### Principals

* Principals are entities that can request access to SQL Server resources. Resources can be anything within SQL Server – Server, Database, Tables, Stored Procedures etc…
* Every Principal has a Security Identifier (SID)
* Windows-level principals - Windows Domain Login, Windows Local Login
* SQL Server-level principals - SQL Server Login, Server Role
* Database-level principals - Database User, Database Role, Application Role
* Principals can be of two types…
  + **Indivisible** – e.g. Windows Local Login, SQL Server Login
  + **Collection** – e.g. Windows Domain Login, Database Role etc
* *sa* login is the only default server level principal that is created as a part of *sysadmin* fixed server role when an instance is created
* *sa* cannot be locked or deleted but can be disabled

### Securables

* **Securables** are the resources to which the SQL Server Database Engine Authorization System works
* Securables can be contained within other securables, creating nested hierarchies called **scopes**
* Securable Scopes are…
  + Server
    - Endpoint
    - Login
    - Database
  + Database
    - User
    - Role
    - Application role
    - Assembly
    - Message Type
    - Route
    - Service
    - Remote Service Binding
    - Fulltext Catalog
    - Certificate
    - Asymmetric Key
    - Symmetric Key
    - Contract
    - Schema
  + Schema
    - Type
    - XML Schema Collection
    - Objects
      * Aggregate
      * Function
      * Procedure
      * Queue
      * Synonym
      * Table
      * View

### Schema

* Schema is a container that holds the securables created within the database
* Each securable must be contained in a schema
* Schemas created by one, can be owned by any principal. When Schema is owned by a database role, the database objects within the schema can be administered by all the database role members
* Default schema can be chosen when an user is created. All objects created by that user will be contained in that schema.
* If no default schema is specified, dbo will be the default schema for the user and the objects created by him unless overridden
* For SQL Server 2000 and before versions, no separate schemas existed. By default, database users were the schemas. From SQL Server 2005, schemas and database users were separated. This feature is called the **user-schema separation**
* Benefits of **user-schema separation**…
  + Simple to drop database users without any Object dependency errors
  + You don’t need to rename objects after dropping a database user
  + Multiple users can own/share a single schema (via roles or windows domain groups)
  + Move objects between schemas

### Permissions

* A principal can be granted, revoked or denied permission
* Granting permissions at…
  + database level enables access to any schema within that database
  + schema level enables access to any securable within that schema
* Granting permission examples…

GRANT SELECT ON [dbo].[table1] TO [domain\user1]

REVOKE SELECT ON [dbo].[table1] TO [domain\user1]

DENY SELECT ON [dbo].[table1] TO [domain\user1]

GRANT UPDATE ON [dbo].[table1] ([Col1]) TO [domain\user1] AS [dbo]

GRANT SELECT ON [dbo].[table1] TO [domain\user1] WITH GRANT OPTION

* Set of Permissions that can be granted/revoked/denied…
  + SELECT – View the data of the securable
  + INSERT – Insert data into a securable
  + UPDATE – Update data inside a securable
  + DELETE – Delete data from a securable
  + CREATE – Create securables within the scope where permission is granted
  + ALTER – Alter properties of the securables defined within the scope where permission is granted
  + ALTER ANY – Enables a principal to change any securable within a specific securable scope. E.g. ALTER ANY LOGIN permission allows the principal to create, alter, drop any login at the server securable scope
  + REFERENCES – With this permission, you can reference a securable from another securable
  + CONTROL – With this permission on a securable, users have all access on that securable including the securables within that securable
  + EXECUTE – Permission to run a securable (SP, synonym, functions etc…)
  + VIEW DEFINITION – View the structure of a securable. E.g. viewing the definition of a table lists all the columns, indexes, keys, constraints etc…
  + TAKE OWNERSHIP
  + VIEW CHANGE TRACKING – When database option “CHANGE\_TRACKING” is turned on, securables that are modified are tracking. With this permission, you can track/view those changes
  + IMPERSONATE – With this permission, users can enable a principal to impersonate a user or login securable

