CRDTs and Databases: A Hands-On Tutorial

Paulo Sérgio Almeida
INESCTEC and U. Minho
Braga, Portugal
psa@di.uminho.pt

Nuno Faria
INESCTEC and U. Minho
Braga, Portugal
nuno.f.faria@inesctec.pt

José Pereira
INESCTEC and U. Minho
Braga, Portugal
jop@di.uminho.pt

ABSTRACT

Traditional isolation models such as Serializability or Snapshot Isolation, with a total commit order, are not compatible with large-scale, highly-available partition-tolerant services. Even a weaker model such as Parallel Snapshot Isolation, while allowing different commit orders at different sites, when defined for the traditional read-write API, must imply either aborting concurrent transactions on shared objects or losing updates. The only way out is to offer transactions over higher-level shared abstractions beyond a read-write API. Conflict-free Replicated Data Types (CRDTs) are such an abstraction. But CRDTs and relational databases have been developed in two worlds apart. Existing CRDT databases have ad hoc non-relational implementations that are not easily combined with database systems, in particular, regarding query optimization which is the mainstay of SQL systems. This tutorial covers how to bring these two worlds together. First, it provides solid coverage of what CRDTs are, and their main approaches, in a self-contained way. Then it addresses how CRDTs can be combined with the relational model, accessed by standard SQL, and implemented over standard relational databases, by the recent notion of Confilct-free Replicated Data Views.

PVLDB Reference Format:
Paulo Sérgio Almeida, Nuno Faria, and José Pereira. CRDTs and Databases: A Hands-On Tutorial. PVLDB, 19(12): 4922 - 4926, 2026.
doi:10.14778/3827998.3828151

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/dbr-haslab/crdts-and-databases.

1 INTRODUCTION

Relational databases aim to provide ACID guarantees, being Serializability [28] the more desirable isolation level for programmers. But for performance reasons, most databases offer the slightly weaker Snapshot Isolation [9] model.

With the advent of cloud services and geo-replication, always-on availability even in the presence of an inter-datacenter partition became important. Amazon's Dynamo [13] design stressed the importance of availability: "even the slightest outage has significant financial consequences and impacts customer trust." But from the CAP Theorem [18], we can have at most two of the three guarantees: strong Consistency (linearizability [19]), Availability, and Partition tolerance. This means that Serializability cannot be offered while ensuring availability under partitions.

More generally, traditional isolation models with a total commit order, such as Serializability or Snapshot Isolation, are not compatible with large-scale, highly-available partition-tolerant services. But even a weaker model such as Parallel Snapshot Isolation [34], while allowing different commit orders at different sites, when defined for the traditional read-write API, must imply either aborting concurrent transactions on shared objects or losing updates.

On the other hand, NoSQL data stores such as Dynamo, while allowing availability and avoiding loss of updates, were difficult to program and error prone, given a low level read-write based API, with the need for ad hoc reconciliation of concurrent updates.

The way to achieve highly-available partition tolerant systems is to offer transactions over higher-level shared abstractions beyond a read-write API. Conflict-free Replicated Data Types (CRDTs) [32] are such an abstraction. The essential concept is providing a higher level API, as in classic data types, but for distributed, replicated objects. They achieve availability through relaxing consistency and allowing immediate local replica updates and queries, with asynchronous communication to make replicas converge. Data type specific concurrency semantics and synchronization are specified as built-in (e.g., *add-wins set*), freeing programmers from writing ad hoc reconciliation code.

Since their appearance, CRDTs became popular in many contexts, namely in NoSQL data stores such as Riak [23] and Redis [30], and in collaborative applications such as Zed [33] or JupyterLab [22]. The latter case usually makes use of CRDTs through client-side library like Yjs [21] or Automerge [12]. Transactional support for CRDTs was added in several databases such as AntidoteDB [1] and SwiftCloud [29].

