DATABASE ADMINISTRATION AND SECURITY NOTES

What Is a Database?

A database can be summarily described as a repository for data. This makes clear that building databases is really a continuation of a human activity that has existed since writing began; it can be applied to the result of any book keepingor recording activity that occurred long before the advent of the computer era.
However, this description is too vague for some of our purposes, and we refine it as we go along.
The creation of a database is required by the operation of an enterprise. We use the term enterprise to designate a variety of endeavors that range from an airline, a bank, or a manufacturing company to a stamp collection or keeping track of people to whom you want to write New Year cards.
DBMS contains information about a particular enterprise. Collection of interrelated data. Set of programs to access the data. An environment that is both convenient and efficient to use
Database Applications: Banking; all transactions,Airlines: reservations, schedules; Universities: registration, grades;Sales: customers, products, purchases;Online retailers: order tracking, customized recommendations; Manufacturing: production, inventory, orders, supply chain; Human resources: employee records, salaries, tax deductions ;Databases touch all aspects of our lives

How does the introduction of a database affect an organisation?
Introduction of a DBMS is likely to have a profound impact Might be positive or negative, depending on how it is administered Three aspects to DBMS introduction: Technological Managerial Cultural One role of DBA department is to educate end users about system uses and benefits

How has the database administration function evolved?
Data administration has its roots in the old, decentralized world of the file system Advent of DBMS produced new level of data management sophistication DP department evolved into information systems (IS) department. Data management became increasingly complex: development of database administration function

What is the DBA's managerial role?

DBA responsible for: Coordinating, monitoring, allocating resources Resources include people and data Defining goals and formulating strategic plans Interacts with end user by providing data and information Enforces policies, standards, procedures Manages security, privacy, integrity Ensures data can be fully recovered Ensures data distributed appropriately

What is the DBA's technical role?
Evaluates, selects, and installs DBMS and related utilities Designs and implements databases and applications Tests and evaluates databases and applications Operates DBMS, utilities, and applications Trains and supports users Maintains DBMS, utilities, and applications

DATABASE ADMINISTRATION

Imagine it's the first day of class in college, and you sit down for the first lecture. Your Head of Department walks in and makes an announcement:
'Welcome back after the holiday break everyone. I have to start off with some bad news. Due to a glitch in the university's database, all grades and credits from last semester have been lost. I'm sorry, but we are going to have to do last semester all over again. Now open your book on page?'
That would be disastrous, right? Could you imagine? How could the university lose all that information? Luckily, these things almost never happen. The important question really is how can we prevent this from happening? Who is responsible for keeping all the data secure in an organization? That would be the database administrator, or DBA.

Database administration refers to the whole set of activities performed by a database administrator to ensure that a database is always available as needed. These activities are closely related to database security, database monitoring and troubleshooting, and planning for future growth.

Who is a Database Administrator?

A database administrator (DBA) is a specialized computer systems administrator who maintains a successful database environment by directing or performing all related activities to keep the data secure. The top responsibility of a DBA professional is to maintain data integrity. This means the DBA will ensure that data is secure from unauthorized access but is available to users.

The DBA is usually a dedicated role in the IT department for large organizations. However, many smaller companies that cannot afford a full-time DBA usually outsource or contract the role to a specialized vendor, or merge the role with another in the ICT department so that both are performed by one person.

Tasks/Functions of a Database Administrator

A DBA must be capable of performing many tasks to ensure that the organization’s data and databases are useful, usable, available, and correct.

1. **Selection of hardware and software**

- Keep up with current technological trends
- Predict future changes
- Emphasis on established off the shelf products
- Evaluate Database features and Database related products.

2. **Database Design**

The DBA must be able to transform a logical data model into a physical database. To properly design and create relational databases DBAs must understand and adhere to sound relational design practices. They must understand both relational theory and the specific implementation of the RDBMS being used to create the database. Database design requires a sound understanding of conceptual and logical data modeling techniques. The ability to create and interpret entity-relationship diagrams is essential to designing a relational database.

2. **Performance Monitoring and Tuning**

Database performance can be defined as the optimization of resource usage to increase throughput and minimize contention, enabling the largest possible workload to be processed. The DBA must be vigilant in monitoring system, database, and application performance. As much as possible this should be accomplished using automated software and scripts. Polling system tables and building alerts based on thresholds can be used to proactively identify problems. Alerts can be set up to e-mail the DBA when performance metrics are not within accepted boundaries.

3. **Ensuring Availability**

The DBA must understand all of these aspects of availability and ensure that each application is receiving the correct level of availability for its needs. Availability of data and databases is often closely aligned with performance. If the DBMS is offline, performance will be horrible because no data can be accessed. Ensuring database availability involves keeping the DBMS up and running and minimizing the amount of downtime required to perform administrative tasks.

4. **Database Security and Authorization**

Once the database is designed and implemented, only authorized programmers and users should have access to prevent security breaches and improper data modification. It is the responsibility of the DBA to ensure that data is available only to authorized users. Security must be administered for many actions required by the database environment including: creating database objects, altering the structure of database objects, reading and modifying data in tables, creating and accessing user-defined functions and data types, running stored procedures and starting and stopping databases and associated database objects.

5. Governance and Regulatory Compliance

Assuring compliance with industry and governmental regulations is an additional task required of database administration, at least in terms of implementing proper controls. The DBA must work with management, auditors, and business experts to understand the regulations that apply to their industry and the manner in which data is treated. Certain aspects of regulatory compliance address standard DBA operating procedures. For example, regulations may contain language enforcing specific security and authorization procedures, auditing requirements, data backup specifications, and change management procedures.

6. Backup and Recovery

The DBA must be prepared to recover data in the event of a problem. “Problem” can mean anything from a system glitch or program error to a natural disaster that shuts down an organization. The majority of recoveries today occur as a result of application software error and human error. The DBA must be prepared to deal with all types of recovery. This involves developing a backup strategy to ensure that data is not lost in the event of an error in software, hardware, or a manual process. The strategy must be applicable to database processing, so it must include image copies of database files as well as a backup/recovery plan for database logs. It needs to account for any non-database file activity that can impact database applications as well.

Types of Database Administrators

Some organizations choose to split DBA responsibilities into separate jobs. Of course, this occurs most frequently in larger organizations, because smaller organizations often cannot afford the luxury of having multiple, specialty DBAs. Some of the most common types of DBAs include:

1. System DBA

A system DBA focuses on technical rather than business issues, primarily in the system administration area. Typical tasks center on the physical installation and performance of the DBMS software. System DBAs rarely get involved with actual implementation of databases and applications. They may get involved in application tuning efforts when operating system parameters or complex DBMS parameters need to be altered.

2. Database Architect

A database architect focuses on designing and implementing new databases. Database architects are involved in new design and development work only; they do not get involved in maintenance, administration, and tuning efforts for established databases and applications. The database architect designs new databases for new applications or perhaps a new database for an existing application. Typical tasks performed by the database architect include: creation of a logical data model, translation of logical data models into physical database designs, analysis of data access and modification requirements to ensure efficient SQL and to ensure that the database design is optimal and creation of backup and recovery strategies for new databases.

3. Database Analyst

Another common staff position is the database analyst. There is really no common definition for database analyst. Sometimes junior DBAs are referred to as database analysts. Sometimes a database analyst performs a role similar to the database architect. And sometimes database analyst is just another term used by some organizations instead of database administrator.

4. Data Modeler

A data modeler is usually responsible for data modeling tasks including: collection of data requirements for development projects, analysis of the data requirements, design of project-based conceptual and logical data models, creation of a corporate data model and keeping the corporate data model up-to-date and working with the DBAs to ensure they have a sound understanding of the data models.

5. Application DBA

Application DBAs focus on database design and the ongoing support and administration of databases for a specific application or applications. The application DBA is likely to be an expert at writing and debugging complex SQL and understands the best ways to incorporate database requests into application programs. The application DBA also must be capable of performing database change management, performance tuning, and most of the other roles of the DBA. The difference is the focus of the application DBA—not on the overall DBMS implementation and database environment, but on a specific subset of applications.

6. Task-Oriented DBA

Larger organizations sometimes create very specialized DBAs who focus on a single specific DBA task. But task-oriented DBAs are quite rare outside of very large IT shops. One example of a task-oriented DBA is a backup and recovery DBA who devotes his entire day to ensuring the recoverability of the organization's databases. Most organizations cannot afford this level of specialization, but when possible, task-oriented DBAs can ensure that very important DBA tasks are tackled by very knowledgeable specialists.

7. Performance Analyst

The performance analyst, more common than other task-oriented DBAs, focuses solely on the performance of database applications. A performance analyst must understand the details and nuances of SQL coding for performance, as well as have the ability to design databases for performance. Performance analysts have knowledge of the DBMS being used at a very detailed technical level so that they can make appropriate changes to DBMS and system parameters when required. The performance analyst is usually the most skilled, senior member of the DBA staff. It is very likely that a senior DBA grows into this role due to his experience and the respect he has gained in past tuning endeavors.

8. Data Warehouse Administrator

Organizations that implement data warehouses (a large store of data accumulated from a wide range of sources within a company and used to guide management decisions) for performing in-depth data analysis often staff DBAs specifically to monitor and support the data warehouse environment. Data warehouse administrators must be capable DBAs, but with a thorough understanding of the differences between a database that supports online transaction processing (OLTP) and a data warehouse.

9. Installation, configuration and upgrading of Database server software and related products.

- Plan growth and changes (capacity planning).
- Work as part of a team and provide 24x7 support when required.

How Many DBAs?

One the most difficult things to determine is the optimal number of DBAs required to keep an organization's databases online and operating efficiently. Many organizations try to operate with the minimal number of DBAs on staff, the idea being that having fewer staff members lowers cost. But that assumption may not be true. An overworked DBA staff may make mistakes that cause downtime and operational problems far in excess of the cost of the salary of an additional DBA.

But determining the optimal number of DBAs is not a precise science. It depends on many factors, including:

1. Number of Databases

The more databases that need to be supported, the more complex the job of database administration becomes. Each database needs to be designed, implemented, monitored for availability and performance, backed up, and administered. There is a limit to the number of databases that an individual DBA can control.

2. Number of users

As additional users are brought online as clients of the applications that access databases, it becomes more difficult to ensure optimal database performance. Additionally, as the number of users increases, the potential for increase in the volume of problems and calls increases, further complicating the DBA's job.

3. Number of applications

A single database can be used by numerous applications. Indeed, one of the primary benefits of the DBMS is to enable data to be shared across an organization. As more applications are brought online, additional pressure is exerted on the database in terms of performance, availability, and resources required, and more DBAs may be required to support the same number of databases.

4. Service-level agreements (SLAs)

The more restrictive the SLA, the more difficult it becomes for the DBA to deliver the service. For example, an SLA requiring subsecond response time for transactions is more difficult to support than an SLA requiring 3-second response time.

5. Availability requirements

When databases have an allowable period of scheduled downtime, database administration becomes easier because some DBA tasks either require an outage or are easier when an outage can be taken. Considerations such as supporting e-business transactions and the Web drive the need for 24/7 database availability.

6. Impact of downtime

The greater the financial impact of a database being unavailable, the more difficult DBA becomes because pressure will be applied to assure greater database availability.

7. Performance requirements

As the requirements for database access become more performance oriented and faster and more frequent access is dictated, DBA becomes more complicated.

8. Type of applications

Organizations implement all kinds of applications. The types of applications that must be supported have an impact on the need for DBA services. The DBMS and database needs of a mission-critical application differ from those of a non-mission critical application. Mission-critical applications are more likely to require constant monitoring and more vigilance to ensure availability.

9. Volatility

The frequency of database change requests is an important factor in the need for additional DBAs. A static database environment requiring few changes will not require the same level of DBA effort as a volatile, frequently changing database environment. Unfortunately, the level of volatility for most databases and applications tends to change dramatically over time. It is difficult to ascertain how volatile an overall database environment will be over its lifetime.

10. DBA staff experience

The skill of the existing DBA staff will impact whether or not additional DBAs are required. A highly skilled DBA staff will be able to accomplish more than a novice team. Skills more than experience dictate DBA staffing level requirements. A highly motivated DBA with two years of experience might easily outperform a ten-year veteran who is burned out and unmotivated.

11. Programming staff experience

The less skilled application developers are in database and SQL programming, the more involved DBAs will need to be in the development process, performing tasks such as complex SQL composition, analysis, debugging, tuning, and ensuring connectivity. As the experience of the programming staff increases, the complexity of DBA decreases.

DBA Certification

Professional certification is an ongoing trend in IT and is available for many different IT jobs. The availability and levels of certification have been growing at an alarming rate for database administration. Certification programs are available for most of the popular DBMS platforms, including IBM DB2, Microsoft SQL Server, and Oracle. The concept behind certification of DBAs is to certify that an individual is capable of performing database administration tasks and duties.

The problem is that passing a test is not a viable indicator of being able to perform a complex job like DBA. Organizations should hire DBAs based on past experience that indicates a level of capability. Certification can make you more employable, but, someone with both experience and certification is better than someone with only one of the two.

DBMS Vendors

There are many DBMS vendors from which to choose. However, there are definite tiers in terms of popularity, support, and leadership in the DBMS market.

In general, the marketplace can be broken down into the following groups:

1. **The Big Three**

   The three market leaders that constitute the greater part of the DBMS installed base as well as the bulk of any new sales.

   They are:
   1. Oracle Corporation - Oracle
   2. IBM Corporation - DB2

    3. Microsoft - Microsoft SQL Server

2. **The second tier**
   Large DBMS vendors with stable products, but lagging behind the Big Three in terms of functionality and number of users.

   They are:
    1. Sybase
    2. Teradata
    3. Informix

3. **Other significant players**
   Other DBMS vendors with viable, enterprise-capable products.

   They are:

    1. Actian Corporation - Ingres
    2. Software AG - Adabas and Tamino

4. **Open source**
   DBMS products supported as open-source software (as opposed to by a single vendor).

   They are:
    1. PostgreSQL
    2. MySQL

5. **Non-relational**
   Vendors that supply pre-relational DBMS products to support legacy applications.