## Fixed Roles

* Fixed Server and Database roles enables users to manage permissions at server and database level respectively
* Adding a SQL Server login or a windows group to a fixed server role grants them permission at the server scope
* Logins that are added to the fixed server role can add other logins
* Server Fixed Roles
  + sysadmin
    - Members of this role can perform any task within SQL Server
    - Contains the Windows Built-in administrator account by default
    - Predefined Permission for this role – CONTROL SERVER
  + securityadmin
    - Members of this role can create and manage logins for SQL Server i.e. GRANT, DENY, REVOKE permissions on server level and database level
    - No members by default are added
    - Predefined Permission for this role – ALTER ANY LOGIN
  + serveradmin
    - Members of this role can change server configuration and shut down the server
    - No members by default are added
    - Predefined Permission for this role – ALTER ANY ENDPOINT, ALTER RESOURCES, ALTER SERVER STATE, ALTER SETTINGS, SHUTDOWN, VIEW SERVER STATE
  + processadmin
    - Members of this role can end processes that are running in a SQL Server instance
    - Predefined Permission for this role – ALTER ANY CONNECTION, ALTER ANY STATE
  + setupadmin
    - Members of this role can add/remove linked servers in an instance
    - Predefined Permission for this role – ALTER ANY LINKED SERVER
  + bulkadmin
    - Members of this role can run BULK INSERT statements
    - Predefined Permission for this role – ADMINISTER BULK OPERATIONS
  + diskadmin
    - Members of this role can manage disk files i.e. add/remove backup devices
  + dbcreator
    - Members of this role can CREATE, ALTER, DROP AND RESTORE any database within SQL Server instance
  + public
    - Members of this role can access all the securables that are not granted or denied permissions specifically
    - Every SQL Server login is a part of public role
* Database Fixed Roles
  + db\_owner
    - Can perform any task on the database including DROP DATABASE.
    - Drop database permission is provided to db\_owner since SQL Server 2005
    - To impersonate this role, GRANT the CONTROL Permission at database level
  + db\_securityadmin
    - Enables the members of the database to manage database roles and permissions at the database level
    - Managing includes – GRANT, REVOKE, DENY
    - To impersonate this role, GRANT the following permissions…
      * ALTER ANY APPLICATION ROLE
      * ALTER ANY ROLE
      * CREATE SCHEMA
      * VIEW DEFINITION
  + Both the above roles have VIEW ANY DATABASE permissions at server level.
  + With this permission, they can view any database but can perform respective tasks for the databases for which they are assigned to
  + db\_reader
    - Members of this role can VIEW any data from tables within database
    - To impersonate this role, GRANT SELECT permission
  + db\_datawriter
    - Members of this role can modify data in any tables including DELETE within a database
    - To impersonate this role, GRANT DELETE, INSERT AND UPDATE permissions
  + db\_ddladmin
    - Members of this role can run DDL statements
    - To impersonate this role, GRANT ALTER, CREATE and DROP within the database
  + db\_denydatareader
    - Members of this role **cannot** view any data in the user tables within database
    - To impersonate this role, DENY SELECT permissions
  + db\_denydatawriter
    - Members of this role **cannot** change any data in the user tables within the database
    - To impersonate this role, DENY DELETE, INSERT and UPDATE permissions
  + db\_backupoperator
    - Members of this role can BACKUP the database
    - To impersonate this role, GRANT the following permissions
      * BACKUP DATABASE
      * BACKUP LOG
      * CHECKPOINT
  + db\_accessadmin
    - Members of this role can manage the access to the database
    - db\_accessadmin permission enables to GRANT or REVOKE CONNECT permission to SQL Server logins, Windows accounts/groups
    - To impersonate this role, GRANT the following permissions…
      * CONNECT
      * ALTER ANY USER
      * CREATE SCHEMA
  + Public
    - All database users are by default added to this role
    - Public role members have SELECT permissions on sys schema

## Authentication

* SQL Server 2008 supports two types of authentication mode
  + Windows Authentication
    - When this mode is selected, users are authenticated using their local or domain windows user accounts
    - Users no need to provide username or password
    - SQL Server authenticates users with the windows token which is already confirmed by the operating system. This is called Trusted connection
    - Windows authentication is the most secure mode for the following reasons…
      * Uses Kerberos security protocol
      * Passwords are governed by local/domain password policies
      * User accounts and passwords are stored and transmitted securely
      * Password Expiration and Account Lockout features can be used against Windows authenticated users
  + Mixed mode Authentication
    - Both Windows and SQL Server authentications are used
    - For SQL Server login, user needs to provide username and password
    - SQL Logins when enabled with “Password expiration policy”, SQL Server uses the password policy that is configured in windows and ensure all SQL logins adhere to this
* While creating passwords with symbols, ensure you enclose the with []

## Logins

While creating the login, these are the settings you would configure…

* Mapped to Certificate – When Logins are mapped to a certificate, the data is encrypted or signed with a digital signature
* Mapped to asymmetric key – same as certificate. Instead you have a key
* Map to Credential – Mapping to login credentials that is used outside SQL Server
* Default Database & Default Language
* Note: Any login accounts that you create from a certificate or asymmetric key cannot connect to SQL Server. They can only be used for signing

