Database Security and Privacy

Unit-2

Administration of Users-Introduction

Database administration is a vast domain that governs and maintains how data works and flows through an organization. Specialist professionals called database administrators are responsible for maintaining and managing data across the enterprise and ensuring maximum efficiency of the database management system. With a growing number of organizations migrating to the cloud, there is an increasing demand for a cloud DBA along with tier DBAs to support the system.

What is Database Administration?

Database administration refers to the set of activities that are performed by a database administrator ensuring a 24*7 availability of databases to be used as and when required. The primary database administration is to maintain and manage the database management system software. Mainstream databases like Oracle, IBM DB2, and Microsoft SQL Server require consistent and ongoing management. This brings up a lot of career opportunities for IT professionals called DBAs or database administrators.

Administrators have the following responsibilities:

- Add new users.
- Delete users.
- Manage user access.
- Set user connection privilege.
- Edit user permissions.
- View existing user permissions.
- Set Password Policies

Database Administrator Responsibilities

Some of the major responsibilities of a database administrator are:

1. Creating and administering databases by identifying user needs

2. Ensuring efficient, effective, and error-free operation of the database

3. Consistently testing the database for adding any new modifications that may be required as per user needs

4. Regular maintenance of database and permissions related to updating

5. Consolidating multiple databases into a refreshed and larger one

6. Ensure data restoration and 24*7 back up to avoid any data loss

Authentication of users

Authorization and authentication are one of the primary services of OS and Database administrator. Database authentication is the process or act of confirming that a user who is attempting to log in to a database is authorized to do so and is only accorded the rights to perform activities that he or she has been authorized to do.

In the context of databases, however, authentication acquires one more dimension because it may happen at different levels. It may be performed by the database itself, or the setup may be changed to allow either the operating system, or some other external method, to authenticate users.

For example, while creating a database in Microsoft's SQL Server, a user is required to define whether to to use database authentication, operating system authentication, or both (the so-called mixed-mode authentication). Other databases in which security is paramount employ near-foolproof authentication modes like fingerprint recognition and retinal scans. Database authentication and authorization can be managed either locally within the database or centrally in a directory service. In most production use cases, database users should be managed centrally like other IT systems for better security, stronger controls over data access and compliance reporting. Primary types of data authentication include.

1. **Local authentication and local authorization:** This method provides simple password authentication out of the box with every database. This allows you to quickly create a database and provide access to it – useful for some environments like development. Authorization is also managed locally by having grants of privileges and roles directly granted to the user schema in the database. With local password authentication and local authorization, every new user for the database, every person that leaves and changes in privileges and roles has to be managed by the local DBA – including password resets.

2. **Central authentication and local authorization:** it is used via various third-party authentication services such as Kerberos, Remote Authentication Dial-In User Service (RADIUS), and SSL Authorization using Certificates. OS authentication can also be considered a form of centralized authentication if the OS is configured in that manner.

- Kerberos is a trusted third-party authentication system that relies on shared secrets. It presumes that the third party is secure, and provides single sign-on capabilities, centralized password storage, database link authentication, and enhanced PC security. It

does this through a Kerberos authentication server. Oracle supports both the original MIT Kerberos services as well as the Kerberos service provided with Microsoft Active Directory.

- RADIUS is a client/server security protocol that is most widely known for enabling remote authentication and access. Oracle Database uses this standard in a client/server network environment to enable use of any authentication method that supports the RADIUS protocol. RADIUS can be used with a variety of authentication mechanisms, including token cards and smart cards. Users frequently use the RADIUS service if they want to implement multi-factor authentication for database users.
- SSL uses digital certificates that comply with the X.509v3 standard and a public and private key pair.
- In OS authentication, the database relies on the OS to authenticate the user to the user schema.

While authentication is centralized in these cases, authorization remains for the most part locally managed in the Database. OS and RADIUS authentication allows for some authorization via their service, most users manage the roles and privileges locally in the database.

3. **Central authentication and central authorization:** Centralized management of users is a key part of IT security, and the database is no exception. Onboarding new users, assigning them to the correct resources with the correct privileges, changing privileges and removing access when they leave is more securely done centrally without having to manage every change within every production database. DBAs could focus on application development or database maintenance instead of the drudgery of fulfilling password reset requests. It can be done via the

Centrally Managed Users (CMU) –CMU provides a simpler integration with Microsoft Active Directory to allow centralized authentication and authorization of users. This direct integration (without an intermediate directory service) enables organizations to use Active Directory to centrally manage users and roles in multiple Oracle databases with a single directory along with other Information Technology services. Active Directory users can authenticate to the Oracle database by using credentials that are stored in Active Directory.

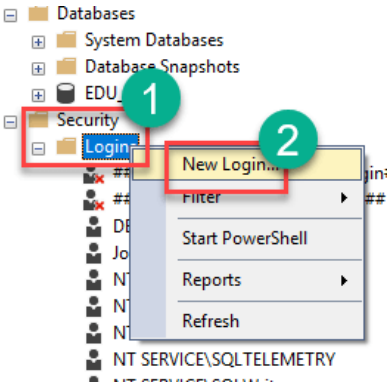
Creating users

Here is how to create login in SQL Server:

Method 1:

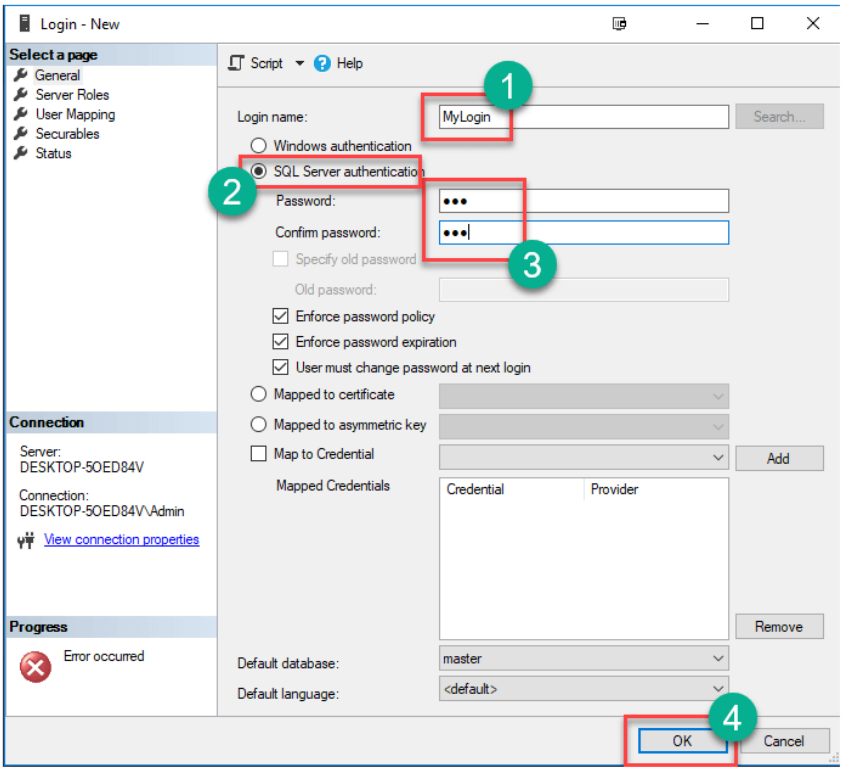
Step 1) To create login SQL server

1. In Object Explorer, expand the **Databases** folder.
2. Expand the database in which to create the new database user.
3. Navigate to Security > Logins
4. Select new login and press Enter

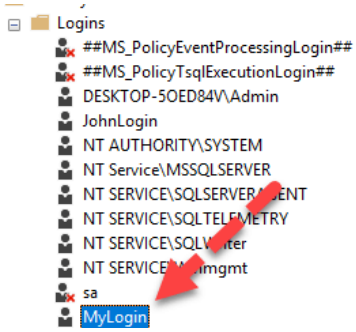


Step 2) In the next screen, Enter

1. Login Name
2. Select SQL Server authentication
3. Enter Password for MySQL create user with password
4. Click Ok



Step 3) Login is created and can be seen by expanding



Method 2:

You can also create a login using the T-SQL command for SQL server create login and user.

