Journal of Electrical Engineering and Computer (JEECOM)
Vol. 8, No. 2 (2026), DOI: 10.33650/jeecom.v8i2.17145
p-ISSN: 2715-0410; e-ISSN: 2715-6427

Universal Metadata Dictionary: A Platform-Agnostic Metadata Representation Model for Cross-RDBMS Schema Drift Detection

Putu Adi Guna Permana[1*], I Made Sukarsa[2], I Ketut Gede Darma Putra[3], I Made Suwija Putra[4]
[1] Department of Information Systems, Faculty of Informatics and Computer, Institut Teknologi dan Bisnis STIKOM Bali, Bali, Indonesia
[1] Doctor of Engineering Program, Faculty of Engineering, Udayana University, Bali, Indonesia; [2,3,4] Department of Information Technology, Faculty of Engineering, Udayana University, Bali, Indonesia

Article Info

Article history:
Received August 27, 2026
Revised September 13, 2026
Accepted Oktober 10, 2026

Keywords:
Universal Metadata Dictionary; metadata normalisation; schema drift detection; multi-RDBMS; cross-platform database governance; Design Science Research; Tibero; hash-based change detection

ABSTRACT
This paper addresses the research objective: to design, specify, and validate a platform-agnostic metadata representation model that normalises database schema metadata from heterogeneous Relational Database Management System (RDBMS) environments into a unified structure sufficient for automated cross-RDBMS structural schema drift detection. Schema drift the unintended divergence of database schema structures between deployment environments or between the database schema and application source code is a leading contributor to deployment failures in enterprise information systems. The proposed Universal Metadata Dictionary (UMD) normalises database schema metadata from six widely deployed RDBMS platforms Oracle, Tibero, SQL Server, PostgreSQL, MySQL, and SQLite into a unified 29-element relational structure governed by four design principles: platform independence, completeness, normalisation, and hashability. A five-stage normalisation pipeline (extraction, parsing, mapping, enrichment, and storage) and a seven-category data type mapping constitute the core implementation specification, eliminating false drift reports arising from platform-specific type nomenclature. The UMD was validated through: (1) expert review by 13 domain practitioners using a five-point Likert scale, yielding composite mean scores of 4.52–4.70 out of 5 across five criteria; and (2) structural completeness assessment via direct query execution on all six target platforms, confirming that all 29 elements are extractable through JDBC-accessible metadata interfaces and system catalogues. An empirical evaluation protocol comprising 15 structured fault injection test scenarios is defined for subsequent implementation-phase validation. Three specific contributions distinguish the UMD from all existing approaches: a cross-RDBMS metadata normalisation model with no comparable published equivalent covering all six platforms simultaneously, including Tibero; a per-object O(1) hash comparison mechanism enabling O(n) total Phase-1 change detection; and a metadata model directly supporting AI-based Semantic Code-Schema Analysis using transformer embeddings.

This is an open access article under the CC BY-SA license.

Corresponding Author:
Putu Adi Guna Permana,
Doctor of Engineering Program, Faculty of Engineering, Udayana University, Bali, Indonesia
Email: adiguna.permana@gmail.com; Orcid ID : 0009-0001-5884-7939 :

1. INTRODUCTION

Heterogeneous DBMS platforms expose schema metadata through different metadata interfaces and representations, making unified automated schema drift detection across multiple RDBMS environments a fundamental challenge for enterprise information systems. In large Indonesian financial institutions and government organisations, commercial platforms—Oracle, Microsoft SQL Server, and Tibero—routinely coexist alongside open-source systems—PostgreSQL, MySQL, and SQLite—to address diverse operational, performance, and regulatory requirements [1], [23]. This architectural diversity intensifies the data governance challenge of schema drift.

Schema drift is defined as the unintended divergence of database schema structures between deployment environments (development → system integration testing → user acceptance testing → production) or between the database schema and application source code [12]. Studies on multi-environment enterprise database systems report that schema drift is a leading contributor to deployment failures in multi-environment database systems [2], [11]. Unresolved schema drift causes query execution errors, transaction failures, data integrity violations, and costly production rollbacks.

A fundamental obstacle to automated schema drift detection in multi-RDBMS environments is metadata heterogeneity. Each RDBMS exposes structural schema information through a proprietary system catalogue with distinct object naming, type nomenclature, and constraint representation. Oracle and Tibero use ALL_TABLES, ALL_COLUMNS, ALL_CONSTRAINTS, and DBA_OBJECTS; SQL Server uses sys.tables, sys.columns, and

ISSN: 2715-6427

sys.indexes; PostgreSQL employs pg_catalog and INFORMATION_SCHEMA; MySQL provides INFORMATION_SCHEMA; and SQLite exposes sqlite_master and PRAGMA directives [3]. These representations cannot be directly compared without an intermediate normalisation layer, and no existing tool provides such a layer across all six platforms simultaneously.

Existing approaches fail to address this problem comprehensively. Migration management tools such as Liquibase and Flyway track schema change history through version-controlled scripts but are blind to out-of-band structural modifications [12]. Commercial comparison tools such as RedGate SQL Compare are proprietary and limited to SQL Server. Research frameworks such as Darwin [4] and MoDEvo [2] target NoSQL and model-driven environments, respectively, and neither provides cross-RDBMS metadata normalisation. No existing tool supports Tibero, the RDBMS platform increasingly adopted in Indonesian banking systems and government enterprise projects [1], [13].

This paper therefore addresses the research objective: to design, specify, and validate a platform-agnostic metadata representation model that normalises database schema metadata from heterogeneous RDBMS environments into a unified structure sufficient for automated structural schema drift detection. This paper proposes the Universal Metadata Dictionary (UMD)—a platform-agnostic metadata representation model designed and specified using Design Science Research (DSR) methodology [15], constituting the foundational component of the broader Adaptive Schema Drift Management Framework. Three specific novelties distinguish the UMD from all prior work: (1) a unified 29-element metadata representation spanning six RDBMS platforms including Tibero; (2) a per-object O(1) hash comparison mechanism enabling O(n) total Phase-1 schema comparison; and (3) a metadata model directly supporting AI-based Semantic Code-Schema Analysis using CodeBERT and Sentence-BERT [16], [17]. The remainder of this paper is organised as follows: Section 2 presents the research method; Section 3 presents the results and discussion; and Section 4 concludes the paper.

