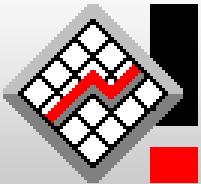


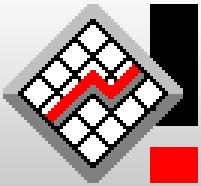
Berechtigungs-Architektur der MS-SQL-Server- Datenbank-Engine

Referent: Sebastian Flucke



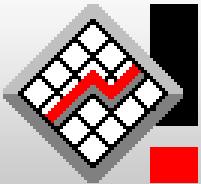
Was erwartet Sie in dieser Session?

- Sicherheit geht alle an
- Permissions, Principals und Securables
- Zusammenspiel Instanzen / Datenbanken
- Rollenkonzept
- effektive Rechte / Impersonation
- (Migrations-Beispiel)
- A-G-DL-P
- SQL-Server Agent
- Installations-Aspekte



Sicherheit ...

- nervt beim Entwickeln
- kostet Zeit und Geld

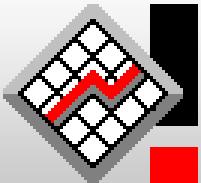


Sicherheit geht alle an ☺

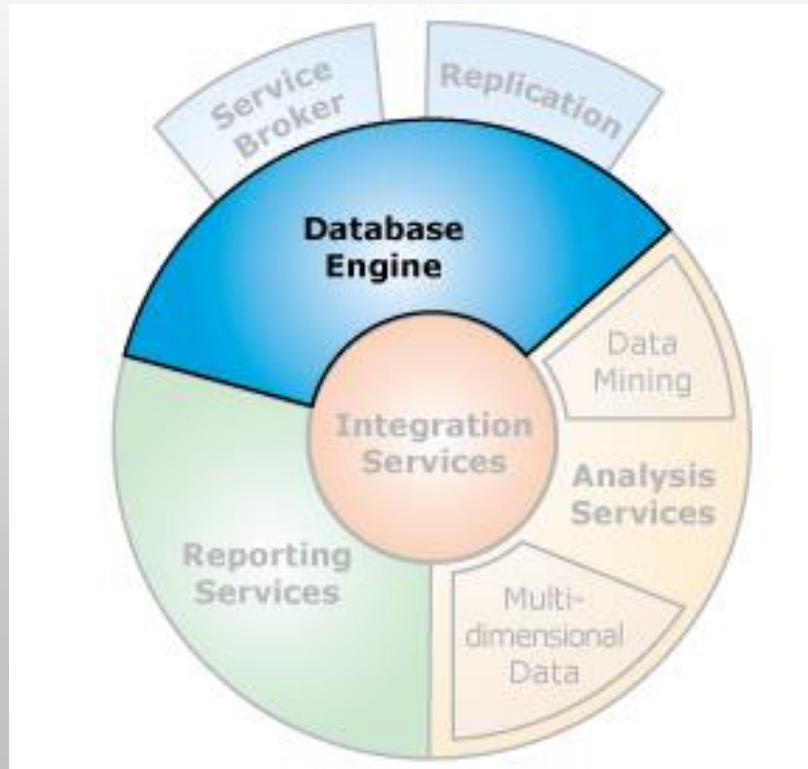
- nervt beim Entwickeln
- kostet Zeit und Geld

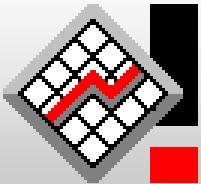
ABER - Sicherheit

- ist unabdingbar
- soll nicht beim Anwenden hindern
- braucht Konzepte, damit sie sicher ist