Syntax :

CREATE LOGIN MyLogin WITH PASSWORD = '123';

Creating SQL Server user

How to Create a User in SQL Server Database

A user is an account that you can use to access the SQL server. To create user SQL server, you can use any of the following two ways:

- Using SQL Server Management Studio
- Using T-SQL

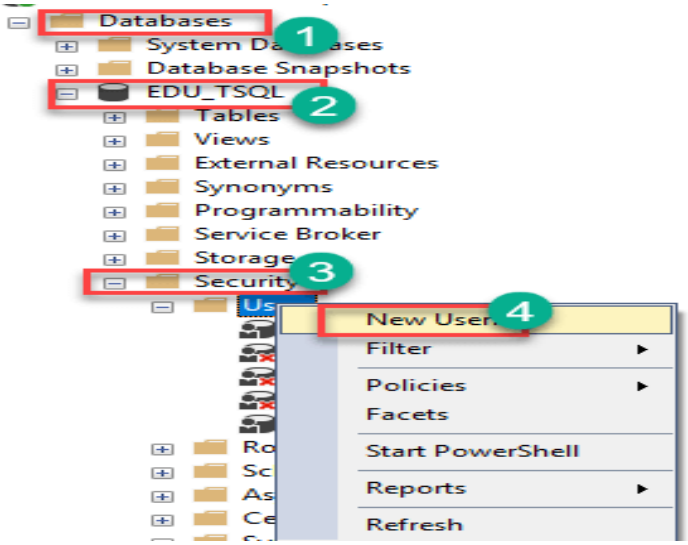
Method 1 :

Here is a step by step process on how to create a user in SQL Server Management Studio:

You will be creating a user for the EDU_TSQL database.

Step 1) Connect to SQL server to create new user.

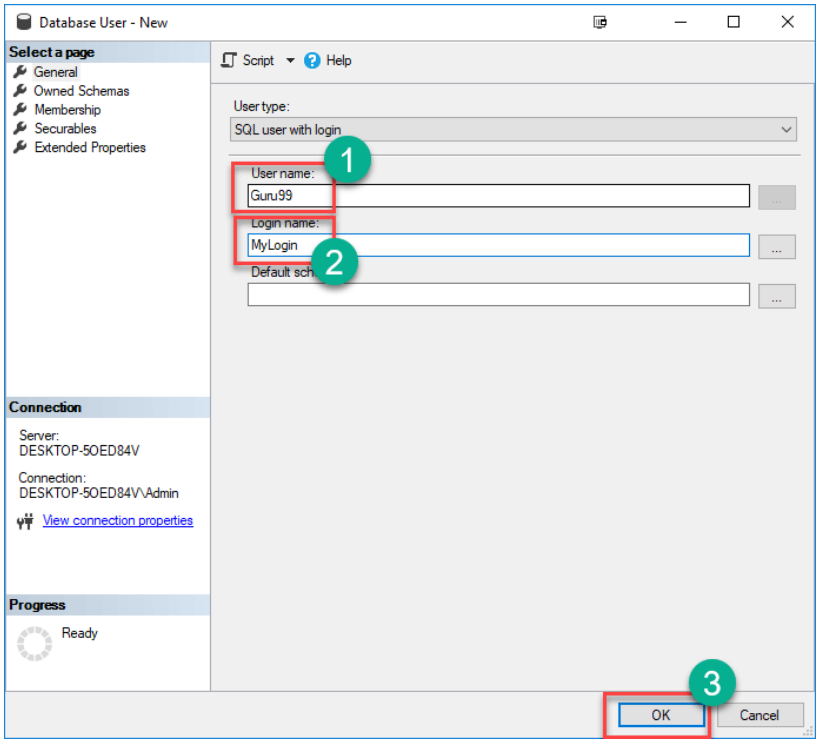
1. Connect to SQL Server then expand the Databases folder from the Object Explorer.
2. Identify the database for which you need to create the user and expand it.
3. Expand its Security folder.
4. Right-click the Users folder then choose “New User…”



Step 2) Enter User details

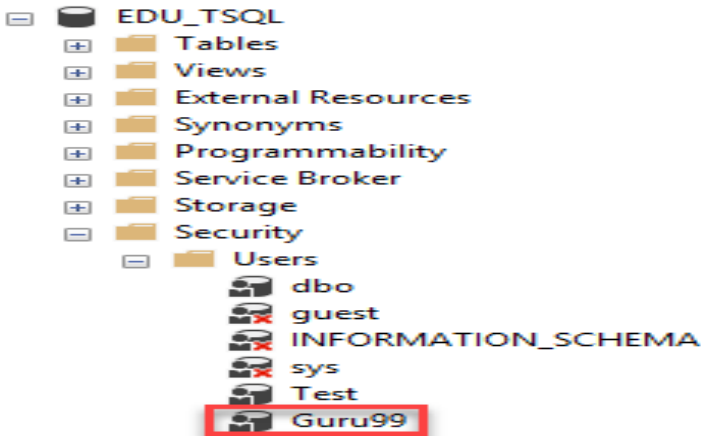
You will get the following screen,

1. Enter desired User name
2. Enter the Login name (created earlier)
3. Click OK



Step 3) User will be created

User is created



Method 2:

Create User using T-SQL

You can use the T-SQL's create user command for SQL server add user to database. The SQL create user command takes the following syntax:

create user <user-name> for login <login-name>
create user Guru99 for login MyLogin

Note: That the query should be executed within the query window. If a user is already created for a Login, SQL Server will throw an error if you create a user for the same login.

Removing, Modifying users

Method 1 :

SQL Server: DROP USER statement

This SQL Server tutorial explains how to use the SQL Server **DROP USER statement** with syntax and examples.

Description

The DROP USER statement is used to remove a user from the SQL Server database.

Syntax

The syntax for the DROP USER statement in SQL Server (Transact-SQL) is:

```
DROP USER user_name;
```

Parameters or Arguments

user_name

The name of the user to remove from the SQL Server database.

Note

- Before you can drop a user, you must delete objects owned by the user or transfer ownership of those objects.
- See also the CREATE USER statement.

Example

Let's look at how to drop a user using the DROP USER statement in SQL Server.

For example:

```
DROP USER SRMNCR;
```

This DROP USER example would drop the user called *techonthenet*. This DROP USER statement will only run if *techonthenet* does not own any objects in the SQL Server database.

