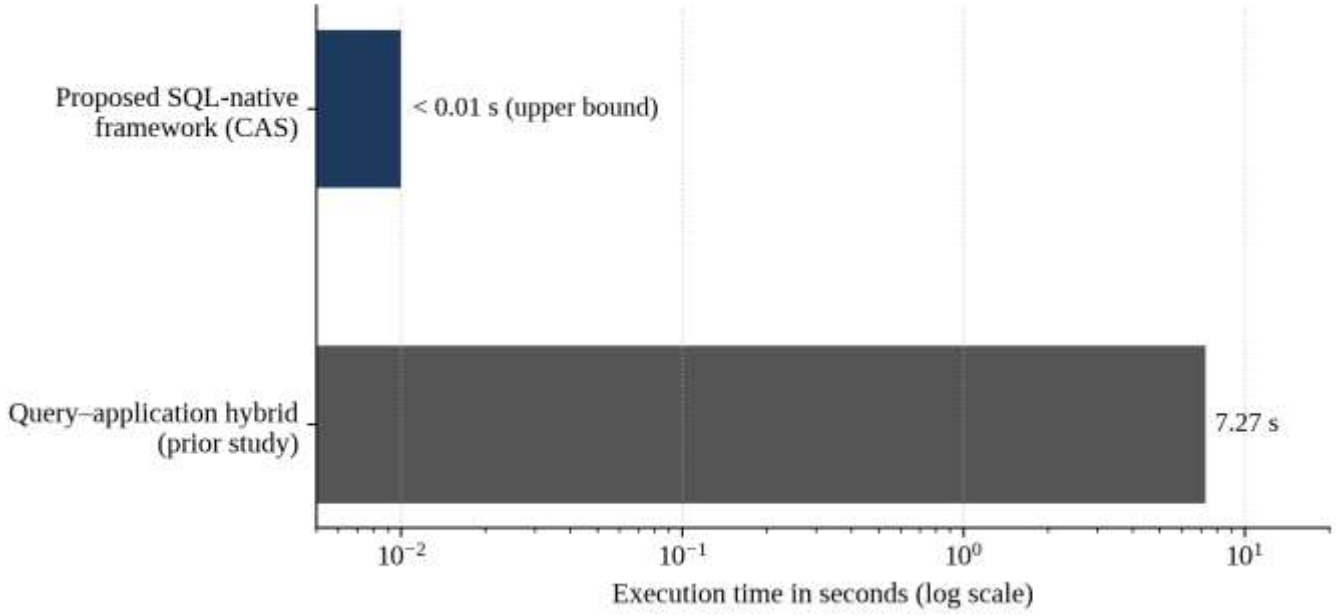

Figure 4. Execution time of SQL-native detection versus a previously reported query–application hybrid approach (logarithmic scale; datasets differ)

DISCUSSION

The CAS findings answer RQ1 directly: every category of redundancy traced back to a business key that had not been declared as a PRIMARY KEY or UNIQUE constraint. This confirms that normalization ensures correct structure but does not by itself prevent repeated inserts (Eessaar, 2016; Silberschatz et al., 2020), and it echoes the observation that duplicate entity storage is amplified wherever data enter through multiple uncontrolled channels (Dearmer, 2021).

With respect to RQ2, grouping-based queries detected the exact duplicates within single tables, and relationship-level grouping exposed duplicate chains across bridge tables. Exact-match detection is deterministic: two rows either share identical business-key values or they do not, so the precision–recall trade-offs that dominate the ER literature (Binette & Steorts, 2022; Papadakis et al., 2023) do not arise for this class of duplicates. The same determinism, however, marks the framework's principal boundary. Depending on the column collation, values that differ only in letter case or trailing whitespace may or may not be grouped together; applying `TRIM()` and `LOWER()` before grouping makes this behavior explicit. Genuine spelling variants and abbreviations require fuzzy or learning-based matching (Selvi & Shanmuga Priyaa, 2019; Y. Li et al., 2020).

Regarding RQ3, the sub-0.01-second response times indicate that native query optimizers are highly effective for exact-duplicate retrieval, particularly when grouped columns are indexed (Adil et al., 2019; IBM Research, 2022). For databases of the size typical of institutional systems, detection can therefore be run interactively during design reviews or scheduled routinely without measurable impact on operations. Performance at much larger scales, where GROUP BY may require temporary tables and external sorting, remains to be verified.