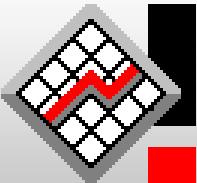
Die SSDE - SQL-Server-Database-Engine





Security-affine Aspekte der SSDE

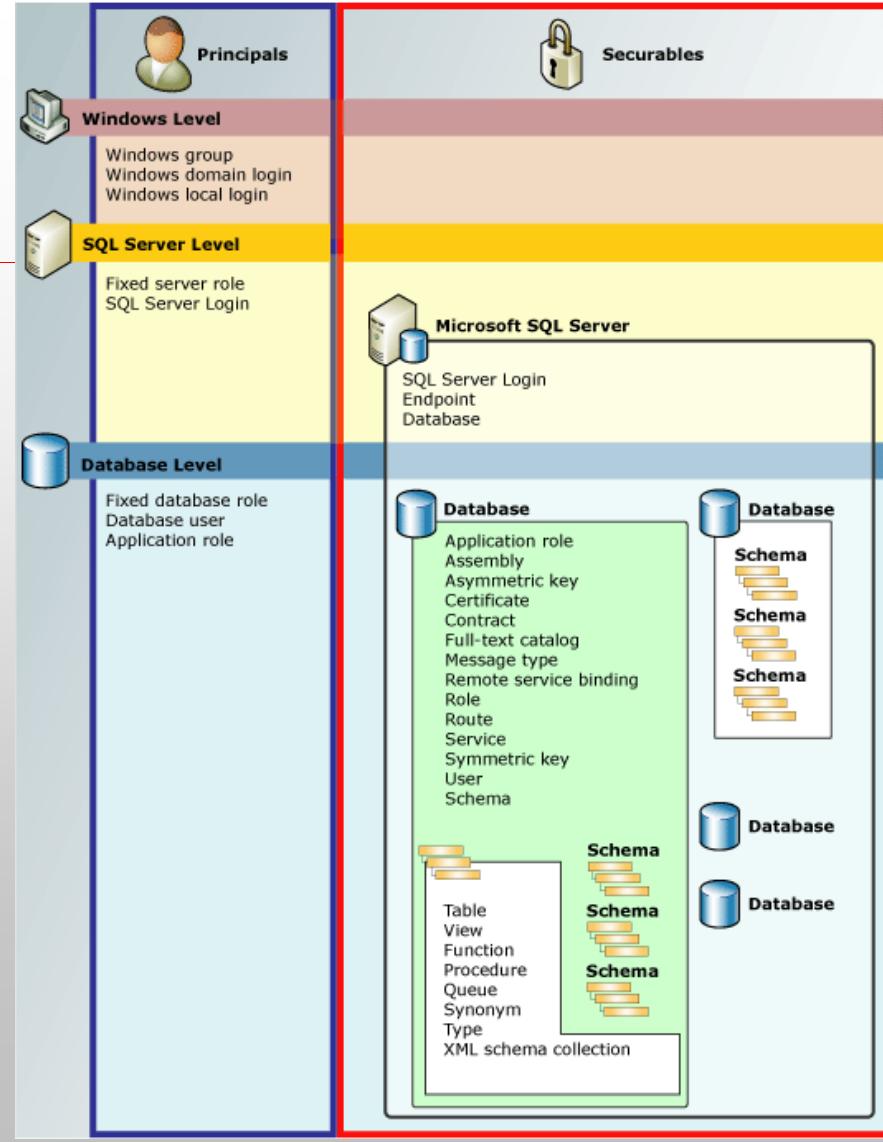
- umgebendes Betriebssystem
- ggf. Cluster-Umgebung
- SQL-Server-Instanzen
 - SQL-Server-Datenbanken
 - Objekte innerhalb der Datenbanken
(Schemas, Tabellen, Views, Prozeduren...)
 - instanzbezogene Verwaltungsaspekte
 - Agent
 - Database Mail
 - Replikationen, Linked Server usw.

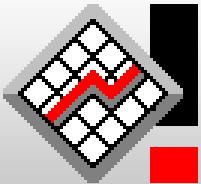


SSDE-Security im Überblick

- Principals
- Securables
- Permissons

Das nebenstehende Schema umfaßt nicht alle Aspekte!





Permissions, Principals und Securables

■ Permission

- Erlaubnis, etwas zu tun

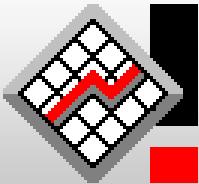
■ Principal

- Anforderer von Ressourcen
- Principals können auch Securables sein

■ Securable

- Ressourcen, für die Permissions erforderlich sind
- Securables können auch Principals sein

ERGO: Ein Principal bekommt Permissions, um auf Securables zugreifen zu dürfen (oder nicht!).



Security auf Ebene der SQL-Server-Instanzen (I)

■ frei definierbare Principals

- Logins

```
SELECT * FROM sys.server_principals  
WHERE [type] IN ('S', 'U', 'G', 'C', 'K')
```

■ Securables

- Datenbanken

```
SELECT * FROM sys.databases
```

– in GUI auf Server-Ebene nicht unter "Security/ Logins/ Securables", sondern unter "Security/ Logins/ User Mapping" (Details dazu später)

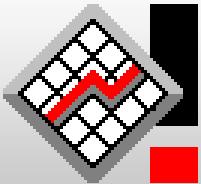
- Logins

- der eigene Server

- (Endpoints

```
SELECT * FROM sys.endpoints
```

■ Einem Login kann man also Rechte bzgl. Datenbanken, anderer Logins und Servern zuweisen.



Security auf Ebene der SQL-Server-Instanzen (II)

■ Typen von Logins

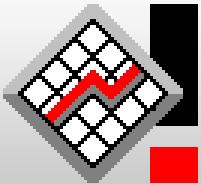
- SQL-Server-Login (`type='S'`)
- Windows-User-Login (`'U'`)
- Windows-Gruppen-Login (`'G'`)
- (Login mapped to a certificate (`'C'`))
- (Login mapped to an asymmetric key (`'K'`))

■ weitere Server-Principals

- Server-Rollen

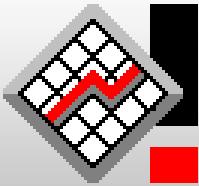
```
EXECUTE sp_helpsrvrole
```

```
SELECT * FROM sys.server_principals  
WHERE [type] = 'R'
```



Security auf Ebene der SQL-Server-Instanzen (III)

- Serverrollen
 - feste Serverrollen (nicht frei definierbar)
 - freie Serverrollen (frei definierbar)
- beinhalten Kombinationen von Securables und Permissions, im Wesentlichen für administrative Zwecke
- organisatorisch eine Art Gruppe – bekommt Logins als Mitglieder zugewiesen
- außerdem auch serverseitige Einzelpermissions



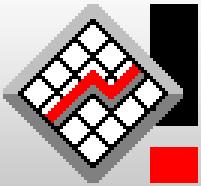
Permissions (I)