Method 2 :

Using SQL Server Management Studio

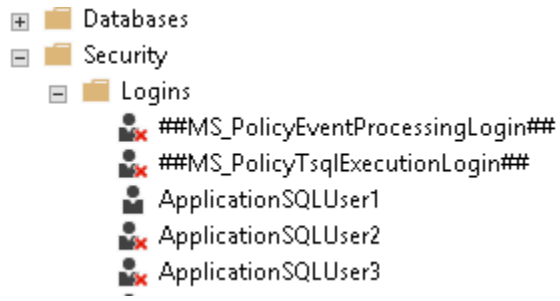
Introduction

Sometimes, we need to drop a SQL Server Login because we gave an access to a person who left, this login was used by an application that has been decommissioned for example.

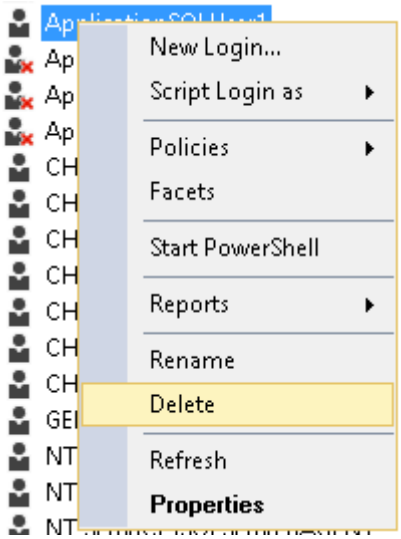
To do so, we can use SQL Server Management Studio (SSMS) as follows:

1. Open SSMS

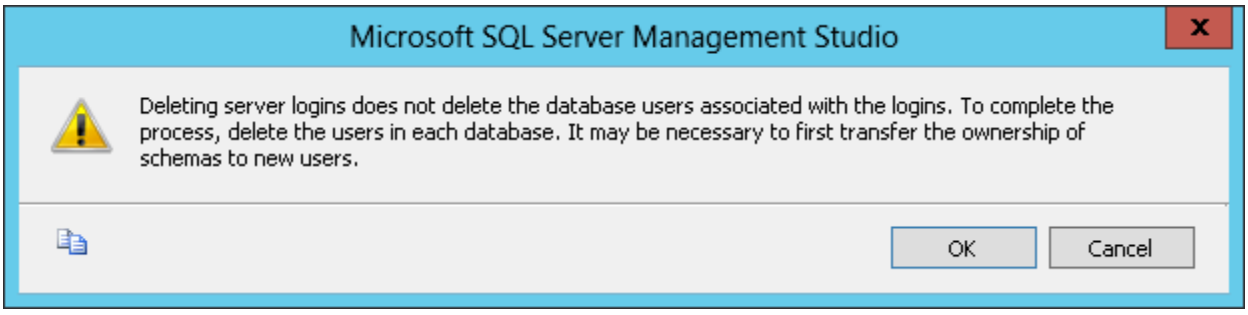
2. Connect to a SQL Server instance
3. In Object Explorer, go to « Security » node then logins



4. Right-click on the SQL Server Login you want to drop then click on “Delete”



5. SSMS will show following warning message



6. Click on “OK”

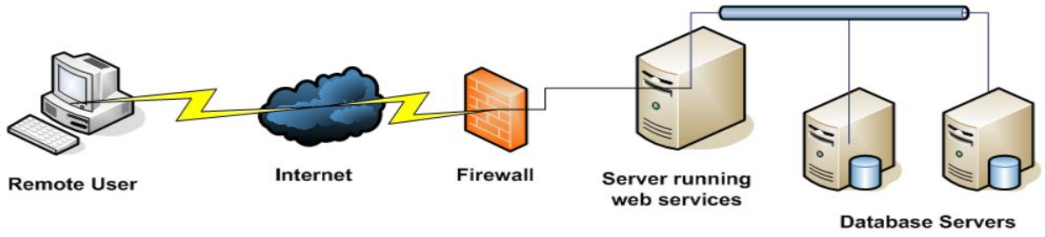
Remote users

In the context of database management systems (DBMS), remote users refer to individuals or applications that access a database from a location different from the physical location where the database is hosted. These users connect to the database over a network, typically using client-server architecture.

Remote users can interact with the database by sending queries, retrieving data, modifying records, or performing other operations, depending on their access privileges and the permissions granted to them by the database administrator. The remote user's requests are transmitted over the network to the DBMS server, which processes the requests and sends back the results or updates.

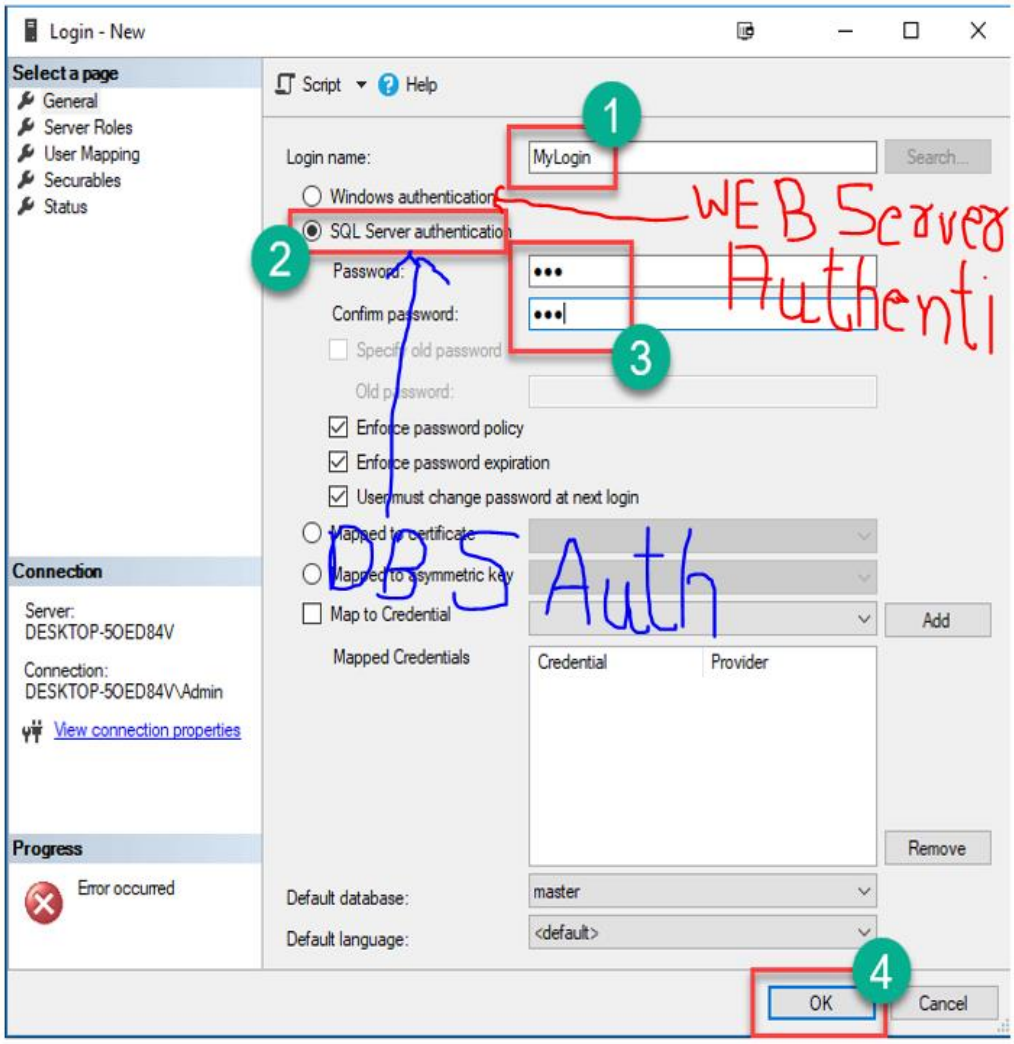
Remote user access is commonly facilitated through database connectivity protocols such as ODBC (Open Database Connectivity), JDBC (Java Database Connectivity), or proprietary protocols specific to a particular DBMS. Security measures, such as authentication and encryption, are usually employed to protect the confidentiality and integrity of the data transmitted between the remote user and the database server.