## Schema

* Schema adds additional security to the database objects by grouping them
* A user can own multiple schemas. User owning the schema will have all permissions on the schema
* Default schema for the user is used when they are creating or accessing objects
* When an user is assigned a schema, all objects created by the user in that database comes under that schema
* Each fixed role has a schema with its name and that role is the owner for that schema.
* Additionally those fixed roles has the default schemas – DBO, guest, INFORMATION\_SCHEMA, sys
* Information schema
  + Stores the informational schema views for the database
  + These views contain the metadata for all the objects in the database
  + Schema owned by database user – INFORMATION\_SCHEMA
* Sys schema
  + Stores the sys informational views, system stored procedures, system views and system functions
  + Owned by user – sys

## Data Encryption

* SQL Server 2008 can use encryption to store data or transmit data over network
* Users can encrypt data stored in a table as long as they have permissions and the key to encrypt
* Encryption process uses keys to encrypt data. SQL Server 2008 uses the below keys…
  + Master keys
    - Service Master Key at server level
      * Created when SQL Server is installed. This can neither be created nor deleted
      * Encrypted by the Windows Data Protection Application Programming Interface (DPAPI) on the local machine when created
      * Encrypted using the SQL Server Service Account Credentials. So only this account can decrypt the key
    - Database Master Key at database level
      * This key is used under Service Master Key and is automatically created and encrypted by the service master key
      * This key is not created when a database is created. So, users have to create explicitly
      * Ensures that all keys inside a database is encrypted using a single database master key
      * This key is encrypted using DES and a password
      * Example…

USE <dbname>

CREATE MASTER KEY ENCRYPTION BY PASSWORD = ‘p@$$w0rd’

* + Certificates
    - Certificates are the convenient way of transferring public keys from one server to another
    - Certificates (also called as digitally-signed security objects) associate public key to private key held by the principal
    - Associating a key to a certificate can be performed
      * using the FROM *existing\_key* option (see CREATE CERTIFICATE syntax for more details)
      * using a new key that is generated using the generate\_new\_keys option
    - Certificates are used for encrypting other keys in the database
    - Certificates are signed by Verisign or Microsoft Certifying Authority
    - Certificates can be generated in either of these ways…
      * Created by SQL Server 2008
      * Generated from an external file
      * Created by user using the CREATE CERTIFICATE option
      * Copy a certificate that is owned by another user (using AUTHORIZATION keyword)
    - Syntax

CREATE CERTIFICATE certificate\_name

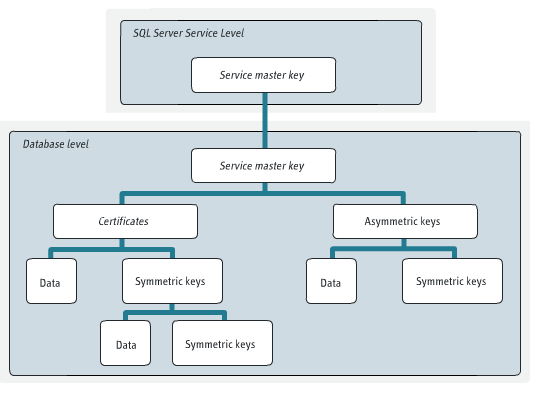
[AUTHORIZATION user\_name] {FROM existing\_keys | generate\_new\_keys}

[ACTIVE FOR BEGIN\_DIALOG = {ON | OFF}]

existing\_keys ::= ASSEMBLY assembly\_name | {[EXECUTABLE] FILE = ‘path\_to\_file’ [WITH PRIVATE KEY (private\_key\_options)] }

private\_key\_options ::= FILE = ‘path\_to\_private\_key’ [,DECRYPTION BY PASSWORD = ‘password’] [,ENCRYPTION BY PASSWORD = ‘password’]

<generate\_new\_keys> ::= [ENCRYPTION BY PASSWORD = ‘password’] WITH SUBJECT = ‘certificate\_by\_subject\_name’ [START\_DATE = ‘mm/dd/yyyy’ | EXPIRY\_DATE = ‘mm/dd/yyyy’]



* Ways of implementing encryption
  + **Symmetric Key Encryption**
    - Encryption and Decryption is carried out using the same key.
    - Since the sender and receiver should have the same key, the key should be transmitted over the network
    - Symmetric keys are preferred over asymmetric for protecting data directly i.e. when data need not be transmitted over the network
    - They can protect large amount of data and is the faster way of encrypting and decrypting data
    - To View the data in the column, open the symmetric key and then select data from table using the DECRYPTBYKEY keyword