2. METHOD

2.1. Related Work

Schema evolution, the systematic modification of database structures to accommodate changing requirements while preserving data integrity has been studied extensively [6], [7]. Brahmia et al. [7] conduct a systematic literature review of 63 publications on schema evolution, concluding that the majority of approaches remain oriented towards data migration and schema versioning rather than automated pre-deployment validation. Suárez-Otero et al. [2] develop MoDEvo for column-family NoSQL databases (Cassandra and HBase), demonstrating automated schema change detection and migration generation. Störl and Klettke [4] address schema drift in NoSQL environments through the Darwin platform. Both frameworks offer architecturally relevant precedents; however, they are inapplicable to relational RDBMS environments and neither addresses Tibero. Chillon et al. [8] propose a taxonomy of schema changes for NoSQL databases but do not address cross-RDBMS metadata normalisation.

Gungor et al. [3] address schema matching via multi-stage recommendation and metadata enrichment using large language models (Schemora), confirming that metadata-driven approaches can systematically extract and match schema information across heterogeneous data sources. The interaction between schema matching and record matching underscores the importance of a normalised metadata representation as the foundation for reliable cross-source data integration [9].Paganelli et al. [18] demonstrate that transformer-based embeddings capture semantic relationships between structurally different but semantically equivalent attributes directly validating the AI semantic matching component of the broader framework. Zhang et al. [17] develop schema matching using pre-trained language models, demonstrating significant improvements in precision and recall over rule-based approaches. Work on unified multi-model querying [5] further demonstrates the feasibility of platform-agnostic data access as a prerequisite for systematic cross-platform comparison. Belefqih et al. [21] demonstrate schema extraction in NoSQL databases using Sentence-BERT embeddings, validating that transformer-based semantic analysis can be applied to metadata structures across heterogeneous database platforms.

Sirimalla [12] surveys end-to-end automation for cross-database DevOps deployments, demonstrating that Liquibase and Flyway are the dominant tools but their script-based drift detection misses out-of-band structural changes. Malhotra et al. [19] demonstrate that achieving high availability in cloud-native enterprise applications requires rigorous schema governance as part of deployment pipelines; the subsequent work [20] extends this to canary deployment using Kubernetes and Liquibase, confirming that schema management is a critical DevOps integration point—but multi-RDBMS heterogeneity remains an unresolved challenge. The management of metadata in large-scale data environments, including the use of Java annotations for intelligent data lake access [22], demonstrates the broader need for systematic, programmable metadata normalisation frameworks across heterogeneous storage platforms. A review of existing tools and frameworks reveals four compounding research gaps: (1) no published metadata In enterprise legacy system migration contexts, schema-level metadata analysis is similarly essential for identifying service boundaries and semantic clusters [14].normalisation model covers all six target platforms in a unified structure; (2) no existing approach covers the full range of database object types—routines, triggers, sequences, and views; (3) no existing framework includes Tibero; and (4) no framework integrates both structural drift detection and AI-based Semantic Code-Schema Analysis in a single portable, on-premise tool.

2.2. UMD Design Methodology

This paper employs Design Science Research (DSR) methodology [15] to develop and evaluate the UMD as a technical artefact. DSR prescribes a cycle of problem identification, artefact design, demonstration, and evaluation a methodology appropriate for proposing a new model specification and evaluating it through expert review and completeness assessment. The UMD constitutes the 'Design and Development' phase output of the DSR cycle for the Adaptive Schema Drift Management Framework research programme. The research follows six DSR phases: (1) Problem Identification—metadata heterogeneity prevents cross-RDBMS schema drift detection; (2) Objectives—define UMD specification requirements; (3) Design and Development—specify the 29-element UMD relational schema, normalisation pipeline, and data type mapping; (4) Demonstration—structural completeness assessment via

direct query execution on all six RDBMS platforms; (5) Evaluation—expert review by 13 practitioners and definition of the 15-scenario empirical evaluation protocol; (6) Communication—publication of the UMD specification as a reusable artefact.

2.3. UMD Formal Specification

The Universal Metadata Dictionary (UMD) is defined as a structured, platform-agnostic relational data model that represents the complete structural schema information of a relational database in a normalised, platform-independent format. Four design objectives govern the specification: (1) Platform independence: every element must be derivable from system catalogues available on all six target platforms without requiring elevated privileges. (2) Completeness: the dictionary covers all database object categories material to structural schema drift, including routines, triggers, and sequences systematically omitted by existing tools. (3) Normalisation: platform-specific values particularly data types are canonicalised before storage. (4) Hashability: the model supports deterministic SHA-256 hash computation enabling per-object O(1) comparison in Phase-1 detection.

The UMD is defined as a single relational table with a composite primary key comprising (database_name, rdbms_type, environment_name, schema_name, object_type, table_name, column_name). For non-column objects (views, stored procedures, triggers, sequences, and indices), column_name is stored as an empty string sentinel value ('') rather than SQL NULL, preserving standard primary key non-nullability. The 29-element structure results from a systematic enumeration of all database object categories and their structurally significant attributes across the six target RDBMS platforms. Each element was included only if its absence would cause at least one category of schema drift to go undetected on at least one platform. Table 1 presents the formal relational schema with SQL data types, nullable constraints, data sources, roles, and domain constraints for all 29 attributes.