   They are:
    1. IBM - IMS
    2. Cullinet - IDMS

6. **NoSQL**
   Non-relational DBMS products for Big Data that are highly scalable to support modern Web applications.

   They include:
    1. CouchDB
    2. MongoDB
    3. Hadoop and HBase

7. **Object oriented**
   Vendors of ODBMS products that are used in conjunction with OO languages and development projects.

They are:

1. Object Store from Progress Software
2. Ontos
3. Poet

8. **PC based**

   Although many of the other vendors create PC versions of their DBMS products, these companies or products focus exclusively on the PC platform.

   They are:

   1. dBase
   2. FileMaker
   3. Lotus Approach (included in Lotus SmartSuite)
   4. Microsoft Access
   5. Paradox (included in the Professional edition of WordPerfect Office)

Creating the Database Environment

One of the primary tasks associated with the job of DBA is the process of choosing and installing a DBMS. Choosing a suitable DBMS for enterprise database management is not as difficult as it used to be as the number of major DBMS vendors has dwindled due to industry consolidation and domination of the sector by a few very large players. However, establishing a usable database environment requires a great deal of skill, knowledge, and consideration.

Large and medium-size organizations typically run multiple DBMS products, from as few as two to as many as ten. For example, it is not uncommon for a large company to use IBM's DB2 on the mainframe, Oracle and MySQL on several different UNIX servers, Microsoft SQL Server on Windows servers, as well as other DBMS products such as PostgreSQL on various platforms, not to mention single-user PC DBMS products such as Microsoft Access.

In order not to complicate the DBA's job, the DBA group should be empowered to make the DBMS decisions for the organization. No business unit should be allowed to purchase a DBMS without the permission of the DBA group.

Choosing a DBMS

The DBA should set a policy regarding the DBMS products to be supported within the organization. Whenever possible, the policy should minimize the number of different DBMS products as most of the major DBMS products have similar features.

When choosing a DBMS, it is wise to select a product from one of the largest vendors having the most heavily implemented and supported products on the market. These include IBM's DB2 and Oracle. Both are popular and support just about any type of database. Another major player is Microsoft SQL Server, but only for Windows platforms. DB2 and Oracle run on multiple platforms ranging from mainframe to UNIX, as well as Windows and even handheld devices. Choosing a DBMS other than these three should be done only under specific circumstances.

If the company is heavily into the open source software movement, PostgreSQL, or MySQL might be viable options. And there are a variety of NoSQL DBMS offerings available, too, such as Hadoop, Cassandra, and MongoDB.

Factors to consider when choosing a DBMS

1. **Operating system support**

Does the DBMS support the operating systems in use at your organization, including the versions that you are currently using and plan on using?

2. **Type of organization**

Take into consideration the corporate philosophy when you choose a DBMS. Some organizations are very conservative and like to keep a tight rein on their environments; these organizations tend to gravitate toward traditional mainframe environments. Government operations, financial institutions, and insurance and health companies usually tend to be conservative. More-liberal organizations are often willing to consider alternative architectures. It is not uncommon for manufacturing companies, dot-coms, and universities to be less conservative. Finally, some companies just do not trust Windows as a mission-critical environment and prefer to use UNIX; this rules out some database vendors (Microsoft SQL Server, in particular).

3. **Benchmarks**

What performance benchmarks are available from the DBMS vendor and other users of the DBMS? The Transaction Processing Performance Council (TPC) publishes official database performance benchmarks that can be used as a guideline for the basic overall performance of many different types of database processing.

4. **Scalability**

Does the DBMS support the number of users and database sizes you intend to implement? How are large databases built, supported, and maintained—easily or with a lot of pain? Are there independent users who can confirm the DBMS vendor's scalability claims?

5. **Availability of supporting software tools**

Are the supporting tools you require available for the DBMS? These items may include query and analysis tools, data warehousing support tools, database administration tools, backup and recovery tools, performance-monitoring tools, capacity-planning tools, database utilities, and support for various programming languages.

6. Technicians

Is there a sufficient supply of skilled database professionals for the DBMS? Consider your needs in terms of DBAs, technical support personnel (system programmers and administrators, operations analysts, etc.), and application programmers.

7. Cost of ownership

What is the total cost of ownership of the DBMS? DBMS vendors charge wildly varying prices for their technology. Total cost of ownership should be calculated as a combination of the license cost of the DBMS; the license cost of any required supporting software; the cost of database professionals to program, support, and administer the DBMS; and the cost of the computing resources required to operate the DBMS.

8. Release schedule

How often does the DBMS vendor release a new version? Some vendors have rapid release cycles, with new releases coming out every 12 to 18 months. This can be good or bad, depending on your approach. If you want cutting-edge features, a rapid release cycle is good. However, if your shop is more conservative, a DBMS that changes frequently can be difficult to support. A rapid release cycle will cause conservative organizations either to upgrade more frequently than they would like or to live with outdated DBMS software that is unlikely to have the same level of support as the latest releases.

9. Reference customers

Will the DBMS vendor supply current user references? Can you find other users on your own who might provide more impartial answers? Speak with current users to elicit issues and concerns you may have overlooked. How is support? Does the vendor respond well to problems? Do things generally work as advertised? Are there a lot of bug fixes that must be applied continuously? What is the quality of new releases? These questions can be answered only by the folks in the trenches.

Database administration tools

Often, the DBMS software comes with certain tools to help DBAs manage the DBMS. Such tools are called native tools. For example, Microsoft SQL Server comes with SQL Server Management Studio and Oracle has tools such as SQL*Plus and Oracle Enterprise Manager/Grid Control. In addition, 3rd parties such as BMC, Quest Software, Embarcadero Technologies, and SQL Maestro

Group offer GUI tools to monitor the DBMS and help DBAs carry out certain functions inside the database more easily.
Another kind of database software exists to manage the provisioning of new databases and the management of existing databases and their related resources. The process of creating a new database can consist of hundreds or thousands of unique steps from satisfying prerequisites to configuring backups where each step must be successful before the next can start. A human cannot be expected to complete this procedure in the same exact way time after time - exactly the goal when multiple databases exist. As the number of DBAs grows, without automation the number of unique configurations frequently grows to be costly/difficult to support. All of these complicated procedures can be modeled by the best DBAs into database automation software and executed by the standard DBAs. Software has been created specifically to improve the reliability and repeatability of these procedures such as Stratavia's Data Palette and GridApp Systems Clarity.

DBMS Architectures

The supporting architecture for the DBMS environment is very critical to the success of the database applications. One wrong choice or poorly implemented component of the overall architecture can cause poor performance, downtime, or unstable applications.

Today the IT infrastructure is distributed and heterogeneous. The overall architecture will probably consist of multiple platforms and interoperating system software. The DBA has to make sure that the DBMS selected is appropriate for the nature and type of processing they plan to implement.

Levels of DBMS architecture

1. **Enterprise DBMS**

An enterprise DBMS is designed for scalability and high performance. An enterprise DBMS must be capable of supporting very large databases, a large number of concurrent users, and multiple types of applications. The enterprise DBMS runs on a large-scale machine, typically a mainframe or a high-end server running UNIX, Linux, or Windows Server. Furthermore, an enterprise DBMS offers all the “bells and whistles” available from the DBMS vendor.
Multiprocessor support, support for parallel queries, and other advanced DBMS features are core components of an enterprise DBMS.

2. **Departmental DBMS**

A departmental DBMS, sometimes referred to as a workgroup DBMS, serves the middle ground. The departmental DBMS supports small to medium-size workgroups within an organization; typically, it runs on a UNIX, Linux, or Windows server. The dividing line between a departmental database server and an enterprise database server is quite gray. Hardware and software upgrades can allow a departmental DBMS to tackle tasks that previously could be performed only by an enterprise DBMS. The steadily falling cost of departmental hardware and software components further contributes to lowering the total cost of operation and enabling a workgroup environment to scale up to serve the enterprise.

3. Personal DBMS

A personal DBMS is designed for a single user, typically on a low- to medium-powered PC platform. Microsoft Access and SQLite are examples of personal database software. Of course, the major DBMS vendors also market personal versions of their higher-powered solutions, such as Oracle Database Personal Edition and DB2 Personal Edition. Sometimes the low cost of a personal DBMS results in a misguided attempt to choose a personal DBMS for a departmental or enterprise solution. However, do not be lured by the low cost. A personal DBMS product is suitable only for very small-scale projects and should never be deployed for multi user applications.

4. Mobile DBMS

A mobile DBMS is a specialized version of a departmental or enterprise DBMS. It is designed for remote users who are not usually connected to the network. The mobile DBMS enables local database access and modification on a laptop or handheld device. Furthermore, the mobile DBMS provides a mechanism for synchronizing remote database changes to a centralized enterprise or departmental database server.

Hardware Issues

When establishing a database environment for application development, selecting the DBMS is only part of the equation. The hardware and operating system on which the DBMS will run will greatly impact the reliability, availability, and scalability of the database environment. That is not to say everything should run on a mainframe; other issues such as cost, experience, manageability, and the needs of the applications to be developed must be considered. The bottom line is that you must be sure to factor hardware platform and operating system constraints into the DBMS selection criteria.

Cloud Database Systems

Cloud computing is increasing in usage, especially at small to medium-size businesses. A cloud implementation can be more cost effective than building an entire local computing infrastructure that requires management and support. A cloud database system delivers DBMS services over the Internet. The trade-off essentially comes down to trusting a cloud provider to store and manage your data in return for minimizing database administration and maintenance cost and effort.

There is no need to install, set up, patch, or manage the DBMS software because the cloud provider manages and cares for these administrative tasks. Of course, the downside is that your data is now stored and controlled by an external agent —the cloud provider. Another inherent risk of cloud computing is the possibility of malicious agents posing as legitimate customers.

An example of a cloud database platform is Microsoft SQL Azure. It is built on SQL Server technologies and is a component of the Windows Azure platform.

Installing the DBMS

Once the DBMS has been chosen, you will need to install it. A DBMS is a complex piece of software that requires up-front planning for installation to be successful. You will need to understand the DBMS requirements and prepare the environment for the new DBMS.

DBMS Installation Basics

The very first thing to do when you install a DBMS for the first time is to understand the prerequisites. Every DBMS comes with an installation manual or guide containing a list of the operating requirements that must be met for the DBMS to function properly. Examples of prerequisites include ensuring that an appropriate version of the operating system is being used, verifying that there is sufficient memory to support the DBMS, and ensuring that any related software to be used with the DBMS is the proper version and maintenance level.

Hardware Requirements

Every DBMS has a basic CPU requirement, meaning a CPU version and minimum processor speed required for the DBMS to operate. Additionally, some DBMSs specify hardware models that are required or unsupported.

Storage Requirements

A DBMS requires disk storage to run. And not just for the obvious reason— to create databases that store data. A DBMS will use disk storage for the indexes to be defined on the databases as well as for the system catalogue, system databases, log files, start-up files, temporary databases, e.t.c.

Memory Requirements

Relational DBMSs, as well as their databases and applications, love memory. A DBMS requires memory for basic functionality and will use it for most internal processes such as maintaining the system global area and performing many DBMS tasks.

Memory is typically required by the DBMS to support other features such as handling lock requests, facilitating distributed data requests, sorting data, optimizing processes, and processing SQL. Ensure that the DBMS has a more-than-adequate supply of memory at its disposal. This will help to optimize database processing and minimize potential problems.

Configuring the DBMS

Configuring the system parameters of the DBMS controls the manner in which the DBMS functions and the resources made available to it. Each DBMS allows its system parameters to be modified in different ways, but the installation process usually sets the DBMS system parameters by means of radio buttons, menus, or panel selections. During the installation process, the input provided to the installation script will be used to establish the initial settings of the system parameters.

Each DBMS also provides a method to change the system parameters once the DBMS is operational. Sometimes you can use DBMS commands to set the system's parameters; sometimes you must edit a file that contains the current system parameter settings. If you must edit a file, be very careful: An erroneous system parameter setting can be fatal to the operational status of the DBMS.

Connecting the DBMS to Supporting Infrastructure Software

Part of the DBMS installation process is the connection of the DBMS to other system software components that must interact with the DBMS. Typical infrastructure software that may need to be configured to work with the DBMS includes networks, transaction processing monitors, message queues, other types of middleware, programming languages, systems management software, operations and job control software, Web servers, and application servers. Typical configuration procedures can include creating new parameter files to establish connections.

Installation Verification

After installing the DBMS, you should run a battery of tests to verify that the DBMS has been properly installed and configured. Most DBMS vendors supply sample programs and installation verification procedures for this purpose. Additionally, you can ensure proper installation by testing the standard interfaces to the DBMS. One standard interface supported by most DBMSs is an interactive SQL interface where you can submit SQL statements directly to the DBMS. Create a set of SQL code that comprises SELECT, INSERT, UPDATE, and DELETE statements issued against sample databases. Running such a script after installation helps you to verify that the DBMS is installed correctly and operating as expected.

DBMS Environments

Generally, installing a DBMS involves more than simply installing one instance or subsystem. To support database development, the DBA needs to create multiple DBMS environments to support, for example, testing, quality assurance, integration, and production work. Of course, it is possible to support multiple environments in a single DBMS instance, but it is not prudent. Multiple DBMS installations are preferable to support multiple development environments for a single database. This minimizes migration issues and won't require complex database naming conventions to support.

Upgrading DBMS Versions and Releases

The DBA must develop an approach to upgrading DBMS software that conforms to the organization's needs and minimizes business disruptions due to outages and database unavailability. A DBMS version upgrade can be thought of as a special case of a new installation. All the procedures required of a new installation apply to an upgrade: You must plan for appropriate resources, reconsider all system parameters, and ensure that all supporting software is appropriately connected. However, another serious issue must be planned for: existing users and applications. An upgrade needs to be planned to cause as little disruption to the existing users as possible. Furthermore, any additional software that works with the DBMS (such as purchased applications, DBA tools, utilities, and so on) must be verified to be compatible with the new DBMS version. Therefore, upgrading can be a tricky and difficult task.

Version or Release?

