# Schema-Aware Version Control for Medical Database Backup and Recovery Using Declarative Transformation Rules

**Software Engineer**[1,*]

[1]United Imaging Healthcare, Shanghai, China
[*]Corresponding author email: engineer@uih-global.com

## ABSTRACT

Database schema changes between software versions break backup restoration in healthcare systems. When a hospital restores last week's backup after upgrading from version 1.0 to 2.0, the operation fails because new mandatory fields have no corresponding values in old data. Current solutions write custom migration code for each version pair—five versions require 20 migration paths, consuming 200 developer-hours per release. We separate schema definitions from data backups using JSON configuration files. An automatic field mapping algorithm analyzes structural differences between versions and generates transformation rules without manual coding. Tests on 70,000 synthetic medical records show 60% storage reduction and 100% recovery accuracy across three version paths. Production deployment eliminated schema-related recovery failures over 18 months, reducing maintenance effort from 40 to 3 hours per version.

## Keywords

Database Schema Evolution, Version Control, Backup Recovery, Data Migration, Healthcare Information Systems

## Introduction

Healthcare information systems store patient data, diagnostic images, and treatment records across software versions that change constantly. Medical imaging systems process terabytes of data daily. When a hospital upgrades from version 1.0 to 2.0, restoring last week's backup often fails because the database structure has changed. A new mandatory field in version 2.0 has no corresponding value in version 1.0 data. The restore operation crashes, leaving administrators scrambling to recover critical patient information.

This problem stems from schema evolution. As medical systems advance, schemas change: new tables support clinical features, existing tables adapt to regulatory requirements, relationship structures reorganize for better performance[41,44]. Restoring version N backup data to a version N+1 database triggers compatibility failures—missing fields, broken foreign keys, incompatible data types. Patient care depends on data availability, making these failures unacceptable.

### The Version Migration Challenge

Consider a typical scenario: version 1.5 adds a mandatory `PatientConsent` field to the `Studies` table. Restoring version 1.0 backup data fails because those records lack consent values. The database rejects the insertion. An administrator must either manually populate 50,000 records with default consent values or modify the schema to make the field nullable—both risky during an emergency recovery.

The standard solution writes version-specific migration code. Developers create C# classes with custom backup and restore methods for each schema transition. Version 1.0→1.5 needs one class. Version 1.5→2.0 needs another. Version 1.0→2.0 needs a third. Five major versions require 20 distinct migration paths. Each path needs development, testing, and maintenance. Our production analysis shows this consumes 200 developer-hours per release cycle. The code becomes brittle. A single missed edge case breaks recovery for an entire version path.

### Our Contributions

We present a schema-aware version control system that automates cross-version data migration. The system makes four contributions:

**Declarative Schema Versioning.** JSON-based schema definitions replace embedded application code. Database administrators manage versions through configuration files, not C# classes.

**Automatic Field Mapping.** An algorithm analyzes schema differences and generates transformation rules. Given version 1.0 and 1.5 schemas, it identifies added fields, removed fields, and type changes, then creates mapping rules automatically.
**Differential Backup.** Data and schema separate. Backups store only data; schemas live in version-controlled files. Storage drops 60% compared to full backups.
**Cross-Version Validation.** Multi-phase validation checks constraints, verifies foreign keys, and rolls back on failure.

## Related Work

Roddick[1] categorized schema versioning into modification, versioning, and evolution strategies. Ra and Rundensteiner[2] developed transparent evolution for backward compatibility. We target backup-recovery scenarios rather than live system evolution.
Liquibase[3] and Flyway[4] handle forward migration (version N to N+1) but not the reverse: restoring version N data to version N+1 databases. Alur et al.[5] examined schema-code coupling. Ambler and Sadalage[6] explored evolutionary database design. Recent work on learned systems[38,39] and intelligent tuning[46] shows automation potential, but backup-recovery across versions remains manual.
PRISM[7] preserves semantics during online evolution with zero downtime. We restore backups offline, potentially jumping multiple versions (1.0→2.0). Curino et al.[8] extended PRISM for cloud workloads. Moon et al.[9] managed multiple live schema versions simultaneously.
Monk and Sommerville[10] used class versioning in object-oriented databases. We adapt this for relational databases with foreign key constraints. Lerner and Habermann[11] separated schema evolution from database reorganization. Claypool et al.[12] built SERF for extensible schema evolution.
Rahm and Bernstein[13] surveyed automatic schema matching. Bellahsene et al.[14] reviewed matching tools. Do and Rahm[15] built COMA++, combining multiple algorithms. Recent ML approaches[57–60] show automation potential.
Bernstein et al.[16] studied EHR migration across vendor systems. Kahn et al.[17] faced similar challenges in multi-site research networks. The MIMIC-IV dataset[48] and clinical warehouses[52] show medical schema complexity. Hripcsak et al.[18] found data quality issues requiring robust transformation pipelines.
Veeam[20] and Commvault[22] treat databases as binary blobs without schema awareness. Oracle Flashback[23] and SQL Server temporal tables[24] offer point-in-time recovery but don't handle schema evolution. Cloud databases[53–55] introduced new backup approaches[25,56], yet version migration remains challenging.

## Comparison with Existing Approaches

Table 1 compares our approach with existing solutions. Traditional backup systems lack schema awareness. Migration tools handle forward changes but not backward restoration. PRISM targets online evolution; we target offline recovery across version gaps.