Remote user access in DBMS allows for distributed and collaborative data management, enabling users in different locations to work with a centralized database simultaneously. It also enables access to databases from applications running on different platforms or devices, promoting flexibility and scalability in database usage.



- When a database administrator or architect works directly on the system running the database servers then its called the local user. It means that when an administrator logs on to the database through the machine where database is stored then it will be called a local user
- When an user or administrator logs onto the database through the Local area network or through the internet or through the virtual private network, then it will be called remote user.

- Remote users can be authenticated in database servers or web servers as shown below



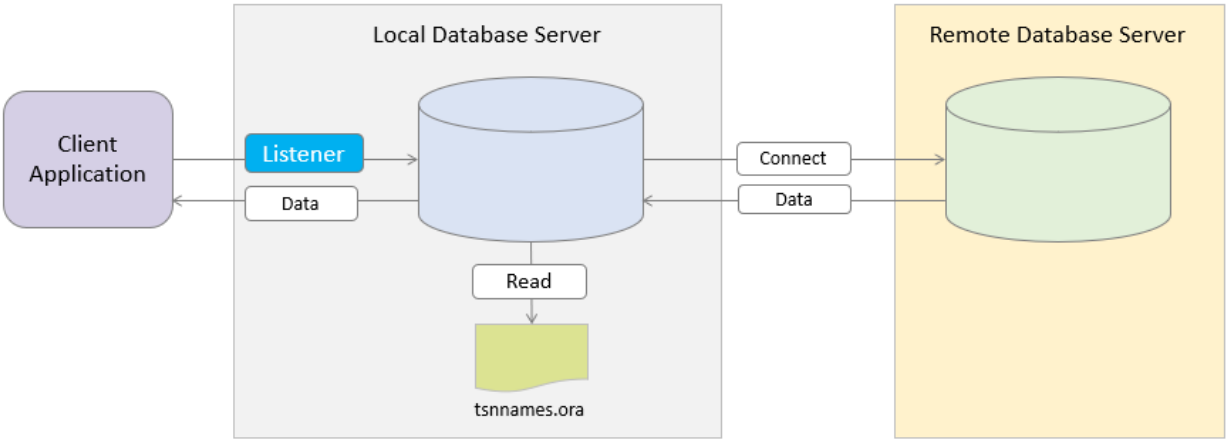
Database links

What is a database link

A database link is a connection from the one database to another remote database. The remote database can be an Oracle Database or any ODBC compliant database such as SQL Server or MySQL.

A database link allows a user or program to access database objects such as tables and views from another database.

A DB link enables a user to perform Data Manipulation Language (DML) or any other valid SQL statements on a DB. ✓ The following figure gives the architecture of DB Link



Generally speaking, you can perform Data Query Language (DQL) and Data Manipulation Language (DML) via a database link, but you cannot perform Data Definition Language (DDL) in this way. Which means, you cannot do `CREATE TABLE`, `ALTER TABLE`, `CREATE INDEX` or `ALTER INDEX` on the remote database through a database link.

There are two types of database links: **public and private.**

Private database links are visible to the owners while public database links are visible to all users in the database. For this reason, public database links may pose some potential security risks.

Managing Database Links

A database link is a schema object in one database that enables you to access objects in another database. You can create, browse, drop a database link and view report.

Once you create a database link, you can access the remote objects by appending `@dblink` to the table or view name, where `dblink` is the name of the database link.

- Creating a Database Link
  Create a database link using Object Browser.

- Browsing a Database Link
  Select a database link from the Object Selection pane and view different reports about the database link.

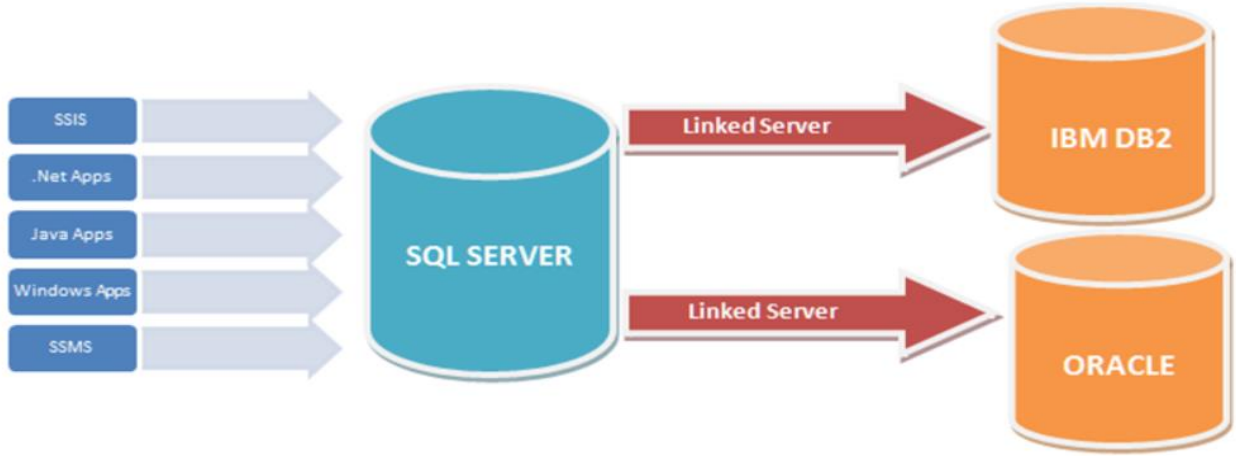
- Reports for Database Links
  Alternative views available when viewing a database links in Object Browser.

- Dropping a Database Link
  Select a database link from the Object Selection pane and click Drop.

Database links are one-way connections, and each link connects a database to one, and only one, other database. For example, database db1 can use a link to connect to database db2. Across that link, database db1 queries or sends updates to objects in database db2.

Linked servers

Linked servers enable the SQL Server Database Engine and Azure SQL Managed Instance to read data from the remote data sources and execute commands against the remote database servers (for example, OLE DB data sources) outside of the instance of SQL Server. Typically linked servers are configured to enable the Database Engine to execute a Transact-SQL statement that includes tables in another instance of SQL Server, or another database product such as Oracle. Many types OLE DB data sources can be configured as linked servers, including third-party database providers and Azure CosmosDB.