Vendors typically make a distinction between a version and a release of a software product. A new version of software is a major concern, with many changes and new features. A release is typically minor, with fewer changes and not as many new features. For example, moving from Version 10g of Oracle Database to Version 11g would be a major change—a version change. However, an in-between point such as Oracle Database 11g Release 2 would be considered a release—consisting of a smaller number of changes.

Upgrading to a new DBMS release offers both rewards and risks.

Benefits of moving to a new DBMS release

- Developers can avail themselves of new features and functionality delivered only in the new release. If development requires a new feature, or can simply benefit from a new feature, program development time can be reduced or made more cost-effective.
- For purchased applications, the application vendor may require a specific DBMS version or release for specific versions of its application to enable specific functionality within the application.
- New DBMS releases usually deliver enhanced performance and availability features that can optimize existing applications. Sometimes a new DBMS release is required to scale applications to support additional users or larger amounts of data.
- DBMS vendors often provide better support and respond to problems faster for a new release of their software. DBMS vendors would not like to allow bad publicity about bugs in a new and heavily promoted version of their products.
- Cost savings may accrue by upgrading to a new DBMS release. Some vendors charge additionally when a company uses multiple versions of a DBMS, such as the new version in a test environment and the old in production. When both are migrated to the same version, the price tag for the DBMS sometimes can be reduced.
- Production migration to a new DBMS release will align the test and production database environments, thereby providing a consistent environment for development and implementation. If a new release is running in the test environment for too long, database administration and application development tasks become more difficult because the test databases will operate differently from the production databases.

Risks of upgrading to a new DBMS release

- An upgrade to the DBMS usually involves some level of disruption to business operations. At a minimum, databases will not be available while the DBMS is being upgraded. This can result in downtime and lost business opportunities if the DBMS upgrade occurs during normal business hours (or if there is no planned downtime). Clustered database implementations may permit some database availability while individual database clusters are migrated to the new DBMS version.
- Other disruptions can occur, such as having to convert database structures or discovering that previously supported features were removed from the new release (thereby causing application errors). Delays to application implementation timelines are another possibility.
- The cost of an upgrade can be a significant barrier to DBMS release migration. First, the cost of the new version or release must be budgeted for (price increases for a new DBMS version can amount to as much as 10 to 25 percent). The upgrade cost must also factor in the costs of planning, installing,

testing, and deploying not just the DBMS but also any applications that use databases. Finally, be sure to include the cost of any new resources (such as memory, storage, additional CPUs) required to use the new features delivered by the new DBMS version.

- DBMS vendors usually tout the performance gains that can be achieved with a new release. However, when SQL optimization techniques change, it is possible that a new DBMS release will generate SQL access paths that perform worse than before. DBAs must implement a rigorous testing process to ensure that new access paths are helping, not harming, application performance. When performance suffers, application code may need to be changed—a very costly and time consuming endeavor. A rigorous test process should be able to catch most of the access path changes in the test environment.
- New DBMS releases may cause features and syntax that are being used in existing applications to be deprecated. When this occurs, the applications must be modified before migration to the new release can proceed.
- To take advantage of improvements implemented in a new DBMS release, the DBA may have to apply some invasive changes. For example, if the new version increases the maximum size for a database object, the DBA may have to drop and recreate that object to take advantage of the new maximum. This will be the case when the DBMS adds internal control structures to facilitate such changes.
- Supporting software products may lack immediate support for a new DBMS release. Supporting software includes the operating system, transaction processors, message queues, purchased applications, DBA tools, development tools, and query and reporting software.

Factors to consider for an effective DBMS upgrade strategy

1. Features and Complexity

The biggest factor in determining when and how to upgrade to a new DBMS release is the functionality supported by the new release. Tightly coupled to functionality is the inherent complexity involved in supporting and administering new features. If DBMS functionality can minimize the cost and effort of application development, the DBA will feel pressure to migrate swiftly to the new release, especially if DBMS problems are fixed in the new release.

2. Complexity of the DBMS Environment

The more complex your database environment is, the more difficult it will be to upgrade to a new DBMS release. The first complexity issue is the size of the environment. The greater the number of database servers, instances, applications, and users, the greater the complexity. Additional concerns include the types of applications being supported. Location of the database servers also affects the release upgrade strategy.

3. Reputation of the DBMS Vendor

DBMS vendors have different reputations for technical support, fixing bugs, and responding to problems, which is why customer references are so important when choosing a database. The better the reputation of the vendor, the greater the likelihood of organizations rapidly adopting a new release.

4. Support Policies of the DBMS

As new releases are introduced, DBMS vendors will retire older releases and no longer support them. The length of time that the DBMS vendor will support an old release must be factored into the DBMS release migration strategy. You should never run a DBMS release in production that is no longer supported by the vendor. If problems occur, the DBMS vendor will not be able to resolve them for you.

5. Organization Style

Every organization displays characteristics that reveal its style when it comes to adopting new products and technologies. Some organizations are technology driven and, as such, are more likely to risk using new and unproven technologies to try to gain a competitive advantage. Some organizations are less willing to take risks but will adopt new technologies once others have shaken out the bugs. However other organizations are very conscious of cost and averse to risk, and will lag behind the majority when it comes to migrating to new technology.

6. DBA Staff Skill Set

Upgrading the DBMS is easier if your DBA staff is highly skilled and/or experienced. The risk of an upgrade increases as the skills of the DBA staff decrease. If your DBAs are not highly skilled, or have never migrated a DBMS to a new release, consider augmenting your DBA staff with consultants for the upgrade. Deploying an integrated team of internal DBAs and consultants will ensure that your upgrade goes as smoothly as possible. Furthermore, the DBA staff will be better prepared to handle the future upgrades alone.

7. Platform Support

When a DBMS vendor unleashes a new release of its product, not all platforms and operating systems are immediately supported. The DBMS vendor usually first supports the platforms and operating systems for which it has the most licensed customers. The order in which platforms are supported for a new release is likely to differ for each DBMS vendor.

8. Supporting Software

Carefully consider the impact of a DBMS upgrade on any supporting software. Supporting software includes purchased applications, DBA tools, reporting and analysis tools, and query tools. Each software vendor will have a different time frame for supporting and exploiting a new DBMS release.

9. Fallback Planning

Each new DBMS version or release should come with a manual that outlines the new features of the release and describes the fallback procedures to return to a prior release of the DBMS. Be sure to review the fallback procedures provided by the DBMS vendor in its release guide. You may need to return to a previous DBMS release if the upgrade contains a bug, performance problems ensue, or other problems arise during or immediately after migration. Keep in mind that fallback is not always an option for every new DBMS release.

Database Change Management

An ever-changing market causes businesses to have to continually adapt. Businesses are striving to meet constantly changing customer expectations while trying to sustain revenue growth and profitability at the same time. To keep pace, businesses must constantly update and enhance products and services to meet and exceed the offerings of competitors.

Change is inevitable but necessary for business survival and success.

Factors that lead to change in database structures

Many factors conspire to force us into changing our database structures.

- Changes to application programs that require additional or modified data elements
- Performance modifications and tweaks to make database applications run faster
- Regulatory changes that mandate storing new types of data, or the same data for longer periods of time
- Changes to business practices, requiring new types of data
- Technological changes that enable databases to store new types of data and more data than ever before

Change Management Requirements

To successfully implement effective change management, understanding a set of basic requirements is essential. To ensure success, the following factors need to be incorporated into your change management discipline.

Factors to incorporate into the change management discipline

1. **Proactivity**.Proactive change, which can eliminate future problems, is an organization’s most valuable type of change. The earlier in the development cycle that required changes are identified and implemented, the lower the overall cost of the change will be.
2. **Intelligence**.When implementing a change, every aspect of the change needs to be examined, because it could result in an unanticipated cost to

the company. The impact of each change must be examined and incorporated into the change process, because a simple change in one area may cause a complex change in another area. Intelligence in the change management process often requires a thorough analysis that includes an efficient and low-risk implementation plan. True intelligence also requires the development of a contingency plan, should the change or set of changes not perform as projected.

3. **Planning analysis**.Planning maximizes the effectiveness of change. A well-planned change saves time. It is always easier to do it right the first time than to do it again after the first change proves to be less than effective. An effective organization will have a thorough understanding of the impact of each change before allocating resources to implement the change. A well-planned change saves time.
4. **Impact analysis**.Comprehensive impact and risk analyses allow the organization to examine the entire problem, and the risk involved, to determine the best course of action. A single change usually can be accomplished in many different ways. However, the impact of each change may be considerably different. Some changes involve more risks: failure, undue difficulty, need for additional changes, downtime, and so on. All considerations are important when determining the best approach to implementing change.
5. **Automation.**With limited resources and a growing workload, automating the change process serves to reduce human error and to eliminate more menial tasks from overburdened staff.
6. **Standardization of procedure**.Attrition, job promotions, and job changes require organizations to standardize processes to meet continued productivity levels. An organized and thoroughly documented approach to completing a task reduces the learning curve, as well as the training time.
7. **Reliable and predictable process**.When creating any deliverable, a business needs to know that none of the invested effort is wasted. Because time is valuable, a high level of predictability will help to ensure continued success and profitability. Reliability and predictability are key factors in producing a consistently high-quality product.
8. **Availability**.Most changes require downtime to implement the change. Applications must come down—the same is true of databases. However, high availability is required of most applications these days, especially for an e-business. This is fast becoming a requirement in the Internet age. Reducing the amount of downtime required to make a change will increase application availability.
9. **Quick and efficient delivery**.Consumers demand quick turnaround for most products and services. Profitability is at its best when a product is first to market. Conversely, the cost of slow or inefficient delivery of products can be enormous. So, when implementing change, faster is better. The shorter the duration of an outage to accomplish the change, the quicker the system can be brought to market.

Types of Changes

Managing change is a big component of the DBA's job. In fact, if systems and databases could be installed into an environment that never changed, most of the

DBA's job would vanish. However, things change. Business changes usually necessitate a change to application code or to database structure. Less obvious business changes also impact the database—for example, when the business grows and more users are added, when additional data is stored, or when transaction volume grows. Additionally, technological changes such as upgrades to the DBMS and changes to hardware components impact the functionality of database software and therefore require DBA involvement.

1. **DBMS Software**
   The DBA must be prepared to manage the migration to new DBMS versions and releases. The complexity involved in moving from one version of a DBMS to another depends on the new features and functions supported by the new version. Additional complexity will be introduced if features are removed from the DBMS in a later version, because databases and programs may need to change if the removed features were being used. Furthermore, as functionality is added to, and removed from, the DBMS, the DBA must create the policies and procedures for the proper use of each new DBMS feature.

2. **Hardware Configuration**
   The DBMS may require hardware upgrades or configuration changes. The DBA will be expected to work in conjunction with the system programmers and administrators responsible for setting up and maintaining the hardware. At times the DBMS may require a different configuration from the one that is commonly used, thereby requiring the DBA to communicate to the SA the reason why a nonstandard configuration is required.

3. **Logical and Physical Design**
   When the database changes, it is important that the blueprints that define the database also change. This means that you need to keep the conceptual and logical data models synchronized with the physical database.

4. **Applications**
   Application changes need to be synchronized with database changes; however, this is easier said than done. Whenever changes are made to a physical database structure, application changes usually accompany those changes. For example, simply adding a column to a database table requires application software to populate, modify, and report on the data in the new column.

5. **Physical Database Structures**
   The most complicated and time-consuming type of change for DBAs is planning, analyzing, and implementing changes to physical database structures. But most databases change over time—indeed, the database that remains static once implemented is very rare. So DBAs must be prepared to make changes to the databases under their care. Some

changes are simple to implement, but others are very complex, error prone, and time-consuming.

Impact of Change on Database Structures

Relational databases are created using Data Definition Language (DDL) statements. DDL consists of three SQL verbs: CREATE, DROP, and ALTER. The CREATE statement is used to create a database object initially, and the DROP statement is used to remove a database object from the system. The ALTER statement is used to make changes to database objects.

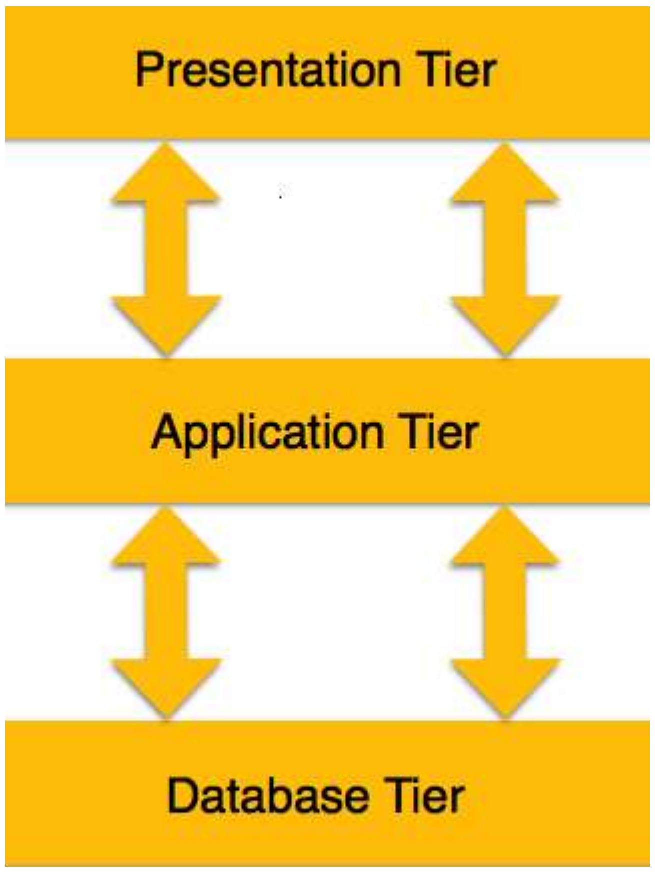
Not every aspect of a database object can be changed by using the ALTER statement. Some types of changes require the database object to be dropped and recreated with the new parameters. The exact specifications for what can, and cannot, be changed using ALTER differ from DBMS to DBMS.