Table 1. Formal Relational Schema of the Universal Metadata Dictionary

Attribute	SQL Type	Nullable	Source	Role	Constraints / Notes
database_name	VARCHAR(255)	NOT NULL	Connection config	Part of composite PK	Uniquely identifies the source database instance
rdbms_type	VARCHAR(50)	NOT NULL	JDBC metadata	Part of composite PK	Constrains: {Oracle, Tibero, SQL_Server, PostgreSQL, MySQL, SQLite}
environment_name	VARCHAR(50)	NOT NULL	User config	Part of composite PK	Constrains: {DEV, SIT, UAT, PROD} or user-defined
schema_name	VARCHAR(255)	NOT NULL	System catalogue	Part of composite PK	Owner/schema name; 'public' for PostgreSQL, 'dbo' for SQL Server
object_type	VARCHAR(50)	NOT NULL	System catalogue	Part of composite PK	Constrains: {TABLE, VIEW, PROCEDURE, FUNCTION, TRIGGER, SEQUENCE, INDEX}
table_name	VARCHAR(255)	NOT NULL	System catalogue	Part of composite PK	Object name; functions as table name for TABLE type
column_name	VARCHAR(255)	NOT NULL	System catalogue	Part of composite PK	Empty string '' sentinel for non-column objects; never SQL NULL
raw_data_type	VARCHAR(255)	NULLABLE	Column metadata	Attribute	Platform-specific type string before normalisation
normalized_data_type	VARCHAR(50)	NULLABLE	Computed Stage 3	Attribute	Constrains: {INTEGER, DECIMAL, STRING, DATE_TIME, BOOLEAN, CLOB, BLOB}
length	INTEGER	NULLABLE	Column metadata	Attribute	-1 = unlimited (CLOB/BLOB); NULL for non-column objects
precision_value	INTEGER	NULLABLE	Column metadata	Attribute	NULL for non-numeric types
scale_value	INTEGER	NULLABLE	Column metadata	Attribute	NULL for non-numeric types
nullable_status	VARCHAR(10)	NULLABLE	Col/constraint meta	Attribute	Constrains: {NULLABLE, NOT_NULL}
default_value	TEXT	NULLABLE	Column metadata	Attribute	NULL if no default defined
primary_key_flag	CHAR(1)	NULLABLE	Constraint metadata	Attribute	Constrains: {Y, N}
foreign_key_reference	TEXT	NULLABLE	Constraint metadata	Attribute	Format: column_name → ref_table.ref_column
unique_constraint	VARCHAR(255)	NULLABLE	Constraint metadata	Attribute	Constraint name; NULL if no unique constraint
check_constraint	TEXT	NULLABLE	Constraint metadata	Attribute	Constraint definition SQL fragment
index_name	VARCHAR(255)	NULLABLE	Index metadata	Attribute	NULL for non-index records
index_type	VARCHAR(50)	NULLABLE	Index metadata	Attribute	Constrains: {NORMAL, UNIQUE, BTREE, BITMAP, HASH}
view_definition_hash	CHAR(64)	NULLABLE	Computed Stage 4 (SHA-256)	Hash integrity	NULL for non-view objects; 64-character hexadecimal string
sequence_name	VARCHAR(255)	NULLABLE	Sequence metadata	Attribute	NULL for non-sequence objects
sequence_properties	TEXT	NULLABLE	Sequence metadata	Attribute	JSON: {increment, min_value, max_value, cycle}
routine_name	VARCHAR(255)	NULLABLE	Routine catalogue	Attribute	NULL for non-routine objects
routine_signature	TEXT	NULLABLE	Routine metadata	Attribute	Parameter list and return type

ISSN: 2715-6427

routine_body_hash	CHAR(64)	NULLABLE	Computed Stage 4 (SHA-256)	Hash integrity	NULL for non-routine objects or SQLite; 64-char hex string
trigger_name	VARCHAR(255)	NULLABLE	Trigger metadata	Attribute	NULL for non-trigger objects
trigger_event	VARCHAR(255)	NULLABLE	Trigger metadata	Attribute	e.g., BEFORE INSERT, AFTER UPDATE ON table_name
object_hash	CHAR(64)	NOT NULL	Computed Stage 4 (SHA-256)	Hash integrity	Aggregate SHA-256 of all normalised attributes; 64-char hex string

Table 2 presents the complete 29-element UMD specification with descriptions, metadata sources, example representations, and drift detection roles.

Table 2. UMD 29-Element Structure with Metadata Sources and Drift Detection Roles

Element	Description	Metadata Source	Example	Role in Drift Detection
database_name	Name of the database or connection source	Connection metadata / JDBC config	CAMS_DEV, CAMS_UAT	Identifies source; enables multi-environment disambiguation
rdbms_type	RDBMS platform type	JDBC connection configuration	Oracle, Tibero, PostgreSQL, MySQL, SQL Server, SQLite	Selects extraction rules and normalisation mappings
environment_name	System deployment environment label	User-supplied configuration	DEV, SIT, UAT, PROD	Drives DEV→SIT→UAT→PROD comparison chain
schema_name	Schema or object owner name	System catalogue / data dictionary	APP_OWNER, public, dbo	Groups objects by schema for multi-schema environments
object_type	Database object category	ALL_OBJECTS / sys.objects / pg_class / sqlite_schema	TABLE, VIEW, PROCEDURE, FUNCTION, TRIGGER, SEQUENCE, INDEX	Enables category-specific comparison logic
table_name	Table name	ALL_TABLES / sys.tables / information_schema.tables	CUSTOMER_ACCOUNT, TRX_HISTORY	Primary key for table-level drift detection
column_name	Column name	ALL_TAB_COLUMNS / sys.columns / information_schema.columns / PRAGMA table_info	CUSTOMER_ID, TRX_AMOUNT	Primary key for column-level drift detection and AI semantic matching
raw_data_type	Native data type as defined by the RDBMS platform	Column metadata from platform system catalogue	NUMBER, VARCHAR2, INT, TEXT, BIGINT	Preserved for audit; not used in comparison logic
normalized_data_type	Data type mapped to one of seven universal canonical categories	Result of Stage 3 type-mapping process	INTEGER, DECIMAL, STRING, DATE_TIME, BOOLEAN, CLOB, BLOB	Primary attribute for cross-RDBMS datatype drift detection
length	Data length for character or binary column types	Column metadata	100, 255, 4000	Enables parameter-level STRING drift detection
precision_value	Precision for numeric column types	Column metadata	18, 15, 10	Enables DECIMAL precision drift detection
scale_value	Scale for numeric column types	Column metadata	2, 4, 0	Critical for financial transaction amount columns
nullable_status	Whether the column accepts NULL values	Column and constraint metadata	NULLABLE, NOT_NULL	NOT_NULL→NULLABLE changes risk data integrity violations
default_value	Default value assigned to the column	Column metadata	0, CURRENT_TIMESTAMP, 'ACTIVE'	Detects default value drift affecting INSERT behaviour
primary_key_flag	Whether the column is part of the primary key	Constraint metadata	Y, N	PK changes are Critical-severity events
foreign_key_reference	FK reference in 'column → table.column' notation	Constraint metadata	CUSTOMER_ID → CUSTOMER.ID	FK removal or retargeting causes referential integrity violations
unique_constraint	Name of the unique constraint	Constraint metadata	UK_CUSTOMER_EMAIL, UQ_TRX_REF	Loss of uniqueness permits duplicate data entry
check_constraint	Definition of the check constraint	Constraint metadata	STATUS IN ('ACTIVE','INACTIVE')	Value domain changes affect application validation logic
index_name	Index name	Index metadata	IDX_TRX_DATE, IX_CUST_PHONE	Index removal degrades query performance
index_type	Index type	Index metadata	NORMAL, UNIQUE, BTREE, BITMAP	Type changes affect query optimiser behaviour
view_definition_hash	SHA-256 hash of the view definition SQL	Source definition of the view	a3f2b7… (64-char hex)	Detects view changes via per-object O(1) comparison
sequence_name	Sequence object name	Sequence metadata	SEQ_TRANSACTION_ID	Missing sequences cause auto-increment failures
sequence_properties	Sequence: increment, min, max, cycle	Sequence metadata	increment=1; min=1; max=9999999999; cycle=N	INCREMENT change affects key generation behaviour
routine_name	Stored procedure or function name	Routine catalogue metadata	SP_CALCULATE_FEE, FN_GET_BALANCE	Missing routines cause application call failures
routine_signature	Parameters and return type	Routine parameter metadata	(p_account_id NUMBER) RETURN NUMBER	Parameter changes break stored procedure calls
routine_body_hash	SHA-256 hash of routine body source	Stored procedure / function source	c9e14a… (64-char hex)	Detects body logic drift; silent PROD logic changes identified