Positioned against the wider literature, the framework is not a competitor to entity resolution systems but a first, low-cost filter. Removing exact duplicates before fuzzy or learning-based matching reduces the number of candidate's pairs those systems must evaluate, complementing blocking strategies (Papadakis et al., 2020; Thirumuruganathan et al., 2021) and lowering the cost of foundation-model-based cleaning (Narayan et al., 2022).

4.4 Implications

Theoretically, the study reinforces the distinction between schema-level normalization and instance-level uniqueness, and it shows that redundancy should be analyzed as a relational phenomenon whose effects

propagate along foreign-key paths rather than as an isolated property of a table. Practically, the query templates give database designers, developers and auditors a transparent checklist that can be executed before go-live, after data migration and as part of scheduled data-quality monitoring, addressing part of the automation gap identified by Ehrlinger and Wöß (2022). Organizationally, accurate attendance and allocation records underpin compliance reporting, resource planning and accountability, so embedding SQL-based redundancy audits into data-governance routines supports evidence-based institutional decision-making. Pedagogically, the framework offers concrete and meaningful exercises in GROUP BY and HAVING, operations that novices find conceptually difficult (Miedema et al., 2021).

4.5 Limitations

Several limitations should be acknowledged. The evaluation was conducted on a single institutional database of modest size in one DBMS, which limits generalizability. The performance comparison relies on a figure reported in a different study rather than on a controlled experiment using identical data. The framework detects exact duplicates only and does not address typographical variants, abbreviations or semantically equivalent records. Finally, choosing the correct business key requires domain knowledge: an inappropriate key may either miss duplicates or wrongly flag legitimate repeated values, such as two different staff members who share a common name.

CONCLUSION AND FUTURE WORK

This study set out to provide a lightweight, transparent and engine-native approach to exact data redundancy in relational databases. Using a Course Attendance System in MySQL, it showed that a small set of standard SQL constructs can reliably detect exact duplicates within tables, expose duplicate relationships that cascade into bridge tables and support safe reduction, all with near-instantaneous execution. The results confirm that normalization alone does not prevent repeated inserts; redundancy is controlled only when business keys are enforced through PRIMARY KEY and UNIQUE constraints. The framework is therefore best understood as a foundational auditing layer for database designers, administrators and educators, one that complements rather than replaces fuzzy, learning-based and storage-level deduplication. By making redundancy visible and preventable with tools already present in every relational DBMS, the approach contributes to more trustworthy institutional data and, in turn, to better-informed organizational decisions.

Future research should pursue five directions:

1. Extend detection to near-duplicates by integrating similarity measures such as Levenshtein or Jaro–Winkler distance through SQL user-defined functions.

2. Develop hybrid pipelines in which SQL removes exact duplicates first and machine-learning or foundation-model-based record linkage is applied only to the remaining candidates.

3. Conduct controlled cross-platform benchmarks on PostgreSQL, MariaDB, Oracle and SQL Server using identical synthetic datasets of increasing size to characterize query-optimizer behaviour at scale.

4. Package the queries as scheduled audit procedures that generate periodic redundancy reports for post-migration data governance.

5. Evaluate the framework as a teaching instrument in introductory database courses.

Declarations

Use of AI Technology: The author used Claude and Gemini to assist with drafting, structural editing, reference verification, and language revision during the preparation of this manuscript. All content, arguments, citations,

and conclusions were reviewed and verified by the author. The author takes full responsibility for the integrity of the work.

Conflicts of Interest: The author declares no conflicts of interest.

Funding: This research received no external funding.

Human Participants: This research did not involve human participants. No informed consent declaration is required.