DATABASE SYSTEMS ARCHITECTURES

The design of a DBMS depends on its architecture. It can be centralized or decentralized or hierarchical. The architecture of a DBMS can be seen as either single tier or multi-tier. An n-tier architecture divides the whole system into related but independent n modules, which can be independently modified, altered, changed, or replaced.
In 1-tier architecture, the DBMS is the only entity where the user directly sits on the DBMS and uses it. Any changes done here will directly be done on the DBMS itself. It does not provide handy tools for end-users. Database designers and programmers normally prefer to use single-tier architecture.
If the architecture of DBMS is 2-tier, then it must have an application through which the DBMS can be accessed. Programmers use 2-tier architecture where they access the DBMS by means of an application. Here the application tier is entirely independent of the database in terms of operation, design, and programming.

3-tier Architecture

A 3-tier architecture separates its tiers from each other based on the complexity of the users and how they use the data present in the database. It is the most widely used architecture to design a DBMS.



Database (Data) Tier − At this tier, the database resides along with its query processing languages. We also have the relations that define the data and their constraints at this level.

Application (Middle) Tier − At this tier reside the application server and the programs that access the database. For a user, this application tier presents an abstracted view of the database. End-users are unaware of any existence of the database beyond the application. At the other end, the database tier is not aware of any other user beyond the application tier. Hence, the application layer sits in the middle and acts as a mediator between the end-user and the database.

User (Presentation) Tier − End-users operate on this tier and they know nothing about any existence of the database beyond this layer. At this layer, multiple views of the database can be provided by the application. All views are generated by applications that reside in the application tier.

Multiple-tier database architecture is highly modifiable, as almost all its components are independent and can be changed independently.

ORACLE DATABASE ARCHITECTURE

An Oracle database is a collection of data treated as a unit. The purpose of a database is to store and retrieve related information. A database server is the key to solving the problems of information management. In general, a server reliably manages a large amount of data in a multiuser environment so that many users can concurrently access the same data. All this is accomplished while delivering high performance. A database server also prevents unauthorized access and provides efficient solutions for failure recovery.

Oracle Database is the first database designed for enterprise grid computing, the most flexible and cost effective way to manage information and applications. Enterprise grid computing creates large pools of industry-standard, modular storage and servers. With this architecture, each new system can be rapidly provisioned from the pool of components. There is no need for peak workloads, because capacity can be easily added or reallocated from the resource pools as needed.

The database has **logical structures and physical structures**. Because the physical and logical structures are separate, the physical storage of data can be managed without affecting the access to logical storage structures.

ORACLE PHYSICAL DATABASE STRUCTURES

The following sections explain the physical database structures of an Oracle database, including datafiles, redo log files, and control files.

1. Datafiles

Every Oracle database has one or more physical datafiles. The datafiles contain all the database data. The data of logical database structures, such as tables and indexes, is physically stored in the datafiles allocated for a database.

The characteristics of datafiles are:

A datafile can be associated with only one database.

Datafiles can have certain characteristics set to let them automatically extend when the database runs out of space.

One or more datafiles form a logical unit of database storage called a tablespace.

Data in a datafile is read, as needed, during normal database operation and stored in the memory cache of Oracle. For example, assume that a user wants to access some data in a table of a database. If the requested information is not already in the memory cache for the database, then it is read from the appropriate datafiles and stored in memory.

Modified or new data is not necessarily written to a datafile immediately. To reduce the amount of disk access and to increase performance, data is pooled in memory and written to the appropriate datafiles all at once, as determined by the database writer process (DBWn) background process.

2. Control Files

Every Oracle database has a control file. A control file contains entries that specify the physical structure of the database. For example, it contains the following information:

- Database name
- Names and locations of datafiles and redo log files
- Time stamp of database creation

When you mount and open the database instance, the control file is read. The entries in thecontrol file specify the physical files that make up the database. When you add additional filesto your database, the control file is automatically updated.

The location of the control files isspecified in the CONTROL_FILES initialization parameter.

To protect against the failure of the database because of the loss of the control file, you mustmultiplex the control file on at least three different physical devices. By specifying multiplefiles through the initialization parameter, you enable the Oracle server to maintain multiplecopies of the control file.

Oracle can multiplex the control file, that is, simultaneously maintain a number of identical control file copies, to protect against a failure involving the control file.

Every time an instance of an Oracle database is started, its control file identifies the database and redo log files that must be opened for database operation to proceed. If the physical makeup of the database is altered (for example, if a new datafile or redo log file is created), then the control file is automatically modified by Oracle to reflect the change. A control file is also used in database recovery.

3. Redo Log Files

Every Oracle database has a set of two or more redo log files. The set of redo log files is collectively known as the redo log for the database. A redo log is made up of redo entries (also called redo records).

The primary function of the redo log is to record all changes made to data. If a failure prevents modified data from being permanently written to the datafiles, then the changes can be obtained from the redo log, so work is never lost.

To protect against a failure involving the redo log itself, Oracle allows a multiplexed redo log so that two or more copies of the redo log can be maintained on different disks.

The information in a redo log file is used only to recover the database from a system or media failure that prevents database data from being written to the datafiles. For example, if an unexpected power outage terminates database operation, then data in memory cannot be written to the datafiles, and the data is lost. However, lost data can be recovered when the database is opened, after power is restored. By applying the information in the most recent redo log files to the database datafiles, Oracle restores the database to the time at which the power failure occurred.

The process of applying the redo log during a recovery operation is called rolling forward.

4. Archive Log Files

You can enable automatic archiving of the redo log. Oracle automatically archives log files when the database is in ARCHIVELOG mode.

5. Parameter Files

Parameter files contain a list of configuration parameters for that instance and database.

Oracle recommends that you create a server parameter file (SPFILE) as a dynamic means of maintaining initialization parameters. A server parameter file lets you store and manage your initialization parameters persistently in a server-side disk file.

6. Alert and Trace Log Files

Each server and background process can write to an associated trace file. When an internal error is detected by a process, it dumps information about the error to its trace file. Some of the information written to a trace file is intended for the database administrator, while other information is for Oracle Support Services. Trace file information is also used to tune applications and instances.

The alert file, or alert log, is a special trace file. The alert log of a database is a chronological log of messages and errors.

7. Backup Files

To restore a file is to replace it with a backup file. Typically, you restore a file when a media failure or user error has damaged or deleted the original file.

User-managed backup and recovery requires you to actually restore backup files before you can perform a trial recovery of the backups.

Server-managed backup and recovery manages the backup process, such as scheduling of backups, as well as the recovery process, such as applying the correct backup file when recovery is needed.

ORACLE LOGICAL DATABASE STRUCTURES

The logical storage structures, including data blocks, extents, and segments, enable Oracle to have fine-grained control of disk space use.

1. Tablespaces

A database is divided into logical storage units called tablespaces, which group related logical structures together. For example, tablespaces commonly group together all application objects to simplify some administrative operations.

Each database is logically divided into one or more tablespaces. One or more datafiles are explicitly created for each tablespace to physically store the data of all logical structures in a tablespace. The combined size of the datafiles in a tablespace is the total storage capacity of the tablespace.

Every Oracle database contains a SYSTEM tablespace and a SYSAUX tablespace. Oracle creates them automatically when the database is created. The system default is to create a smallfile tablespace, which is the traditional type of Oracle tablespace. The SYSTEM and SYSAUX tablespaces are created as smallfile tablespaces.

Oracle also lets you create bigfile tablespaces. This allows Oracle Database to contain tablespaces made up of single large files rather than numerous smaller ones. This lets Oracle Database utilize the ability of 64-bit systems to create and manage ultralarge files. The consequence of this is that Oracle Database can now scale up to 8 exabytes in size. With Oracle-managed files, bigfile tablespaces make datafiles completely transparent for users. In other words, you can perform operations on tablespaces, rather than the underlying datafiles.

2. *Online and Offline Tablespaces*

A tablespace can be online (accessible) or offline (not accessible). A tablespace is generally online, so that users can access the information in the tablespace. However, sometimes a tablespace is taken offline to make a portion of the database unavailable while allowing normal access to the remainder of the database. This makes many administrative tasks easier to perform.

3. Oracle Data Blocks

At the finest level of granularity, Oracle database data is stored in data blocks. One data block corresponds to a specific number of bytes of physical database space on disk. The standard block size is specified by the DB_BLOCK_SIZE initialization parameter. In addition, you can specify up to five other block sizes. A database uses and allocates free database space in Oracle data blocks.

4. Extents

The next level of logical database space is an extent. An extent is a specific number of contiguous data blocks, obtained in a single allocation, used to store a specific type of information.

5. Segments

Above extents, the level of logical database storage is a segment. A segment is a set of extents allocated for a certain logical structure. The following table describes the different types of segments.

Segment	Description
Data segment	Each nonclustered table has a data segment. All table data is stored in the extents of the data segment. For a partitioned table, each partition has a data segment. Each cluster has a data segment. The data of every table in the cluster is stored in the cluster's data segment.
Index segment	Each index has an index segment that stores all of its data. For a partitioned index, each partition has an index segment.
Temporary segment	Temporary segments are created by Oracle when a SQL statement needs a temporary database area to complete execution. When the statement finishes execution, the extents in the temporary segment are returned to the system for future use.
Rollback segment	If you are operating in automatic undo management mode, then the database server manages undo space using tablespaces. Oracle recommends that you use automatic undo management. Earlier releases of Oracle used rollback segments to store undo information. The information in a rollback segment was used during database recovery for generating read-consistent database information and for rolling back uncommitted transactions for users. Space management for these rollback segments was complex, and Oracle has deprecated that method. This book discusses the undo tablespace method of managing undo; this eliminates the complexities of managing rollback segment space, and lets you exert control over how long undo is retained before being overwritten. Oracle does use a SYSTEM rollback segment for performing system

Segment	Description
	transactions. There is only one SYSTEM rollback segment and it is created automatically at CREATE DATABASE time and is always brought online at instance startup. You are not required to perform any operations to manage the SYSTEM rollback segment.

Oracle dynamically allocates space when the existing extents of a segment become full. In other words, when the extents of a segment are full, Oracle allocates another extent for that segment. Because extents are allocated as needed, the extents of a segment may or may not be contiguous on disk.

Database Writer (DBW*n*)

The server process records changes to undo and data blocks in the database buffer cache.DBWn writes the dirty buffers from the database buffer cache to data files. It ensures that a sufficient number of free buffers (buffers that can be overwritten when server processes needto read blocks from the data files) are available in the database buffer cache. Server processes make changes only in the database buffer cache. This is a prerequisite for high databaseperformance.

Checkpoint (CKPT)

Every three seconds (or more frequently), the CKPT process stores data in the control file to document which modified data blocks DBWn has written from the SGA to disk. This is called a “checkpoint”. The purpose of a checkpoint is to identify that place in the online redo log file where instance recovery is to begin (which is called the “checkpoint position”). In the event of a log switch, the CKPT process also writes this checkpoint information to the headers of data files.
Checkpoints exist for the following reasons:
• To ensure that modified data blocks in memory are written to the disk regularly so that data is not lost in case of a system or database failure
• To reduce the time required for instance recovery. Only the online redo log file entries following the last checkpoint need to be processed for recovery.
• To ensure that all committed data has been written to data files during shutdown
The checkpoint information written by the CKPT process includes checkpoint position, system change number, location in the online redo log file to begin recovery, information about logs, and so on.

Redo Log Files and LogWriter

Redo log files record changes to the database as a result of transactions and internal Oracle server actions. (A transaction is a logical unit of work, consisting of one or more SQL statements run by a user). Redo log files protect the database from the loss of integrity
because of system failures caused by power outages, disk failures, and so on. Redo log files must be multiplexed to ensure that the information stored in them is not lost in the event of a disk failure. The redo log consists of groups of redo log files. A group consists of a redo log file and its multiplexed copies. Each identical copy is said to be a member of that group, and each group is identified by a number. The LogWriter (LGWR) process writes redo records from the redo log buffer to all members of a redo log group until the file is filled or a log switch operation is
requested. Then it switches and writes to the files in the next group. Redo log groups are used in a circular fashion.

Archiver (ARC*n*)

ARCn is an optional background process. However, it is crucial to recovering a database after the loss of a disk. As online redo log files get filled, the Oracle instance begins writing to the next online redo log file. The process of switching from one online redo log file to another is called a log switch. The ARCn process initiates

backing up or archiving of the filled log group at every log switch. It automatically archives the online redo log file before the log can be reused, so all of the changes made to the database are preserved. This enables recovery of
the database to the point of failure even if a disk drive is damaged.
One of the important decisions that a DBA has to make is whether to configure the database to operate in ARCHIVELOG mode or in NOARCHIVELOG mode.
• In NOARCHIVELOG mode, the online redo log files are overwritten each time a log switch occurs.
• In ARCHIVELOG mode, inactive groups of filled online redo log files must be archived before they can be used again.

System Monitor (SMON)

The system monitor (SMON) process performs recovery, if necessary, at instance startup.SMON is also responsible for cleaning up temporary segments that are no longer in use and for coalescing contiguous free extents within dictionary-managed tablespaces. If any terminated transactions are skipped during instance recovery because of file-read or offline errors, SMON recovers them when the tablespace or file is brought back online. SMON checks regularly to see whether it is needed. Other processes can call SMON if they detect a need for it.

Process Monitor (PMON)

The process monitor (PMON) performs process recovery when a user process fails. PMON is responsible for cleaning up the database buffer cache and freeing resources that the user process has been using.
PMON periodically checks the status of server processes and restarts any processes that have stopped running (but not those that are intentionally terminated by the Oracle instance).
PMON checks regularly to see whether it is needed and can be called if another process detects the need for it.

INSTANCE MEMORY STRUCTURES

Oracle creates and uses memory structures to complete several jobs. For example, memory stores program code being run and data shared among users. Two basic memory structures are associated with Oracle: the system global area and the program global area. The following subsections explain each in detail.

a.System Global Area

The **System Global Area (SGA)** is a shared memory region that contains data and control information for one Oracle instance. Oracle allocates the SGA when an instance starts and deallocates it when the instance shuts down. Each instance has its own SGA.

Users currently connected to an Oracle database share the data in the SGA. For optimal performance, the entire SGA should be as large as possible (while still fitting in real memory) to store as much data in memory as possible and to minimize disk I/O.