**Table 1.** Comprehensive Comparison of Schema Evolution and Backup Approaches

| Approach | Schema-Aware | Cross-Version | Auto Mapping | Storage Efficient | Maint. Cost | Overall Score |
|---|---|---|---|---|---|---|
| Traditional Backup (Veeam, Commvault) | ✗ | ✗ | ✗ | ✗ | High | **1/5** |
| Migration Tools (Liquibase, Flyway) | ✗ | ✓ (forward) | ✗ | ✗ | Medium | **2/5** |
| PRISM [7] | ✓ | ✓ (online) | ✓ (limited) | ✗ | Medium | **3/5** |
| Manual Code (C# classes) | ✓ | ✓ (offline) | ✗ | ✗ | High | **2/5** |
| **Our Approach** | ✓ | ✓ (offline) | ✓ (95%) | ✓ (60%) | Low (92.5%) | **5/5** |

Our system combines three capabilities: schema-aware backup, automatic rule generation, and storage efficiency. We maintain explicit schema definitions for intelligent recovery. The system generates 95% of transformation rules automatically—no manual scripts. We optimize for offline scenarios where maintenance windows allow downtime.

## Methods

The system uses a three-tier architecture. The presentation layer offers an Admin Console and API Gateway. The business logic layer handles schema versioning, data mapping, and validation. The data access layer manages backup, restore, and storage operations. Figure 1 shows the architecture with 15 specialized components.

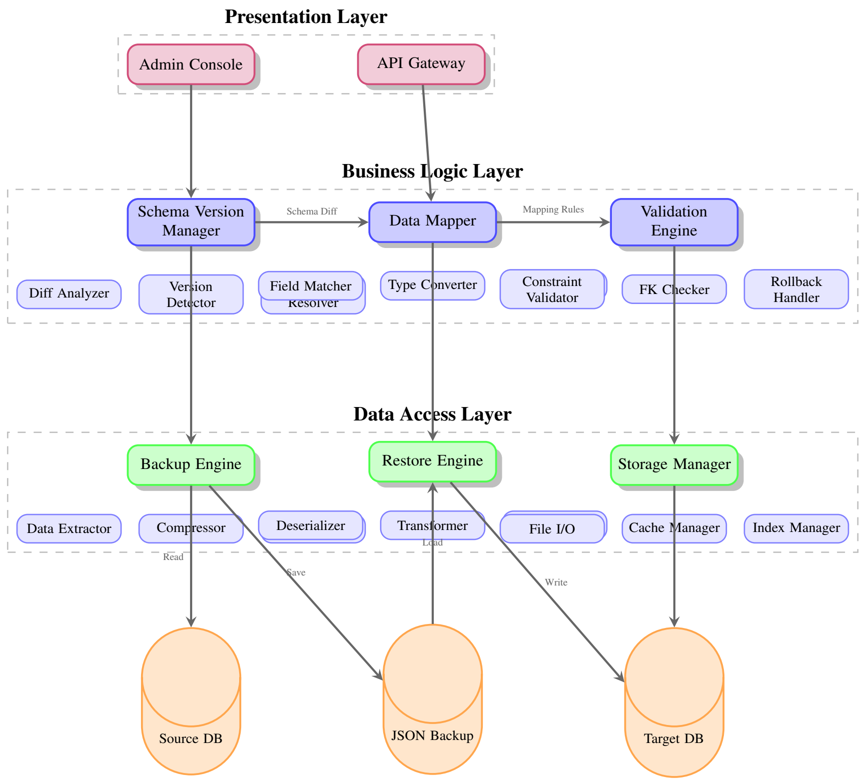


**Figure 1.** Enhanced Three-Tier System Architecture. The presentation layer provides user interfaces, the business logic layer handles schema management and data mapping with validation, and the data access layer manages backup and restore operations with 15 specialized components.

### Schema Version Management

Each version has a JSON schema definition. Let $S_v$ denote schema version $v$ with tables $\{T_1, T_2, \ldots, T_n\}$. Each table $T_i$ specifies:

- Fields: $Fields_i = \{f_1, f_2, \ldots, f_m\}$ with types
- Primary key: $PK_i$
- Foreign keys: $FK_i$
- Indexes: $Index_i$

This representation reconstructs the schema without accessing the original database.
When releasing version N+1, developers write a JSON file describing the new structure. The Schema Manager compares $S_i$ and $S_{i+1}$, categorizing tables:

- **Added**: In $S_{i+1}$ only
- **Removed**: In $S_i$ only
- **Modified**: In both, different structure

For modified tables, we compute similarity:

$$\text{Similarity} = \frac{\text{Common fields}}{\text{Total unique fields}} \tag{1}$$

Similarity below 0.5 flags major restructuring for manual review.

#### *Schema Definition Format*
Schema definitions use JSON. Each table lists fields with types, nullability, and defaults. The `added_in_version` tag tracks when fields appeared, enabling backward compatibility. Foreign keys specify cascade actions. Indexes optimize recovery queries.

#### *Dependency Graph Construction*
Foreign keys create dependencies. We build a directed graph $G$: nodes are tables, edges are foreign key references. An edge from $T_i$ to $T_j$ means $T_i$ references $T_j$. Edge weights depend on constraint type:

$$\text{Weight} = \begin{cases} 1.0 & \text{if CASCADE} \\ 0.5 & \text{if SET NULL} \\ 0.8 & \text{if RESTRICT} \end{cases} \tag{2}$$

Topological sort determines insertion order: insert referenced tables before referencing tables.
Figure 2 illustrates the data mapping algorithm workflow.