There has also been an increasing interest in coordination avoidance in database systems in general [5] and, specifically, in the combination of CRDTs with SQL database systems [24]. Table 1 lists a number of proposals for CRDTs in database systems or that have been, in some form, used together with database systems. However, these proposals tend to either restrict the variety of data types and merge strategies available to application developers, for example, by simply modeling relational tables as CRDT maps, or interact badly with SQL query optimization, by forcing early materialization and preventing operation reordering.

This tutorial covers how to bring these two worlds together. First, it provides a solid coverage of what CRDTs are, and their main approaches, in a self-contained way. Then it addresses how CRDTs can be offered in a relational interface, accessed by standard SQL, and implemented over standard relational databases, demonstrating this using the recent notion of Conflict-free Replicated Data Views [16] (CRDVs).

This work is licensed under the Creative Commons BY-NC-ND 4.0 International License. Visit https://creativecommons.org/licenses/by-nc-nd/4.0/ to view a copy of this license. For any use beyond those covered by this license, obtain permission by emailing info@vldb.org. Copyright is held by the owner/author(s). Publication rights licensed to the VLDB Endowment.
Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828151

1.1 Format and length

The tutorial is proposed for two sessions with a total duration of 3 hours and structured as follows:

(1) **Introduction to CRDTs** *(60 min.)*
   (a) Historical tour, consistency and the CAP theorem
   (b) Operation-based CRDTs
   (c) State-based CRDTs
   (d) Comparison of Approaches
(2) **CRDTs in Databases** *(30 min.)*
   (a) Key challenges and requirements
   (b) A survey of existing approaches
   (c) Conflict-free Replicated Data Views
(3) **CRDVs hands-on** *(60 min.)*
   (a) A tour of CRDVs in PostgreSQL
   (b) The CRDVs cookbook for application developers
   (c) Performance analysis
(4) **Open challenges** *(15 min.)*
(5) **Q&A** *(15 min.)*

1.2 Target audience and related tutorials

This tutorial is intended for a broad audience of professionals engaged in the development, operation, or research of database systems. For developers and operators, it provides an introduction to the theory of conflict-free replication and provides hands on experience with techniques that can be readily deployed. For researchers in database systems, it provides a solid background in distributed systems topics that are relevant to core challenges in current leading database systems, and it also points out outstanding challenges and interesting research directions.

A short version of this tutorial, with similar structure but much shorter duration (1 hour), was presented to *TUMuchData, the Munich Database Club*.[1] A related tutorial with similar duration has been presented at the *Third ACM Europe Summer School on Distributed and Replicated Environments (DARE 2025)*.[2] It included most topics in parts 2 and 3 of this proposal, but, as it was targeted at distributed systems graduate students and researchers, it omitted part 1 and instead included an introduction to query processing and optimization in database systems, which is not necessary for the typical VLDB audience.

1.3 Artifacts

The tutorial will use existing software that implements CDRVs in PostgreSQL and provides a range of demonstrations and benchmarks on various implementations of CRDTs in database systems [16]. Moreover, materials used for the presentation and the demonstrations are available online.[3]

2 OVERVIEW

The tutorial is structured in four main parts, including an in-depth introduction to replication with CRDTs; a survey of CRDTs in database systems, focusing on the CRDVs approach; a hands on demonstration of CRDVs on PostgreSQL; and finally, a discussion of open challenges.

[1] https://tumuchdata.club/events/38/
[2] https://dare-lisbon.github.io/program/#jose-orlando-pereira
[3] https://github.com/dbr-haslab/crdts-and-databases

2.1 Conflict-free Replicated Data Types (CRDTs)

The first part of the tutorial is an in-depth introduction to CRDTs. It is essentially based on a CRDT tutorial paper published in ACM *Computing Surveys* [2] by one of the tutorial authors. After a historical tour of the evolution from sequential data types to CRDTs, it presents in detail the two main approaches to CRDTs, operation-based and state-based, together with two important variations, the pure operation-based and the delta-state based. It provides solid coverage of the essential concepts, clarifies some misconceptions which frequently occur, and also presents some novel insights gained from considerable experience of the author in designing specific CRDTs and approaches to CRDTs.