The information stored in the SGA is divided into several types of memory structures, including the **database buffer**s, **redo log buffer**, and the **shared pool**.

i.Database Buffer Cache of the SGA

Database buffers store the most recently used blocks of data. The set of database buffers in an instance is the database **buffer cache**. The buffer cache contains modified as well as unmodified blocks. Because the most recently (and often, the most frequently) used data is kept in memory, less disk I/O is necessary, and performance is improved.

ii.Redo Log Buffer of the SGA

The **redo log buffer** stores **redo entries**—a log of changes made to the database. The redo entries stored in the redo log buffers are written to an **online redo log**, which is used if database recovery is necessary. The size of the redo log is static.

iii.Shared Pool of the SGA

The shared pool contains shared memory constructs, such as shared SQL areas. A shared SQL area is required to process every unique SQL statement submitted to a database. A shared SQL area contains information such as the parse tree and execution plan for the corresponding statement. A single shared SQL area is used by multiple applications that issue the same statement, leaving more shared memory for other uses.

Statement Handles or Cursors

A **cursor** is a handle or name for a private SQL area in which a parsed statement and other information for processing the statement are kept. (Oracle Call Interface, OCI, refers to these as **statement handles**.) Although most Oracle users rely on automatic cursor handling of Oracle utilities, the programmatic interfaces offer application designers more control over cursors.

For example, in precompiler application development, a cursor is a named resource available to a program and can be used specifically to parse SQL statements embedded within the application. Application developers can code an application so it controls the phases of SQL statement execution and thus improves application performance.

b.Program Global Area

The **Program Global Area (PGA)** is a memory buffer that contains data and control information for a server process. A PGA is created by Oracle when a server process is started. The information in a PGA depends on the Oracle configuration.

STARTING UP ORACLE DATABASE

The three steps to starting an Oracle database and making it available for system wide use are:

1. Start an instance.
2. Mount the database.
3. Open the database.

A database administrator can perform these steps using the SQL*Plus STARTUP statement or Enterprise Manager. When Oracle starts an instance, it reads the server parameter file (SPFILE) or initialization parameter file to determine the values of initialization parameters. Then, it allocates an SGA and creates background processes.

How Oracle Works

The following example describes the most basic level of operations that Oracle performs. This illustrates an Oracle configuration where the user and associated server process are on separate computers (connected through a network).

1. An **instance** has started on the computer running Oracle (often called the **host** or **database server**).
2. A computer running an application (a **local computer** or **client workstation**) runs the application in a **user process**. The client application attempts to establish a **connection** to the server using the proper Oracle Net Services driver.
3. The server is running the proper Oracle Net Services driver. The server detects the connection request from the application and creates a dedicated server process on behalf of the user process.
4. The user runs a SQL statement and commits the transaction. For example, the user changes a name in a row of a table.
5. The server process receives the statement and checks the **shared pool** for any shared SQL area that contains a similar SQL statement. If a shared SQL area is found, then the server process checks the user's access privileges to the requested data, and the previously existing shared SQL area is used to process the statement. If not, then a new shared SQL area is allocated for the statement, so it can be parsed and processed.
6. The server process retrieves any necessary data values from the actual datafile (table) or those stored in the SGA.
7. The server process modifies data in the system global area. The DBW*n* process writes modified blocks permanently to disk when doing so is efficient. Because the transaction is committed, the LGWR process immediately records the transaction in the redo log file.
8. If the transaction is successful, then the server process sends a message across the network to the application. If it is not successful, then an error message is transmitted.
9. Throughout this entire procedure, the other background processes run, watching for conditions that require intervention. In addition, the database server manages other users' transactions and prevents contention between transactions that request the same data.

SHUTDOWN [ABORT|IMMEDIATE|NORMAL|TRANSACTIONAL [LOCAL]]

Shuts down a currently running Oracle instance, optionally closing and dismounting a database.

ABORT

Proceeds with the fastest possible shutdown of the database without waiting for calls to complete or users to disconnect.

Uncommitted transactions are not rolled back. Client SQL statements currently being processed are terminated. All users currently connected to the database are implicitly disconnected and the next database startup will require instance recovery.

You must use this option if a background process terminates abnormally.

IMMEDIATE

Does not wait for current calls to complete or users to disconnect from the database.

Further connects are prohibited. The database is closed and dismounted. The instance is shutdown and no instance recovery is required on the next database startup.

NORMAL

NORMAL is the default option which waits for users to disconnect from the database.

Further connects are prohibited. The database is closed and dismounted. The instance is shutdown and no instance recovery is required on the next database startup.

TRANSACTIONAL [LOCAL]

Performs a planned shutdown of an instance while allowing active transactions to complete first. It prevents clients from losing work without requiring all users to log off.

No client can start a new transaction on this instance. Attempting to start a new transaction results in disconnection. After completion of all transactions, any client still connected to the instance is disconnected. Now the instance shuts down just as it would if a SHUTDOWN IMMEDIATE statement was submitted. The next startup of the database will not require any instance recovery procedures.

The LOCAL mode specifies a transactional shutdown on the local instance only, so that it only waits on local transactions to complete, not all transactions. This is useful, for example, for scheduled outage maintenance.

Usage

SHUTDOWN with no arguments is equivalent to SHUTDOWN NORMAL.

You must be connected to a database as SYSOPER, or SYSDBA. You cannot connect via a multi-threaded server.

Types of Failures

Several circumstances can halt the operation of an Oracle database. The most common types of failure are described in the following table.

Failure	**Description**
User error	Requires a database to be recovered to a point in time before the error occurred. For example, a user could accidentally drop a table. To enable recovery from user errors and accommodate other unique recovery requirements, Oracle provides exact point-in-time recovery. For example, if a user accidentally drops a table, the database can be recovered to the instant in time before the table was dropped.
Statement failure	Occurs when there is a logical failure in the handling of a statement in an Oracle program. When statement failure occurs, any effects of the statement are automatically undone by Oracle and control is returned to the user.
Process	Results from a failure in a user process accessing Oracle, such as an

Failure	Description
failure	abnormal disconnection or process termination. The background process PMON automatically detects the failed user process, rolls back the uncommitted transaction of the user process, and releases any resources that the process was using.
Instance failure	Occurs when a problem arises that prevents an instance from continuing work. Instance failure can result from a hardware problem such as a power outage, or a software problem such as an operating system failure. When an instance failure occurs, the data in the buffers of the system global area is not written to the datafiles. After an instance failure, Oracle automatically performs **instance recovery**. If one instance in a RAC environment fails, then another instance recovers the redo for the failed instance. In a single-instance database, or in a RAC database in which all instances fail, Oracle automatically applies all redo when you restart the database.
Media (disk) failure	An error can occur when trying to write or read a file on disk that is required to operate the database. A common example is a disk head failure, which causes the loss of all files on a disk drive. Different files can be affected by this type of disk failure, including the datafiles, the redo log files, and the control files. Also, because the database instance cannot continue to function properly, the data in the database buffers of the system global area cannot be permanently written to the datafiles. A disk failure requires you to restore lost files and then perform **media recovery**. Unlike instance recovery, media recovery must be initiated by the user. Media recovery updates restored datafiles so the information in them corresponds to the most recent time point before the disk failure, including the committed data in memory that was lost because of the failure.

Data Availability

Availability is the holy grail of database administrators. If the data is not available, the applications cannot run. If the applications cannot run, the company loses business. Therefore, DBAs are responsible for doing everything in their power to ensure that databases are kept online and operational. This has been the duty of the DBA since the first days of the database.

Defining Availability

Simply stated, availability is the condition where a given resource can be accessed by its consumers. This means that if a database is available, the users of its data—that is, applications, customers, and business users—can access it. Any condition that renders the resource inaccessible causes the opposite of availability: unavailability.

Another definition of availability is the percentage of time that a system can be used for productive work. The required availability of an application will vary from organization to organization, within an organization from system to system, and even from user to user.

Database availabilityand ***database performance***are terms that are often confused with each other, and indeed, there are similarities between the two. The major difference lies in the user's ability to access the database. It is possible to access a database suffering from poor performance, but it is not possible to access a database that is unavailable. So, when does poor performance turn into unavailability? If performance suffers to such a great degree that the users of the database cannot perform their jobs, the database has become, for all intents and purposes, unavailable. Nonetheless, keep in mind that availability and performance are different and must be treated by the DBA as separate issues—even though a severe performance problem is a potential availability problem.

Components of Availability

Availability comprises four distinct components, which, in combination, assure that systems are running and business can be conducted:

1. **Manageability**— the ability to create and maintain an effective environment that delivers service to users
2. **Recoverability**— the ability to re-establish service in the event of an error or component failure
3. **Reliability**— the ability to deliver service at specified levels for a stated period
4. **Serviceability**— the ability to determine the existence of problems, diagnose their cause(s), and repair the problems

Cost of Downtime

The cost of downtime varies from company to company, each organization needs to determine the actual cost of downtime based on its customers, systems, and business operations. Some businesses can handle downtime better than others.

For other businesses that can "get by" using manual systems during an outage, downtime is not as much of a disaster. The truth is, outages impact every business, and any nontrivial amount of downtime will impose a cost on the organization. When estimating the cost of downtime, remember to factor in all of the costs, including:

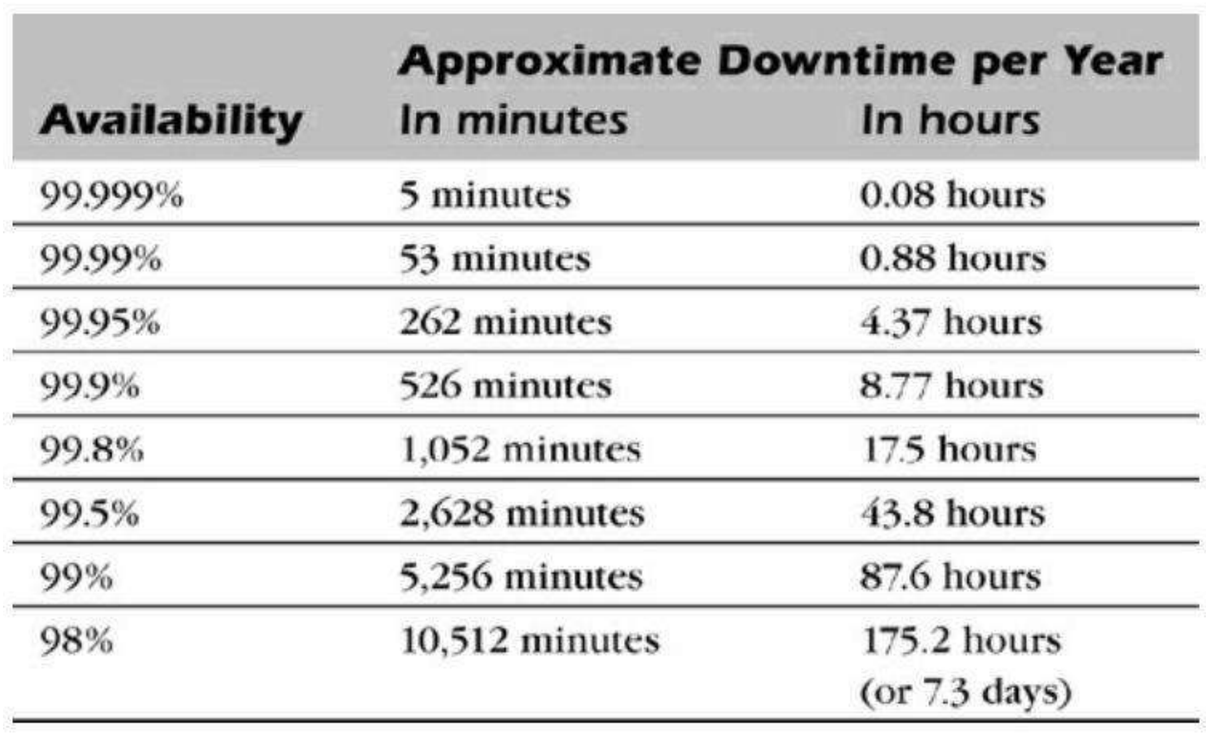
- Lost business during the outage
- Cost of catching up after systems are once again available
- Legal costs of any lawsuits
- Impact of reduced stock value (especially for dot-coms that rely on computerized systems for all of their business)

Failure to prepare an estimate of the cost of downtime will make it more difficult to cost-justify the measures a DBA needs to take to ensure data availability.

How Much Availability Is Enough?

Availability is traditionally discussed in terms of the percentage of total time that a service needs to be up. For example, a system with 99 percent availability is up and running 99 percent of the time and down, or unavailable, 1 percent of the time.

So, just how much availability is enough? In this Internet age, the push is on to provide never-ending uptime, 365 days a year, 24 hours a day. At 60 minutes an hour, that mean 525,600 minutes of uptime a year. Clearly, to achieve 100 percent availability is a laudable goal, but just as clearly an unreasonable one. The term five nines is often used to describe highly available systems. Meaning 99.999 percent uptime, five nines describes what is essentially 100 percent availability, but with the understanding that some downtime is unavoidable



	Approximate Downtime per Year	
Availability	**In minutes**	**In hours**
99.999%	5 minutes	0.08 hours
99.99%	53 minutes	0.88 hours
99.95%	262 minutes	4.37 hours
99.9%	526 minutes	8.77 hours
99.8%	1,052 minutes	17.5 hours
99.5%	2,628 minutes	43.8 hours
99%	5,256 minutes	87.6 hours
98%	10,512 minutes	175.2 hours (or 7.3 days)

Even though 100 percent availability is not reasonable, some systems are achieving availability approaching five nines. DBAs can take measures to design databases and build systems that are created to achieve high availability. However, just because high availability can be built into a system does not mean that every system should be built with a high availability design. That is so because a highly available system can cost many times more than a traditional system designed with unavailability built into it. The DBA needs to negotiate with the end users and clearly explain the costs associated with a highly available system.

The amount of availability that should be built into the database environment will be based on cost. How much availability can the application owner afford? That is the ultimate question. Although it may be possible to achieve high availability, it may not be cost-effective, given the nature of the application and the budget available to support it. The DBA needs to be proactive in working with the application owner to make sure the application owner fully understands the cost aspect of availability.