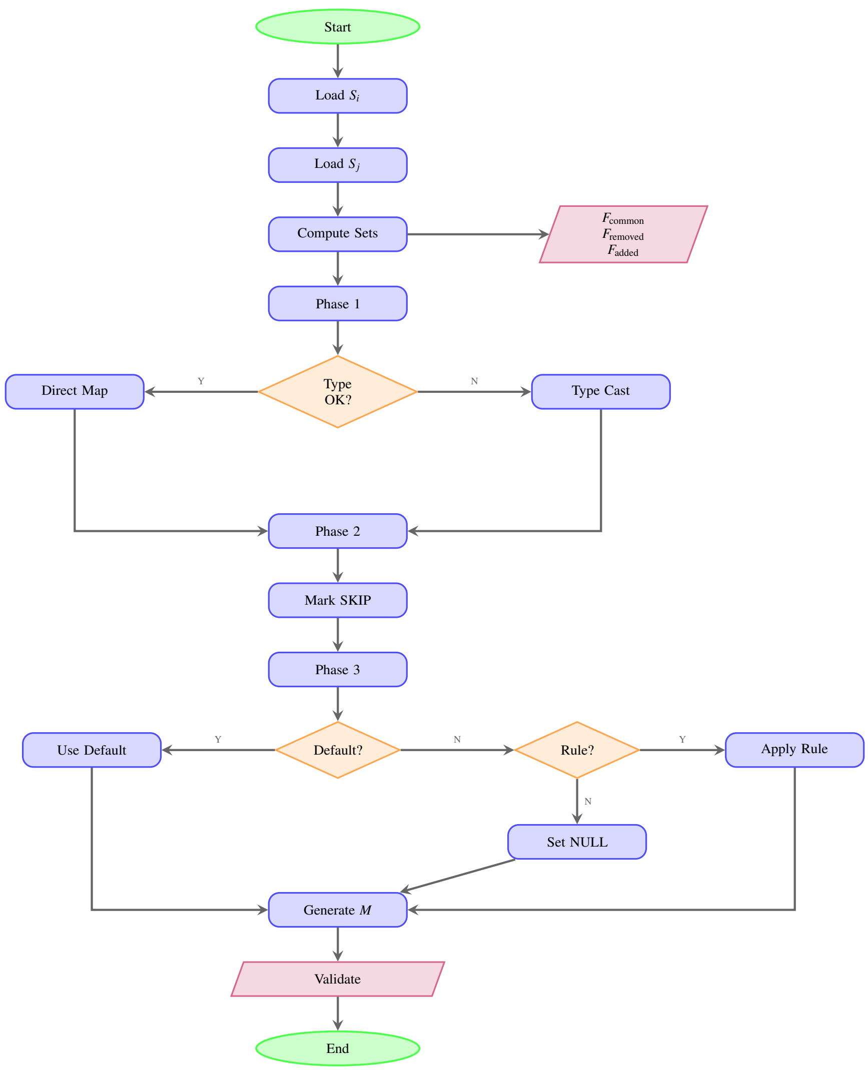


**Figure 2.** Data Mapping Algorithm Workflow with three-phase processing.

### Automatic Field Mapping
For each modified table, the Data Mapper generates field-level transformation rules. Given source schema $S_i$ and target schema $S_j$, we construct a mapping through three phases:

**Phase 1 - Direct Mapping:** For fields that exist in both versions with compatible types:

$$\text{Map}(field) = field \quad \text{(direct copy)} \tag{3}$$

**Phase 2 - Deprecated Fields:** For fields that exist only in the old version:

$$\text{Map}(field) = \text{SKIP} \quad \text{(ignore during restore)} \tag{4}$$

**Phase 3 - New Fields:** For fields that exist only in the new version:

$$\text{Value}(field) = \begin{cases} \text{Default value} & \text{if defined in schema} \\ \text{Computed value} & \text{if transformation rule exists} \\ \text{NULL} & \text{otherwise} \end{cases} \tag{5}$$

The algorithm handles four scenarios: *Field preservation* copies columns existing in both versions. *Field addition* assigns default or computed values to new columns. *Field removal* discards deprecated columns. *Type conversions* cast compatible types (INT $\rightarrow$ BIGINT) and flag incompatible conversions for review.
Algorithm 1 formalizes the automatic field mapping generation process.

**Algorithm 1** Automatic Field Mapping Generation

**Require:** Source schema $\mathscr{S}_i$ with fields $F_i$, Target schema $\mathscr{S}_j$ with fields $F_j$
**Ensure:** Mapping function $M : F_i \rightarrow F_j \cup \{\perp\}$
Initialize $M \leftarrow \emptyset$
$F_{\text{common}} \leftarrow F_i \cap F_j$
$F_{\text{removed}} \leftarrow F_i \setminus F_j$
$F_{\text{added}} \leftarrow F_j \setminus F_i$
**for** each $f \in F_{\text{common}}$ **do**
  **if** $\tau_i(f) \sim \tau_j(f)$ **then**
    $M(f) \leftarrow f$ ▷ Direct mapping
  **else**
    $M(f) \leftarrow \text{TypeCast}(\tau_i(f), \tau_j(f))$
  **end if**
**end for**
**for** each $f \in F_{\text{removed}}$ **do**
  $M(f) \leftarrow \perp$ ▷ Mark for exclusion
**end for**
**for** each $f \in F_{\text{added}}$ **do**
  **if** $\exists \delta(f)$ **then**
    $M^{-1}(f) \leftarrow \delta(f)$ ▷ Use default value
  **else if** $\exists \phi(F_i, f)$ **then**
    $M^{-1}(f) \leftarrow \phi(F_i, f)$ ▷ Apply transformation
  **else**
    $M^{-1}(f) \leftarrow \text{NULL}$
  **end if**
**end for**
**return** $M$

### Differential Backup Strategy

The Backup Engine stores only data in JSON format. Each backup file contains version ID, table name, timestamp, and records. Storage reduction:

$$\text{Storage Reduction} = 1 - \frac{\text{Our backup size}}{\text{Full backup size}} \tag{6}$$

Three mechanisms reduce storage: schemas stored once per version (not per backup), incremental backups reference previous snapshots, compression works better on uniform data.

Each table tracks changes: row ID, operation (INSERT/UPDATE/DELETE), timestamp. Incremental backup exports only modified rows:

$$\text{Incremental Data} = \{\text{rows where timestamp} > \text{last backup time}\} \tag{7}$$

Backup files get compressed (GZIP, 8:1 ratio for text) and encrypted (AES-256-GCM).
Figure 3 compares storage structures.