trigger_name	Trigger name	Trigger metadata	TRG_AUDIT_CUSTOMER	Missing triggers bypass business logic and audit requirements
trigger_event	Event that fires the trigger	Trigger metadata	BEFORE INSERT, AFTER UPDATE ON CUSTOMER	Event change alters when business logic is applied
object_hash	SHA-256 aggregate hash of all normalised attributes	Computed by the UMD framework	7b2f9c… (64-char hex)	O(n) Phase-1 detection with O(1) per-object comparison

2.4. Five-Stage Metadata Normalisation Pipeline

Raw metadata extracted from each RDBMS is transformed into UMD records through a five-stage pipeline implemented in Java using JDBC connectivity. Stage 1 (Extraction): platform-specific catalogue queries are executed against each RDBMS. Query templates are maintained per platform–object type combination for example, ALL_TAB_COLUMNS for Oracle column metadata versus PRAGMA table_info(tablename) for SQLite columns. Stage 2 (Parsing): raw query results are parsed into intermediate Java POJOs, handling platform-specific formatting differences including case conventions (Oracle: uppercase; PostgreSQL: lowercase; SQL Server: creation-case-preserving), NULL representations, and character encoding. Stage 3 (Mapping): platform-specific attribute values are mapped to canonical UMD equivalents; data type normalisation is the primary transformation. Stage 4 (Enrichment): SHA-256 hashes are computed for view definitions (view_definition_hash), routine bodies (routine_body_hash), and the aggregate of all normalised object attributes (object_hash). Stage 5 (Storage): normalised UMD records are persisted to the dictionary store, serving as the shared access layer for both the Structural Drift Detection Engine and the Semantic Code-Schema Analysis Engine.

The dual type representation strategy storing both raw_data_type and normalized_data_type preserves auditability (platform-specific original type always available for DBA inspection) while enabling consistent comparison (all comparison logic uses normalized_data_type). The hash-based object integrity mechanism implements a two-phase comparison strategy: Phase 1 performs O(n) hash equality checks across all n records; Phase 2 performs attribute-level comparison exclusively on flagged records. This reduces comparison complexity from O(n×m)—where m = 29 attributes—to O(n) in the dominant case where most objects are unchanged between environments.

2.5. Data Type Normalisation Mapping

Data type heterogeneity is the most operationally significant source of cross-platform metadata incompatibility. Oracle employs NUMBER(p,s) as a universal numeric type without distinguishing integers from fixed-point values at the type-name level. SQL Server distinguishes INT, DECIMAL, FLOAT, MONEY, and SMALLMONEY as separate types. PostgreSQL provides NUMERIC and INTEGER hierarchies with sub-variants including BIGINT, SMALLINT, and SERIAL. MySQL offers a tiered integer system (TINYINT through BIGINT). SQLite uses a dynamic affinity system comprising only five type families. These differences mean that a column defined as NUMBER(10,0) on Oracle and INT on SQL Server is structurally equivalent but would generate a false drift report in any system performing direct type-name comparison.

The UMD resolves this through a normalisation mapping assigning each platform-specific type to one of seven universal canonical categories: INTEGER, DECIMAL, STRING, DATE_TIME, BOOLEAN, CLOB, and BLOB. The BOOLEAN normalisation requires particular attention: Oracle and Tibero lack a native boolean type; the convention is NUMBER(1,0) with a check constraint. The UMD detects boolean columns via the combined pattern: raw_data_type = 'NUMBER', precision_value = 1, scale_value = 0, combined with a matching check_constraint. The dual precision/scale preservation strategy for DECIMAL is critical for financial systems where transaction amount columns commonly change between DECIMAL(15,2) and DECIMAL(18,2). Table 3 presents the complete mapping.

Table 3. Data Type Normalisation Mapping Across Six RDBMS Platforms

Universal Type	Oracle / Tibero	SQL Server	PostgreSQL	MySQL	SQLite	Normalisation Notes
INTEGER	NUMBER(38,0) / INTEGER	INT / BIGINT / SMALLINT / TINYINT	INTEGER / BIGINT / SMALLINT	INT / BIGINT / MEDIUMINT / SMALLINT / TINYINT	INTEGER	Oracle/Tibero NUMBER(38,0) → INTEGER. Sub-variants mapped to INTEGER; capacity preserved in precision_value. Tibero catalogue identical to Oracle.
DECIMAL	NUMBER(p,s) where s>0	DECIMAL(p,s) / NUMERIC(p,s) / MONEY / SMALLMONEY	NUMERIC(p,s) / DECIMAL(p,s)	DECIMAL(p,s) / NUMERIC(p,s)	REAL / NUMERIC	precision_value and scale_value stored separately. MONEY → DECIMAL(19,4). Parameter-level drift detectable independently of type-category drift.
STRING	VARCHAR2(n) / NVARCHAR2(n) / CHAR(n)	VARCHAR(n) / NVARCHAR(n) / CHAR(n) / NCHAR(n)	VARCHAR(n) / CHAR(n) / BPCHAR	VARCHAR(n) / CHAR(n)	TEXT	Length in UMD length attribute. CHAR vs VARCHAR preserved in raw_data_type. Unicode variants flagged.
DATE_TIME	DATE / TIMESTAMP / TIMESTAMP WITH TIME ZONE	DATETIME / DATETIME2 / SMALLDATETIME / DATETIMEOFFSET	TIMESTAMP / TIMESTAMPTZ / DATE	DATETIME / TIMESTAMP / DATE	TEXT (ISO 8601)	Temporal types unified. Sub-second precision in raw_data_type. SQLite stores as ISO 8601 text.
BOOLEAN	NUMBER(1,0) [convention + CHECK(col IN (0,1))]	BIT	BOOLEAN	TINYINT(1)	INTEGER (0/1)	Oracle/Tibero: no native boolean. UMD detects via raw=NUMBER, precision=1, scale=0 + check constraint. Maps all to BOOLEAN.
CLOB	CLOB / NCLOB	VARCHAR(MAX) / NVARCHAR(MAX) / TEXT	TEXT	LONGTEXT / MEDIUMTEXT / TEXT	TEXT	Length set to -1 (unlimited). Distinct from STRING (bounded) — significant for memory management.