Potential Causes of Data Unavailability

1. **Loss of the Data Center**
   Quite obviously, data will not be available if the data center is lost due to a natural disaster or some other type of catastrophe. Whether the disaster is small from a business perspective and impacts only the computing resources of the business or whether it is large and impacts an entire building or the entire organization, losing the computer means losing the database and any data it contains.

   To restore availability in a disaster situation usually requires recreating the database environment (and perhaps much more) at a remote location. From an availability perspective, losing the data center is the worst type of availability problem the DBA could ever encounter.

2. **Network Problems**
   If a database server is on a single network switch and that switch incurs an outage, the database will be unavailable. Consider implementing redundant network switches to prevent such outages.

   Loss of network access also can cause a database outage. Such problems are usually caused by malfunctioning hardware, such as the network card in the database server. It is wise to have spare networking hardware available for immediate replacement in case such a problem occurs.

   DBAs should build good working relationships with the networking specialists in their organization. When networking problems occur, the DBA can consult with the network specialists right away and ideally learn something from the experience of working with them.

3. **Loss of the Server Hardware**
   At a basic level, the database server hardware consists of the CPU, the memory, and the disk subsystems holding the databases. Obviously, if the CPU is damaged or becomes unavailable for any reason, the database also will not be available. However, the database files should remain usable and could possibly be connected to another CPU to bring the database back online.

   To avoid outages due to CPU failure, consider using hardware cluster failover techniques. When using cluster failover, the loss of a single server causes the system to process on another node of the cluster. The data need not be moved, and failover is automatic.

If system memory fails, the database may or may not be available. If all system memory fails, any database on the system will be unavailable because memory is required for a relational database system to operate. To resolve this situation you will need to replace the failing RAM chips or modules.

If the entire database server is lost or damaged, the failure will be more difficult to address. Loss of an entire server implies that the CPU, memory, and disk subsystem are unavailable. Once again, in such a situation the databases on the server also will be unavailable. If the entire server platform fails, the database will need to be recreated.

4. **Disk-Related Outages**
   Because databases rely on physical disk structures to actually store the data, database availability is particularly vulnerable to disk failures. Of course, the degree of database unavailability depends on the type of disk system and the type of outage the disk system has suffered. Disk drives fail for numerous reasons: The physical drive mechanism may fail, the controller could fail, or perhaps a connecting wire has loosened.

   To restore the database, the DBA can recover the data using backup copies of the database files and logs.

5. **Operating System Failure**
   Not all database availability problems are caused by hardware problems. Software also can be the culprit. For example, data will not be available during an operating system (OS) failure or outage, even if all of the server hardware is operational. Typical causes of operating system outages include general OS instability due to inherent bugs, problems encountered when upgrading an OS version, or problems with patches applied to the operating system.

   When an OS failure occurs, the only viable options for restoring data availability are to fix the OS problem or to restore the database on another server with a functional operating system. Once again, a failover system can be used to minimize downtime caused by a software failure.

6. **DBMS Software Failure**
   Similar to the failure of an operating system, a failure in the DBMS software will cause unavailability. If the DBMS is not operational, the data in its databases cannot be accessed. DBMS failure occurs for similar reasons that an OS fails: general DBMS instability due to inherent bugs, problems encountered when upgrading to a new version of the DBMS, or problems when patches are applied to the DBMS software. The DBMS may also fail when resources it needs to operate are not available—such as start-up parameters, certain system files, and memory structures. For example, if the

database log file is corrupted or missing, the DBMS will not allow data to be modified.

7. **Application Problems**
   Application software also can cause availability problems. If the application is the only way that end users can access data, a data unavailability problem arises when the application fails. An application software failure is unlikely to cause a database outage, but an operational database means little if the users can’t use their usual programs to access the data. The DBA and other sophisticated users will be able to access data in the database using alternative methods such as interactive SQL or query tools.

8. **Security and Authorization Problems**
   Security-related problems are another cause of database unavailability. This type of problem is caused by improper use or administration of the database. Before data can be accessed, either directly by an end user or by means of a program, authorization must be granted—typically by the DBA or security administrator. If, for any reason, the requesting agent does not have current authorization to access the database, the data will be unavailable to the requester. Security-related problems usually occur immediately after an application goes into production or when a new user attempts to use the system but has yet to receive permission. DBA error can also cause security related problems: If the DBA accidentally overwrites or removes valid authorizations from the DBMS, valid users will not be able to access data.

9. **Loss of Data**
   It is possible for data to be unavailable because it no longer exists, which can occur if data is deleted or overwritten erroneously. Accidental mass deletes, application program bugs, or even malicious attacks can all cause data loss. When data is deleted in error, the DBA may need to recover the database to a point in time before the data was deleted. This can be accomplished using the RECOVER or RESTORE function of the DBMS in use. The DBA reviews the transaction logs to determine when the error occurred and then restores the data to the point in time just before the error. Analyzing and interpreting the contents of a database transaction log can be a difficult proposition, but there are third-party tools on the market that simplify the task.

10. **DBA Mistakes**
    One of the biggest causes of database downtime is human error. In fact, one major DBMS vendor states that 70 percent of the calls it receives for database outages are the result of DBA mistakes. While there is nothing you can do to guarantee that mistakes will not be made, proper DBA training and tools can minimize mistakes.

Be sure that all DBAs have received proper training before giving them responsibility for critical-production database systems. Training should consist of both course material (either instructor-led or computer-based training) and on-the-job experience. A DBA's first administration experience should always be with a test system—never with production data.

Ensuring Availability

Now that we have established that life is challenging for today's DBAs, we will shift our focus to some techniques that help promote higher availability. Faced with shrinking budgets and resources, and an ever-increasing volume of data to manage, IT organizations need to evaluate their critical needs and implement a series of key strategic steps to ensure availability.

Good strategy should include steps to:

- Perform routine maintenance while systems remain operational
- Automate DBA functions
- Exploit the features of the DBMS that promote availability
- Exploit hardware technologies e.g. clustering

Database Performance

Database performance focuses on tuning and optimizing the design, parameters, and physical construction of database objects, specifically tables and indexes, and the files in which their data is stored. The actual composition and structure of database objects must be monitored continually and changed accordingly if the database becomes inefficient. No amount of SQL tweaking or system tuning can optimize the performance of queries run against a poorly designed or disorganized database.

Techniques for Optimizing Databases

Most of the major DBMSs support the following techniques although perhaps by different names. Each of the following techniques can be used to tune database performance.

1. **Partitioning**
   Breaking a single database table into sections stored in multiple files.

2. **Indexing**
   Choosing the proper indexes and options to enable efficient queries.

3. **Denormalization**
   Varying from the logical design to achieve better query performance.

4. **Clustering**
   Enforcing the physical sequence of data on disk.

5. **Interleaving data**
   Combining data from multiple tables into a single, sequenced file.

6. **Free space**
   Leaving room for data growth.

7. **Compression**
   Algorithmically reducing storage requirements.

8. **File placement and allocation**
   Putting the right files in the right place.

9. **Page size**
   Using the proper page size for efficient data storage and I/O.

10. **Reorganization**
   Removing inefficiencies from the database by realigning and restructuring database objects.

Database Security

Database security and protection are receiving more attention and budget from organizations due to the steady increase in data breaches and the resultant regulations designed to abate them. However, database security still requires more focus and effort.

So the cost of a data breach can be quite steep. And database servers are favorite sources for hackers to target. Thus, it makes sense to spend some time and money up-front to better secure and protect the data in our database systems.

Database Security Basics

The basic security and authorization approach taken by DBMS vendors to secure database access is that **all database resources are controlled by the DBMS**.No default authorizations are given to any user just because the user logs in to the DBMS. Therefore, for a user to be able to perform any DBMS operation or function, one of the following conditions must exist:

1. The user has been granted the ability to perform that function or operation, or
2. That operation or function has been granted generically to all users.

Using the security features of the DBMS, the DBA can set up the environment such that only certain users or certain application programs are allowed to perform certain operations on certain data within the database. Each user's function within the organization should determine the authorized level of database access. For example, only general-ledger programmers, batch jobs, and programs can access and modify the general-ledger databases. Different checks can be established for each type of access to each type of information, and different users can be assigned different access rights to different database objects.
At a high level, database security boils down to answering four questions:

1. Who is it? (authentication)
2. Who can do it? (authorization)
3. Who can see it? (encryption)
4. Who did it? (audit)

Strong authentication is the cornerstone of any security implementation plan. It is impossible to control authorization and track usage without it. Before authorization to use database resources can be granted, a **login**needs to be established for each user of the DBMS. Logins are sometimes referred to as accounts, or user IDs. The login will have a password associated with it such that only those who know the password can use the login ID.

Passwords should be changed regularly over time to make it difficult for hackers to gain access to the DBMS surreptitiously. As a DBA, you may decide to set up automated procedures—such as an e-mail notification system—to coerce users into changing their login password every month or so. Users who do not change their login password can be disabled until they call to complain. Of course, this adds to the workload of the DBA, but it does enhance the security of the DBMS.

When a DBMS user no longer requires access to the DBMS, or leaves the company, the DBA should drop that user's login from the system as soon as possible.

Privileges

Two types of privileges are important relating to database security within the database environment: system privileges and object privileges.

1. System Privileges

System privileges allow a user to perform administrative actions in a database. These include privileges (as found in SQL Server) such as: create database, create procedure, create view, backup database, create table, create trigger, and execute.

2. Object Privileges

Object privileges allow for the use of certain operations on database objects as authorized by another user. Examples include: usage, select, insert, update, and references.

The Principal of least Privilege, and Separation of duties:

Databases fall under internal controls, that is, data used for public reporting, annual reports, etc.) are subject to the separation of duties, meaning there must be segregation of tasks between development, and production.

Each task has to be validated (via code walk-through/fresh eyes) by a third person who is not writing the actual code. The database developer should not be able to execute anything in production without an independent review of the documentation/code for the work that is being performed. Typically, the role of the developer is to pass code to a DBA; however, given the cutbacks that have resulted from the economic downturn, a DBA might not be readily available. If a DBA is not involved, it is important, at minimum, for a peer to conduct a code review. This ensures that the role of the developer is clearly separate.

Another point of internal control is adherence to the principle of providing the least amount of privileges, especially in production. To allow developers more access to get their work done, it is much safer to use impersonation for exceptions that require elevated privileges (e.g. EXECUTE AS or sudo to do that temporarily). Often developers may dismiss this as “overhead” while on their path to coding glory. Please be aware, however, that DBAs must do all that is considered responsible because they are the de facto data stewards of the organization and must comply with regulations and the law.

Database Security Threats

Databases face a number of security threats. Many of these threats are common to all computer systems, but large databases in organizations are particularly vulnerable because they often contain sensitive information and are used by many different people.

One of the basic threats is data loss, which means that parts of a database can no longer be retrieved. The earlier scenario where the university lost part of their records has disastrous consequences.

This could be the result of physical damage to the storage medium, like fire or water damage, human error or hardware failures. Every single computer system is to some degree vulnerable, so a common strategy employed by DBAs is to create multiple backups on different computer systems. So if one system were to fail, the data is still secure somewhere else.

Another security threat is unauthorized access. Many databases contain sensitive information, and it could be very harmful if this information were to fall in the wrong hands. Imagine someone getting a hold of your social security number, date of birth, address and bank information. It would be relatively easy for someone to open up a credit card under your name and start spending without your knowledge.

Getting unauthorized access to computer systems is known as hacking. Computer hackers have developed sophisticated methods to obtain data from databases, which they may use for personal gain or to harm others. Have you ever received an e-mail with a notification that you need to log in to your credit card account with a link for you to follow? Most likely, this is a hacker trying to obtain your log in details, so be careful.

A third category of security threats consists of viruses and other harmful programs. A computer virus is a computer program that can cause damage to a computer's software, hardware or data. It is referred to as a 'virus' because it has the capability to replicate itself and hide inside other computer files. There are many types of viruses and new ones are being developed all the time.

Once a virus is present on a computer, it typically performs some type of harmful action, such as corrupting data or obtaining sensitive information. Computer viruses are one type of malware, short for 'malicious software.' Malware is used by attackers to disrupt computer operation. In addition to computer viruses, this includes spyware (used to collect information about a person without their knowledge), adware (used to display advertising) and Trojan horses (used to create unauthorized access to someone's computer).

One of the most common ways to get a virus on your computer is to download a file from the Internet that is infected. So if you get an e-mail from someone you don't know with a file attached to it, be careful opening up these attachments. As the database administrator of your own computer, you should use some type of anti-virus software.

Authorization Steps

You may be tempted to skip ahead because you think you already have access controls in place, and then promptly fail a security review. Just because you have an access control system does not mean your system is secure. Database authentication and domain authentication are different, and great care must be taken so these two systems coordinate access and don't allow users to bypass database authorization entirely.

This is your first line of defense for database and data security, and it warrants close inspection to ensure proper configuration of accounts, as well as proper deployment of the two systems. Keep in mind that the longer a database has been in operation, the more access rights drifts away from a secure baseline.

Change default user passwords immediately upon installing the database. Periodically verify they have not reverted because of a reinstall or account reset.

Lock user accounts not in use. If you are certain they will never be used, remove them. This is especially important with canned database testing, tutorials or demonstration users. These known accounts are packaged with the database, and are exploitable to gain access to data and database functions.

Enforce stronger passwords. If you are using domain-level access to control database authorization, then you can set policies for stronger passwords. Our recommendation is you rotate passwords, and move away from static passwords.

Remove public accounts, as well as public access from all accounts. There is no use case where you want the general public to have access to your database.

Choose domain authentication or database authentication for your database users, and stick with it. Do not mingle the two. Confusion of responsibility will create security gaps.

Examine roles and groups closely. List out user permissions, roles and group participation, and review it to make sure users have just enough authorization to do their job. Unfortunately, depending upon the number of users in your database, this can be time consuming. Even when automation tools collect permissions assigned to each user account, manual review of settings is still required to detect problems. The bad news is this is not fun, and for large databases, you should plan on

spending a day to get the permissions mapping right. The good news is once it is set, it is much easier to detect unwanted changes and stop escalated privileges.
Protect administrative functions from users. Database vendors list functions, roles, stored procedures and utilities dedicated for administration. Do not delegate these functions to users.
Divide database admin duties. For companies with more than one database administrator, divide administrative tasks between different admins, operating under different administrator accounts. The relational database platforms provide advanced access control provisions to accomplish this, allowing both for separation of duties, as well as locking down the master DBA account.