USE <dbname>

OPEN SYMMETRIC KEY sym\_name DECRYPT BY CERTIFICATE certificate\_name

SELECT EMPID, firstname, lastname, CONVERT(varchar, DECRYPTBYKEY (employeessn) FROM tablename

CLOSE ALL SYMMETRIC KEYS

* + - Syntax…

CREATE SYMMETRIC KEY symm\_key\_name

[AUTHORIZATION owner\_name]

[FROM PROVIDER provide\_name]

WITH <key\_options> [,…n] | ENCRYPTION BY encryption\_mechanism [,…n]

* + - To create a symmetric key, there must be a database master key. To check whether a database master key exists run the query

USE <dbname>

SELECT \* FROM sys.symmetric\_keys WHERE name LIKE ‘%DatabaseMasterKey%’

* + **Asymmetric Key Encryption**
    - One Public Key and one private key
    - Public keys can be used by any user of the database
    - Private keys can be used by the owner of the key or permissions to use the key
    - If One key is used to encrypt data, the other key is used to decrypt the data and vice versa
    - Asymmetric keys are more secure than symmetric but can be CPU intensive
    - They can be used encrypt symmetric keys within a database and can be used instead of Certificates
    - Syntax…

CREATE ASYMMETRIC KEY asym\_key\_name

[AUTHORIZATION database\_principal\_name] {FROM Asysm\_key\_source | WITH ALGORITHM = {RSA\_512 | RSA\_1024 | RSA\_2048}}

[ENCRYPTION BY PASSWORD = ‘password’]

* + **Digital Certificates**
    - A form of Asymmetric key encryption, digital certificate is the digital representation of information
    - Digital Certificates contain the following…
      * Certification authority Signature
      * Information about its subscribers
      * User’s Public key
      * Operational period of the certificate
  + **Transparent Data Encryption**
    - Uses a symmetric key called the Database Encryption Key (DEK) to encrypt an entire database
    - DEK is stored beneath master key or as an asymmetric key in an Extensible Key Management (EKM) module
* SQL Server supports the following algorithm for encryption…

1. 128-bit Advanced Encryption Standard (AES)
2. 192-bit AES
3. 256-bit AES
4. Data Encryption Standard (DES)
5. Triple DES
6. Triple\_DES\_3KEY
7. Rivest’s Cipher (RC) 2
8. RC4
9. 128-bit RC4

* The AES Algorithm is not supported in Windows XP and Windows Server 2000
* Guidelines for choosing the appropriate algorithm
  + Cipher Strength – A block cipher using a long encryption key is much stronger than a stream cipher
  + Compression – Encrypted data cannot be compressed. So compress before encrypting
  + Encrypting Twice – Encrypting first with a symmetric key and second time with an asymmetric key adds more security to the data
  + CPU Usage – Stronger encryption used more CPU. So choose accordingly
  + Key and password length – Longer ones are more secure.
* **Transparent Data Encryption**
* TDE encrypts the entire database including data and log files and the backups
* The database is encrypted using an encryption key (Database Encryption Key) that is stored in the database boot record on disk
* DEK is encrypted by database master key and is stored in master database
* Even if the databases are stolen, the data cannot be read without the DEK
* TDE encrypts the database at page level before data is written to disk and decrypts the data when the pages are read into memory for use
* TDE does not increase the size of the database
* Considerations for TDE…
  + For database mirroring/log shipping all involved databases are encrypted
  + If any file groups are configured read-only, the encryption process fails
  + Data compression works efficiently when combined with TDE
  + FILESTREAM data is not encrypted
  + If any user database is encrypted, tempdb is automatically encrypted
* To enable TDE…
  + Create Master key in the master database

USE master

CREATE MASTER KEY ENCRYPTION BY PASSWORD = ‘password’

* + Create/Obtain a certificate that is encrypted by master key

USE master

CREATE CERTIFICATE CertName WITH SUBJECT = ‘Certification for TDE’

* + Create Database Encryption key

USE <dbname>

CREATE DATABASE ENCRYPTION KEY WITH ALGORITHM = AES\_128 ENCRYPTION BY SERVER CERTIFICATE CertName

* + Backup the certificate and private key to recover the database in case key is lost. The certificate is backed up to

USE master

BACKUP CERTIFICATE CertName TO FILE = ‘FileLocation1’ WITH PRIVATE KEY (FILE = ‘FileLocation2’, ENCRYPTION BY PASSWORD = ‘password’)