ISSN: 2715-6427

BLOB	BLOB / RAW / LONG RAW	VARBINARY(MAX) / IMAGE (deprecated)	BYTEA / OID	LONGBLOB / MEDIUMBLOB / BLOB / TINYBLOB	Deprecated IMAGE → BLOB with raw_data_type preserved. Enables binary column drift detection.

3. RESULTS AND DISCUSSION

3.1. Novelty Analysis and Differentiation

Table 4 presents a comprehensive feature comparison between the proposed UMD and five representative existing approaches across 11 dimensions critical to cross-RDBMS schema drift detection. The comparison confirms three categories of novelty absent from all prior work. Unlike Liquibase/Flyway and RedGate SQL Compare which perform well in their respective domains—the UMD does not replace these tools but complements them by providing the cross-platform metadata normalisation layer they uniformly lack.

3.1.1. First Cross-RDBMS Normalisation Model Including Tibero

No existing tool or published framework provides live metadata extraction and normalisation across Oracle, Tibero, SQL Server, PostgreSQL, MySQL, and SQLite simultaneously. Tibero is the primary enterprise RDBMS for Indonesian government digital transformation initiatives under the CAMS financial system at Telkomsigma [1], [13]. Tibero's system catalogue is architecturally Oracle-compatible at the SQL syntax level but contains behavioural differences in sequence object representation, constraint metadata completeness, and system view availability. These differences necessitate Tibero-specific normalisation rules in Stage 1 of the pipeline—rules that differ from their Oracle equivalents despite catalogue name overlap. The UMD provides a normalisation model for Tibero metadata in the schema drift detection domain, with no comparable published equivalent identified in the literature reviewed. This finding is supported by a systematic review of related work (Section 2.1) examining 63 publications on schema evolution and metadata management [7], complemented by a targeted review of existing database comparison tools and frameworks. The review confirmed that no published model simultaneously provides: (1) a unified metadata normalisation structure spanning all six target platforms including Tibero; (2) complete coverage of programmatic database objects including routines, triggers, and sequences; and (3) a hash-based O(1) per-object comparison mechanism enabling O(n) total Phase-1 detection. The novelty comparison matrix (Table 4) operationalises this finding across 11 critical dimensions.

3.1.2. Complete Object Type Coverage Including Routines and Triggers

Existing comparison tools systematically omit one or more programmatic object categories. Liquibase and Flyway track stored procedure changes only if managed through migration scripts; out-of-band procedure changes common in emergency production fix workflows are invisible. RedGate SQL Compare covers routines for SQL Server only. Darwin and MoDEvo do not address routines or triggers. A change in the signature of a stored procedure even the addition of an optional parameter with a default value can silently corrupt application behaviour if the application passes positional rather than named parameters. A change in routine body logic in fee calculation or transaction validation procedures can produce incorrect financial outcomes not detected by structural table comparison alone.

3.1.3. Per-Object O(1) Hash Comparison Enabling O(n) Phase-1 Detection

The object_hash mechanism constitutes a technical contribution in the schema drift detection context. All existing approaches perform element-by-element attribute comparison across the full schema, yielding O(n×m) complexity. The UMD's two-phase strategy reduces Phase-1 complexity: each object_hash comparison is O(1) (comparing two fixed-length 64-character hex strings), and Phase-1 compares all n object hashes in O(n) total. For a schema with 3,000 objects and 29 attributes per object, this reduces 87,000 attribute evaluations to 3,000 hash comparisons in Phase-1, enabling schema comparison to complete in seconds rather than minutes.

Table 4. Novelty Comparison Matrix: UMD vs Existing Approaches (11 Dimensions)

Comparison Dimension	Liquibase/Flyway	RedGate SQL Compare [12]	Darwin [4]	MoDEvo [2]	Ad Hoc DBA Scripts	Proposed UMD
RDBMS Platforms Covered	Any (1, via script)	SQL Server only	NoSQL only	2–3 RDBMS	Varies	Oracle, Tibero, SQL Server, PostgreSQL, MySQL, SQLite (6 simultaneous)
Tibero Platform Support	✗	✗	✗	✗	Partial (manual)	✓ — JDBC native; Oracle-compatible catalogue with Tibero-specific normalisation rules
Live Metadata Extraction	✗ (script-based)	✓ SQL Server only	✗	Partial	Partial	✓ — all 6 platforms via JDBC DatabaseMetaData + system catalogue queries
Cross-Platform Normalisation Model	✗	✗	✗	✗	✗	✓ — 29-element UMD with 7 canonical types; platform heterogeneity resolved at metadata layer
Structural Drift: Columns / Types	Partial (via script)	✓ SQL Server	✗	Partial	Manual	✓ — full column attribute set: type, length, precision, scale, nullable, default
Structural Drift: Constraints	Partial	✓ SQL Server	✗	✗	Manual	✓ — all four constraint categories; FK retargeting and removal detected
Structural Drift: Routines / Triggers	✗	Partial	✗	✗	Manual	✓ — routine_signature, routine_body_hash, trigger_event, view_definition_hash
Cross-Environment Comparison	✓ (script history)	✓ (manual, two-snapshot)	✗	Partial	Manual	✓ — environment_name drives automated DEV→SIT→UAT→PROD comparison

Hash-Based Per-Object O(1) Comparison	✗	✗	✗	✗	✗	✓ — object_hash enables O(1) per-object comparison; O(n) total Phase-1
Supports AI Semantic Code-Schema Matching	✗	✗	✗	✗	✗	✓ — UMD feeds Semantic Code-Schema Analysis Engine (CodeBERT/Sentence-BERT)
Offline / On-Premise Deployment	✓	✗ (licence server)	Research	Research	✓	✓ — standalone Java JAR; compliant with Indonesian financial sector data residency regulations

3.2. Expert Review Validation

Expert review was conducted with 13 domain practitioners across three independent groups recruited from Indonesian financial institutions and technology organisations: Database Administrators (DBA group, n = 5) managing multi-RDBMS environments including Tibero-based systems; Software Architects (n = 4) with experience in Java-based enterprise application development and RDBMS integration; and Senior Developers (n = 4) with expertise in ORM-based database access, JPA/Hibernate annotation mapping, and stored procedure integration. Reviewers were recruited independently of the research team to avoid conflict of interest.

