All Categories

- HR Policies (40)
- Employee Performance Management System (0)
- Training (16)
- Utilities (2)
- Knowledge Sharing (15)
  - Mobile (0)
  - Others / Misc (6)
  - Web (3)
  - Database Guidelines
  - Hardware - Software Performance Guidelines
  - Hosting Guidelines
  - HTML-CSS-JS Guidelines
  - Neosoft Secure Coding Practices
  - Security Guidelines

There are no categories to display in knowledge base.

Recently Viewed

- Business Conduct And Ethics
- Working Hours
- Workplace Policies
- Guidelines for Working in Night Shifts
- Database Guidelines

Home » Group Categories » Knowledge Sharing

Database Guidelines

Article Number: 119 | Rating: 4/5 from 1 votes | Last Updated: Wed, Mar 30, 2016 at 1:03 PM

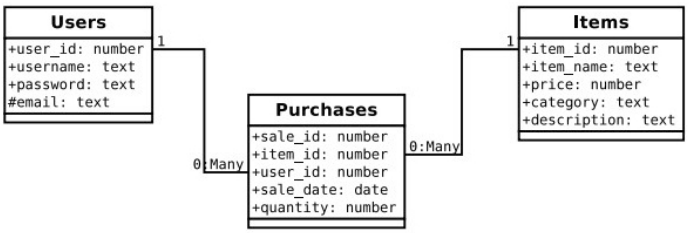
Database Selection criteria

- **Relational Databases**

  A relational database is a type of database that organizes data into tables, and links them, based on defined relationships. These relationships enable you to retrieve and combine data from one or more tables with a single query.

  Relational databases use the notion of databases separated into tables where each column represents a field and each row represents a record. Tables can be related or linked with each other with the use of foreign keys or common columns.Relational databases use the notion of databases separated into tables where each column represents a field and each row represents a record. Tables can be related or linked with each other with the use of foreign keys or common columns.

  The design of a database schema can be visualised using following diagram



ACID properties

An important aspect of relational databases which guarantees the reliability of transactions is their adherence to the ACID properties: Atomicity, Consistency, Isolation, Durability. Each property is explained below in the context of databases.

- Atomicity: Either all parts of a transaction must be completed or none.
- Consistency: The integrity of the database is preserved by all transactions. The database is not left in an invalid state after a transaction.
- Isolation: A transaction must be run isolated in order to guarantee that any inconsistency in the data involved does not affect other transactions.
- Durability: The changes made by a completed transaction must be preserved or in other words be durable.

There are following different types of databases available.

  - MySQL / MariaDB
    - Default Choice with Innodb / Extra DB
    - Suitable for any CRUD application without heavy workload.
    - Suitable for read optimized application.
  - SQLite
  - MSSQL
  - Oracle
  - PostgresSQL

- **Non relational Databases**

  There are many types of non-relational or NoSQL databases, each type tries to solve different problem faced by RDBMS systems.

  Here are some types of NoSQL databases and software which implement those types

  - **Document** - Store data in document format such as JSON and XML, no structure is forced on data Example: MongoDB, RethinkDB, CouchDB
  - **Key-Value** - Data is stored in key-value pairs, speed of retrieval is emphasized. Softwares like Redis, Memcache store data in memory. Example: MemcacheDB, Redis, OrientDB, Riak, Dynamo,
  - **Graph** - Uses graph structures for semantic queries with nodes, edges and properties to represent and store data. Benefit of using graph database is the speed & scalability of database operations like finding friend of friends on large dataset. Examples: Neo4j, Allegro, OrientDB
  - **Column** - Stores data in columns instead of rows to speed up aggregate operations such as AVG, SUM. Examples: Vertica, Cassandra, HBase
  - **Multi-Model** - Implement multiple models, for example OrientDB allows data to be stored in Key-Value, Graph as well as SQL.