Assess Database Configuration

This is very important for determining security and operational integrity.

- **Find** out how your databases are configured, either through database queries and analysis of configuration files, or via a freely available assessment tools. All database vendors recommend configuration and security settings, and it does not take very long to compare your configuration to the standard.
- **Remove** modules and services you don't need. Not using replication, for example? Then remove those packages. These services may or may not be secure, but their absence assures you are not leaving an open door for hackers.
- **Document** approved configuration baseline for databases. This should be used for reference by all administrators, as well as a guideline to detect misconfigured systems.
- **Use scanning tools** to discover the databases you have, and be consistent when applying configuration settings.

Assess Database/Platform Interaction

All databases provide means to directly call operating system commands for administrative tasks. These functions are comprised of OS and database code, running under administrative permissions, and offering a bidirectional portal to the database. These recommendations are meant to close security holes along this boundary.

- **Disable** extended or external stored procedures.
- **Ensure** the database owner account on local platform, under which the database is installed, is not assigned domain administrator functions.
- **Make** sure domain administrators are not database administrators.
- **Tie import/export** utilities, startup scripts, registry entries or properties files to the local database owner credentials.

Secure Communications

You want to make sure that communications to the database are kept private.

- Encrypt sessions between applications and the database, especially Web application connections.
- Reset database port numbers to non-default value. For example, moving Oracle's default port of 1521 to a random value defeats automated attacks, and makes it harder for an attacker to probe for information.
- Block ad-hoc connections. Ad-hoc connections from undesired locations, time of day or through unapproved applications can be detected and rejected by simple login triggers, database firewalls and some access control systems.

PATCH THE DATABASE

Your goal is to leverage the security knowledge and expertise of the database vendor, allowing them to find and address security issues. This requires certifying and installing patches on a regular basis.

- Create an environment and process to perform a sanity functions check on database patches prior to production deployment.
- Don't allow patch downloads by individual DBAs; Rather have centralized, approved and verified copies available.
- Synch internal patch cycles with vendor patch releases.
- Reconfigure in the cases where patch or alteration of functions is unacceptable. If you employ database/Web application firewalls, determine if you can block the threat until a suitable patch is available.

Application Usage Of The Database

Enterprise and Web applications leverage more than basic data storage, using service accounts that are provisioned with a broad range of capabilities.

- Segment the authorization between common users and application administration accounts.
- Restrict connection pooling where a single database account is leveraged by all database users. If possible, divide the application processing into different groupings, and perform these operations under different database user accounts. In addition, access permissions can be minimized in accordance with the role as it provides segregation of duties and makes log file analysis easier.
- Modify the application-to-database connection to allow for database queries to be associated with an end user. This makes audit analysis and policy enforcement easier.

Media Protection

Protecting backup media is not optional because lost media is the leading cause of data breaches. There are several methods available that do not require alteration of processes or applications, including database transparent encryption, which requires no changes to application code and is often available free from the database vendor.

Log And Event Review

- Use logging if you can.
- Create a log retention policy, determine what events you don't need and filter them out.
- Review logs periodically, focusing on failures of system functions and logins that indicate system probing.
- Review log settings periodically.

Embrace Insecurity

No matter what you do, you will never be 100 percent secure. Take this into account, and plan your response to security events.

- Inventory your databases.
- Discover and catalog your sensitive data.
- Have a plan on what to do if data is lost or stolen.
- Have a disaster recovery plan.

- Create a cooperative culture, get to know the applications developers and administrators, and make sure they understand that everyone needs to work together. If you have a compliance group, get to know them, and ask for their advice and opinions.

COMPLIANCE FORCES ENCRYPTION, AUDITING
If you have valuable data, odds are you have an industry or governmental obligation to perform the following recommendations. They are good security practices for everyone, but as it costs additional time and money to implement, they were largely ignored prior to regulatory pressure. The two most commonly prescribed regulatory requirements are auditing and encryption. These functions can be accomplished with the tools provided by the database vendor, but given the difficulty of implementation, deployment and management of these systems, you will be purchasing additional tools and platforms to alleviate day-to-day management and performance issues.

Auditing. Database auditing is used to capture a record of database transactions, which is then used to detect suspect activity and perform forensic audits. All of the relational database platforms have auditing features that capture transactions on the data and administrative operations against the database system. Depending upon the vendor and how the feature is deployed, you can gather detailed transactional information, filter unwanted transactions to capture a succinct view of database activity, and do so with only modest performance impact.

Using the standard software provided by database vendors will be ample to collect needed data, but you will need to develop a review process and reports to demonstrate compliance. Database auditing monitoring and log management tools are also available to automate these efforts. While the latter requires additional investment, these tools provide better performance, are easier to use, and have prebuilt policies and reports specifically designed for regulations.

Encryption. There are many forms of database encryption available, but they typically break into two families: transparent encryption that covers the entire database and requires no modifications to business processes, and user encryption applied only to select objects within the database that requires alteration of the application code. Transparent encryption is really designed to protect data on media, such as disk drives and backup tapes, from being accessed outside of the database. User encryption can be used for both media protection and protecting data from misuse.

When we discuss encryption to meet regulatory mandates, transparent encryption options are not suitable measures to meet requirements such as the Payment Card Industry Data Security Standard (PCI DSS), but they do satisfy most state data breach notification law requirements. As the cost and complexity is radically different between these two options, you will need to discuss which is appropriate to satisfy your auditors before deciding upon a course of action. Make sure you are addressing the right threat before deciding how you are going to implement database encryption.

You can begin any of these actions today, and they are proven effective at preventing the most common database attacks. Better still, the basic steps are free, and with a little of your time and energy, they can be completed without having to purchase specialized products or services. You can buy aftermarket products to perform these tasks, and they will do the job better, faster and with less effort on your part.

You need not let cost stop you from performing these security steps, and in this economic climate where we are all expected to do more with less, that is a good thing.

Granting and Revoking Authority

The DBA controls database security and authorization using Data Control Language, or DCL. DCL is one of three subtypes of SQL. (The other two are DDL and DML.) DCL statements are used to control which users have access to which objects and commands. These statements are the manner in which database security is enacted. DCL statements comprise two basic types:

- GRANT assigns a permission to a database user.
- REVOKE removes a permission from a database user.

Creating User Accounts in Oracle

In Oracle, we can specify not only who can connect to the database server but also from which host that the user connects. Therefore, a user account in Oracle consists of a username and a host name separated by the **@**character.

For example, if the **admin**user connects to the Oracle database server from **localhost**,the user account is **admin@localhost**.

The **admin**user only can connect to the Oracle database server from the **localhost**,not from a remote host such as **tukenya.ac.ke**.This makes the **Oracle** database server even more secure.

In addition, by combining the username and host, it is possible to setup multiple accounts with the same name but can connect from different hosts with the different privileges.

Oracle stored the user accounts in the **user**grant table of the oracledatabase.

Oracle provides the **CREATE USER**statement that allows us to create a new user account. The syntax of the **CREATE USER**statement is as follows:

```
CREATE USER user_account IDENTIFIED BY password;
```

The ***user_account***is in the format **'username'@'hostname'**.

The password is specified in the **IDENTIFIED BY**clause. The password must be in clear text. Oracle will encrypt the password before saving the user account into the **user**table.

For example, to create a new user **dbadmin**that connects to the Oracle database server from the **localhost**with the password **secret**,we use the **CREATE USER** statement as follows:

```
CREATE USER dbadmin@localhost IDENTIFIED BY secret;
```

To view the privileges of a user account, we use the **SHOW GRANTS**statement as follows:

```
SHOW GRANTS FOR dbadmin@localhost;
```

The **`*.*`**in the result shows that the **`dbadmin`**user account can only login to the database server and has no other privileges. To grant permission to the user, we shall use the **`GRANT`**statement.

Note that the part before the dot (**`.`**)represents the database and the part after the dot (**`.`**)represents the table e.g. **`database.table`**.

To allow a user account to connect from any host, we use the percentage (**`%`**) wildcard as shown in the following example:

```
CREATE USER superadmin@'%' IDENTIFIED BY 'secret';
```

EXAMPLE

```
SQL >  connect system/tuk123;           (or the password )

  SQL > create user student
          identified by student123;

SQL > grant create session to student;

SQL > grant dba to student;
```

Removing User Accounts in Oracle

To remove one or more user accounts, we use the **`DROP USER`**statement as follows:

```
DROP USER user [,user]...;

DROP USER [IF EXISTS] user [,user]...;
```

Besides removing the user account, the **`DROP USER`**statement also removes all privileges from all grant tables.

Suppose a user account is logged in and has active session running. If we drop the user account, it won't stop the open sessions. The active session will continue until user exits. Typically, in this case, we should shutdown user's session immediately right before executing the **`DROP USER`**statement. It's important to note that if we don't terminate the active sessions, the removed user, if connected to the database server, still can perform all operations until the session ends.

Granting Privileges to Users in Oracle

After creating a new user account, the user doesn't have any privileges. To grant privileges to a user account, we use the **`GRANT`**statement.

The following illustrates the syntax of the **`GRANT`**statement:

```
GRANT privilege, [privilege,] ...ON privilege_level
TO user[IDENTIFIED BY password] [REQUIRE tsl_option
]
[WITH [GRANT OPTION | resource_option]];
```

Let's examine the **GRANT**statement in greater detail.

- First, specify one or more privileges after the **GRANT**keyword. If we grant the user multiple privileges, each privilege is separated by a comma. (see a list of privilege in the table below).
- Next, specify the **privilege_level**that determines the level at which the privileges apply.Oracle supports global (***.***), database (**database.***), table (**database.table**)and column levels. If we use column privilege level, we must specify one or a list of comma-separated columns after each privilege.
- Then, place the **user**that we want to grant privileges. If user already exists, the **GRANT**statement modifies its privilege. Otherwise, the **GRANT** statement creates a new user. The optional clause **IDENTIFIED BY**allows us to set a new **password**for the user.
- After that, we specify whether the user has to connect to the database server over a secure connection such as SSL.
- Finally, the optional **WITH GRANT OPTION**clause allows the user to grant other users or remove from other users the privileges that we are granting them. In addition, we can use the **WITH**clause to allocate Oracle database server's resource e.g. to set how many connections or statements that the user can use per hour. This is very helpful in the shared environments such as Oracle shared hosting.

Notice that in order to use the **GRANT**statement, the currently logged in user must have the **GRANT OPTION**privilege and the privileges that they are granting. If the **read_only**system variable is enabled, the currently logged in user needs to have the **SUPER**privilege to execute the **GRANT**statement.

Let's practice with some examples of using Oracle **GRANT**statement to have a better understanding.

Typically, we use the **CREATE USER**statement to create a new user account first and then use the **GRANT**statement to grant privileges to the user.

For example, the following **CREATE USER**statement creates a new super user account.

```
CREATE USER super@localhost IDENTIFIED BY 'dolphin';

CREATE USER rfc IDENTIFIED BY 'shark';
GRANT SELECT, UPDATE, DELETE ON imax.* TO rfc;
```

Oracle User Roles

Typically, we have a number of users with the same set of privileges. Previously, the only way to grant and revoke privileges to multiple users is to change privileges of each user individually, which is time-consuming.

To make it easier, Oracle provided a new object called **role** that is a named collection of privileges.

If we want to grant the same set of privileges to multiple users, we should do it as follows:

1. First, create a new role.
2. Second, grant privileges to the role.
3. Third, grant the role to the users.

In case we want to change the privileges of the users, we need to change the privileges of the granted role only. The changes will take effect to all users to which the role granted.

Creating Roles

Suppose we develop an application that uses the **imax**database. To interact with the **imax**database, we need to create accounts for developers who need full access to the database. In addition, we need to create accounts for users who need only read access and others who need both read/write access.

To avoid granting privileges to each user account individually, we create a set of roles and grant the appropriate roles to each user account.

To create new roles, we use **CREATE ROLE**statement:

```
CREATE ROLE imax_dev, imax_read, imax_write;
```

The role name is similar to user account that consists of user and host parts: **role_name@host_name**.

If we omit the host part, it will default to '**%**'that mean any hosts.

Granting Privileges to Roles

To grant privileges to a role, we use **GRANT**statement. The following statement grants all privileges to **imax_dev**role:

```
GRANT ALL ON imax.* TO imax_dev;
```

The following statement grants **SELECT**privilege to **imax_read**role:

```
GRANT SELECT ON imax.* TO imax_read;
```

The following statement grants **INSERT**,**UPDATE**,and **DELETE**privileges to **imax_write**role:

```
GRANT INSERT, UPDATE, DELETE ON imax.* TO imax_write;
```

Assigning Roles to User Accounts

Suppose we need one user account as the developer, one user account that can have read-only access and two user accounts that can have read/write access.

To create new users, we use **CREATE USER**statements as follows:

```
-- developer user
CREATE USER imax_dev1@localhost IDENTIFIED BY 'Secure$dev1';

-- read access user
CREATE USER imax_read1@localhost IDENTIFIED BY 'Secure$read1';

-- read/write users
CREATE  USER  imax_write1@localhost  IDENTIFIED  BY  'Secure$w1';
CREATE USER imax_write2@localhost IDENTIFIED BY 'Secure$w2';
```

To assign roles to users, we use **GRANT**statement:

```
GRANT imax_dev TO imax_dev1@localhost;

GRANT imax_read TO imax_read1@localhost;

GRANT imax_read, imax_write
TO imax_write1@localhost, imax_write2@localhost;
```

Note that the **GRANT**statement for the **imax_write1@localhost**and **imax_write2@localhost**accounts grant both **imax_read**and **imax_write**roles.