■ Permissions

- daten-bezogen
 - SELECT, UPDATE, INSERT, DELETE
- programmodul-bezogen
 - EXECUTE
- allgemeines Objekt-Management
 - VIEW DEFINITION, ALTER, TAKE OWNERSHIP, CONTROL
- für spezielle Objekte
 - VIEW CHANGE TRACKING, REFERENCES, RECEIVE

■ für unterschiedliche Securables

- siehe `SELECT * FROM fn_builtin_permissions('')`

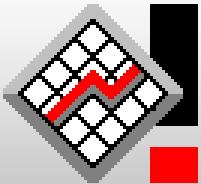


Principals auf Datenbank-Ebene (I)

■ Database User

- basierend auf Server-Logins

```
SELECT * FROM sys.database_principals  
WHERE [type] IN ('S', 'U', 'G')
```



Principals auf Datenbank-Ebene (II)

■ Roles

- Fixed Database Roles

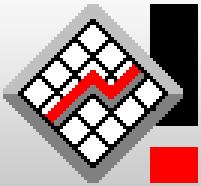
```
EXECUTE sp_helpdbfixedrole  
SELECT * FROM sys.database_principals  
WHERE [type] = 'R' AND is_fixed_role = 1
```

- Database Roles

```
EXECUTE sp_helprole  
SELECT * FROM sys.database_principals  
WHERE [type] = 'R' AND is_fixed_role = 0
```

- Application Roles

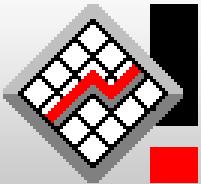
```
EXECUTE sp_helprole  
SELECT * FROM sys.database_principals  
WHERE [type] = 'A'
```



Principals auf Datenbank-Ebene (III)

■ weitere Database-Principals

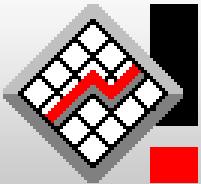
- User mapped to a certificate
- User mapped to an asymmetric key
- `SELECT * FROM sys.database_principals
WHERE [type] IN ('C', 'K')`
- nur der Vollständigkeit halber!



Securables auf Datenbank-Ebene

■ Schema

- Objects
 - Table
 - Column
 - View
 - Column
 - Procedure
 - Function
 - (Sonstiges: Aggregate, Constraint, Queue, Statistic, Synonym)
 - (Type)
 - (XML Schema Collection)
- | |
|-------------------------------|
| SELECT * FROM sys.tables |
| SELECT * FROM sys.columns |
| SELECT * FROM sys.views |
| SELECT * FROM sys.columns |
| SELECT * FROM sys.sql_modules |
| SELECT * FROM sys.sql_modules |
| SELECT * FROM sys.objects |



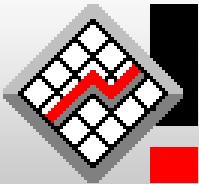
Weitere Securables auf Datenbank-Ebene

■ Database Principals

- Database User
- Roles
- usw.

■ Sonstiges

- (Assembly, Message Type, Route, Service, Remote Service Binding, Fulltext Catalog, Certificate, Asymmetric Key, Symmetric Key, Contract)



Database Permissions

■ abrufbar in einer wilden Kreuztabelle

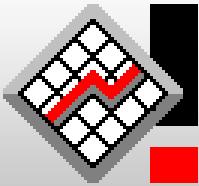
- `SELECT * FROM sys.database_permissions`

class	class_desc	major_id	minor_id	grantee_principal_id	grantor_principal_id	type	permission_name	state	state_desc
1	OBJECT_OR_COLUMN	-101	0	0	1	SL	SELECT	G	GRANT
1	OBJECT_OR_COLUMN	581577110	3	5	1	SL	SELECT	D	DENY
1	OBJECT_OR_COLUMN	1298103665	0	5	1	DL	DELETE	G	GRANT
1	OBJECT_OR_COLUMN	1298103665	0	5	1	IN	INSERT	G	GRANT
1	OBJECT_OR_COLUMN	1298103665	0	5	1	UP	UPDATE	G	GRANT

■ je nach Inhalt von Class, Major_Id und Minor_Id Relationen zu

- `sys.columns` bzw. `sys.objects` und von dort weiter...
- ...oder zu einer von ca. 10 anderen Tabellen

■ außerdem Relationen zu `sys.database_principals`



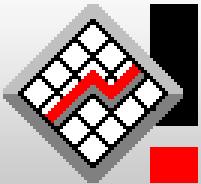
Server Permissions

- abrufbar in einer ähnlich wilden Kreuztabelle

- `SELECT * FROM sys.server_permissions`

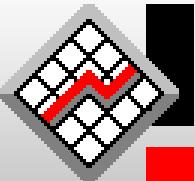
class	class_desc	major_id	minor_id	grantee_principal_id	grantor_principal_id	type	permission_name	state	state_desc
100	SERVER	0	0	262	1	COSQ	CONNECT SQL	G	GRANT
100	SERVER	0	0	263	1	COSQ	CONNECT SQL	G	GRANT
101	SERVER_PRINCIPAL	260	0	263	260	Vw	VIEW DEFINITION	D	DENY
105	ENDPOINT	2	0	2	1	CO	CONNECT	G	GRANT
105	ENDPOINT	3	0	2	1	CO	CONNECT	G	GRANT