Comparison between SQL and NoSQL

Sr.	SQL	NoSQL
1	Called as Relational Databases (RDBMS)	Called as non-relational or distributed database
2	Table based	Document based, key-value pairs, graph databases or wide-column stores.
3	Have predefined schema.	Have dynamic schema.
4	Are vertical scalable	Are horizontally scalable
5.	SQL databases uses SQL (structured query language) for defining and manipulating the data, which is very powerful.	In NoSQL database, queries are focused on collection of documents. Sometimes it is also called as UnQL (Unstructured Query Language). The syntax of using UnQL varies from database to database

6.	SQL database examples: MySql, Oracle, Sqlite, Postgres and MS-SQL.	NoSQL database examples: MongoDB, BigTable, Redis, RavenDb, Cassandra, Hbase, Neo4j and CouchDb
7	For complex queries: SQL databases are good fit for the complex query intensive environment.	NoSQL databases are not good fit for complex queries. On a high-level, NoSQL don't have standard interfaces to perform complex queries, and the queries themselves in NoSQL are not as powerful as SQL query language.
8	For the type of data to be stored: SQL databases are not best fit for hierarchical data storage.	NoSQL database fits better for the hierarchical data storage as it follows the key-value pair way of storing data similar to JSON data.
9	SQL databases are vertically scalable. You can manage increasing load by increasing the CPU, RAM, SSD, etc, on a single server.	NoSQL databases are horizontally scalable. You can just add few more servers easily in your NoSQL database infrastructure to handle the large traffic.

Design requirement

- Column commenting and data type
    - Comments must not too long nor too short. It must be descriptive enough.
    - Data type definition must be related to content type length. eg for name, length must not be more than 100 or for status column use boolean instead of int and so on.
- i18n - Set utf8_general_ci
- Design tool (Workbench)
- Normalization
- Naming conventions
    - Frameworks / CMS based
    - Standard practises
        - Plural names for tables(Pascal casing), unless specified otherwise
        - Singular names for columns (Underscore separated)
        - Use table prefixes on shared database server, avoid prefixes

          otherwise
        - Don't use spaces in naming any database entities.

Development

- Development styles
    - Business logic in SP
    - Business logic in SP + Query (Parameterized query)
    - Business logic in ORM
- Development Guidelines
    - We need to add all the data here.
- Migration or versioning

Debugging and troubleshooting

Deployment

Documentation

- Tools

Security

- Disable networking
- Securing MySQL Accounts

  The MySQL installation procedure creates a database named mysql containing the grant tables that the server uses to determine which MySQL accounts can do what. In particular, the user table in the mysql database lists all valid accounts and indicates which global privileges they have, if any. This section provides some guidelines that you can use to evaluate existing accounts, and that you should keep in mind when creating new accounts. These guidelines apply to servers running on any platform.

A general set of principles for securing the MySQL accounts listed in your grant tables is as follows:

- Remove anonymous accounts
- Make sure that each account has a password
- Don't grant global privileges unnecessarily
- Avoid using wildcards in the hostname value associated with accounts
- Instruct your MySQL users how to keep their passwords secure

- Users & Privileges
    - Define who can access database.
    - Define what users can do when inside database.
    - No support for user groups
        - Group-like mappings available through some plugins

- SQL injection
- Password should be salt based
- Avoid common usernames like admin
- Avoid common password like p@ssw0rd.
- Use password check dictionary
- Use symmetric encryption for highly sensitive data
- Use GUID/UUID as primary key, if exposing ID will cause security issues.

Maintenance

- Log size
    - Define max log size for error logs and access logs
    - Set expiration date on logs
- temp file size
    - Define max log size for error logs and access logs
- Zombie process
- Backup and recovery
    - Automated incremental backup should be schedule and should be taken offsite
    - Use Repair tables, Optimize tables and Check tables queries, if unsuccessful should be recover from backup location.
    - If shared server use backup plugins / scripts for CMS or any other framework.

Optimization (https://dev.mysql.com/doc/refman/5.5/en/optimization.html)

Database performance depends on several factors at the database level, such as tables, queries, and configuration settings. These software constructs result in CPU and I/O operations at the hardware level, which you must minimize and make as efficient as possible. As you work on database performance, you start by learning the high-level rules and guidelines for the software side, and measuring performance using wall-clock time. As you become an expert, you learn more about what happens internally, and start measuring things such as CPU cycles and I/O operations.