To verify the role assignments, we use the **SHOW GRANTS**statement as the following example:

```
SHOW GRANTS FOR imax_dev1@localhost;
```

As we can see, it just returned granted roles. To show the privileges the roles represent, we use the **USING**clause with the name of the granted roles as follows:

```
SHOW GRANTS FOR imax_write1@localhost USING imax_write;
```

Setting Default Roles

Now if we connect to the Oracle using the **imax_read1**user account and try to access the **imax**database:

```
>mysql -u imax_read1 -p
Enter password: ***********
mysql>USE imax;
```

The statement issued the following error message:

This is because when we granted roles to a user account, it did not automatically make the roles to become active when the user account connects to the database server.

If we invoke the **CURRENT_ROLE()**function:

```
SELECT CURRENT_ROLE();
```

It returned **NONE**,meaning no active roles.

To specify which roles should be active each time a user account connects to the database server, we use the **SET DEFAULT ROLE**statement.

The following statement set the default for the **imax_read1@localhost**account all its assigned roles.

```
SET DEFAULT ROLE ALL TO imax_read1@localhost;
```

Now, if we connect to the Oracle database server using the **imax_read1**user account and invoke the **CURRENT_ROLE()**function:

```
>mysql -u imax_read1 -p
Enter password: ***********
mysql>SELECT CURRENT_ROLE();
```

We will see the default roles for **imax_read1**user account.

We can test the privileges of **imax_read**account by switching the current database to **imax**,executing a **SELECT**statement and a **DELETE**statement as follows:

```
mysql> USE imax;
mysql> SELECT * FROM movies;
mysql> DELETE FROM movies;
```

It worked as expected. When we issued the **DELETE**statement, we received an error because **imax_read1**user account has read access only.

Setting Active Roles

A user account can modify the current user's effective privileges within the current session by specifying which granted role are active.

The following statement set the active role to **NONE**,meaning no active role.

```
SET ROLE NONE;
```

To set active roles to all granted role, we use:

```
SET ROLE ALL;
```

To set active roles to default roles that set by the **SET DEFAULT ROLE**statement, we use:

```
SET ROLE DEFAULT;
```

To set active named roles, we use:

```
SET ROLE granted_role_1, granted_role_2,
```

Revoking Privileges from Roles

To revoke privileges from a specific role, we use the **REVOKE**statement. The **REVOKE** statement takes effect not only on the role but also any account granted the role.

For example, to temporarily make all read/write users read only, we change the **imax_write**role as follows:

```
REVOKE INSERT, UPDATE, DELETE ON imax.* FROM imax_write;
```

To restore the privileges, we need to re-grant them as follows:

```
GRANT INSERT, UPDATE, DELETE ON imax.* TO imax_write;
```

Removing Roles

To remove one or more roles, we use the **DROP ROLE**statement as follows:

```
DROP ROLE role_name, role_name, ...;
```

Like **REVOKE**statement, the **DROP ROLE**statement revokes roles from every user account to which it was granted.

For example, to remove the **imax_read**,**imax_write**roles, we use the following statement:

```
DROP ROLE imax_read, imax_write;
```

Copying Privileges from a User Account to another

Oracle treats user account like a role, therefore, we can grant a user account to another user account like granting a role to that user account. This allows us to copy privileges from a user to another user.

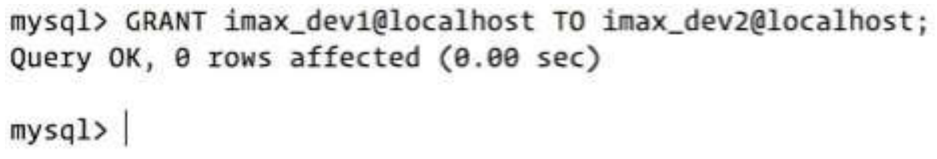
Suppose we need another developer account for the **imax**database:

First, we create the new user account:

```
CREATE USER imax_dev2@localhost IDENTIFIED BY 'Secure$dev2';
```

Second, copy privileges from the **imax_dev1**user account to **imax_dev2**user account as follows:

```
GRANT imax_dev1@localhost TO imax_dev2@localhost;
```



```
mysql> GRANT imax_dev1@localhost TO imax_dev2@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql> |
```

Changing Oracle User Password

Oracle provides various statements that we can use to change the password of a user including the **`SET PASSWORD`**and **`ALTER USER`**statements.

Changing oracle User Password using `SET PASSWORD` Statement

The first way to change the password is by using the **`SET PASSWORD`**statement.

We use the user account in **`user@host`**format to update the password. If we need to change the password for other accounts, our account needs to have at least **`UPDATE`**privilege.

The following statement changes the password of **`dbadmin`**user account using the **`SET PASSWORD`**statement.

```
SET PASSWORD FOR 'dbadmin'@'localhost' = 'dolphin';
```

With no **`FOR`** ***`user`***clause, the statement sets the password for the current user:

```
SET PASSWORD = 'auth_string';
```

Changing Oracle User Password using `ALTER USER` Statement

The second way to change the password for a user account is to use the **`ALTER USER`**statement with the **`IDENTIFIED BY`**clause.

The following **`ALTER USER`**statement changes the password of the **`dbadmin`**user to **`bigshark`**:

```
ALTER USER dbadmin@localhost IDENTIFIED BY 'bigshark';
```

SAMPLE ORACLE ADMINISTRATION SCRIPTS
(Run the scripts below as system user)

Connect system/password;
 Or
Connect /as sysdba;

(Oracle Dba scripts)

SHOW SGA;

SELECT *
FROM   v$database;

SELECT *
FROM   v$instance;

```
SELECT *
FROM   v$version;
--------------------------------------------

SELECT *
FROM   v$license;


SELECT a.role,
       a.password_required,
       a.authentication_type
FROM   dba_roles a
ORDER BY a.role;
-------------------------------------------

SELECT statistic#,
       name
FROM   v$segstat_name
ORDER BY statistic#;

-------------------------------------------


SELECT sp.name,
       sp.type,
       sp.value,
       sp.isses_modifiable,
       sp.issys_modifiable,
       sp.isinstance_modifiable
FROM   v$system_parameter sp
ORDER BY sp.name;

SELECT grantee,
       privilege,
       admin_option
FROM   dba_sys_privs
ORDER BY grantee, privilege;

-------------------------------------------

SELECT username,
       account_status,
       TO_CHAR(lock_date, 'DD-MON-YYYY') AS lock_date,
       TO_CHAR(expiry_date, 'DD-MON-YYYY') AS expiry_date,
       default_tablespace,
       temporary_tablespace,
       TO_CHAR(created, 'DD-MON-YYYY') AS created,
       profile,
       initial_rsrc_consumer_group,
       editions_enabled,
       authentication_type
FROM   dba_users
ORDER BY username;

-------------------------------------------
CREATE DATABASE mynewdb
USER SYS IDENTIFIED BY sys_password
USER SYSTEM IDENTIFIED BY system_password
```

```
EXTENT MANAGEMENT LOCAL
UNDO TABLESPACE undotbs
DEFAULT TEMPORARY TABLESPACE tempts1
DEFAULT TABLESPACE users;
```

```
CREATE DATABASE mynewdb
USER SYS IDENTIFIED BY sys_password
USER SYSTEM IDENTIFIED BY system_password
SET DEFAULT BIGFILE TABLESPACE
UNDO TABLESPACE undotbs
DEFAULT TEMPORARY TABLESPACE tempts1;
--------------------------------------------
```

```
SELECT * FROM DBA_SERVICES;
SHOW ALL_SERVICES; or  SELECT * FROM V$SERVICES;
SELECT * FROM V$ACTIVE_SERVICES;
SELECT * FROM V$SERVICE_STATS;
SELECT * FROM V$SERVICE_EVENT;
SELECT * FROM V$SERVICE_WAIT_CLASSES;
SELECT * FROM V$SERV_MOD_ACT_STATS;
SELECT * FROM V$SERVICE_METRICS;
SELECT * FROM V$SERVICE_METRICS_HISTORY;

SELECT * FROM V$SESSION;
SELECT * FROM V$ACTIVE_SESSION_HISTORY;
SELECT * FROM DBA_RSRC_GROUP_MAPPINGS;
SELECT * FROM DBA_SCHEDULER_JOB_CLASSES;
SELECT * FROM DBA_THRESHOLDS;
SELECT * FROM V$DATABASE
```

```
ALTER SYSTEM SUSPEND;
SELECT DATABASE_STATUS FROM V$INSTANCE;
ALTER SYSTEM RESUME;
SELECT DATABASE_STATUS FROM V$INSTANCE;
```

```
SELECT * FROM V$DISPATCHER_RATE;
SELECT NAME, NETWORK FROM V$DISPATCHER;
```

```
SELECT * FROM V$SHARED_SERVER_MONITOR;
SELECT * FROM V$SGA;
SELECT * FROM V$SGASTAT;
SELECT * FROM V$SHARED_POOL_RESERVED;
SELECT * FROM V$SHARED_SERVER;
SELECT * FROM V$CIRCUIT
```

```
SELECT * FROM DBA_CPOOL_INFO;
SELECT * FROM V$CPOOL_CONN_INFO;
SELECT * FROM V$CPOOL_STATS;
SELECT * FROM V$CPOOL_CC_STATS;
```

```
SELECT * FROM V$PROCESS;
SELECT * FROM V$SESSION;
SELECT * FROM V$SESS_IO;
SELECT * FROM V$SESSION_LONGOPS;
SELECT * FROM V$SESSION_WAIT;
SELECT * FROM V$SESSION_WAIT_HISTORY
```

```
SELECT * FROM V$RESOURCE_LIMIT;
SELECT * FROM V$SESSTAT;
SELECT * FROM V$WAIT_CHAINS;
```

```
SHOW PARAMETER TARGET
```

```
SELECT * FROM V$SYSSTAT;
SELECT * FROM V$SESSTAT;
SELECT * FROM V$PGASTAT;
SELECT * FROM V$SQL_WORKAREA;
SELECT * FROM V$SQL_WORKAREA_ACTIVE;
```

SQL*Plus Command Summary

Command	Description
@ ("at" sign)	Runs the SQL*PLus statements in the specified script. The script can be called from the local file system or from a web server.
@@ (double "at"	Runs a script. This command is similar to the @ ("at" sign) command. It is useful for

Command	Description
sign)	running nested scripts because it looks for the specified script in the same path as the script from which it was called.
/ (slash)	Executes the SQL command or PL/SQL block.
ACCEPT	Reads a line of input and stores it in a given user variable.
APPEND	Adds specified text to the end of the current line in the buffer.
ARCHIVE LOG	Starts or stops the automatic archiving of online redo log files, manually (explicitly) archives specified redo log files, or displays information about redo log files.
ATTRIBUTE	Specifies display characteristics for a given attribute of an Object Type column, and lists the current display characteristics for a single attribute or all attributes.
BREAK	Specifies where and how formatting will change in a report, or lists the current break definition.
BTITLE	Places and formats a specified title at the bottom of each report page, or lists the current BTITLE definition.
CHANGE	Changes text on the current line in the buffer.
CLEAR	Resets or erases the current clause or setting for the specified option, such as BREAKS or COLUMNS.
COLUMN	Specifies display characteristics for a given column, or lists the current display characteristics for a single column or for all columns.
COMPUTE	Calculates and prints summary lines, using various standard computations, on subsets of selected rows, or lists all COMPUTE definitions.
CONNECT	Connects a given user to Oracle.
COPY	Copies results from a query to a table in a local or remote database.
DEFINE	Specifies a user variable and assigns it a CHAR value, or lists the value and variable type of a single variable or all variables.
DEL	Deletes one or more lines of the buffer.
DESCRIBE	Lists the column definitions for the specified table, view, or synonym or the specifications for the specified function or procedure.
DISCONNECT	Commits pending changes to the database and logs the current user off Oracle, but does not exit SQL*Plus.
EDIT	Invokes a host operating system text editor on the contents of the specified file or on the contents of the buffer.
EXECUTE	Executes a single PL/SQL statement.
EXIT	Terminates SQL*Plus and returns control to the operating system.
GET	Loads a host operating system file into the SQL buffer.
HELP	Accesses the SQL*Plus help system.
HOST	Executes a host operating system command without leaving SQL*Plus.
INPUT	Adds one or more new lines after the current line in the buffer.
LIST	Lists one or more lines of the SQL buffer.
PASSWORD	Allows a password to be changed without echoing the password on an input device.
PAUSE	Displays the specified text, then waits for the user to press [Return].
PRINT	Displays the current value of a bind variable.
PROMPT	Sends the specified message to the user's screen.
QUIT	Terminates SQL*Plus and returns control to the operating system. QUIT is identical to EXIT.
RECOVER	Performs media recovery on one or more tablespaces, one or more datafiles, or the entire database.

Command	Description
REMARK	Begins a comment in a script.
REPFOOTER	Places and formats a specified report footer at the bottom of each report, or lists the current REPFOOTER definition.
REPHEADER	Places and formats a specified report header at the top of each report, or lists the current REPHEADER definition.
RUN	Lists and executes the SQL command or PL/SQL block currently stored in the SQL buffer.
SAVE	Saves the contents of the SQL buffer in a host operating system file (a script).
SET	Sets a system variable to alter the SQL*Plus environment for your current session.
SHOW	Shows the value of a SQL*Plus system variable or the current SQL*Plus environment.
SHUTDOWN	Shuts down a currently running Oracle instance.
SPOOL	Stores query results in an operating system file and, optionally, sends the file to a printer.
START	Executes the contents of the specified script. The script can only be called from a *url*.
STARTUP	Starts an Oracle instance and optionally mounts and opens a database.
STORE	Saves attributes of the current SQL*Plus environment in a host operating system file (a script).
TIMING	Records timing data for an elapsed period of time, lists the current timer's title and timing data, or lists the number of active timers.
TTITLE	Places and formats a specified title at the top of each report page, or lists the current TTITLE definition.
UNDEFINE	Deletes one or more user variables that you defined either explicitly (with the DEFINE command) or implicitly (with an argument to the START command).
VARIABLE	Declares a bind variable that can be referenced in PL/SQL.
WHENEVER OSERROR	Performs the specified action Exits SQL*Plus if an operating system command generates an error.
WHENEVER SQLERROR	Performs the specified action Exits SQL*Plus if a SQL command or PL/SQL block generates an error.