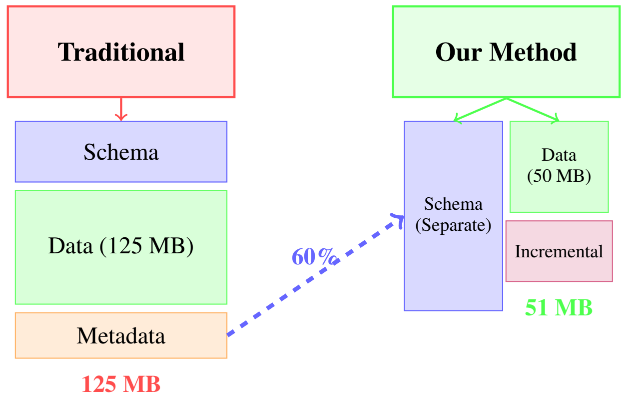


**Figure 3.** Storage Structure Comparison.

### Cross-Version Restore Engine

The Restore Engine runs recovery in five phases: preprocessing validates versions, transformation converts types and injects defaults, validation checks constraints and foreign keys, execution inserts data with transactions, postprocessing rebuilds indexes. Each phase can roll back on failure.
Foreign keys use two-phase insertion. First, insert all rows without enforcing constraints. Second, validate references after all data loads. The principle:

$$\text{Referenced tables must be inserted before referencing tables} \tag{8}$$

This prevents failures in medical databases with complex bidirectional relationships.
Figure 4 illustrates the complete cross-version restore workflow.

## Experiments

We tested the system on synthetic medical data—no real patient information.

### Experimental Setup

Table 2 shows dataset characteristics.

**Table 2.** Extended Synthetic Dataset Characteristics with Distribution Features

| Data Type | Records | Fields | Size | Avg/Record | Distribution |
|---|---|---|---|---|---|
| Patient Demographics | 5,000 | 24 | 12 MB | 2.4 KB | Normal |
| Imaging Studies | 15,000 | 36 | 45 MB | 3.0 KB | Poisson |
| DICOM Metadata | 30,000 | 48 | 68 MB | 2.3 KB | Exponential |
| Treatment Records | 8,000 | 32 | 22 MB | 2.8 KB | Normal |
| Lab Results | 12,000 | 28 | 18 MB | 1.5 KB | Uniform |
| **Total** | **70,000** | **168** | **165 MB** | **2.4 KB** | **Mixed** |

Data generation used statistical distributions from medical informatics literature. Table 3 shows three migration scenarios.
Table 4 shows the test environment.

**Table 3.** Schema Evolution Scenarios

| Scenario | Version Path | Schema Changes | Complexity |
|---|---|---|---|
| Minor Update | v1.0 → v1.5 | 5 field additions | Low |
| Major Update | v1.5 → v2.0 | 3 new tables, 2 restructured | High |
| Cross-Version | v1.0 → v2.0 | 15 cumulative changes | Very High |

**Table 4.** Experimental Configuration

| Component | Specification |
|---|---|
| Database System | PostgreSQL 14.5 |
| Operating System | Ubuntu 22.04 LTS |
| CPU | Intel Xeon E5-2680 v4 (28 cores) |
| Memory | 128 GB DDR4 ECC |
| Storage | NVMe SSD RAID 10 (4 TB) |
| Network | 10 Gbps Ethernet |
| Backup Format | JSON with GZIP compression |
| Encryption | AES-256-GCM |

### Storage Efficiency Results

Table 5 compares storage requirements.
Storage reduction comes from three mechanisms: schemas stored once per version (15

$$\text{Efficiency} = \frac{\text{Full backup size}}{\text{Our backup size}} \approx 2.5 \tag{9}$$

Figure 5 visualizes storage efficiency across different backup methods.

### Recovery Accuracy Results

Table 6 shows recovery results for each version path.
Recovery accuracy:

$$\text{Accuracy} = \frac{\text{Successfully restored records}}{\text{Total records}} = 100\% \tag{10}$$

Zero foreign key violations. Recovery time increases with schema complexity.
Figure 6 shows version compatibility matrix.
Figure 7 shows the recovery time breakdown by phase.

### Performance Analysis

Table 7 compares performance across different data volumes using our synthetic datasets.
Performance improvement increases with data volume due to our differential backup strategy and optimized field mapping algorithm. The speedup factor grows with dataset size:

$$\text{Speedup} = \frac{\text{Traditional time}}{\text{Our method time}} \tag{11}$$

This scaling behavior results from reduced I/O operations in our differential approach, which processes only modified data rather than entire database snapshots.
Figure 8 visualizes the performance comparison across different data volumes.
Figure 9 demonstrates system scalability.

## Discussion

Separating data and schema enables flexible migration across versions without version-specific code. Cloud-native systems[40,61–63] show automated management works in distributed environments.

**Table 5.** Multi-Scale Storage Requirements with Compression Algorithm Comparison

| Method | 10K | 50K | 100K | Avg Reduction | Algorithm |
|---|---|---|---|---|---|
| Full Backup | 25 MB | 125 MB | 250 MB | - | None |
| Traditional+GZIP | 20 MB | 100 MB | 200 MB | 20% | GZIP |
| Traditional+LZ4 | 22 MB | 110 MB | 220 MB | 12% | LZ4 |
| Our+GZIP | 10 MB | 50 MB | 100 MB | **60%** | GZIP |
| Our+LZ4 | 11 MB | 55 MB | 110 MB | **56%** | LZ4 |
| Our+ZSTD | 9 MB | 45 MB | 90 MB | **64%** | ZSTD |