- je nach Inhalt von Class, Major_Id und Minor_Id Relationen zu
 - `sys.servers`, `sys.endpoints` bzw. `sys.server_principals`
- außerdem Relationen zu `sys.database_principals`



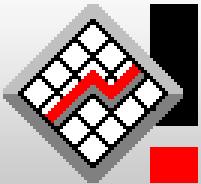
Effektive Rechte (I)

- entstehen durch Kombination von
 - verschiedenen Logins bei Windows-Authentifizierung
 - Mitgliedschaft in Server-Rollen
 - explizit vergebenen Permissions auf Server-Ebene
 - Mitgliedschaft in Datenbank-Rollen
 - explizit vergebenen Permissions auf Datenbank-Ebene
 - implizite Permissions auf Grund explizit vergebener übergeordneter Permissions (permission hierarchy)
- DENY ist stärker als GRANT - fast immer ☺
 - an einigen Stellen gibt es Dominanz durch die Objekt-Hierarchie (wg. Backward Compatibility)



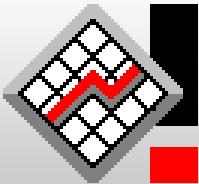
Permission Hierarchy (I)

```
SELECT * FROM fn_builtin_permissions('')
```



Permission Hierarchy (II)

- class_desc
 - Securable-Ebene
- permission_name / type
 - Permission
- covering_permission_name
 - übergeordnete Permission
- parent_class_desc
 - übergeordnetes Securable
- parent_covering_permission_name
 - relevante Permission des übergeordneten Securables



Effektive Rechte (II)

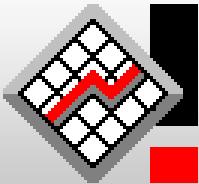
■ Permissions eines Principals auf ein Securable

- `SELECT * FROM fn_my_permissions('AdventureWorks2008.Sales.CreditCard' , 'OBJECT')`

entity_name	subentity_name	permission_name
AdventureWorks2008.Sales.CreditCard	CardType	SELECT
AdventureWorks2008.Sales.CreditCard	CreditCardID	SELECT
AdventureWorks2008.Sales.CreditCard	ExpMonth	SELECT
AdventureWorks2008.Sales.CreditCard	ExpYear	SELECT
AdventureWorks2008.Sales.CreditCard	ModifiedDate	SELECT

■ Einzel-Permission eines Principals auf ein Securable gegeben?

- `SELECT HAS_PERMS_BY_NAME('AdventureWorks2008.Sales.CreditCard', 'OBJECT', 'SELECT', 'CreditCardNumber', 'COLUMN')`
--> 0
- `SELECT HAS_PERMS_BY_NAME('AdventureWorks2008.Sales.CreditCard', 'OBJECT', 'SELECT', 'CardType', 'COLUMN')`
--> 1



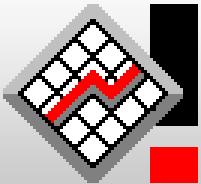
Effektive Rechte (III)

■ implizite Permission-Hierarchie

- ```
SELECT * FROM
 dbo.ImplyingPermissions
 ('database' , 'view definition')
```
- ```
SELECT * FROM
    dbo.ImplyingPermissions
    ( 'object' , 'update' )
```

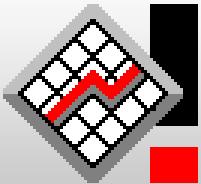
permname	class	height	rank
VIEW DEFINITION	DATABASE	0	0
CONTROL	DATABASE	0	1
VIEW ANY DEFINITION	SERVER	1	1
CONTROL SERVER	SERVER	1	2

permname	class	height	rank
UPDATE	OBJECT	0	0
CONTROL	OBJECT	0	1
UPDATE	SCHEMA	1	1
CONTROL	SCHEMA	1	2
UPDATE	DATA... BASE	2	2
CONTROL	DATA... BASE	2	3
CONTROL SERVER	SERVER	3	4



Monitoring von Usern und Rechten

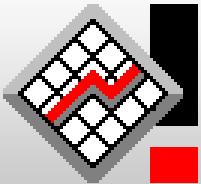
- Auditing
- Profiler-Traces
- Extended Events
- turnusmäßige Zustands-Snapshots mit Vergleichsmöglichkeiten (Baseline)
- kommerzielle Monitoring-Tools



Migrations-Beispiel

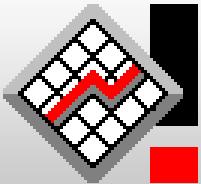
- Migration von SQL-Server-DBs von einem RZ-Anbieter "A" zu einem anderen RZ "I"
- Betrachtungsaspekt: Benutzerrechte
- diverse Hürden, Klippen, Traps etc. ...
- ...sowohl organisatorischer als auch technischer Art

- Die Aufgabe: Die Datenbanken "C" und "D" aus einer MS-SQL-Server-Instanz "O" sind zu migrieren.



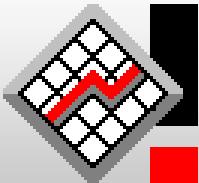
Migrations-Beispiel: Die Katastrophe beginnt

- Das RZ "A" liefert die entsprechenden Datenbanken "C" und "D" im MDF-/LDF-Format – so weit, so gut.
- Die Nachfrage
"Und was ist mit der MASTER-DB?"
wird beantwortet mit
"Die kann aus Gründen geistigen Eigentums nicht geliefert werden."
- Houston – wir haben ein Problem!