Historical tour, consistency and the CAP theorem. Starts with a historical tour about the evolution from sequential data abstractions to CRDTs, along concurrent data abstractions, strongly consistent replication, the CAP theorem, the importance of causal consistency as the limit for highly available partition tolerant systems, optimistic replication, and the two main approaches (operation- vs state-based).

Operation-based CRDTs. Presents the standard execution model of operation-based CRDTs, discussing how sequential data types with commutative operations can be trivially reused as op-based CRDTs, and how data types with non-commutative operations can be addressed, by defining concurrent semantics for the data type. As explained, this may lead to outcomes that cannot be explained by any sequential execution. Sometimes, even the data type API itself is modified; the classic multi-value register example is presented. The notion of observed-cancel semantics is discussed as an important design choice, with the classic observed-remove set presented as an example. This motivates a discussion for the need to optimize CRDT implementations, which frequently start by being very naive. Then, an important special case of the op-based approach, *pure op-based CRDTs* [6, 7] are briefly presented. These are a subset of general op-based CRDTs, restricted to essentially sending only the operations themselves, and not arbitrary messages (resulting from transforming operations under the current state).

State-based CRDTs. Starts by a self-contained brief introduction to lattices and order, introducing the concept of join-semilattices as the basic ingredient for the state-based approach, with the lattice join for replica reconciliation, as well as state mutators that are inflations to achieve monotonic evolution of state. Conditions for reusing sequential data types as CRDTs are discussed, conveying the greater difficulty in reuse, compared with the op-based approach, due to non-idempotent operations, even for simple data types. The single-writer principle design pattern is presented and exemplified with classic CRDTs, such as counters. Causal CRDTs, making use of a causal context, are shown as the way to avoid tombstones and achieve observed-cancel semantics. Delta-state CRDTs [3] are then presented, in which mutators produce a delta-state, to be joined with the current state. These aim to achieve the best of both worlds, allowing unreliable messaging and having small messages to allow more frequent synchronization. But as discussed, naive delta propagation algorithms may easily fail to achieve the goals. The solution is then introduced, by optimal deltas and smart propagation through join decompositions.

Table 1: Summary of CRDT implementations in database systems [16].

Solution	Data Model	Interface	Implementation	Sync.	Conflict Resolution	Optimization
CRDV [16]	Relational	SQL	Views+Triggers	State	Custom (SQL)	Yes
Automerge [12]	Document	Custom API	Library (Multi-lang)	Operation[a]	LWW, MVR	No
Yjs [21]	Document	Custom API	Library (Multi-lang)	Both[b]	Highest Id	No
D. Types [17]	Text	Custom API	Library (Rust, JS)	Operation	-	No
OrbitDB [11]	Key-value	Custom API	Library (JS)	Operation[c]	LWW	No
RxDB [27]	Document	MQL[d]	Library (JS)	Operation	Highest Id	Simple[e]
Riak KV [36]	Key-value	Custom API	Internal (Erlang)	State	AW, EW, LWW, MVR	No
Redis [31]	Key-value	Custom API	Internal (C)	Operation	AW, LWW	No
AntidoteDB [4]	Key-value	Custom API	Internal (Erlang)	Operation	[A|R]W, [E|D]W, LWW, MVR	No
AntidoteSQL [4]	Relational	AQL	Library (Erlang)	Operation	[A|R]W, EW, LWW	Simple[e]
Pg_crdt[f] [35]	Relational	Custom API	PG Ext., Lib. (Multi-lang)	Operation	LWW, MVR	No
ElectricSQL [25]	Relational	SQL	Lib. (JS), Triggers+Views	State	LWW	Partial[g]
CRR [38]	Relational	SQL	Library (Erlang)	State	LWW	Partial[g]
CR-SQLite [37]	Relational	SQL	SQLite Extension	State	AW, LWW	Partial[g]
Dart sql_crdt [10]	Relational	SQL	Library (Dart)	State	LWW	Partial[g]
Lasp [26]	Key-value	Custom API	Library (Erlang)	State	Custom (Erlang)[h]	No

[a] As documents store all changes, it also supports replicating the entire state. [b] Operation-based for inserts, state-based for deletes [20]. [c] While the synchronization is operation-based, each operation contains the entire state. [d] MongoDB Query Language. [e] Support secondary indexes but are otherwise limited when compared to major relational DBMSs. [f] Using Automerge as the backend. [g] Unlike CRDV, conflict resolution is not considered by the optimizer. [h] Unlike CRDV, rules must defined at build time.

