Different Ways of Representing History

Historical data refers to past records that capture changes over time.

It's important as it is crucial for auditing, analytics, reporting, machine learning, and compliance.

Various models help store, retrieve, and track historical changes in data.

Common Techniques

There are different ways to manage history in databases. The most common techniques include:

- Slowly Changing Dimensions (SCD)
- Temporal Tables
- Audit Logging
- Event Sourcing

Slowly Changing Dimensions (SCD)

Used in Data Warehousing to track changes in data.

- SCD Type 1: Overwrites old data.

CustomerID	Name	Address
101	Alice	New York

- SCD Type 2: Stores multiple versions with validity dates.

CustomerID	Name	Address	Valid_From	Valid_To
101	Alice	New York	2023-01-01	2024-02-01
101	Alice	Los Angeles	2024-02-02	NULL

- SCD Type 3: Stores the latest and previous value.

CustomerID	Name	Current_Address	Previous_Address
101	Alice	Los Angeles	New York

Analysis (SCD)

Type	Description	Advantages	Disadvantages	PostgreSQL Implementation
SCD Type 1 (Overwrite)	Overwrites old data with new values	Simple, saves storage space	No history tracking	UPDATE customers SET address = 'Los Angeles' WHERE customerid = 101;
SCD Type 2 (Full History)	Maintains historical records with validity dates	Complete history is preserved	Increased storage, complex queries	UPDATE customers_scd2 SET valid_to = '2024-02-01' WHERE customerid = 101 AND valid_to IS NULL; INSERT INTO customers_scd2 (customerid, name, address, valid_from, valid_to) VALUES (101, 'Alice', 'Los Angeles', '2024-02-02', NULL);
SCD Type 3 (Limited History)	Stores only the latest and previous value	Small storage footprint, easy access to previous value	Only keeps one historical record	UPDATE customers_scd3 SET previous_address = current_address, current_address = 'Los Angeles' WHERE customerid = 101;

Temporal Data Models

Temporal databases are designed to track time-based changes in data. There are three types:

- **Valid-Time Model:** Tracks real-world validity.

EmployeeID	Name	Salary	Valid_From	Valid_To
201	Bob	50000	2023-01-01	2024-01-01
201	Bob	60000	2024-01-02	NULL

- **Transaction-Time Model:** Tracks when changes were recorded.

EmployeeID	Name	Salary	Transaction_Timestamp
201	Bob	50000	2023-01-02
201	Bob	60000	2024-01-02

- **Bi-Temporal Model:** Stores both valid and transaction time.

EmployeeID	Name	Salary	Valid_From	Valid_To	Recorded_On
201	Bob	50000	2023-01-01	2024-01-01	2023-01-02
201	Bob	60000	2024-01-02	NULL	2024-01-02

Analysis (Temporal Data Models)

Model	Description	Advantages	Disadvantages	PostgreSQL Implementation
Valid-Time Model	Tracks when a record is valid in the real world	Useful for time-based data analysis	Requires proper indexing for fast querying	CREATE TABLE employees (employeeid INT, name TEXT, salary INT, valid_from TIMESTAMPTZ DEFAULT now(), valid_to TIMESTAMPTZ DEFAULT 'infinity');
Transaction-Time Model	Tracks when data was recorded in the database	Enables rollback to previous states	More storage needed for logs	Implemented via system-versioned tables
Bi-Temporal Model	Combines both valid-time and transaction-time tracking	Most comprehensive history tracking	High storage and processing cost	CREATE TABLE employees_bitemporal (employeeid INT, name TEXT, salary INT, valid_from TIMESTAMPTZ, valid_to TIMESTAMPTZ, recorded_from TIMESTAMPTZ DEFAULT now(), recorded_to TIMESTAMPTZ DEFAULT 'infinity');

Audit Logging & Change Tracking

- Maintains logs for database modifications.
- Useful for compliance and debugging.

ChangeID	TableName	Action	ChangedBy	Timestamp
1	Employee	UPDATE	Admin	2024-02-10

Analysis (Audit Logging & Change Tracking)

Technique	Description	Advantages	Disadvantages	PostgreSQL Implementation
Audit Logging	Captures changes made to database records	Good for compliance and debugging	Requires additional tables and storage	PostgreSQL provides built-in logging using pgAudit extension
Change Tracking	Maintains a history of modifications	Helps track who changed what and when	May slow down write operations	CREATE TABLE audit_log (changeid SERIAL PRIMARY KEY, tablename TEXT, action TEXT, changedby TEXT, timestamp TIMESTAMPTZ DEFAULT now());

Event Sourcing

- Instead of storing the latest state, every action is stored as an event.
- Used in banking transactions, inventory management, and auditing.

EventID	AccountID	EventType	Amount	Timestamp
1	501	Deposit	1000	2023-06-01
2	501	Withdraw	500	2023-06-10

Analysis (Event Sourcing)

Concept	Description	Advantages	Disadvantages	PostgreSQL Implementation
Event Sourcing	Stores changes as a sequence of events instead of state updates	Full history is preserved	Querying current state requires reconstructing events	CREATE TABLE events (eventid SERIAL PRIMARY KEY, entityid INT, eventtype TEXT, data JSONB, timestamp TIMESTAMPTZ DEFAULT now());

Challenges in Storing Historical Data

Challenge	Explanation
Data Volume	Increased storage needs.
Query Complexity	Fetching historical states can be slow.
Storage Costs	Keeping old records takes up space.

Conclusion

Technique	Best Use Case	Main Challenge
SCD (Slowly Changing Dimensions)	Data Warehousing, Tracking Customer Changes	Can become storage-intensive
Temporal Tables	Financial Systems, Policy Tracking	Querying performance issues
Audit Logging	Compliance, Security	Can slow down transactions
Event Sourcing	Banking, E-commerce	Complexity in reconstructing state

Thank you