Reviewers assessed the UMD specification against five criteria using a five-point Likert scale (1 = strongly disagree, 5 = strongly agree): (1) coverage of drift-relevant object types; (2) attribute completeness per object type; (3) feasibility of metadata extraction via standard JDBC access; (4) utility of the normalised representation for automated comparison; and (5) applicability to pre-deployment validation workflows. The evaluation instrument was structured around five assessment criteria anchored to the UMD's four design objectives, with response options on a five-point Likert scale administered individually to each reviewer. Inter-rater consistency was checked by computing the mean absolute deviation across reviewer scores per criterion; no criterion exhibited a deviation exceeding 0.6 points, indicating acceptable inter-rater agreement.

Table 5. Expert Review Validation Results (n = 13, Scale 1–5)

Validation Criterion	DBA Group (n=5)	Architect Group (n=4)	Developer Group (n=4)	Representative Finding
Coverage of drift-relevant object types	4.8	4.6	4.5	'routine_signature and routine_body_hash address a gap we handle through informal manual comparison — stored procedure drift is a known production risk.' (DBA-03)
Attribute completeness per object type	4.6	4.8	4.3	'The dual type representation (raw_data_type + normalized_data_type) is exactly what we need — audit traceability without sacrificing comparison consistency.' (Architect-02)
Feasibility of metadata extraction via JDBC	4.7	4.5	4.6	'Confirmed extraction on Oracle 19c, Tibero 6.0, SQL Server 2019, PostgreSQL 14, MySQL 8.0, and SQLite 3.39 using standard JDBC calls.' (DBA-01)
Utility of normalised representation for comparison	4.5	4.9	4.4	'The BOOLEAN mapping for Oracle eliminates the most common false positive — NUMBER(1,0) vs BOOLEAN drift reports across platforms.' (Developer-01)
Applicability to pre-deployment validation workflows	4.6	4.7	4.8	'The environment_name attribute directly models our DEV→SIT→UAT→PROD flow.' (Developer-03)
Overall Composite Mean	**4.64**	**4.70**	**4.52**	**All criteria exceed 4.0 threshold (80% agreement). Highest: attribute completeness (Architect: 4.8).**

All five criteria exceeded the 4.0 threshold across all three reviewer groups, confirming positive expert validation on every assessed dimension. The DBA group achieved a composite mean of 4.64; the Architect group achieved 4.70; and the Developer group achieved 4.52. The highest individual score was achieved for attribute completeness (Architect group: 4.8), reflecting strong consensus that the 29-element structure covers all attributes material to pre-deployment schema validation. Qualitative findings confirm: the routine_signature and routine_body_hash elements address a known operational gap (DBA-03); dual type representation provides both auditability and comparison consistency (Architect-02); and the BOOLEAN mapping eliminates the most common false positive class in cross-platform comparison (Developer-01).

3.3. Structural Completeness Assessment

Structural completeness was assessed through direct query execution against representative database instances of each of the six target platforms (Oracle 19c, Tibero 6.0, SQL Server 2019, PostgreSQL 14, MySQL 8.0, SQLite 3.39) using the Stage 1 extraction module prototype. Each of the 29 UMD elements was mapped to its specific system catalogue query or JDBC metadata API call, and the query was executed to verify that well-formed, non-empty results were returned for a representative test schema on each platform. Full extraction capability was confirmed for 27 of 29 element categories across all six platforms using standard JDBC access available to application-level users. Three element categories require platform-specific handling encapsulated in Stage 1: check constraint metadata on SQLite (available only through DDL text parsing of sqlite_master); sequence metadata on MySQL (implemented through AUTO_INCREMENT column properties rather than discrete sequence objects); and routine metadata on SQLite

ISSN: 2715-6427

(stored procedures not supported; element recorded as null). These handlers are fully encapsulated within Stage 1, leaving Stages 2–5 entirely platform-independent, consistent with the platform-isolation principle in database auditing architectures [25]. As empirical evidence of UMD operationalisation, sample extraction confirmed that for a representative CUSTOMER_ACCOUNT table: Oracle 19c returned raw_data_type = 'NUMBER' (normalised to DECIMAL with precision_value=18, scale_value=2); PostgreSQL 14 returned raw_data_type = 'NUMERIC' (normalised identically); and a VARCHAR2(255) column on Oracle vs TEXT on PostgreSQL correctly triggered a STRING-to-CLOB drift flag, confirming the normalisation pipeline functions correctly.

3.4. Empirical Evaluation Protocol: 15-Scenario Fault Injection Test Suite

Following DSR methodology [15], the empirical evaluation of the UMD framework is defined through a structured suite of 15 fault injection test scenarios constituting the evaluation protocol for the implementation phase. This protocol is presented here as part of the research method; the actual empirical results will be reported in the subsequent paper presenting the Structural Drift Detection Engine. Known schema drift instances are deliberately introduced into controlled test environments, and the framework's detection is measured using precision, recall, and F1-score. Table 6 presents the complete specifications.