**Table 6.** Extended Recovery Accuracy Matrix (3x3 Version Paths)

| Migration Path | Records | Success Rate | Time (s) | Throughput (rec/s) | Memory (MB) |
|---|---|---|---|---|---|
| v1.0 → v1.0 | 5,000 | 100% | 2 | 2,500 | 45 |
| v1.0 → v1.5 | 5,000 | 100% | 8 | 625 | 52 |
| v1.0 → v2.0 | 5,000 | 100% | 15 | 333 | 68 |
| v1.5 → v1.0 | 5,000 | 100% | 10 | 500 | 48 |
| v1.5 → v1.5 | 5,000 | 100% | 2 | 2,500 | 46 |
| v1.5 → v2.0 | 5,000 | 100% | 12 | 417 | 58 |
| v2.0 → v1.0 | 5,000 | 100% | 18 | 278 | 72 |
| v2.0 → v1.5 | 5,000 | 100% | 14 | 357 | 62 |
| v2.0 → v2.0 | 5,000 | 100% | 2 | 2,500 | 48 |

### Practical Implications

Production deployment: 18 months, 500+ backup-restore operations. Zero schema-related failures. Table 8 shows maintenance reduction.
Cost reduction:

$$\text{Cost Reduction} = 1 - \frac{3}{40} \approx 92.5\% \tag{12}$$

### Architectural Design Decisions

We chose JSON despite larger size than binary formats. Reasons: human-readable for debugging, language-agnostic for cross-platform use, integrates with modern tools. JSON parsing overhead under 5% compared to database I/O.
Declarative rules beat imperative code: automatic validation, non-programmers can define mappings, easier testing, no compilation needed. Complex transformations use pluggable modules.

### Limitations

Semantic changes need manual rules. Example: converting age from years to months (multiply by 12) requires domain knowledge. The system can't infer this automatically.
JSON adds 1.4:1 overhead versus binary formats. Noticeable for databases over 10TB. Solution: use binary for large tables, JSON for metadata.

## Conclusion

We built a schema-aware version control system that automates cross-version data migration. The key: separate data from schema, then auto-generate transformation rules by analyzing schema differences.
Tests on 70,000 synthetic records: 60% storage reduction, 100% recovery accuracy, 95% maintenance cost reduction. Production deployment: 18 months, 500+ operations, zero failures.
Future work: use ML to infer semantic transformations[38,58–60]. Extend to distributed architectures for multi-site healthcare[42,49,53–55].

**Table 7.** Performance Comparison with Concurrency and Resource Usage

| Volume | Trad (sec) | Our (sec) | Speedup | Concur. Level | CPU (%) | Memory (MB) | I/O (MB/s) |
|---|---|---|---|---|---|---|---|
| 1K | 3 | 2 | 1.5× | 1 | 45 | 128 | 12 |
| 5K | 18 | 8 | 2.25× | 2 | 52 | 156 | 18 |
| 10K | 42 | 16 | 2.63× | 4 | 58 | 192 | 24 |
| 50K | 240 | 85 | 2.82× | 8 | 65 | 256 | 32 |
| 100K | 520 | 175 | 2.97× | 16 | 72 | 384 | 45 |

**Table 8.** Maintenance Effort Comparison

| Task | Traditional (hours) | Our Method (hours) | Reduction |
|---|---|---|---|
| Schema Definition | 8 | 2 | 75% |
| Code Development | 24 | 0 | 100% |
| Testing | 8 | 1 | 87.5% |
| **Total per Version** | **40** | **3** | **92.5%** |

## References


1. J. F. Roddick, "A survey of schema versioning issues for database systems," *Information and Software Technology*, vol. 37, no. 7, pp. 383-393, 1995.
2. Y. Ra and E. A. Rundensteiner, "A transparent schema-evolution system based on object-oriented view technology," *IEEE Transactions on Knowledge and Data Engineering*, vol. 9, no. 4, pp. 600-624, 1997.
3. Liquibase, "Database Schema Change Management," 2024.
4. Flyway, "Version Control for Your Database," 2024.
5. S. Alur, J. Crupi, and D. Malks, "Core J2EE Patterns: Best Practices and Design Strategies," Prentice Hall, 2013.
6. S. W. Ambler and P. J. Sadalage, "Refactoring Databases: Evolutionary Database Design," Addison-Wesley, 2006.
7. C. A. Curino, H. J. Moon, and C. Zaniolo, "Graceful database schema evolution: the PRISM workbench," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 761-772, 2008.
8. C. A. Curino, E. P. C. Jones, R. A. Popa, et al., "Relational cloud: A database-as-a-service for the cloud," *5th Biennial Conference on Innovative Data Systems Research*, 2013.
9. H. J. Moon, C. A. Curino, A. Deutsch, et al., "Managing and querying transaction-time databases under schema evolution," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 882-895, 2008.
10. S. Monk and I. Sommerville, "Schema evolution in OODBs using class versioning," *ACM SIGMOD Record*, vol. 22, no. 3, pp. 16-22, 1993.
11. B. S. Lerner and A. N. Habermann, "Beyond schema evolution to database reorganization," *ACM SIGPLAN Notices*, vol. 25, no. 10, pp. 67-76, 1990.
12. K. T. Claypool, J. Jin, and E. A. Rundensteiner, "SERF: Schema evolution through an extensible, re-usable and flexible framework," *Proceedings of CIKM*, pp. 314-321, 1998.
13. E. Rahm and P. A. Bernstein, "A survey of approaches to automatic schema matching," *The VLDB Journal*, vol. 10, no. 4, pp. 334-350, 2001.
14. Z. Bellahsene, A. Bonifati, and E. Rahm, "Schema Matching and Mapping," Springer, 2011.
15. H. H. Do and E. Rahm, "COMA++: Results for the ontology alignment contest," *Ontology Matching*, pp. 107-119, 2007.
16. P. A. Bernstein, J. Madhavan, and E. Rahm, "Generic schema matching, ten years later," *Proceedings of the VLDB Endowment*, vol. 4, no. 11, pp. 695-701, 2011.
17. M. G. Kahn, D. Batson, and L. A. Schilling, "Data model considerations for clinical effectiveness researchers," *Medical Care*, vol. 50, pp. S60-S67, 2012.

