

Database Authorization: Roles, Grants, and Locking It to Business Hours

Mon, Aug 10, 9:00 AM EDT · <https://scottslab.io/posts/database-authorization-time-of-day>



TL;DR

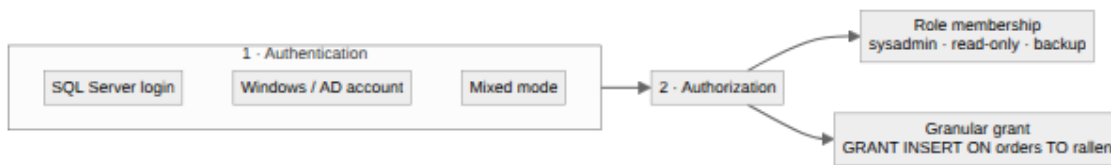
Database access is two questions stacked: authentication (SQL Server logins, Windows/AD accounts, or mixed mode) then authorization (broad role membership like sysadmin or read-only, or granular per-table GRANTS, revoked with REVOKE). Layer on time-of-day restrictions from Active Directory so an account only works during business hours, and scale the strictness to the data's sensitivity: risk-based access.

A database is where authorization gets concrete, because you can dial it from "you administer everything" down to "you may insert into this one table." And you should.

It starts with authentication, and SQL Server is the clean example. You can authenticate with SQL Server logins managed by the database itself, with Windows/Active Directory accounts managed by the domain, or with mixed mode that allows both. AD-backed auth is usually what you want, because it means database access rides your existing identity lifecycle: disable the person in Active Directory and their database access dies with the same click.

Then authorization, which comes at two granularities. Role membership is the broad brush: drop a login into `sysadmin`, or a read-only role, or a backup-admin role, and it inherits that role's whole

reach. Granular grants are the scalpel: `GRANT INSERT ON orders TO rallen` hands over exactly one privilege on exactly one table, and `REVOKE` takes it back just as precisely. Least privilege lives right here: reach for the narrowest grant that lets the job get done, not the role that happens to be convenient.



Two more levers are worth knowing. Time-of-day restrictions cap *when* an account can even log in. In Active Directory Users and Computers you can limit logon hours, so Alice's account only works weekdays from 8 to 6, and a 3am login with her credentials simply can't succeed no matter who's typing them. And risk-based access ties it together: the more sensitive the data, the more controls you stack on top of it. Not every table deserves the same ceremony, and not every table can be left wide open. Match the friction to what's actually at stake.

