Database Administration

Server machine Linux Windows Disk Configuration
Configuring server software Logs Configuring for InnoDB
Optimising for speed and size
Reliability
And logs Replication RAID Cluster Symlinks Load balancing Distribution
OLTP/OLAP Disaster Planning - checks, backups, repairs

If you have a database, small or large, you have a database administrator (DBA). If your organisation is small and you are its one and only software developer, you *are* the DBA. At the other end of the continuum, your organisation may be big enough to support a DBA group and you may be but one cog in it, with sharply defined responsibilities.

A simple, troublesome paradox governs the working life of most DBAs. To the extent a database-driven application succeeds, its performance degrades. The better you do your job, the closer you get to failure.

Why? Because if the application is a success then in all likelihood the number of users is increasing, their expectations are driving more and more change, data is accumulating at increasing rates, and demands on all parts of the system inexorably push the system closer to critical slowdown and failure thresholds.

Years ago, Leo Gerstner, a former CEO of IBM, said “Inside IBM we talk about ten times more connected people, a hundred times more network speed, a thousand times more devices, and a million times more data”. To contain the paradox of success, you build success and its consequences into all aspects of system design. So DBA responsibilities extend to all aspects of servers and all their databases.

Table 17-1: Application Life Cycle

Life cycle stage	***Product***
Requirements analysis	Functional specification, UML diagrams, use cases
Logical design	Logical data design, user interface design, business rules
Database design	Database
Application design	Application model
Application development	Application
Application testing	Deployable application
Deployment	Working application
Maintenance	Mature working application
Application retirement	Data moved to new DBMS

In the life cycle of a MySQL application, the DBA enters at the *database design* stage, and stays till the application is retired, with specific responsibilities at each stage:

- *Database design*: Transform the logical data model into a physical database implementation that meets application demands; plan server hardware acquisition and allocation; set up database support for security and maintenance functions.
- *Application design*: Help application designers develop a security scheme, queries, updates, backup and recovery, and replication or mirroring.
- *Application development*: Analyze and manage database modifications; begin to log and analyze the application's interaction with the database.
- *Testing:* Thoroughly analyse and test mockups of DBMS workload, throughput, optimization, contention, backups, disaster planning and recovery.
- *Deployment*: Install the database on production servers, and ensure that the database is responding correctly to the application.
- *Maintenance*: Continue analysis, maintenance and tuning of every aspect of the database; participate in code refactoring; prevent disasters and manage recovery from them.
- *Retirement*: copy data to the new DBMS, perhaps set up an OLAP archive.

Looking at these responsibilities end-on as it were, or cross-sectionally, we see that the skills required of the DBA are wide and deep. They include management of

- data structures,
- data content,
- physical databases, including startup, shutdown, backup, recovery, failover,
- security,
- performance,

and liaison with management, users, developers, operations staff, network administrators, software vendors and hardware vendors. Not trivial—all the more surprising it is, then, that so many organisations leave database administration almost to chance.

We discuss MySQL server setup, application life cycles and how they relate to databases in this chapter, administrative tools in *Chapter 18*, and security in *Chapter 19*.

In the era of worldwide web applications, 24x7 availability is the standard requirement, but increasingly hard to meet as databases grow larger, applications grow more complex, users ask for more and demand it sooner, and IT grows more complex. What's more, downsizing has reduced many IT departments from large cadres of skilled specialists to leaner but less well prepared cadres of generalists. One of us saw fifty-odd DBAs and application analysts downsized in a day; for the next year or two in that organisation, IT database services were nothing like they had been.

When the cost of a database failure can run to millions of dollars an hour, you need serious availability metrics and strategies. The simplest metric is percent availability over time. More and more, the goal is what’s now called *five nines*, 99.999% availability, or just *five minutes of downtime a year*. Almost perfect, you say? Not if the business is a brokerage house and the downtime cost is seven million dollars a minute.

The strategy that gets you to five nines with MySQL is easy to formulate and expensive to execute: design maximal reliability and inexhaustible redundancy into every aspect of the system—networking hardware and connectivity, server hardware, disks, operating systems, the DBMS, the database, the application and security.

To make matters worse for the DBA, a major DBMS vendor has estimated that *70% of database outages are due to DBA error*. This is due partly because in the world of database applications and their management, change does not stop, most changes require planned outages, and such changes are difficult to execute. As much as half of all database downtime arises from problems encountered while executing planned changes.

Configuring a server machine

A database server offers six loci of potential bottleneck and breakdown: disk i/o, CPU i/o and cache memory, memory i/o, network i/o, bus i/o and the operating system's limits.

Dedicated servers

What "big" means for databases keeps growing. Hundreds of GB or TB now are common.The first problem to solve is contention for resources with other software, and the ideal solution is simple: move all other software off the database server machine(s).

For small webapps a shared hosting site may let you bypass many such issues. If you have a busy MySQL-driven web site served by a modest machine running recent versions of Linux, Apache and MySQL, what's the quickest and cheapest way to boost performance? Separate the MySQL database from the web server. It gives you a huge boost in performance for very little effort: simply move the database to a new machine and change the database hostname in the web applications. Plan for success. Start with a server machine dedicated to the database.

Or for big apps, you may be able to hire a server management site to create, configure and manage server storage. Or you may be able to export such details to a cloud service. If you have one of these arrangements, *and if it works*, you may want to skip the next section.

TOC ***Previous*** ***Next***