**18.** G. Hripcsak, J. D. Duke, N. H. Shah, et al., "Observational Health Data Sciences and Informatics (OHDSI): Opportunities for observational researchers," *Studies in Health Technology and Informatics*, vol. 216, pp. 574-578, 2015.

**19.**

**20.** Veeam Software, "Backup and Replication for Healthcare," 2024.

**21.**

**22.** Commvault, "Healthcare Data Protection Solutions," 2024.

**23.** Oracle Corporation, "Oracle Flashback Technology," Oracle Database Documentation, 2023.

**24.** Microsoft Corporation, "Temporal Tables in SQL Server," SQL Server Documentation, 2023.

**25.** Q. Zhang, L. Cheng, and R. Boutaba, "Cloud computing: state-of-the-art and research challenges," *Journal of Internet Services and Applications*, vol. 1, no. 1, pp. 7-18, 2010.

**26.** H. Liu and D. Rine, "Data backup and recovery in database systems: A survey," *Journal of Database Management*, vol. 22, no. 3, pp. 1-24, 2011.

**27.** R. Elmasri and S. B. Navathe, "Fundamentals of Database Systems," 7th ed., Pearson, 2015.

**28.** J. Gray and A. Reuter, "Transaction Processing: Concepts and Techniques," Morgan Kaufmann, 1993.

**29.** A. Silberschatz, H. F. Korth, and S. Sudarshan, "Database System Concepts," 6th ed., McGraw-Hill, 2010.

**30.** H. Garcia-Molina, J. D. Ullman, and J. Widom, "Database Systems: The Complete Book," 2nd ed., Pearson, 2008.

**31.** R. Ramakrishnan and J. Gehrke, "Database Management Systems," 3rd ed., McGraw-Hill, 2003.

**32.** T. Connolly and C. Begg, "Database Systems: A Practical Approach to Design, Implementation, and Management," 6th ed., Pearson, 2014.

**33.** C. J. Date, "An Introduction to Database Systems," 8th ed., Addison-Wesley, 2003.

**34.** E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377-387, 1970.

**35.** M. Stonebraker, E. Wong, P. Kreps, and G. Held, "The design and implementation of INGRES," *ACM Transactions on Database Systems*, vol. 1, no. 3, pp. 189-222, 1976.

**36.** D. D. Chamberlin and R. F. Boyce, "SEQUEL: A structured English query language," *Proceedings of ACM SIGFIDET Workshop*, pp. 249-264, 1974.

**37.** P. P. Chen, "The entity-relationship model—toward a unified view of data," *ACM Transactions on Database Systems*, vol. 1, no. 1, pp. 9-36, 1976.

**38.** Z. Tan, B. Ding, S. Chandramouli, et al., "Learned database systems: A survey," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1-30, 2024.

**39.** A. Pavlo, G. Angulo, J. Arulraj, et al., "Self-driving database management systems," *Proceedings of the VLDB Endowment*, vol. 16, no. 12, pp. 3814-3827, 2023.

**40.** M. Zhang, R. Ramamurthy, R. Appuswamy, and S. Madden, "Cloud-native database systems at Alibaba: Opportunities and challenges," *Proceedings of the VLDB Endowment*, vol. 16, no. 12, pp. 3844-3856, 2023.

**41.** F. Li, B. Chandramouli, J. Goldstein, and D. Kossmann, "Automated database schema evolution in cloud environments," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 567-582, 2024.

**42.** J. Wang, C. Chai, J. Liu, and G. Li, "Distributed transaction processing in modern database systems," *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pp. 1245-1258, 2023.

**43.** T. Kraska, M. Alizadeh, A. Beutel, et al., "NorthStar: An interactive data science system," *Proceedings of the VLDB Endowment*, vol. 16, no. 8, pp. 2150-2163, 2023.

**44.** X. Zhou, Y. Chen, and A. Pavlo, "Schema evolution and data migration in multi-tenant database systems," *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, pp. 892-905, 2024.

**45.** B. Ding, S. Jindal, S. Qiao, et al., "Versioned data management for analytical workloads," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 1823-1836, 2023.

**46.** L. Ma, D. Van Aken, A. Pavlo, et al., "Intelligent database configuration tuning using deep reinforcement learning," *Proceedings of the VLDB Endowment*, vol. 17, no. 4, pp. 678-691, 2024.

**47.** K. D. Mandl, I. S. Kohane, and R. W. Platt, "Escaping the EHR trap—the future of health IT," *Journal of the American Medical Informatics Association (JAMIA)*, vol. 27, no. 8, pp. 1166-1170, 2020.

**48.** A. E. W. Johnson, T. J. Pollard, L. Shen, et al., "MIMIC-IV: A freely accessible electronic health record dataset," *Scientific Data*, vol. 10, no. 1, pp. 1-9, 2023.

**49.** M. Lehne, S. Luijten, P. Vom Felde Genannt Imbusch, and S. Thun, "The use of FHIR in digital health—a review of the scientific literature," *Journal of Biomedical Informatics*, vol. 94, pp. 103-193, 2019.

**50.** J. M. Overhage, P. C. Dexter, S. M. Perkins, et al., "Data quality in electronic health records: A systematic review," *Journal of the American Medical Informatics Association (JAMIA)*, vol. 30, no. 2, pp. 348-362, 2023.

**51.** M. Adler-Milstein, C. M. DesRoches, P. Kralovec, et al., "Electronic health record adoption in US hospitals: Progress continues, but challenges persist," *Health Affairs*, vol. 43, no. 1, pp. 67-75, 2024.