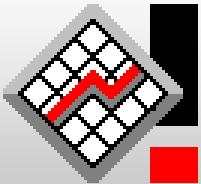
Server-Logins und Datenbank-User

- Datenbank-User und deren Rechte sind in den jeweiligen Datenbanken abgelegt
- Datenbank-User werden über SQL-Server-Logins authentifiziert
- zwei Arten von SQL-Server-Logins
 - windows-authentifiziert
 - sql-server-authentifiziert
- SQL-Server-Logins sind in der MASTER-Datenbank gespeichert



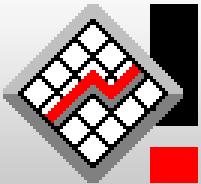
Migrations-Beispiel: Datenbank-User ohne SQL-Server-Logins

- zu den Datenbank-Usern fehlen die korrespondierenden SQL-Server-Logins
 - damit sind die Datenbank-User sogenannte Orphans (Waisen)
- außerdem:
 - die fehlenden SQL-Server-Logins könnten serverweite Berechtigungen oder Gruppenzugehörigkeiten haben
- zu lösende Probleme:
 - Erzeugen und Berechtigen der fehlenden SQL-Server-Logins
 - Verbinden der Logins mit den Datenbank-Usern



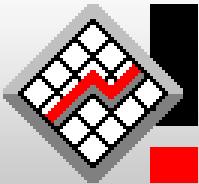
Transfer von Logins ohne MASTER-DB-Direktzugriff (I)

- "Logins könnt Ihr bekommen, die MASTER-DB nicht"
- Ausweg: ein Skript erzeugen, welches von der Quell-Instanz die login-spezifischen Informationen einsammelt...
 - Logins und deren Attribute
 - Zugehörigkeit zu Server-Rollen
- ...und gleich ausführbar liefert
 - ein SQL-Skript, daß ausführbaren SQL-Code erzeugt
 - Skript: ServerLogins_And_RoleMemberships.sql



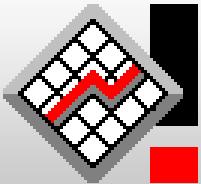
Transfer von Logins ohne MASTER-DB-Direktzugriff (II)

- damit sind die SQL-Logins erzeugt
- Einschränkungen der Demo
 - es sind weitere Properties der SQL-Logins einzubeziehen, z.B. "Default Language" u.a., aber das Prinzip bleibt
 - ein paar Logins sollte man wohl rauslassen (NT-AUTORITÄT\SYSTEM und so)
 - Besonderheiten für windows-basierte Logins
 - diese müssen natürlich windowsseitig existieren (im ActiveDirectory o.ä.)
 - ggf. benötigt man beim "CREATE LOGIN" eine Umbenennung des zugehörigen Domänen-Namens



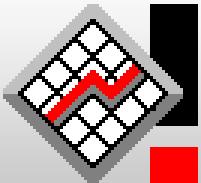
Reconnect der Datenbank-User mit den neuen Logins...

- ...geht nicht, jedenfalls nicht für windows-basierte Logins <schnief>
 - ist im MS-SQL-Server aus Sicherheitsgründen nicht vorgesehen
- Und nun?
- Die Datenbank-User
 - sichern (inklusive aller Eigenschaften)
 - löschen
 - auf Basis der neuen Logins neu erzeugen



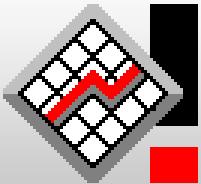
Datenbank-User sichern, löschen, neu erzeugen (V1)

- DROP USER ...
- CREATE USER ...
- EXEC sp_AddRoleMember ...
- set default schema
- Database-Permissions (GRANT, DENY ...)
- Object-Permissions (GRANT, DENY ...)
- Column-Permissions (GRANT, DENY ...)
- Schema-Permissions (GRANT, DENY ...)



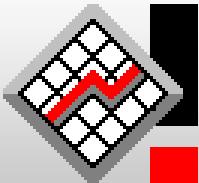
Datenbank-User sichern, löschen, neu erzeugen (V2)

- Schema-Ownership lösen
- DROP USER ...
- CREATE USER ...
- Schema-Ownership wieder herstellen
- EXEC sp_AddRoleMember ...
- set default schema
- Database-Permissions (GRANT, DENY ...)
- Object-Permissions (GRANT, DENY ...)
- Column-Permissions (GRANT, DENY ...)
- Schema-Permissions (GRANT, DENY ...)



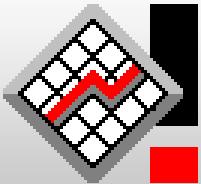
Datenbank-User sichern, löschen, neu erzeugen (V3)

- Schema-Ownership lösen
- DROP USER ... (alte Domäne)
- DROP USER ... (neue Domäne)
- CREATE USER ...
- Schema-Ownership wieder herstellen
- EXEC sp_AddRoleMember ...
- set default schema
- Database-Permissions (GRANT, DENY ...)
- Object-Permissions (GRANT, DENY ...)
- Column-Permissions (GRANT, DENY ...)
- Schema-Permissions (GRANT, DENY ...)