When to use linked servers?

Linked servers enable you to implement distributed databases that can fetch and update data in other databases. They are a good solution in the scenarios where you need to implement database

sharding without need to create a custom application code or directly load from remote data sources. Linked servers offer the following advantages:

- The ability to access data from outside of SQL Server.
- The ability to issue distributed queries, updates, commands, and transactions on heterogeneous data sources across the enterprise.
- The ability to address diverse data sources similarly.

You can configure a linked server by using SQL Server Management Studio or by using the sp_addlinkedserver (Transact-SQL) statement. OLE DB providers vary greatly in the type and number of parameters required. For example, some providers require you to provide a security context for the connection using sp_addlinkedsrvlogin (Transact-SQL). Some OLE DB providers allow SQL Server to update data on the OLE DB source. Others provide only read-only data access. For information about each OLE DB provider, consult documentation for that OLE DB provider.

Linked server components

A linked server definition specifies the following objects:

- An OLE DB provider
- An OLE DB data source

An *OLE DB provider* is a DLL that manages and interacts with a specific data source. An *OLE DB data source* identifies the specific database that can be accessed through OLE DB. Although data sources queried through linked server definitions are ordinarily databases, OLE DB providers exist for a variety of files and file formats. These include text files, spreadsheet data, and the results of full-text content searches.

Remote Servers

Remote servers are supported in SQL Server for backward compatibility only. New applications should use linked servers instead.

A remote server configuration allows for a client connected to one instance of SQL Server to execute a stored procedure on another instance of SQL Server without establishing a separate connection. Instead, the server to which the client is connected accepts the client request and sends the request to the remote server on behalf of the client. The remote server processes the request and returns any results to the original server. This server in turn passes those results to the client. When you set up a remote server configuration, you should also consider how to establish security.

If you want to set up a server configuration to execute stored procedures on another server and do not have existing remote server configurations, use linked servers instead of remote servers. Both stored procedures and distributed queries are allowed against linked servers; however, only stored procedures are allowed against remote servers.

Remote Server Details

Remote servers are set up in pairs. To set up a pair of remote servers, configure both servers to recognize each other as remote servers.

Most of the time, you should not have to set configuration options for remote servers. SQL Server Set sets the defaults on both the local and remote computers to allow for remote server connections.

For remote server access to work, the **remote access** configuration option must be set to 1 on both the local and remote computers. (This is the default setting.) **remote access** controls logins from remote servers. You can reset this configuration option by using either the Transact-SQL **sp_configure** stored procedure or SQL Server Management Studio. To set the option in SQL Server Management Studio, on the **Server Properties Connections** page, use **Allow remote connections to this server**. To reach the **Server Properties Connections** page, in Object Explorer, right-click the server name, and then click **Properties**. On the **Server Properties** page, click the **Connections** page.

From the local server, you can disable a remote server configuration to prevent access to that local server by users on the remote server with which it is paired.

A local server gives you exclusive access to data and objects in a set of Windows folders called data directories. During the TM1 client session, only you can create, browse, and modify data or objects that a local server stores. You can also control where the data directories are located.

Remote servers provide access to shared data and objects in your organization. A user's level of access depends on the security group that the administrator assigns to the user name (client ID) that the user employs to access the remote server. For example, a user might be able to update March sales data that is stored on a department's remote server, but that user can browse only the campaign data that is stored on the Marketing department's remote server.

Practices for Administrators and Managers

1. Set business goals

An actionable, targeted database management strategy should reflect your business needs and outline the metrics you'll use to track your success. If you don't spend enough time deciding what data to collect and how you can use this data effectively, you run the risk of wasting internal resources gathering the wrong data, piling up too much data to efficiently use, or missing important data opportunities.

Setting relevant business goals gives you a lodestar to follow so you don't lose your way. These uses for business data are worthwhile to consider:

- Creating profiles and targeting: Customer profiles are a common way to use data collection and analysis, but the user, partner, and audience data you are collecting could also be

valuable. Forming profiles from accumulated data is a smart way to start making sense of it.

- Identifying trends and patterns: Customer trends, sales trends, and other patterns provide you with strategic insight into your industry and the purchasing behaviors of those who patronize your products and services. Usage patterns, consumption trends, and other conditions can be tracked, and this information is particularly valuable for SaaS companies and other organizations that could strategically benefit from understanding trends.
- Automating and improving processes: You can also use data to help you revamp your processes, implement automation, and make adjustments. Looking closely at your data may reveal opportunity areas.
- Informing business decisions: The next best thing to a fortune-telling crystal ball is the ability to dial in on your past experiences and collected data.

With a close eye on your data, database administrators are in a good position to help with sorting through these data uses and determining which other opportunities exist for your organization.

2. Establish policies and procedures, including backup and recovery procedures

Crafting specific backup and recovery procedures and policies prepares your team to act more effectively if the worst happens. Determine smart actions you can plan in advance—this exercise will keep the team focused and give you a chance to work through your worst-case scenarios.

As you plan your disaster response, you can use flowcharts and process mapping to visualize everything and provide your team with a helpful overview.

- Collecting and organizing data: Your team should be trained on your procedures and policies for collecting data and adding it to the database. With the role automation now plays in this, make sure everyone is on the same page with how software contributes and what role AI and automation plays.
- Guard data integrity: Database errors can be devastating, so make a plan for keeping them minimal and allowing your team to find and correct them when they occur.
- Monitor your data: On a regular basis, check your databases for accuracy and possible data corruption.
- Set benchmarks: Your DBA should help with establishing alerts to protect the database by bringing up problems when they occur. Your team should know what your organization’s goals are for the database and be prepared to act accordingly if changes happen.
- Map your processes: Being able to visualize how your database works and see the entire process, from data collection to processing, assists your organization with troubleshooting and planning.

3. Make security your priority

Although not every disaster is entirely predictable or preventable, you can improve your data security and manage the risks associated with worst-case scenarios for your database. Maintenance, backup, and recovery planning are your best bets for protecting what's important.

DBAs who know industry best practices for database security and are prepared to manage your database security effectively are valuable allies in the fight against data loss, security breaches, and database compromise.

- Create a comprehensive maintenance plan: Maintain your database regularly. Data security should stay at the forefront and not become an afterthought. You don't want to be in the position of trying to "catch up" on security after a breach, for instance. Make a plan your team can use as a preventative treatment—it's easier than a cure.
- Develop your backup and recovery procedures: Have your backup and recovery plan together and review it to make sure it still fits your security strategy, your team, and your database.
- Build your team's security skills: Security issues change along with technology changes, business growth, and database characteristics. Your team should stay up to date with the industry and strive to stay ahead of your database's needs.
- Leverage automation to help with security: Automation can support your DBAs, too—for example, you can schedule frequent automated backups.

4. Focus on the quality of the data

Your DBA should work to promote a high standard of data quality, removing data that doesn't meet the standards and adapting quality standards to fit your changing strategy.