Comparison of Approaches. This part of the tutorial closes with a taxonomy and comparison of the approaches, in aspects such as open vs closed systems, reusing sequential data types, partition tolerance, state size, amortizing metadata overhead, messaging requirements, and consistency achieved.

2.2 CRDTs in database systems

The second part of the tutorial describes how CRDTs are combined with database systems for high concurrency and availability in geo-replicated databases. We discuss two desirable properties of implementing CRDTs in database systems, why most proposals listed in Table 1 have not been able to combine them, and how it can be achieved with Conflict-free Replicated Data Views (CRDVs) approach, which was awarded a *Honorable Mention for Best Paper* at ACM SIGMOD 2025 [16].

Key challenges and requirements. The first requirement is that an implementation supports a broad spectrum of merge semantics which defines what is the result when conflicting concurrent operations are executed. For example, on the one hand, in the well known shopping cart application [14], when there are conflicting *add* and *remove* operations of an item, the system should prefer to keep the item in the cart. On the other hand, when operating on a access control list and there are conflicting operations to add and remove a user, the system should conservatively not allow the user. These lead to the *add-wins* (AW) and *remove-wins* (RW) policies. In fact, one might even have cases in which the right policy depends on the data (e.g., AW in the shopping cart unless the item is discontinued) or the purpose of the computation (e.g., RW in the ACL unless it is for auditing purposes). The second requirement is that it enables effective query optimization. In a SQL database system, the combination of query optimization and database administration allows defining and using different materialization strategies and access paths, optimally for different workloads.

A survey of existing approaches. The discussion then focuses on how different existing implementations (do not) address these requirements, as well as their main features, as summarized in Table 1. The preferred approach is often to encapsulate existing CRDTs implementations as user defined data types. However, such data types are defined outside of the SQL schema, and the encapsulation makes their operation invisible to the query optimizer, that is, regarding how the current value is computed and when it is materialized. Others directly model relational tables as CRDTs maps and thus greatly restrict the possibility to model different merge semantics. Although the resulting relational model is exposed to the query optimizer, the operations needed for replication and merging of conflicting updates are still outside the scope of the query processor, usually resulting in eager materialization, which is not optimal for write intensive workloads.

Conflict-free Replicated Data Views. Introduces the Conflict-free Replicated Data Views (CRDVs) architecture [16] and discusses how it simultaneously achieves the main requirements. In a nutshell, CRDVs build stacked views on a replicated log. At the lowest level, tables store updated rows with additional meta-information necessary to manage replication. In addition, there is a view that computes the current data, exploiting causality represented as vector clocks. Finally, an application defined layer merges conflicting current data according to application semantics. The application queries are then redirected to this top layer, which provides the current merged data.

2.3 Hands-on with CRDVs

The third part of the tutorial is a hands-on demonstration of CRDVs [16] as implemented on PostgreSQL.[4] This implementation has won a *Honorable Mention for Best Artifact* at ACM SIGMOD 2025 [15].

[4] Available at https://github.com/nuno-faria/crdv.

A tour of CRDVs in PostgreSQL. Shows a step-by-step guide on how to install the CRDVs artifact, which can be followed by the audience. The opportunity is used to explain how each component is implemented in PostgreSQL, taking advantage of features such as logical replication, rules, and views. This is also useful for developers who aim to improve CRDVs or port them to other database management systems. A critical component of this implementation is the representation of vector clocks used to track causality in a way that can be manipulated by SQL queries, specifically, that can be indexed for efficiently accessed.

The CRDVs cookbook for application developers. Takes the perspective of the application developer, assembling an application making use of conflict-free replicated data. This section focuses on how different data structures and merge semantics can be implemented and combined to achieve the desired application goals.