Optimizing at the Database Level

The most important factor in making a database application fast is its basic design:

- Are the tables structured properly? In particular, do the columns have the right data types, and does each table have the appropriate columns for the type of work? For example, applications that perform frequent updates often have many tables with few columns, while applications that analyze large amounts of data often have few tables with many columns.
- Are the right indexes in place to make queries efficient?
- Are you using the appropriate storage engine for each table, and taking advantage of the strengths and features of each storage engine you use? In particular, the choice of a transactional storage engine such as InnoDB or a non transactional one such as MyISAM can be very important for performance and scalability.

  Note

  In MySQL 5.5 and higher, InnoDB is the default storage engine for new tables. In practice, the advanced InnoDB performance features mean that InnoDB tables often outperform the simpler MyISAM tables, especially for a busy database.
- Does each table use an appropriate row format? This choice also depends on the storage engine used for the table. In particular, compressed tables use less disk space and so require less disk I/O to read and write the data. Compression is available for all kinds of workloads with InnoDB tables, and for read-only MyISAM tables.
- Does the application use an appropriate locking strategy? For example, by allowing shared access when possible so that database operations can run concurrently, and requesting exclusive access when appropriate so that critical operations get top priority. Again, the choice of storage engine is significant. The InnoDB storage engine handles most locking issues without involvement from you, allowing for better concurrency in the database and reducing the amount of experimentation and tuning for your code.
- Are all memory areas used for caching sized correctly? That is, large enough to hold frequently accessed data, but not so large that they overload physical memory and cause paging. The main memory areas to configure are the InnoDB buffer pool, the MyISAM key cache, and the MySQL query cache.

Optimizing at the Hardware Level

Any database application eventually hits hardware limits as the database becomes more and more busy. A DBA must evaluate whether it is possible to tune the application or reconfigure the server to avoid these bottlenecks, or whether more hardware resources are required. System bottlenecks typically arise from these sources:

- Disk seeks. It takes time for the disk to find a piece of data. With modern disks, the mean time for this is usually lower than 10ms, so we can in theory do about 100 seeks a second. This time improves slowly with new disks and is very hard to optimize for a single table. The way to optimize seek time is to distribute the data onto more than one disk.
- Disk reading and writing. When the disk is at the correct position, we need to read or write the data. With modern disks, one disk delivers at least 10–20MB/s throughput. This is easier to optimize than seeks because you can read in parallel from multiple disks.
- CPU cycles. When the data is in main memory, we must process it to get our result. Having large tables compared to the amount of memory is the most common limiting factor. But with small tables, speed is usually not the problem.
- Memory bandwidth. When the CPU needs more data than can fit in the CPU cache, main memory bandwidth becomes a bottleneck. This is an uncommon bottleneck for most systems, but one to be aware of.

Balancing Portability and Performance

- To use performance-oriented SQL extensions in a portable MySQL program, you can wrap MySQL-specific keywords in a statement within /*! */ comment delimiters. Other SQL servers ignore the commented keywords.
- SQL statements
- Indexes
- Database structure
- INNODB Table
- Configuration
- Buffering and caching
- Benchmarking

Database

1. Use MySQL workbench for relational diagram
2. Use indexes wherever required
3. Select queries should have column names. Never use select * from
4. Use limit 1 clause when retrieving unique row like max or min values we found out
5. Reduce column size - if we just need 20 characters in the column then using varchar(20) is the best option rather than assigning more bytes like varchar(250). It takes more memory and slows down the performance. Same goes for integers
6. Do not use wildcard on search condition in the beginning of the parameter

for ex. Select name, age from table where name like '%ab'. Rather than using above reversing the strong keeps indexing effect intact

Ex select name, age from table where name like 'ba%';

1. Avoid spaces in table names
2. Always use bit field for Boolean values. Using integer or varchar is memory consuming. Field with Boolean data types should be prefixed with Is_
3. Provide authentication for database access. Don't give admin role to each user
4. In big application it is recommended to use Orm
5. Partition big and unused table parts to different physical storages for better query performance
6. For big sensitive and mission critical database systems use disaster recovery and security services like fail-over clustering, auto backups, replication
7. Use constraints foreign key, check, not null.. For data integrity.
8. For queries retrieving a range of rows cluster indexes are usually better. For point queries non clustered indexes are usually better
9. Ideal practice is to keep database on remote servers to protect data from hackers and point remote server to local IP in process keeping db unexposed to internet and having it in intranet
10. Normalization must be used to optimize performance of database. Over normalization will cause excessive joins and under normalization would cause excessive repetitions of data

ddfd

MySQL Guidelines

Don’t use ‘SELECT *’

```
SELECT *
FROM Users
WHERE Username = 'bob'
AND Password = 'password';
```