- Have SMART data quality metrics: Ideally, you would create metrics that meet SMART (specific, measurable, achievable, relevant, and time-bound) standards. SMART helps you make data quality metrics that are usable and truly useful for your organization. At best, metrics that don't fit SMART criteria represent "nice to have" goals that are subjective. At worst, these metrics leave your team aiming for moving targets.
- Empower your data steward: As your DBA goes about their job protecting your data quality, make sure they have everything necessary to do their job successfully. Loop them into communications with the rest of your team, allow them to enforce your data quality standards, and make sure organizational resources are available to help them protect your data. The last thing you want is a situation where the data czar doesn't have management or team support.

5. Reduce duplicate data

Duplicate data reduces your database performance and can hinder your efforts. Often, duplicates also lead to wasted internal resources and doubled effort by your team. If a customer record is

duplicated in a CRM, for instance, the service team might literally spend twice as much time fixing the same problem all over again.

- Share data quality basics throughout the organization: Your entire company should know a few basics about protecting data quality, even if they don't work directly with the DBA or with the database. Someone who doesn't know the harm of creating duplicate records can create more work for your team. Teach everyone what good data looks like and how to contribute high-quality data.
- Eliminate siloed data access and management: When individual departments manage separate areas of a database or manage their own without any outside direction or input within your organization, you risk duplication and errors. In some cases, these silo databases exist without following data quality and duplication standards.
- Have a plan for duplicate data and test your database: If duplication seems to happen more often to your organization than you want it to, then develop a plan to address the sources of duplication. Test your database regularly to make sure you're managing the problem effectively with these changes in place.

6. Make the data easily accessible

You need to make sure that your users can benefit from the data. Internal users, end-users, and other stakeholders that access your database should know how to use it and be comfortably able to benefit from it.

- Design and manage for the user: Think of how your database is used and design it accordingly. Keep in mind that some shortcuts or development and management strategies might work well for your team but be poor choices from a UX/usability or performance perspective.
- Gather feedback on your database: Your users should be able to provide feedback on how the database is working for them. How you gather this feedback is up to you and depends on the stakeholders you have—choosing a survey, forming a panel or committee meeting, or designating a DBA as a point of contact are potential approaches.

1. Build strong file naming and cataloging conventions

If you are going to utilize data, you have to be able to find it. You can't measure it if you can't manage it. Create a reporting or file system that is user- and future-friendly—descriptive, standardized file names that will be easy to find and file formats that allow users to search and discover data sets with long-term access in mind.

- To list dates, a standard format is YYYY-MM-DD or YYYYMMDD.
- To list times, it is best to use either a Unix timestamp or a standardized 24-hour notation, such as HH:MM:SS. If your company is national or even global, users can take note of where the information they are looking for is from and find it by time zone.

2. Carefully consider metadata for data sets

Essentially, metadata is descriptive information about the data you are using. It should contain information about the data's content, structure, and permissions so it is discoverable for future use. If you don't have this specific information that is searchable and allows for discoverability, you cannot depend on being able to use your data years down the line.

Catalog items such as:

- Data author
- What data this set contains
- Descriptions of fields
- When/Where the data was created
- Why this data was created and how

This information will then help you create and understand a data lineage as the data flows to tracking it from its origin to its destination. This is also helpful when mapping relevant data and documenting data relationships. Metadata that informs a secure data lineage is the first step to building a robust data governance process.

3. Data Storage

If you ever intend to be able to access the data you are creating, storage plans are an essential piece of your process. Find a plan that works for your business for all data backups and preservation methods. A solution that works for a huge enterprise might not be appropriate for a small project's needs, so think critically about your requirements.

A variety of storage locations to consider:

- Desktops/laptops
- Networked drives
- External hard drives
- Optical storage
- Cloud storage
- Flash drives (while a simple method, remember that they do degrade over time and are easily lost or broken)

The 3-2-1 methodology

A simple, commonly used storage system is the 3-2-1 methodology. This methodology suggests the following strategic recommendations: 3: Store three copies of your data, 2: using two types of storage methods, 1: with one of them stored offsite. This method allows smart access and makes sure there is always a copy available in case one type or location is lost or destroyed, without being overly redundant or overly complicated.

4. Documentation

Within data management best practices, we can't overlook documentation. It's often smart to produce multiple levels of documentation that will provide full context to why the data exists and how it can be utilized.

Documentation levels:

- Project-level
- File-level
- Software used (include the version of the software so if future users are using a different version, they can work through the differences and software issues that might occur)

- Context (it is essential to give any context to the project, why it was created, if hypotheses were trying to be proved or disproved, etc.)

5. Commitment to data culture

A commitment to data culture includes making sure that your department or company's leadership prioritizes data experimentation and analytics. This matters when leadership and strategy are needed and if budget or time is required to make sure that the proper training is conducted and received. Additionally, having executive sponsorship as well as lateral buy-in will enable stronger data collaboration across teams in your organization.

6. Data quality trust in security and privacy

Building a culture committed to data quality means a commitment to making a secure environment with strong privacy standards. Security matters when you are working to provide secure data for internal communications and strategy or working to build a relationship of trust with a client that you are protecting the privacy of their data and information. Your management processes must be in place to prove that your networks are secure and that your employees understand the critical nature of data privacy. In today's digital market, data security has been identified as one of the most significant decision-making factors when companies and consumers are making their buying decisions. One data privacy breach is one too many. Plan accordingly.

7. Invest in quality data-management software

When considering these best practices together, it is recommended, if not required, that you invest in quality data-management software. Putting all the data you are creating into a manageable working business tool will help you find the information you need. Then you can create the right data sets and data-extract scheduling that works for your business needs. Data management software will work with both internal and external data assets and help configure your best governance plan. Tableau offers a Data Management Add-On that can help you create a robust analytics environment leveraging these best practices. Using a reliable software that helps you build, catalog, and govern your data will build trust in the quality of your data and can lead to the adoption of self-service analytics. Use these tools and best practices to bring your data management to the next level and build your analytics culture on managed, trusted, and secure data.

Profiles, Password Policies, Privileges and roles: Introduction

Users, Roles & Permissions

Users, roles, and permissions work together to determine *who can access what* inside your database. Users are the individual accounts for authenticating into the project. Each user is assigned a role which defines its access permissions.

In order to understand how users, roles, and permissions work in Directus, a conceptual understanding of *how they work in general* will be helpful. The following few paragraphs will introduce you to how users, roles, and permissions work within a relational database. If you're already familiar with these concepts, feel free to skip to How it Works in Directus.

A user profile **is a collection of settings and information associated with a user. It contains critical information that is used to identify an individual, such as their name, age, portrait photograph and individual characteristics such as knowledge or expertise**

Types of User Profiles

- Local User Profiles. A local user profile is created the first time that a user logs on to a computer. ...
- Roaming User Profiles. A roaming user profile is a copy of the local profile that is copied to, and stored on, a server share. ...
- Mandatory User Profiles. ...
- Temporary User Profiles.

Roles

In many cases, your project will have multiple users doing the same thing *(managers, writers, subscribers, etc)*. If we assigned permissions directly to the user, we would have to configure the same permissions over and over, which makes it tedious to change configurations for all users doing the same job and also leads to a higher chance of misconfiguration. This problem is an example of data duplication. To avoid this, we create roles, configure the role's permissions once, then assign the role to users as desired.

Regardless of your project, your SQL database will *always* need an administrator role and a public role. In addition, you may need any number of custom roles.