Performance analysis. Ends with an evaluation of the resulting system and a comparison to competing alternatives and the PostgreSQL baseline. This includes a set of experiments that show the importance of configurable materialization strategies and how the SQL query optimizer combines application code with merge semantics and optimizes across the entire operation. It also includes a demonstration of simple benchmarks and a comparison with other implementation approaches.

2.4 Open challenges

The fourth and final part of the tutorial describes open challenges on highly available data processing, mainly, as resulting from the combination of CRDTs and SQL database systems.

Integration. First, we discuss how the architecture and programming abstractions in mainstream database systems can be improved to support this approach. Although CRDVs can be encapsulated with rules and views, the result is a leaky abstraction, in which the application developer is not shielded from possible errors. However, this fact is also the key to being able to express application-specific merge strategies. Therefore, it is an open problem to strike the optimal balance between encapsulation and flexibility.

Deployment. Second, we discuss how current database system implementations cope with the workloads generated by this approach. This includes how the query optimizer deals with queries resulting from the stacked views and the use of logical clocks, and motivates research in customizing optimizers specifically for this purpose. Moreover, the current replication infrastructure is not designed for the scale that is achievable and desirable with conflict-free replication. Therefore, it is interesting to improve existing replication mechanisms so that they can be used to build larger and more complex overlays, with arbitrary forwarding rules.

Correctness. Finally, building CRDVs takes some of the responsibility of handling concurrency to libraries and application code on top of the baseline database management system, namely, what changes are visible considering their vector timestamps. As concurrency is notoriously difficult to get right and to test, it is important to develop techniques and tools. Unfortunately, current transactional isolation checkers cannot handle arbitrary data types with complex semantics. Therefore, it is important to pursue configurable and extensible tools that can be used for this purpose [8].

3 PRESENTERS

Paulo Sérgio Almeida is an assistant professor at the Department of Informatics at University of Minho and a senior researcher at INESC TEC. He obtained a Ph.D. degree in Computer Science from Imperial College London in 1998. His research activities have been in the area of distributed systems, namely in causality tracking, eventually consistent databases, distributed aggregation, distributed graph algorithms, exactly-once messaging, and Bloom filters. The main focus of recent research has been on Conflict-free Replicated Data Types and on consistency models for distributed systems.

Nuno Faria completed his M.Sc. in 2020 and is now a researcher at INESC TEC and a teaching assistant at the University of Minho. His main research focuses on large-scale distributed data management, being the main author of recent publications in conferences such as SIGMOD and VLDB on transactional-analytical processing and distributed database systems (including an honorable mention for best paper). He is also an expert database systems software developer and administrator and member of the board of directors at Graph Data Council (formerly known as Linked Data Benchmark Council).

José Pereira is a research coordinator at INESCTEC and a professor at the University of Minho. His main research focus is on distributed data management, having over time addressed multiple topics such as data replication, transactions, distributed-parallel processing, and polyglot systems. He has published in top distributed systems conferences such as IEEE DSN and data management conferences such as VLDB and ACM SIGMOD. He has also been a member of the program committee of conferences such as IEEE DSN, ACM EuroSys, and ACM SIGMOD. He is the co-chair of the PaPoC workshop with EuroSys 2026.

ACKNOWLEDGMENTS

This work is financed by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia; Paulo Almeida under the support UID/50014/2025 (DOI:10.54499/UID/50014/2025); Nuno Faria and José Pereira by CMU–Portugal project ADAPQO (DOI:10.54499/2024.12585.CMU).

