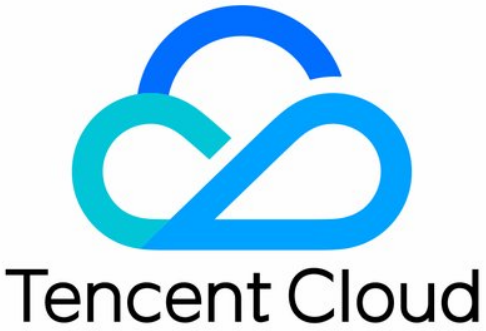
TencentDB for TcaplusDB Practice Tutorial



Copyright Notice

Trademark Notice

Service Notice

This document provides an overview of the as-is details of Tencent Cloud's products and services in their entirety or part. The descriptions of certain products and services may be subject to adjustments from time to time.

The commercial contract concluded by you and Tencent Cloud will provide the specific types of Tencent Cloud products and services you purchase and the service standards. Unless otherwise agreed upon by both parties, Tencent Cloud does not make any explicit or implied commitments or warranties regarding the content of this document.

Contact Us

We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

Contents

Practice Tutorial
Best Practice for Table Structure Design

Last updated: 2024-10-15 17:52:06

Database design plays a pivotal role in the entire software development process. This article will introduce the principles and schemes to be addressed in database design.

Criteria for Table Structure Definition

- It is not recommended for column names and table names to be too long. It is best to keep them within 32 bytes and free of special characters.
- Naming normalization of column names and table names should aim to express the actual meaning clearly and avoid excessive abbreviations.
- Pay attention to selecting appropriate data types to prevent precision loss.
- The choice of primary key and sharding factor should have a high degree of discretization to facilitate CLB and scaling processes.
- More indexes are not always better; create indexes specifically based on inquiries to avoid duplication with primary key definitions.
- For IP address fields in business, it is recommended to use int type.
- For time types, it is better to choose long type to store seconds.
- Fields that are logically related should maintain atomicity in operations and are advised to be merged into one table.
- If performance is crucial, appropriate redundant data design can be applied.
- The choice of primary key and sharding factor should have a high degree of discretization to facilitate scaling processes.
- Primary keys should be highly readable and not use a binary type to facilitate review and issue tracking.
- Primary key field sizes should not be too large. Use the shortest possible primary keys to speed up query performance.
- Field sizes should be defined according to actual use. Avoid defining large sizes when actual usage is small.
- All tables and fields need to have comments added.

Design principles related to game business

- For generating globally unique IDs, it is recommended to use the increment operation.
- To avoid database (DB) overload in architecture design, consider adding a queuing mechanism.
- Data content with relevance should be placed in a single table to avoid data inconsistency.
- The weight of tables needs to be separated. It is not recommended to put all features in one table; independent features should be considered for independent tables.
- If a table is used frequently and the records are large, consider designing a brief table to avoid increasing DB load by directly retrieving the raw data table.
- Lobby chat can use shared memory, and in-game chat can be pushed in real-time; it is not recommended to store in DB. For supporting offline private chat, consider using DB.
- Utilize the array data structure provided by DB, e.g., for historical performance, emails, and report records. This ensures data elimination and supports TopN operations based on insertion order.
- When designing leaderboards, if a sorting component can be used, then use it directly; if it needs to be implemented by the gameserver, it is recommended to asynchronously persist ranking results to the DB.
- For operations that are marginal and time-consuming in the game, it is recommended to handle them with a standalone process to avoid affecting the main logic of the gameserver.
- In game business with long open cycles and complex logic, changes in logical data structures are common during development. For scalability and easy maintenance, it is recommended that some changeable data structures be designed as blobs in the game data table, serialized and stored in the database to avoid frequent changes to DB tables due to data structure changes.

TDR table definition

- The primary key field requires high discreteness to facilitate the distribution of requests to multiple access layer nodes by the gameserver.
- When defining the table, it is recommended that the index key should not be identical to the primary key; if they are the same, it will consume network and disk resources.
- When defining arrays of common fields, add the refer attribute (count is the defined size, refer is the actual size) to facilitate future expansion of the count size and reduce network transmission and disk occupancy of data.
- The value fields should primarily be primary fields to reduce field nesting. The nested depth should be limited to 3.
- It is not recommended to use the binary type for primary key fields, as it is not conducive to troubleshooting.