Administrators

An administrator role provides complete, unrestricted control over the database, including the data model and all it data. This cannot be limited, as by definition it would no longer be an administrator role. You need at least one user in an administrator role. Otherwise, it would be impossible to fully manage the database.

Public

A public role defines access permissions for unauthenticated requests to the database. That means that if you enable an access permission for this role, *everybody has that permission enabled*. Remember, the database has no idea which data you'd want the public to see. So to be safe, all permissions begin turned off by default. It is up to the administrators to re-configure these and define exactly what the public role has access to.

Custom Roles

In addition to these two extreme types of roles, you may need to create more roles each with their own unique set of permissions. The roles you create and the permissions you configure for them are completely open-ended and dependent on your project's needs.

Permissions

Remember, for the majority of projects, it wouldn't be safe or ethical to give every user full access to the data. Users could accidentally damage data or even take malicious actions against the project and its users. For example, a student may need to be able to *see their grade, but not be able to change it*.

Thus, there are four types of permissions for each data table, based on the four CRUD actions you can do to data in a database: *create, read, update, and delete*... Hence you often hear the term CRUD permissions. You can configure CRUD permissions on each data table as desired. For example, you can grant:

- read-only permission
- read and write but not update or delete permissions
- *any other combination of the four*

Business Rules

In many cases, you will need to grant permissions to data based on its value, or by some other conditional logic. This type of conditional permission is often called a business rule.

To give an example, students should be able to read to their own grades, but not the grades of other students. So you could create a business rule for the student role, so that a user can only see his or her own grade.

Taking this example one step further, we'd also want to allow students to read and create answers to an online test, but not update or delete their test answers once submitted. Then you may need a business rule to crate a submission deadline. Finally, you likely want to restrict each student's CRUD access to all other student tests.

Defining and Using Profiles

Introduction to Oracle CREATE PROFILE statement

A user profile is a set of limits on the database resources and the user password. Once you assign a profile to a user, then that user cannot exceed the database resource and password limits.

The CREATE PROFILE statement allows you to create a new user profile. The following illustrates the basic syntax of the CREATE PROFILE statement:

```
CREATE PROFILE profile_name
LIMIT { resource_parameters | password_parameters};
```
Code language: SQL (Structured Query Language) (sql)

In this syntax:

- First, specify the name of the profile that you want to create.
- Second, specify the LIMIT on either database resources or password.

resource_parameters

You use the following clauses to set the limit for resource parameters:

- SESSIONS_PER_USER – specify the number of concurrent sessions that a user can have when connecting to the Oracle database.
- CPU_PER_SESSION – specify the CPU time limit for a user session, represented in hundredth of seconds.
- CPU_PER_CALL – specify the CPU time limit for a call such as a parse, execute, or fetch, expressed in hundredths of seconds.
- CONNECT_TIME – specify the total elapsed time limit for a user session, expressed in minutes.

- IDLE_TIME – specify the number of minutes allowed periods of continuous inactive time during a user session. Note that the long-running queries and other operations will not subject to this limit.
- LOGICAL_READS_PER_SESSION – specify the allowed number of data blocks read in a user session, including blocks read from both memory and disk.
- LOGICAL_READS_PER_CALL – specify the allowed number of data blocks read for a call to process a SQL statement.
- PRIVATE_SGA – specify the amount of private memory space that a session can allocate in the shared pool of the system global area (SGA).
- COMPOSITE_LIMIT – specify the total resource cost for a session, expressed in service units. The total service units are calculated as a weighted sum of CPU_PER_SESSION, CONNECT_TIME, LOGICAL_READS_PER_SESSION, and PRIVATE_SGA.

password_parameters

You use the following clauses to set the limits for password parameters:

- FAILED_LOGIN_ATTEMPTS – Specify the number of consecutive failed login attempts before the user is locked. The default is 10 times.
- PASSWORD_LIFE_TIME – specify the number of days that a user can use the same password for authentication. The default value is 180 days.
- PASSWORD_REUSE_TIME – specify the number of days before a user can reuse a password.
- PASSWORD_REUSE_MAX – specify the number of password changes required before the current password can be reused. Note that you must set values for both PASSWORD_REUSE_TIME and PASSWORD_REUSE_MAX parameters make these parameters take effect.
- PASSWORD_LOCK_TIME – specify the number of days that Oracle will lock an account after a specified number of a consecutive failed login. The default is 1 day if you omit this clause.
- PASSWORD_GRACE_TIME – specify the number of days after the grace period starts during which a warning is issued and login is allowed. The default is 7 days when you omit this clause.

Note that to create a new profile, your user needs to have the CREATE PROFILE system privilege.

Oracle CREATE PROFILE examples

To find the current profile of a user, you query it from the dba_users view as shown in the following statement:

```
SELECT
   username,
   profile
FROM
```

```
    dba_users
WHERE
    username = 'OT';
```
Code language: SQL (Structured Query Language) (sql)

Here is the output:

USERNAME	PROFILE
OT	DEFAULT

So the user OT has the DEFAULT profile.

When you create a user without explicitly specifying a profile, Oracle will assign the DEFAULT profile to the user.

To find the parameters of DEFAULT profile, you query the dba_profiles as shown in the following query:

```
SELECT
  *
FROM
    dba_profiles
WHERE
    PROFILE = 'DEFAULT'
ORDER BY
    resource_type,
    resource_name;
```
Code language: SQL (Structured Query Language) (sql)

PROFILE	RESOURCE_NAME	RESOURCE_TYPE	LIMIT	COMMON	INHERITED	IMPLICIT
DEFAULT	COMPOSITE_LIMIT	KERNEL	UNLIMITED	NO	NO	NO
DEFAULT	CONNECT_TIME	KERNEL	UNLIMITED	NO	NO	NO
DEFAULT	CPU_PER_CALL	KERNEL	UNLIMITED	NO	NO	NO
DEFAULT	CPU_PER_SESSION	KERNEL	UNLIMITED	NO	NO	NO
DEFAULT	IDLE_TIME	KERNEL	UNLIMITED	NO	NO	NO
DEFAULT	LOGICAL_READS_PER_CALL	KERNEL	UNLIMITED	NO	NO	NO
DEFAULT	LOGICAL_READS_PER_SESSION	KERNEL	UNLIMITED	NO	NO	NO
DEFAULT	PRIVATE_SGA	KERNEL	UNLIMITED	NO	NO	NO
DEFAULT	SESSIONS_PER_USER	KERNEL	UNLIMITED	NO	NO	NO
DEFAULT	FAILED_LOGIN_ATTEMPTS	PASSWORD	10	NO	NO	NO
DEFAULT	INACTIVE_ACCOUNT_TIME	PASSWORD	UNLIMITED	NO	NO	NO
DEFAULT	PASSWORD_GRACE_TIME	PASSWORD	7	NO	NO	NO
DEFAULT	PASSWORD_LIFE_TIME	PASSWORD	180	NO	NO	NO
DEFAULT	PASSWORD_LOCK_TIME	PASSWORD	1	NO	NO	NO
DEFAULT	PASSWORD_REUSE_MAX	PASSWORD	UNLIMITED	NO	NO	NO
DEFAULT	PASSWORD_REUSE_TIME	PASSWORD	UNLIMITED	NO	NO	NO
DEFAULT	PASSWORD_VERIFY_FUNCTION	PASSWORD	NULL	NO	NO	NO