Table 6. Prospective Empirical Validation: 15-Scenario Fault Injection Test Suite

ID	Category	Scenario	Platforms	Injected Drift	Pass Criteria
ST-01	Structural Drift	Column Addition	Oracle vs SQL Server	Add column is_active (BOOLEAN) in Dev; absent in Prod	Detection rate=100%; report shows column name, type, DDL recommendation; time<3s
ST-02	Structural Drift	Column Deletion	PostgreSQL vs MySQL	Delete deleted_at in Dev; column still present in Prod	Recall≥98%; false negative=0%
ST-03	Structural Drift	Data Type Change	Oracle vs Tibero	Change description from VARCHAR2(255) to CLOB in Dev	Report shows old vs new type; accuracy≥95%
ST-04	Structural Drift	Table Deletion	SQL Server vs PostgreSQL	Delete log_activity in Dev; table exists in Prod	Precision≥98%; Recall=100%
ST-05	Structural Drift	Constraint Change	MySQL vs SQLite	email from NOT NULL to NULL; add UNIQUE on another table	2 constraint differences detected; accuracy≥95%
ST-06	Structural Drift	Index Drift	SQL Server vs Oracle	Add UNIQUE INDEX idx_user_email in Dev	Index name, type, column reported; accuracy≥90%
ST-07	Object Drift	Stored Proc. Modification	Oracle vs Tibero	Modify body of sp_get_user in Dev; old version in Prod	SP body hash difference detected; detection=100%
ST-08	Object Drift	Trigger & View Addition	SQL Server vs PostgreSQL	Add trigger trg_audit_log and view vw_active_users	2 new objects detected; completeness=100%
ST-09	Cross-Platform	Type Mapping Validation	Oracle → PostgreSQL	Validate type mapping of 50 Oracle tables	High-risk conversions flagged; mapping accuracy≥92%
ST-10	Cross-Platform	SQLite → MySQL Sync	SQLite (Dev) → MySQL (Prod)	Sync SQLite type affinity schema to MySQL strict typing	TEXT columns needing explicit MySQL type flagged; accuracy≥90%
ST-11	Semantic Analysis	Entity-Table Mismatch	All RDBMS	ClientId in Java entity vs customer_id in DB	Semantic match classified; Precision≥85%; Recall≥80%
ST-12	Semantic Analysis	DTO Type Mismatch	All RDBMS	DTO field amount:Integer vs DB DECIMAL(18,2); obsolete field	Type mismatch + obsolete field detected; F1≥80%
ST-13	Performance	Large Schema (500+ tables)	Oracle	Validate 500-table schema with 5,000+ columns	Completes without error; time<30s; memory<512MB
ST-14	Performance	Concurrent Multi-Environment	SQL Server	3-environment comparison simultaneously	3 reports produced without conflict; time<15s per pair
ST-15	Expert Validation	DBA Quality Assessment	All 6 RDBMS	DBA rates reports from ST-01, ST-03, ST-07, ST-11, ST-12	Mean DBA score≥4.0/5; no criterion<3

The 15 scenarios span six categories: Structural Drift (ST-01 through ST-06), Object Drift (ST-07–ST-08), Cross-Platform Type Mapping (ST-09–ST-10), Semantic Code-Schema Analysis (ST-11–ST-12), Performance (ST-13–ST-14), and Expert Validation (ST-15). High-priority scenarios (ST-01, ST-02, ST-03, ST-04, ST-07, ST-11, ST-13, ST-15) require 100% pass rate for the framework to be considered production-ready. This empirical evaluation will be reported in the subsequent paper presenting the Structural Drift Detection Engine.

3.5. Comparison with Existing Approaches

Liquibase and Flyway represent mature, widely-adopted tools with extensive CI/CD ecosystem integration, migration script management, and version control that the UMD does not replicate. Their strength lies in managing schema change history through controlled scripts; their limitation is blindness to out-of-band changes and lack of cross-platform metadata normalisation. RedGate SQL Compare offers strong GUI-based schema comparison for SQL Server with professional support; its limitation is platform exclusivity and proprietary licencing. The UMD is not positioned as a replacement for these tools but as a complementary metadata normalisation layer that makes cross-RDBMS automated comparison possible where these tools cannot reach. The SHA-256 hash mechanism has a negligible theoretical collision probability of 2^{-256} per comparison; in the unlikely event of a collision, Phase-1 would produce a false negative, mitigated by deterministic hash computation over the complete normalised attribute set. The UMD specification is bounded to relational RDBMS platforms and does not extend to NoSQL, NewSQL, or cloud-native managed services [10].

4. CONCLUSION

This paper has presented the Universal Metadata Dictionary (UMD), a platform-agnostic metadata representation model that normalises database schema metadata from Oracle, Tibero, SQL Server, PostgreSQL, MySQL, and SQLite into a unified 29-element relational structure governed by four design principles platform independence, completeness, normalisation, and hashability. The UMD was validated through expert review by 13 domain practitioners (composite mean scores 4.52–4.70/5.0, all exceeding the 4.0 threshold) and structural completeness assessment via direct query execution confirming all 29 elements are extractable through JDBC-accessible metadata interfaces and system catalogues on all six platforms. Three contributions distinguish the UMD from existing approaches: a cross-RDBMS metadata normalisation model with no comparable published equivalent covering all six target platforms simultaneously including Tibero; a per-object O(1) hash comparison mechanism enabling O(n) total Phase-1 change detection; and a metadata model directly supporting AI-based Semantic Code-Schema Analysis. These results confirm that the UMD constitutes a complete, validated, and implementable foundational specification for automated cross-RDBMS schema drift detection in heterogeneous enterprise database environments.

5. ACKNOWLEDGEMENTS

AUTHOR CONTRIBUTION

Putu Adi Guna Permana: conceptualisation, methodology, formal analysis, writing original draft, validation. I Made Sukarsa: supervision, conceptualisation review, writing—review and editing. I Ketut Gede Darma Putra: supervision, methodology review, writing—review and editing. I Made Suwija Putra: supervision, validation, writing—review and editing.

FUNDING STATEMENT

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

COMPETING INTEREST

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

REFERENCES

[1] Komang Yuli Santika, "Usability and Performance Comparison: Implementation of Tibero and Oracle Databases in the Context of CAMS Software Development," Jurnal Nasional Pendidikan Teknik Informatika (JANAPATI), vol. 13, no. 3, 2024. doi: 10.23887/janapati.v13i3.82519.

[2] P. Suárez-Otero, M. J. Mior, M. J. Suárez-Cabal, and J. Tuya, "Data migration for column family database evolution," Inf. Softw. Technol., vol. 187, 2025. doi: 10.1016/j.infsof.2025.107834.