REFERENCES

1. Adil, S. H., Ali, S. S. A., Raza, K., & Ebrahim, M. (2019). Performance analysis of duplicate record detection techniques. Engineering, Technology & Applied Science Research, 9(5), 4755–4758.
2. Binette, O., & Steorts, R. C. (2022). (Almost) all of entity resolution. Science Advances, 8(12), eabi8021. https://doi.org/10.1126/sciadv.abi8021
3. Christophides, V., Efthymiou, V., Palpanas, T., Papadakis, G., & Stefanidis, K. (2020). An overview of end-to-end entity resolution for big data. ACM Computing Surveys, 53(6), Article 127. https://doi.org/10.1145/3418896
4. Dearmer, A. (2021). Data redundancy in organizations. Xplenty.
5. Eessaar, E. (2016). The database normalization theory and the theory of normalized systems: Finding a common ground. Baltic Journal of Modern Computing, 4(1), 5–33.
6. Ehrlinger, L., & Wöß, W. (2022). A survey of data quality measurement and monitoring tools. Frontiers in Big Data, 5, 850611. https://doi.org/10.3389/fdata.2022.850611
7. IBM Research. (2022). Storage optimization using RDBMS deduplication. IBM.
8. Li, B.-H., Liu, Y., Zhang, A.-M., Wang, W.-H., & Wan, S. (2020). A survey on blocking technology of entity resolution. Journal of Computer Science and Technology, 35, 769–793.
9. Li, Y., Li, J., Suhara, Y., Doan, A., & Tan, W.-C. (2020). Deep entity matching with pre-trained language models. Proceedings of the VLDB Endowment, 14(1), 50–60. https://doi.org/10.14778/3421424.3421431
10. Lin, D. (2015). Efficient data quality and redundancy elimination strategies for large scale relational databases. IEEE Data Engineering Bulletin.
11. Lithmee, B. (2018). Difference between data corruption and data redundancy. Pediaa.com.
12. Mahdavi, M., & Abedjan, Z. (2020). Baran: Effective error correction via a unified context representation and transfer learning. Proceedings of the VLDB Endowment, 13(11), 1948–1961. https://doi.org/10.14778/3407790.3407801
13. Miedema, D., Aivaloglou, E., & Fletcher, G. (2021). Identifying SQL misconceptions of novices: Findings from a think-aloud study. In Proceedings of the 17th ACM Conference on International Computing Education Research (pp. 355–367). Association for Computing Machinery. https://doi.org/10.1145/3446871.3469759
14. Mueller, J. P., & Massaron, L. (2017). Algorithms for dummies. John Wiley & Sons.
15. Naeem, T. (2021). Database design techniques. Astera Software.
16. Narayan, A., Chami, I., Orr, L., & Ré, C. (2022). Can foundation models wrangle your data? Proceedings of the VLDB Endowment, 16(4), 738–746. https://doi.org/10.14778/3574245.3574258
17. Oracle Corporation. (2024). MySQL 8.0 reference manual. https://dev.mysql.com/doc/refman/8.0/en/
18. Palani, S., Reddy, P., & Roja, K. (2018). Secure cloud deduplication. International Journal of Scientific Research in Computer Science, Engineering and Information Technology.
19. Papadakis, G., Efthymiou, V., Thanos, E., Hassanzadeh, O., & Christen, P. (2023). An analysis of one-to-one matching algorithms for entity resolution. The VLDB Journal, 32(6), 1369–1400.
20. Papadakis, G., Skoutas, D., Thanos, E., & Palpanas, T. (2020). Blocking and filtering techniques for entity resolution: A survey. ACM Computing Surveys, 53(2), Article 31. https://doi.org/10.1145/3377455
21. Peeters, R., & Bizer, C. (2021). Dual-objective fine-tuning of BERT for entity matching. Proceedings of the VLDB Endowment, 14(10), 1913–1921.

22. Prajapati, P., & Shah, P. (2022). A review on secure data deduplication: Cloud storage security issue. Journal of King Saud University – Computer and Information Sciences, 34(7), 3996–4007.
23. Selvi, P., & Shanmuga Priyaa, D. (2019). Fuzzy-EM clustering deduplication. International Journal of Recent Technology and Engineering.
24. Sharma, O. (2019). Database fundamentals. C# Corner.
25. Silberschatz, A., Korth, H. F., & Sudarshan, S. (2020). Database system concepts (7th ed.). McGraw-Hill Education.
26. Talend. (2021). Drawbacks of data redundancy. Talend Resources.
27. Thirumuruganathan, S., Li, H., Tang, N., Ouzzani, M., Govind, Y., Paulsen, D., Fung, G., & Doan, A. (2021). Deep learning for blocking in entity matching: A design space exploration. Proceedings of the VLDB Endowment, 14(11), 2459–2472.
28. Thor, A., & Rahm, E. (2018). Duplicate detection in relational databases: A survey. Datenbank-Spektrum.
29. Vighio, M., Khanzada, T., & Kumar, M. (2018). Query normalization and redundancy elimination tool. Sindh University Research Journal.
30. Mohammadi Koushki, N., El-Shekeil, I., & Kant, K. (2025). ConfExp: Root-Cause Analysis of Service Misconfigurations in Enterprise Systems. Journal of Network and Systems Management, 33(1).