- The number of indexes in a single table definition is limited to 8. It is recommended to use 2 to 3 indexes, set according to actual needs; too many indexes may reduce overall performance.

Protocol Buffers table definition

- The primary key field requires high discreteness to facilitate the distribution of requests to multiple access layer nodes by the gameserver.
- Supports struct types with nested Definition `non-primary key` fields. Excessive nesting depth will impact data access performance.

Examples of poor design

- Design inconsistent with requirements
  Large modification amounts, especially when the project is close to launch, resulting in higher modification costs.
- Low performance
  Excessive associations among tables with large data volumes; lack of reasonable field designs for queries leading to complex SQL query statements; ineffective methods for handling tables with large data volumes; abuse of views, etc.
- Data integrity loss
  Unreasonable design of associated fields among tables with primary–foreign key relationships, causing errors or imperfections during update and delete operations; use of deleted or lost data.
- Poor scalability
  Table design is too tightly bound to business, making it too single-purpose, resulting in poor scalability and modifiability, unable to meet new requirements.
- Excessive unnecessary data redundancy
  Storing too much useless garbage data, which not only occupies resources but also affects query efficiency.
- Not conducive to computation or statistics
  Lack of necessary connectivity or statistical fields, or fields for computational statistics are scattered across multiple tables, making computation and statistical steps cumbersome, or even impossible.
- Lack of detailed data recording information
  Missing necessary fields, making it impossible to track data changes, user operations, and also unable to perform data analysis.
- High coupling among tables
  Overly tight associations among multiple tables, causing changes in one table to affect other tables.

- Field design considerations are inadequate
  Field lengths are too short or field types are too specific, leading to limited flexibility and scalability.

Best Practice for Database Interaction

Last updated: 2024-10-15 17:52:25

Best Practices for Game Servers

- In use cases where not all fields need to be read from or written to, we recommend reading and updating only the required fields to avoid fetching invalid data. This can reduce the size of data transmitted over a network and reduce the number of disk reads and writes.
- If the data record needs to be returned when a write operation is performed, it is recommended to set result_flag as needed. For example, if the updated data record is required after an update operation, the result_flag should be set to 2. If the data record is not required during a write operation, the result_flag is set to 0.
- For batch processing command words, such as batchget, listgetall, and getbypartkey, it is recommended to fetch data records according to offset and limit. The recommended limit is 200, and subpackage return should be enabled on the game server side.
- For list-related tables, when a listaddafter operation is performed, a phase-out rule should be set in case that the list is full. It is recommended for the rule to add a new record to the front and delete a record from the rear, or vise versa.
- For list-related tables, a correct index should be passed for listreplace, listdelete, and listbatchdelete operations.
- Before the game server reads data, make sure that the value fields of the data record to be read are not empty. For example, when there are three value fields: A, B, and C, if the game server only gets A and B, then it indicates that field C is empty. Please check the fields before reading a data record.
- The game server should control the timeout locally, and we do not recommend determining the timeout based on the response package sent from the game server.
- When the game server performs traversal, we recommend accessing data from secondary nodes at the storage layer, so as to avoid affecting the performance of primary nodes.
- It is not recommended to perform memset operation on large fields to avoid consuming CPU resources. Set the values of some fields for large data structures to 0 before initialization.
- When processing requests to TcaplusDB and responses from TcaplusDB, the game server should use the divide-and-conquer method to process partial requests sent to TcaplusDB first and then process responses returned from TcaplusDB.

Best Practices for System Design

- Create separate tables for the frequently used fields and the fields on which a one-time atomic operation needs to be performed.
- When the game server writes data back, we recommend limiting traffic and discretize the data by time.
- It is recommended to support dynamic modification of the table structure and provide a table data conversion plugin.
- Rollback to the point in time of your cold backup or to an exact time at the table/logging level in all-server/all-region mode is supported.
- Nodes at the access layer and the storage layer can be dynamically scaled for both all-server/all-region mode and multi-server/multi-region mode. We recommend the multi-server/multi-region mode.
- Fields with a logical relation should be merged into one table to avoid distributed transaction problems.
- We recommend enabling the compression feature, including the compression of request/response packages and logs.