REFERENCES

[1] 2013. *AntidoteDB: A planet scale, highly available, transactional database built on CRDT technology.* https://github.com/AntidoteDB/antidote
[2] Paulo Sérgio Almeida. 2025. Approaches to Conflict-free Replicated Data Types. *ACM Comput. Surv.* 57, 2 (2025), 51:1–51:36. https://doi.org/10.1145/3695249
[3] Paulo Sérgio Almeida, Ali Shoker, and Carlos Baquero. 2018. Delta state replicated data types. *J. Parallel Distributed Comput.* 111 (2018), 162–173. https://doi.org/10.1016/j.jpdc.2017.08.003
[4] AntidoteDB. 2026. AntidoteDB Documentation - Datatypes in Antidote. https://antidotedb.gitbook.io/documentation/architecture/datatypes.
[5] Peter Bailis, Alan Fekete, Michael J Franklin, Ali Ghodsi, Joseph M Hellerstein, and Ion Stoica. 2014. Coordination avoidance in database systems. *Proceedings VLDB Endowment* 8, 3 (Nov. 2014), 185–196. https://doi.org/10.14778/2735508.2735509
[6] Carlos Baquero, Paulo Sérgio Almeida, and Ali Shoker. 2014. Making Operation-Based CRDTs Operation-Based. In *Distributed Applications and Interoperable Systems - 14th IFIP WG 6.1 International Conference, DAIS 2014, Held as Part of the 9th International Federated Conference on Distributed Computing Techniques,*

DisCoTec 2014, Berlin, Germany, June 3-5, 2014, Proceedings (Lecture Notes in Computer Science), Kostas Magoutis and Peter R. Pietzuch (Eds.), Vol. 8460. Springer, 126–140. https://doi.org/10.1007/978-3-662-43352-2_11

[7] Carlos Baquero, Paulo Sérgio Almeida, and Ali Shoker. 2017. Pure Operation-Based Replicated Data Types. *CoRR* abs/1710.04469 (2017). arXiv:1710.04469 http://arxiv.org/abs/1710.04469

[8] Manuel Barros, Alcino Cunha, Jose Pereira, and Eunsuk Kang. 2026. Reasoning about Transactional Isolation Levels with Isolde. arXiv:2604.00159 [cs.DB] https://arxiv.org/abs/2604.00159

[9] Hal Berenson, Phil Bernstein, Jim Gray, Jim Melton, Elizabeth O'Neil, and Patrick O'Neil. 1995. A critique of ANSI SQL isolation levels. *ACM SIGMOD Record* 24, 2 (1995), 1–10.

[10] Daniel Cachapa et al. 2026. Dart sql_crdt: Dart implementation of Conflict-free Replicated Data Types (CRDTs) using SQL databases. https://github.com/cachapa/sql_crdt.

[11] OrbitDB Community. 2026. OrbitDB: Peer-to-Peer Databases for the Decentralized Web. https://github.com/orbitdb/orbitdb.

[12] Automerge contributors. 2026. Automerge: Version control for your data. https://automerge.org/.

[13] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: amazon's highly available key-value store. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, Thomas C. Bressoud and M. Frans Kaashoek (Eds.). ACM, 205–220. https://doi.org/10.1145/1294261.1294281

[14] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's highly available key-value store. *Oper. Syst. Rev.* 41, 6 (Oct. 2007), 205–220. https://doi.org/10.1145/1323293.1294281

[15] Lisa Ehrlinger, Nuno Faria, and Anjiang Wei. 2026. Reproducibility Report for ACM SIGMOD 2025 Paper: 'CRDV: Conflict-free Replicated Data Views'. In *Reproducibility Reports of the 2025 International Conference on Management of Data* (Germany) *(SIGMOD ARI Reports '25)*. Association for Computing Machinery, New York, NY, USA, 69–73. https://doi.org/10.1145/3785021.3787996

[16] Nuno Faria and José Pereira. 2025. CRDV: Conflict-free Replicated Data Views. *Proc. ACM Manag. Data* 3, 1, Article 25 (Feb. 2025), 27 pages. https://doi.org/10.1145/3709675

[17] Seph Gentle et al. 2026. Diamond Types: The world's fastest CRDT. https://github.com/josephg/diamond-types.

[18] Seth Gilbert and Nancy A. Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News* 33, 2 (2002), 51–59. https://doi.org/10.1145/564585.564601

[19] Maurice Herlihy and Jeannette M. Wing. 1990. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.* 12, 3 (1990), 463–492. https://doi.org/10.1145/78969.78972

[20] Kevin Jahns et al. 2026. Yjs Internals. https://github.com/yjs/yjs/blob/main/INTERNALS.md.