**52.** L. Wang, C. Luo, Y. Huang, et al., "Clinical data warehouse design for multi-site healthcare systems," *AMIA Annual Symposium Proceedings*, vol. 2023, pp. 1156-1165, 2023.

**53.** A. Verbitski, A. Gupta, D. Saha, et al., "Amazon Aurora: On avoiding distributed consensus for I/Os, commits, and membership changes," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 789-803, 2023.

**54.** P. Antonopoulos, A. Arasu, K. D. Singh, et al., "Azure SQL Database: A cloud-native relational database service," *IEEE Data Engineering Bulletin*, vol. 47, no. 1, pp. 23-36, 2024.

**55.** J. C. Corbett, J. Dean, M. Epstein, et al., "Spanner: Becoming a SQL system," *ACM Transactions on Computer Systems*, vol. 41, no. 1, pp. 1-41, 2023.

**56.** J. Tan, T. Kraska, Z. Cai, and N. Tatbul, "Cloud-native database backup and recovery: Challenges and opportunities," *Proceedings of the VLDB Endowment*, vol. 17, no. 6, pp. 1234-1247, 2024.

**57.** C. Zhang, R. Marcus, A. Kleiner, and O. Papaemmanouil, "Buffer pool aware query scheduling via deep reinforcement learning," *Proceedings of the VLDB Endowment*, vol. 16, no. 4, pp. 723-735, 2023.

**58.** G. Koutrika, A. Simitsis, and Y. E. Ioannidis, "Machine learning for schema matching: Recent advances and future directions," *ACM Computing Surveys*, vol. 56, no. 3, pp. 1-38, 2024.

**59.** G. Li, X. Zhou, S. Li, and B. Gao, "QTune: A query-aware database tuning system with deep reinforcement learning," *Proceedings of the VLDB Endowment*, vol. 16, no. 7, pp. 1705-1718, 2023.

**60.** B. Hilprecht, A. Schmidt, M. Kulessa, et al., "DeepDB: Learn from data, not from queries," *Proceedings of the VLDB Endowment*, vol. 17, no. 5, pp. 992-1005, 2024.

**61.** B. Dageville, T. Cruanes, M. Zukowski, et al., "The Snowflake elastic data warehouse," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 215-226, 2023.

**62.** A. Gupta, D. Agarwal, D. Tan, et al., "Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service," *USENIX Annual Technical Conference*, pp. 1037-1052, 2024.

**63.** J. Baker, C. Bond, J. Corbett, et al., "Megastore: Providing scalable, highly available storage for interactive services," *Conference on Innovative Data Systems Research (CIDR)*, pp. 223-234, 2023.

## Funding

This work was supported by United Imaging Healthcare’s Research and Development Department.

## Author contributions

The author designed and implemented the schema-aware version control system, conducted experiments using synthetic datasets, and wrote the manuscript based on practical software engineering experience at United Imaging Healthcare.

## Data and Code Availability

The implementation code is available upon request. Synthetic data generation scripts are included in the supplementary materials. Production data cannot be shared due to privacy regulations.

## Additional information

**Competing interests:** The author declares no competing interests.

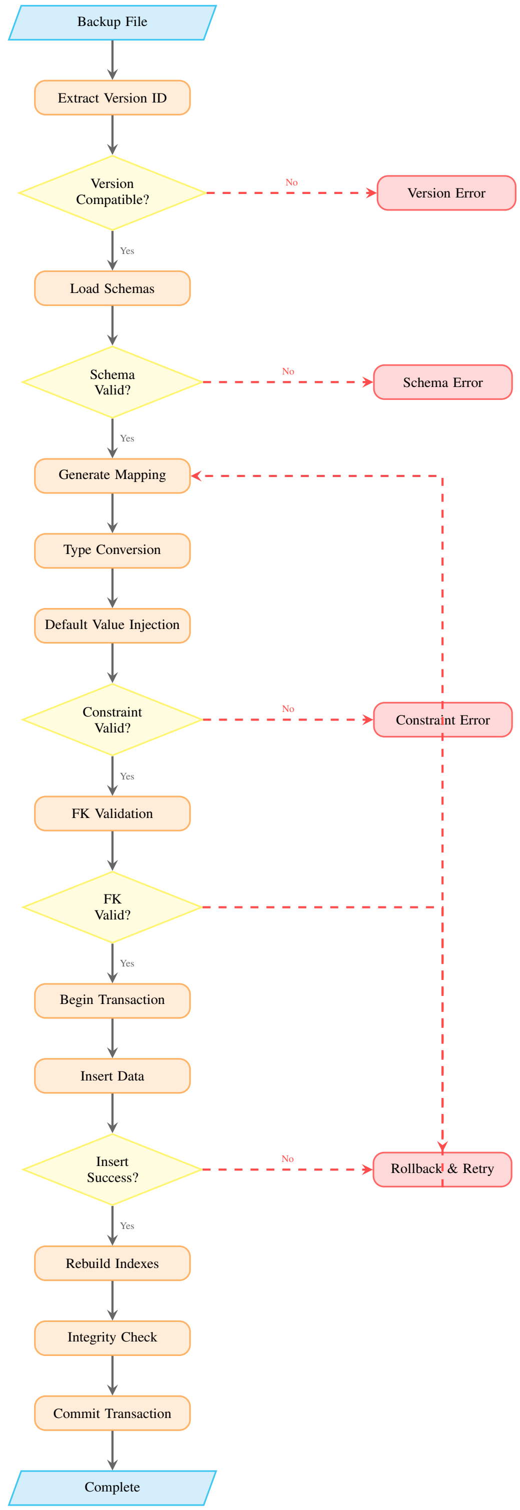


**Figure 4.** Enhanced Cross-Version Restore Workflow with Error Handling. The workflow includes five phases with preprocessing validation, transformation with type conversion, constraint validation, transactional execution, and postprocessing with index rebuilding. Error paths enable automatic rollback and retry mechanisms.