Datenbank-User sichern, löschen, neu erzeugen (V4)

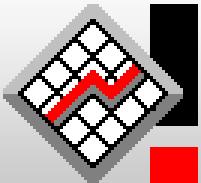
- Schema-Ownership lösen
- DROP USER ... (alte Domäne)
- DROP USER ... (neue Domäne)
- CREATE USER ...
- Schema-Ownership wieder herstellen
- EXEC sp_AddRoleMember ...
- set default schema
- Database-Permissions (GRANT, DENY ...)
- Object-Permissions (GRANT, DENY ...)
- Column-Permissions (GRANT, DENY ...)
- Schema-Permissions (GRANT, DENY ...)
- ggf. weitere Permissions



Migrations-Beispiel: Datenbank-User ohne SQL-Server-Logins

■ Fazit:

- eine spannende Angelegenheit
- die zudem durchaus aufwandsintensiv war



Empfehlung für ein Authentifizierungskonzept

■ "A-G-DL-P"-Konzept

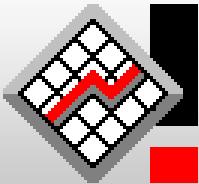
- Windows-Accounts...
- ...werden in g- diese wiederum in domänen-lokalen Windows-Gruppen zusammengefaßt
- und diese dann letztendlich im SQL-Server mit Permissions versehen
- **Also kein Einsatz von reinen SQL-Server-Logins!**

■ Nachteil

- einmaliger Definitionsaufwand

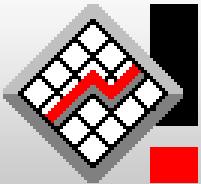
■ Vorteil

- einfaches Rechte-Management



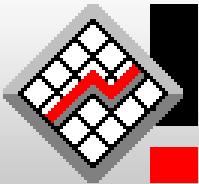
Dies & das: Kleinkram

- für "A-G-DL-P" müssen die beteiligten DCs korrekt konfiguriert sein bzgl.
 - Domain functional level
 - Forest functional level
- keine verschachtelten RoleMemberships
- Reconnect-Feature von DB-Users zu Server-Logins nicht für Windows-Authentifizierung!
- beim Detach werden ACLs auf Dateiebene verändert (alles weg außer dem Detach-User)



Dies & das: Orphansproblem wg. 2-stufiger Autorisierung

- Szenario "Windows-Authentifizierung"
- Stufe 1 "Server-Ebene"
 - Windows-Login gegen SQL-Server-Logins prüfen (explizit oder über Mitgliedschaft in AD-Gruppen)
- unabhängige Stufe 2 "Datenbank-Ebene"
 - Prüfen der Zugehörigkeit des authentifizierten Windows-Logins bzgl. expliziter oder impliziter (per AD-Gruppenmitgliedschaft) Berechtigung
 - **Problem:** Autorisierung wird auch erteilt, wenn der auf einer Windows-Gruppe basierende DB-User mit gar keinem Server-Login mehr verbunden ist!



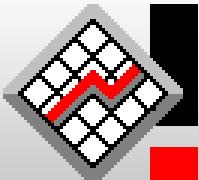
Loop über Datenbanken (I)

```
SQLQuery17.sql ...istrator (69)* UIT-Rechte_R...istrator (67)) SQLQuery16.sql ...istrator (68)* Berechtigu
1 DECLARE @xDatabaseName sysname
2 DECLARE xDatabases CURSOR SCROLL STATIC FOR
3     SELECT Name
4         FROM sys.Databases
5         ORDER BY Name
6     FOR READ ONLY
7
8 OPEN xDatabases
9 FETCH NEXT FROM xDatabases INTO @xDatabaseName
10
11 WHILE @@fetch_status = 0
12 BEGIN
13     PRINT 'Und jetzt tun wir irgendwas mit der DB: ' + @xDatabaseName
14     PRINT '...'
15     PRINT ''
16     FETCH NEXT FROM xDatabases INTO @xDatabaseName
17 END
18
19 CLOSE xDatabases
20 DEALLOCATE xDatabases
```

Messages

Und jetzt tun wir irgendwas mit der DB: AdventureWorks2008
...

Und jetzt tun wir irgendwas mit der DB: AdventureWorksDW2008
...



Loop über Datenbanken (II): Dynamisches SQL

```
SQLQuery17.sql ...istrator (69)* UIT-Rechte_R...istrator (67) SQLQuery16.sql .
11: WHILE @@fetch_status = 0
12: BEGIN
13:
14:   USE @xDatabaseName
15:   SELECT * FROM sys.database_principals
16:
17:   FETCH NEXT FROM xDatabases INTO @xDatabaseName
18: END
```

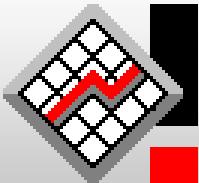
■ ← so geht es nicht

Messages

```
Msg 102, Level 15, State 1, Line 14
Incorrect syntax near '@xDatabaseName'.
```