[21] Kevin Jahns et al. 2026. Yjs: Shared data types for building collaborative software. https://github.com/yjs/yjs.

[22] Project Jupyter. 2026. JupyterLab Documentation. https://jupyterlab.readthedocs.io/en/latest/index.html.

[23] Rusty Klophaus. 2010. Riak Core: building distributed applications without shared state. In *ACM SIGPLAN Commercial Users of Functional Programming* (Baltimore, Maryland) *(CUFP '10)*. Association for Computing Machinery, New York, NY, USA, Article 14, 1 pages. https://doi.org/10.1145/1900160.1900176

[24] Shadaj Laddad, Conor Power, Mae Milano, Alvin Cheung, Natacha Crooks, and Joseph M Hellerstein. 2022. Keep CALM and CRDT on. *Proceedings VLDB Endowment* 16, 4 (Dec. 2022), 856–863. https://doi.org/10.14778/3574245.3574268

[25] Electric DB Limited. 2024. ElectricSQL - Sync for Modern apps. https://electric-sql.com.

[26] Christopher Meiklejohn and Peter Van Roy. 2015. Lasp: A language for distributed, coordination-free programming. In *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming*. 184–195.

[27] Daniel Meyer et al. 2026. RxDB: A fast, local first, reactive Database for JavaScript Applications. https://github.com/pubkey/rxdb.

[28] Christos H Papadimitriou. 1979. The serializability of concurrent database updates. *Journal of the ACM (JACM)* 26, 4 (1979), 631–653.

[29] Nuno M. Preguiça, Marek Zawirski, Annette Bieniusa, Sérgio Duarte, Valter Balegas, Carlos Baquero, and Marc Shapiro. 2014. SwiftCloud: Fault-Tolerant Geo-Replication Integrated all the Way to the Client Machine. In *33rd IEEE International Symposium on Reliable Distributed Systems Workshops, SRDS Workshops 2014, Nara, Japan, October 6-9, 2014*. IEEE Computer Society, 30–33. https://doi.org/10.1109/SRDSW.2014.33

[30] Redis. 2022. *Diving into Conflict-Free Replicated Data Types (CRDTs)*. https://redis.io/blog/diving-into-crdts

[31] Redis. 2026. Redis Enterprise - Active-Active geo-distribution (CRDTS-based). https://redis.io/active-active/.

[32] Marc Shapiro, Nuno M. Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-Free Replicated Data Types. In *Stabilization, Safety, and Security of Distributed Systems - 13th International Symposium, SSS 2011, Grenoble, France, October 10-12, 2011. Proceedings (Lecture Notes in Computer Science)*, Xavier Défago, Franck Petit, and Vincent Villain (Eds.), Vol. 6976. Springer, 386–400. https://doi.org/10.1007/978-3-642-24550-3_29

[33] Nathan Sobo. 2022. How CRDTs make multiplayer text editing part of Zed's DNA. https://zed.dev/blog/crdts.

[34] Yair Sovran, Russell Power, Marcos K Aguilera, and Jinyang Li. 2011. Transactional storage for geo-replicated systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP '11)*. Association for Computing Machinery, New York, NY, USA, 385–400. https://doi.org/10.1145/2043556.2043592

[35] Supabase. 2024. Pg_crdt - POC CRDT support in Postgres. https://github.com/supabase/pg_crdt.

[36] Basho Technologies. 2026. Riak KV Documentation - Data Types. https://docs.riak.com/riak/kv/latest/developing/data-types/index.html.

[37] Matt Wonlaw et al. 2026. Cr-SQLite: Convergent, Replicated SQLite. Multi-writer and CRDT support for SQLite. https://github.com/vlcn-io/cr-sqlite.

[38] Weihai Yu and Claudia-Lavinia Ignat. 2020. Conflict-free replicated relations for multi-synchronous database management at edge. In *2020 IEEE International Conference on Smart Data Services (SMDS)*. IEEE, 113–121.