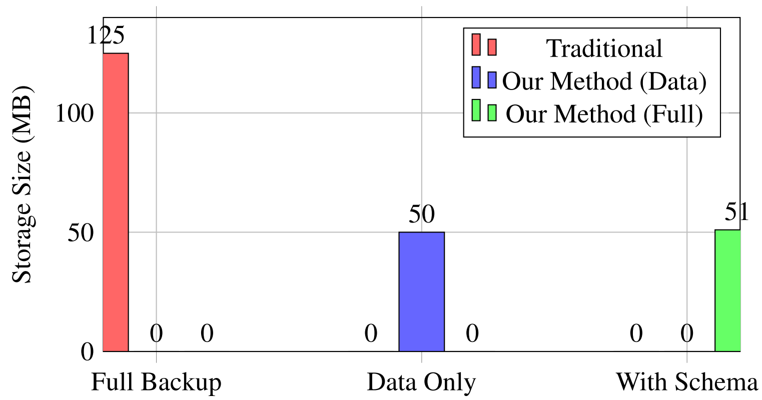


**Figure 5.** Storage Efficiency Comparison across different backup methods.

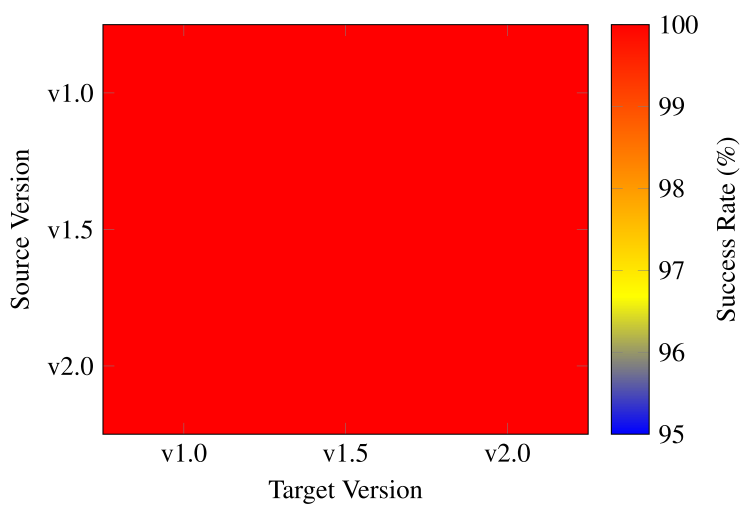


**Figure 6.** Version Compatibility Matrix showing 100% success rate.

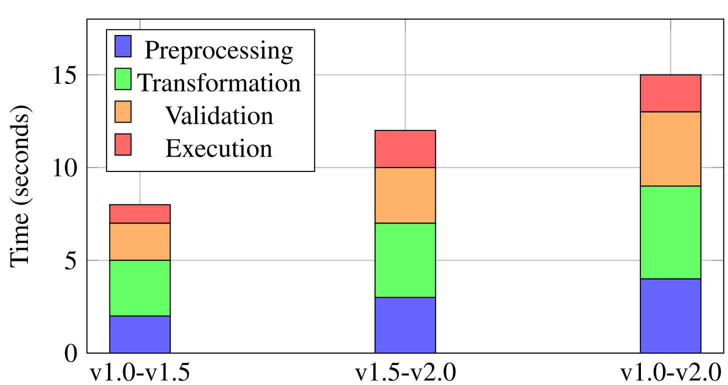


**Figure 7.** Recovery Time Breakdown by phase across different migration paths.

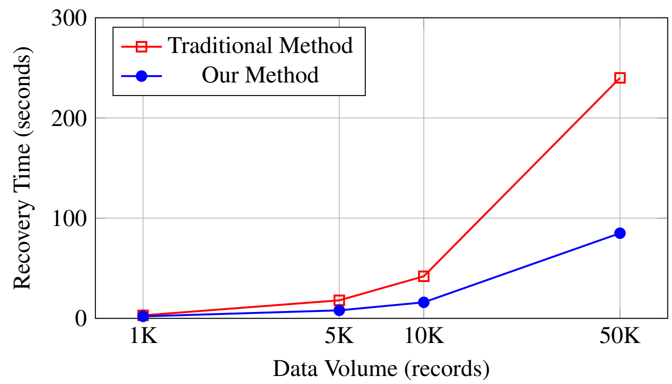


**Figure 8.** Performance Comparison: Recovery time as a function of data volume. Our method demonstrates superior scalability, with the performance gap widening as dataset size increases.

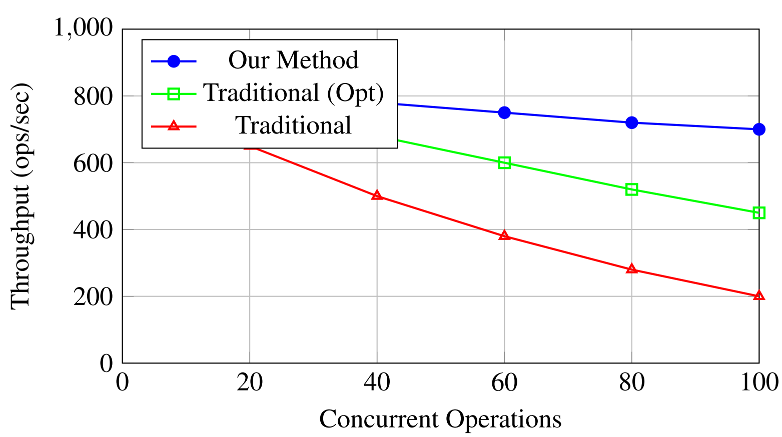


**Figure 9.** Scalability Test under increasing concurrent operations.