1) Using Oracle CREATE PROFILE to set the resource limit example

First, create a profile called CRM_USERS that set the resource limits:

```
CREATE PROFILE CRM_USERS LIMIT
   SESSIONS_PER_USER       UNLIMITED
   CPU_PER_SESSION         UNLIMITED
   CPU_PER_CALL            3000
   CONNECT_TIME            15;
```
Code language: SQL (Structured Query Language) (sql)

Second, create a user called CRM:

```
CREATE USER crm IDENTIFIED BY abcd1234
PROFILE crm_users;
```
Code language: SQL (Structured Query Language) (sql)

Third, verify the profile of the CRM user:

```
SELECT
   username,
   profile
FROM
   dba_users
WHERE
   username = 'CRM';
```
Code language: SQL (Structured Query Language) (sql)

USERNAME	PROFILE
CRM	CRM_USERS

The user CRM is subject to the following limits: the CRM user can have any number of concurrent sessions (SESSIONS_PER_USER). In each session, it can consume any amount of CPU time (CPU_PER_SESSION). In addition, the CRM user cannot consume more than 30 seconds of CPU time in a single call. (CPU_PER_CALL) and each session cannot last for more than 15 minutes.

2) Using Oracle CREATE PROFILE to set the password limit example

First, create a new profile called erp_users with password limits:

```
CREATE PROFILE erp_users LIMIT
   FAILED_LOGIN_ATTEMPTS 5
   PASSWORD_LIFE_TIME 90;
```

Code language: SQL (Structured Query Language) (sql)

Then, create a user named sap and set its profile to erp_users:

```
CREATE USER sap IDENTIFIED BY abcd1234
PROFILE erp_users;
```

Granting and Revoking User Privileges in SQL

Granting and revoking user privileges in SQL involves managing the access rights and permissions of users on database objects. The specific syntax and commands may vary slightly depending on the database management system (DBMS) you are using, but I will provide you with some general examples that should work in most SQL databases.

Granting Privileges: To grant privileges to a user, you typically use the **GRANT** statement followed by the specific privileges and the object(s) on which the privileges should be granted. Here's a general syntax:

scssCopy code

```
GRANT privilege_type(s) ON object_name TO user_name;
```

- **privilege_type(s)** can be one or more of the following privileges:
  - **SELECT**: Allows the user to retrieve data from the specified table(s) or view(s).
  - **INSERT**: Allows the user to insert new rows into the specified table(s).
  - **UPDATE**: Allows the user to modify existing rows in the specified table(s).
  - **DELETE**: Allows the user to delete rows from the specified table(s).
  - **ALL**: Grants all available privileges on the specified object(s).
- **object_name** refers to the table(s), view(s), or other database object(s) on which the privileges should be granted.
- **user_name** is the name of the user to whom the privileges should be granted.

Here's an example to grant **SELECT** and **INSERT** privileges on a table named **employees** to a user called **john**:

sqlCopy code
```
GRANT SELECT, INSERT ON employees TO john;
```

Revoking Privileges: To revoke privileges from a user, you use the **REVOKE** statement followed by the specific privileges and the object(s) from which the privileges should be revoked. Here's a general syntax:

scssCopy code
```
REVOKE privilege_type(s) ON object_name FROM user_name;
```

- **privilege_type(s)** and **object_name** have the same meanings as described above.
- **user_name** is the name of the user from whom the privileges should be revoked.

Here's an example to revoke the **INSERT** privilege on the **employees** table from the user **john**:

sqlCopy code
```
REVOKE INSERT ON employees FROM john;
```

Remember to adjust the specific privileges, object names, and user names according to your database schema and requirements. Additionally, keep in mind that the exact syntax and available privileges may vary depending on the database system you are using.

Creating , Assigning and Revoking User Roles

In SQL, creating, assigning, and revoking user roles involves working with the database's security features. Here's how you can perform these actions using SQL:

1. **Creating User Roles:** To create a user role in SQL, you typically use the **CREATE ROLE** statement. Here's an example:

   sqlCopy code
   ```
   CREATE
   ```

   Replace **role_name** with the desired name for the role you want to create.
2. **Assigning User Roles:** To assign a role to a user in SQL, you use the **GRANT** statement. This statement grants specific privileges associated with a role to a user account. Here's an example:

   sqlCopy code
   ```
   GRANT TO
   ```

   Replace **role_name** with the name of the role you want to assign, and **username** with the name of the user you want to assign the role to.
3. **Revoking User Roles:** To revoke a role from a user in SQL, you use the **REVOKE** statement. This statement removes specific privileges associated with a role from a user account. Here's an example:

```
sqlCopy code
REVOKE role_name FROM username;
```

Replace **role_name** with the name of the role you want to revoke, and **username** with the name of the user you want to revoke the role from.

It's important to note that the syntax and functionality of user roles can vary slightly depending on the specific database management system (DBMS) you are using. The examples provided above demonstrate the general approach, but you should consult the documentation for your specific DBMS to ensure accurate usage and to explore additional features or options related to user roles and permissions.

Best Practices

When it comes to creating, assigning, and revoking user roles in SQL, it's essential to follow best practices to ensure a secure and well-managed database system. Here are some best practices to consider:

1. **Least Privilege Principle:** Apply the principle of least privilege when assigning roles. Only grant the minimum privileges necessary for users to perform their required tasks. This minimizes the risk of unauthorized access and potential damage caused by accidental or malicious actions.
2. **Separation of Duties:** Implement a separation of duties by creating distinct roles for different functional areas or responsibilities. This helps prevent conflicts of interest and reduces the risk of unauthorized activities. For example, separate roles for application administrators, database administrators, and end users.
3. **Regular Review and Auditing:** Conduct regular reviews of user roles and their associated permissions. Periodically audit the access rights granted to ensure they align with the current needs of users and the organization. Remove any unnecessary or outdated roles and permissions to maintain a clean and secure database environment.
4. **Avoid Default Privileges:** Avoid granting default privileges to all users or granting excessive permissions by default. Instead, create specific roles and assign them on an as-needed basis. This ensures that users only receive the necessary privileges when they require them.
5. **Secure Role Management:** Protect the management of roles by restricting access to privileged accounts. Only authorized personnel should have the authority to create, assign, or revoke roles. Secure the administrative accounts with strong passwords and implement multi-factor authentication to prevent unauthorized changes to user roles.
6. **Testing and Validation:** Before assigning or revoking roles, thoroughly test and validate the changes in a controlled environment. Ensure that the new roles or modifications do not inadvertently disrupt application functionality or cause unintended consequences. Implement a structured testing process to minimize the impact on production systems.
7. **Documentation and Documentation Review:** Maintain clear documentation of user roles, their associated permissions, and the rationale behind their assignment. Document the process for creating, assigning, and revoking roles to provide a clear understanding for

administrators and auditors. Regularly review the documentation to keep it up to date with any changes made to user roles.

8. **Regular Security Training:** Provide regular security training and awareness programs for users, administrators, and developers. Educate them about the importance of role-based access control, best practices for managing user roles, and potential risks associated with mishandling roles and permissions.

By following these best practices, you can establish a secure and well-managed user role system in SQL that aligns with the principle of least privilege and ensures appropriate access to your database resources.