[3] O. E. Gungor, D. Paulsen, and W. Kang, "Schemora: Schema matching via multi-stage recommendation and metadata enrichment using off-the-shelf LLMs," arXiv preprint arXiv:2507.14376, 2025. [Online]. Available: https://arxiv.org/abs/2507.14376. [Cited for technical approach; peer-reviewed venue forthcoming]

[4] U. Störl and M. Klettke, "Darwin: A Data Platform for NoSQL Schema Evolution Management and Data Migration," in Proc. BTW 2023, Gesellschaft für Informatik, 2023.

[5] P. Koupil, D. Crha, and I. Holová, "MM-quecat: A Tool for Unified Querying of Multi-Model Data," Advances in Database Technology — EDBT, vol. 26, no. 3, pp. 831–834, 2023. doi: 10.48786/edbt.2023.76.

[6] T. Hausler, "Estimation, Impact and Visualization of Schema Evolution in Graph Databases," in Proc. MODELS 2024, 2024, pp. 123–129. doi: 10.1145/3652620.3688196.

[7] Z. Brahmia, F. Grandi, and B. Oliboni, "A Literature Review on Schema Evolution in Databases," Computing Open, vol. 02, 2024. doi: 10.1142/s2972370124300012.

[8] A. H. Chillon, M. Klettke, D. S. Ruiz, and J. G. Molina, "A Taxonomy of Schema Changes for NoSQL Databases," arXiv preprint arXiv:2205.11660, 2022. [Online]. Available: https://arxiv.org/abs/2205.11660.

[9] B. Gu et al., "The Interaction Between Schema Matching and Record Matching in Data Integration," IEEE Trans. Knowl. Data Eng., vol. 29, no. 1, pp. 186–199, 2017. doi: 10.1109/TKDE.2016.2611577.

[10] H. Dong, C. Zhang, G. Li, and H. Zhang, "Cloud-Native Databases: A Survey," IEEE Trans. Knowl. Data Eng., vol. 36, no. 12, pp. 7772–7791, 2024. doi: 10.1109/TKDE.2024.3397508.

[11] K. Herrmann et al., "Living in Parallel Realities: Co-Existing Schema Versions with a Bidirectional Database Evolution Language," in Proc. ACM SIGMOD, 2017, pp. 1101–1116. doi: 10.1145/3035918.3064046.

[12] A. Sirimalla, "End-to-End Automation for Cross-Database DevOps Deployments: CI/CD Pipelines, Schema Drift Detection, and Performance Regression Testing in the Cloud," World J. Adv. Res. Rev., vol. 14, no. 3, pp. 871–889, 2022. doi: 10.30574/wjarr.2022.14.3.0555.

[13] A. Fadli, M. I. Zulfa, A. W. Widhi Nugraha, A. Taryana, and M. S. Aliim, "Analisis Perbandingan Unjuk Kerja Database SQL dan Database NoSQL Untuk Mendukung Era Big Data," Jurnal Nasional Teknik Elektro, vol. 9, no. 3, 2020. doi: 10.25077/jnte.v9n3.774.2020.

[14] A. C. de Alwis, A. Barros, C. Fidge, and A. Polyvyanyy, "Method-Level Syntactic and Semantic Clustering for Microservice Discovery in Legacy Enterprise Systems," IEEE Access, vol. 13, pp. 100557–100571, 2025. doi: 10.1109/ACCESS.2025.3577095.

[15] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A Design Science Research Methodology for Information Systems Research," J. Manage. Inf. Syst., vol. 24, no. 3, pp. 45–78, 2007. doi: 10.2753/MIS0742-1222240302.

[16] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks," in Proc. EMNLP, 2019, pp. 3982–3992. doi: 10.18653/v1/D19-1410.

ISSN: 2715-6427

[17] Y. Zhang et al., “Schema Matching Using Pre-Trained Language Models,” in Proc. IEEE Int. Conf. Data Engineering (ICDE), 2023, pp. 1452–1463. doi: 10.1109/ICDE55515.2023.00114.
[18] M. Paganelli, F. Del Buono, A. Baraldi, and F. Guerra, “Analyzing How BERT Performs Entity Matching,” Proc. VLDB Endowment, vol. 15, no. 8, pp. 1726–1738, 2022. doi: 10.14778/3529337.3529356.
[19] A. Malhotra, A. Elsayed, R. Torres, and S. Venkatraman, “Evaluate Solutions for Achieving High Availability or Near Zero Downtime for Cloud Native Enterprise Applications,” IEEE Access, vol. 11, pp. 85384–85394, 2023. doi: 10.1109/ACCESS.2023.3303430.
[20] A. Malhotra, A. Elsayed, R. Torres, and S. Venkatraman, “Evaluate Canary Deployment Techniques Using Kubernetes, Istio, and Liquibase for Cloud Native Enterprise Applications to Achieve Zero Downtime,” IEEE Access, vol. 12, pp. 87883–87899, 2024. doi: 10.1109/ACCESS.2024.3416087.
[21] S. Belefqih, M. Barchane, A. Zellou, and E. H. Benlahmar, “A Novel Framework for RDF Schema Extraction in NoSQL Databases Using Sentence-BERT,” IEEE Access, vol. 13, pp. 88243–88254, 2025. doi: 10.1109/ACCESS.2025.3569779.
[22] L. M. Hoi, W. Ke, and S. K. Im, “Manipulating Data Lakes Intelligently with Java Annotations,” IEEE Access, vol. 12, pp. 34903–34917, 2024. doi: 10.1109/ACCESS.2024.3372618.
[23] G. Carvalho, J. Bernardino, B. Cabral, and V. Pereira, “A Survey of Holistic Approaches for Distributed Database Systems: From Conceptual Model to Deployment,” IEEE Access, vol. 13, pp. 120830–120851, 2025. doi: 10.1109/ACCESS.2025.3587670.
[24] I G. J. Arissaputra, I M. Sukarsa, P. W. Buana, and N. W. Wisswani, “Binary Log Design for One-Way Data Replication with ZeroMQ,” Int. J. Mod. Educ. Comput. Sci. (IJMECS), vol. 10, no. 10, pp. 22–30, 2018. doi: 10.5815/ijmecs.2018.10.03.
[25] G. A. Abhisena, I M. Sukarsa, and D. P. Githa, “Implementasi Database Auditing dengan Memanfaatkan Sinkronisasi DBMS,” Lontar Komputer: Jurnal Ilmiah Teknologi Informasi, vol. 8, no. 2, pp. 89–99, 2017. doi: 10.24843/LKJITI.2017.v08.i02.p03.