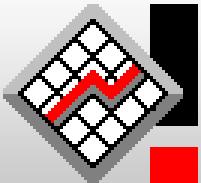
■ so dagegen schon→

```
SQLQuery17.sql ...istrator (69)* UIT-Rechte_R...istrator (67) SQLQuery16.sql .
11: DECLARE @nvcSQL nVarChar (max)
12: WHILE @@fetch_status = 0
13: BEGIN
14:
15:   SET @nvcSQL =
16:
17:   USE ' + @xDatabaseName + '
18:   SELECT * FROM sys.database_principals
19:
20:   EXEC sp_executesql @nvcSQL
21:
22:   FETCH NEXT FROM xDatabases INTO @xDatabaseName
23: END
```



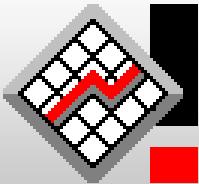
Loop über Datenbanken (III): Dynamisches SQL ohne Cursor

```
SQLQuery1.sql - (lo...Administrator (56))* X
1 DECLARE @nvcSql AS nvarchar( max ) = N''
2 SELECT
3     @nvcSql = @nvcSql +
4     '
5     USE [ ' + name + ' ]
6     SELECT DB_NAME(), * FROM .sys.database_principals
7     '
8     FROM sys.databases
9
10    PRINT @nvcSql
11    EXECUTE sp_executesql @nvcSql
12
```



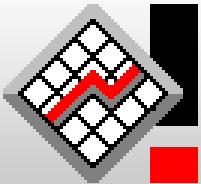
Loop über Datenbanken (IV): Unsupported StoredProcedure

```
SQLQuery2.sql - (lo...Administrator (52))* X
1  DECLARE @nvcSql AS nvarchar( 2000 ) = N''
2
3  SET @nvcSql =
4  '
5  USE [?]
6  SELECT DB_NAME(), * FROM .sys.database_principals
7
8  '
9  PRINT @nvcSql
10 EXECUTE sp_MSforeachdb @nvcSql
11'
```



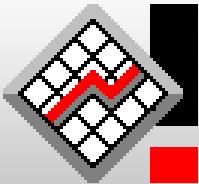
Dies & das: Kleinkram

- Ownership von Jobs kann störend sein...
...wenn man sie nicht hat ;-)
 - Ausweg:
 - aus dem Job ein CREATE-Skript erzeugen
 - in diesem dann die Ownerschaft und den Jobnamen anpassen, laufen lassen – fertig
- gern unterschätztes Sicherheits-Risiko sind Historien, Log-Files und Protokolle
 - SQL-Server-Agent-Historie
 - SQL-Server-Logs
 - Mailhistorie "msdb.sys.sysmail_*
 - Backuphistorie "msdb.sys.backup"



Impersonation

- Wechsel des Security-Kontextes
- diverse Befehle und Funktionen in diesem Umfeld
- übliche Architektur
 - User habe keinen Zugriff auf Tabellen, Views etc. ...
 - ...sondern nur Execute-Rechte auf bestimmten Stored Procedures, die ihrerseits Rechte auf die Daten haben

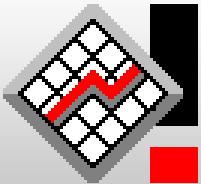


Security-Aspekte im SQL-Server-Agent

■ vier Authentifizierungs-Einflüsse

- bei TSQL-Steps
 - Advanced / Run as user
 - hier kann man einen Datenbank-User eintragen
- bei allen Steps
 - General / Run as
 - hier agiert man mit Proxy-Usern
- Owner des gesamten Jobs
 - General / Owner
- Konto, unter dem der SQL-Server-Agent läuft

■ viel Potential zur Verwirrung



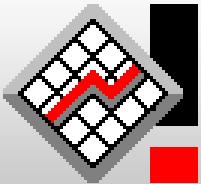
Security-Aspekte in speziellen Bereichen

■ MSDB

- diverse vordefinierte Rollen, z.B. für SQL-Server-Agent, SSIS u.ä.
- Administration hat manchmal ein paar Ecken und Kanten

■ SSIS Catalog/ SSIS DB

- paßwortgeschützter Key für verschlüsselte Daten
- intuitiv zu administrieren
- bestimmte Features gehen nur gut oder überhaupt mit "Windows Authentication"



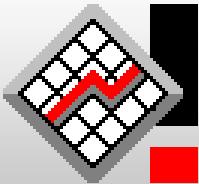
Verschlüsselung

■ Data Encryption

- Transparent Data Encryption (SQL2K8)
- Column-level Encryption (SQL5K5)
- Encrypting and Decrypting Data with the .NET Framework (unabhängig von SQL Server)
- Encrypting File Systems (W2K)
- BitLocker (W2K8R2)

■ Transport Level Encryption

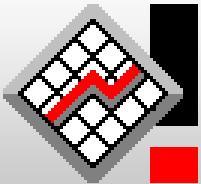
- IPsec (W2K)
- SSL (???)