* + Enable the Encryption on the database

USE <dbname>

ALTER DATABASE <dbname> SET ENCRYPTION ON

## Impersonation

* Impersonation simplifies the permissions that need to be granted in the database and ensures that the end user does not have direct access to the database and its objects.
* Three ways of implementing impersonation
  + Module Signing
    - A Module refers to Stored Procedures, Functions, Triggers (excluding DDL Triggers), Assemblies
    - Module signing does not change the execution context as like ‘EXECUTE AS’ statement
    - To impersonate using module signing…
      * Create a Certificate on server
      * Create a login based on the certificate. The login does not need CONNECT permission, so revoke connect permission and grant required permission (for e.g. execute permission to a SP)
      * Now Sign the SP with the certificate using the ADD SIGNATURE statement
  + Execution Context
    - Execution context uses security tokens (combination of login and user token) to connect using a different user context
    - Performed by EXECUTE AS statements
    - EXECUTE AS supports these contexts…
      * EXECUTE AS CALLER – The module executes in the context of the user who calls the object
      * EXECUTE AS SELF - The module executes in the context of the user who created the SP/Function/Trigger
      * EXECUTE AS OWNER – The module executes in the context of the module owner. This can’t be used when calling DDL Triggers or the module owner is a group/role
      * EXECUTE AS login\_name or user\_name – This can be used only with DDL Triggers at server level
  + Ownership Chaining
    - You can create trust between databases, by granting the authenticator “AUTHENTICATE“ permission
    - Setting AUTHENTICATE permission needs TRUSTWORTHY database option to be ON which results in ownership chains

USE db1

GRANT AUTHENTICATE TO user\_on\_db2

ALTER DATABASE db2 SET TRUSTWORTHY ON

* + - Apart from creating trust between entire databases, you can set trust for specific set of modules to access another database using certificates and asymmetric keys without setting TRUSTWORTHY option to ON
    - The certificates and asymmetric keys are used as authenticators rather than database owner by a method called signing

## Auditing

* Auditing data access can be done at server level, database level or individual object level
* Each time an auditable action is met, an audit event will occur
* Different methods for auditing…
  + DDL Triggers
    - Affects all databases in the instance as it is scoped at server level
    - However, when DDL triggers are scoped to the database level, to monitor changes to a particular database
    - EVENTDATA function captures detailed information about the event
  + Logon Triggers
    - Logon triggers respond to a logon event. A logon event occurs when a user session is initiated within an instance of SQL server. If user authentication fails, Logon triggers will not fire
    - Logon triggers are used to audit and control server sessions
    - Using login triggers, you can restrict the number of sessions for particular login, track login activity etc
    - Many triggers can be fired for a single event and the order of the triggers is determined by sp\_settriggerorder statement
  + Event Notifications
    - Captures T-SQL DDL statements as well as SQL Trace events to a service broker service
    - They run outside the scope of a transaction so they can be run from a remote server. Also they are not coupled with the event that triggered them
    - When an event notification is created, a dialogue is created between SQL Server, target server and any number of service brokers
    - Event information is delivered to the service broker as variable of type XML
    - To create an event notification, create the target service that will receive the event notifications. For creating the target service,
      * + Create a queue to receive the event notification messages

CREATE QUEUE queue\_name

WITH STATUS = ON | OFF

,RETENTION = ON | OFF

,ACTIVATION ( STATUS = ON | OFF

,PROCEDURE\_NAME = <procedure\_name>

,MAX\_QUEUE\_READERS = maxreaders

,EXECUTE AS {SELF | 'user\_name' | OWNER}

)

ON filegroup | [DEFAULT]

* + - * + Create a service and attach it to the message queue that references the event notification contract

CREATE SERVICE service\_name [AUTHORIZATION owner\_name]

ON QUEUE [schema\_name.]queue\_name (contract\_name | | DEFAULT)

* + - * + Create a route on that service to specify the address to which the service broker sends messages

CREATE ROUTE route\_name AUTHORIZATION owner\_name

WITH SERVICE\_NAME = 'service\_name'

,BROKER\_INSTANCE = 'broker\_instance\_identifier'

,LIFETIME = 'route\_lifetime'

,ADDRESS = 'next\_hop\_address'

,MIRROR\_ADDRESS = 'next\_hop\_mirror\_address'

* + C2 Auditing
    - Auditing at server level
    - C2 tracks Successful and failed login & Track security violations
    - The data from C2 audit are saved into a file and once this file reaches 200 MB, a new file is generated and forthcoming records are written to the new one
  + Common Criteria Auditing
    - Common Criteria is a guide for the development, evaluation and procurement of IT products with security functionality
    - To enable CC compliance, use sp\_configure
    - Advantages of enabling the CC Compliance…
      * Ability for a table-level DENY to assume priority over a column-level GRANT
      * Capacity to view login statistics
      * Use of Residual Information Protection