Let’s create a scenario, you have a login form on your website and to check if the details are correct you query the database. This database could contain loads of fields that are totally unnecessary for this certain query like address or phone number. You would only need the username, password and possibly email to check the user input against. So why grab all that unneeded data and use up resources when its totally avoidable. Instead of structuring a query like:

Change it so that it only takes what it needs

```
SELECT Username, Password
FROM Users
WHERE Username = 'bob'
AND Password = 'password';
```

As you can see from that, it will only get the two fields that you need, reducing the time it takes to run the query. Good for your database server and good for the end user.

Limit queries where possible

When you know your query will only return one result, is there any point in continuing to search the database after it has been found? That’s what’s happening if you don’t limit, it will search the entire database even after it has found the record you are looking for. A login query will only ever return one result, so let’s add it into the query to stop searching when that result is found.

```
SELECT FirstName
FROM Users
WHERE Username = 'bob'
AND Password = 'password'
LIMIT 1
```

Use smaller columns where possible

In a database, size is everything. A large data size can often cause a bottleneck in your database server, so the best solution to this? Use types that are smaller in size. If you can get away with using SMALLINT or even TINYINT instead of the regular INT then go for it! When large amounts of rows are created, it will make all the difference.

FirstName should be VARCHAR(100)

LastName should be VARCHAR(150)

Gender Should be ENUM(‘M’,’F’)

Use TIMESTAMP data type instead of DATETIME

We can convert Timestamp value to any time zone. So that It will be very helpful to show time and date to user from different time zone.

Keywords should be in uppercase in sql statements.

```
SELECT FirstName
FROM Users
WHERE Username = 'bob'
AND Password = 'password'
LIMIT 1
```

All keywords should be upper-case in SQL query, Stored Procedures and Functions. Eg

NAMING CONVENTIONS

Lowercase. Identifiers should be written entirely in lower-case. This includes tables, views, column, and everything else too.

Underscores separate words. Object name that are comprised of multiple words should be separated by underscores. This includes tables, views, column and everything else too.

Full words, not abbreviations. Object names should be full English words. In general avoid abbreviations, especially if they're just the type that removes vowels. Most SQL databases support at least 30-character names which should be more than enough for a couple English words. PostgreSQL supports up to 63-character for identifiers.

Ex: Use middle_name, not mid_nm.

Avoid reserved words. Avoid using any word that is considered a reserved word in the database that you are using. There aren't that many of them so it's not too much effort to pick a different word.

You will find list of keywords and reserved word here

Primary Keys

Single column primary key fields should be named id. It's short, simple, and unambiguous. This means that when you're writing SQL you don't have to remember the names of the fields to join on.

Foreign key fields should be a combination of the name of the referenced table and the name of the referenced fields

Ex : foo_id.

https://www.percona.com/doc/percona-toolkit/2.1/pt-query-advisor.html

http://alternativeto.net/software/jet-profiler-for-mysql/
http://sourceforge.net/projects/myprofi/
http://www.profilesql.com/use/
http://www.jetprofiler.com/buy/

Posted by: PHP - Administrator - Wed, Mar 30, 2016 at 12:49 PM This article has been viewed 448 times.
Filed Under: Knowledge Sharing

Article Rating (1 Votes)

You've Already Voted.

Bookmark Article (CTRL-D)

Attachments

There are no attachments for this article.

Related Articles

HTML-CSS-JS Guidelines
Viewed 445 times since Wed, Mar 30, 2016

Neosoft Secure Coding Practices
Viewed 908 times since Wed, Mar 30, 2016

Hosting Guidelines
Viewed 248 times since Wed, Mar 30, 2016

Hardware - Software Performance Guidelines
Viewed 316 times since Wed, Mar 30, 2016

Security Guidelines
Viewed 620 times since Wed, Mar 30, 2016