Erweiterung des Horizonts

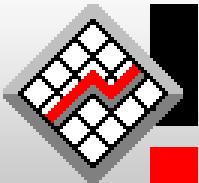
- Contained Database aus Security-Sicht:
 - pro: höhere Kapselung
 - con: zusätzliche Risiken bei inkorrektener Konfiguration

- Azure
 - ein paar Unterschiede zu "on premise"
 - zusätzlich:
 - Verschlüsselung der Verbindungen
 - Sicherung des Zugriffs

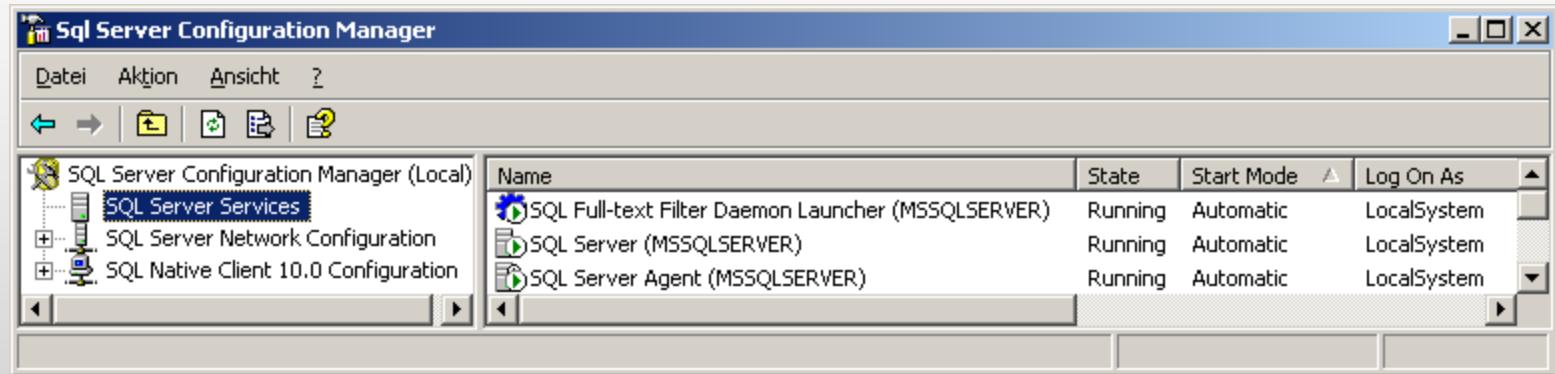


Installations-Überlegungen

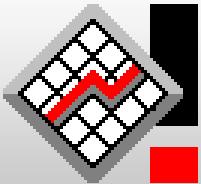
- Erhöhen der physikalischen Sicherheit
 - Personenzugang, Datensicherungen, sichere Räume, sichere Netzwerkzone
- Verwenden von Firewalls
- Isolieren von Diensten
 - separate Dienste-Konten
- Konfigurieren eines sicheren Dateisystems
- Deaktivieren von NetBIOS und Server Message Block
- siehe auch <http://msdn.microsoft.com/de-de/library/ms144228.aspx>



Etwas außer der Reihe: Die SSDE aus OS-Sicht (I)



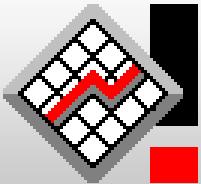
- Die vollständige SSDE läuft als drei Dienste:
 - SQL Server
 - SQL Server Agent
 - SQL Full-text Filter Daemon Launcher



Etwas außer der Reihe: Die SSDE aus OS-Sicht (II)

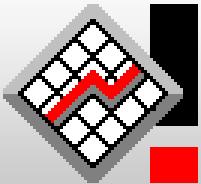
- "SQL Server"-Dienst
 - die eigentliche SSDE
- "SQL Server Agent"-Dienst
 - der Scheduling-Dienst
- "SQL Full-text Filter Daemon Launcher"-Dienst
 - die Volltext-Indizierung

Diese Dienste sollten an Stelle von "LocalSystem" mit dezidierten Accounts betrieben werden (die genau zugeschnittene Rechte haben)!



Berechtigungs-Aspekte des "SQL Server"-Dienst

- Konfiguration als klassisches Service-Konto
 - kein interaktives Login etc.
- neben dem eigentlichen Laufenlassen des Dienstes:
 - Zugriff auf Verzeichnisse für
 - Datenbank-Dateien (MDF, NDF, LDF)
 - Backup-Dateien
 - Protokolle und Errorlogs
 - alle sonstigen zur Installation gehörenden Dateien



Berechtigungs-Aspekte weiterer Dienste

■ "SQL Server Agent"-Dienst

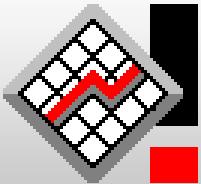
- benötigt einige windows-spezifische Permissions (LogonAsService u.ä.)
- das Security-Konzept des "SQL Server Agent" wird gesondert betrachtet

■ "SQL Full-text Filter Daemon Launcher"-Dienst

- benötigt einige windows-spezifische Permissions (LogonAsService u.ä.)

■ SQL-Mail / Database Mail

- Mailversand-Konfiguration

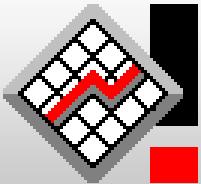


Service-Accounts

■ mögliche Account-Typen

- Local User Account
- Local Service Account (dedicated)
- Local System Account
- Network Service Account
- Domain Account
- Managed Service Accounts
- Virtual Accounts

■ hat alles seine Vor- und Nachteile

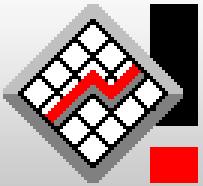


Zusammenfassung

- Da hilft Kollege Theodor Fontane:

Es ist ein weites Feld...

- Viel Spaß beim Experimentieren!



Danke für Ihre
Aufmerksamkeit!