Session D-SECU

Sicherheit geht alle an - Security in der SSDE

Sebastian Flucke

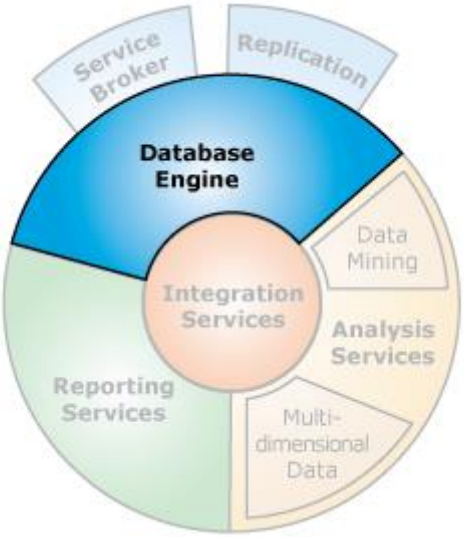
Vorbemerkung

Die Berechtigungsproblematik innerhalb der MS-SQL-Server-Datenbankengine ist sehr vielfältig - was man spätestens merkt, wenn man seine frisch entwickelte Applikation erstmals nicht unter dem allmächtigen Administrator-Konto laufen läßt ☺! Die Session gibt einen allgemeinen Überblick über die security-relevanten Aspekte der Datenbank-Engine – die entscheidenden Stichpunkte hier sind Principals, Securables und Permissions. Weiterhin werden Codebeispiele zum Ermitteln, Konservieren und Restaurieren von Security-Informationen gezeigt (z. B. für Migrationsszenarien). Außerdem gibt es Einblicke in diverse Fallen und leider auch Lücken beim Management von Server-Logins und DB-Usern sowie zum Einsatz von Impersonation in Stored Procedures.

> Sicherheit nervt beim Entwickeln und kostet Zeit sowie Geld...
> ...ist aber unabdingbar, sollte nicht beim Anwenden hindern und benötigt deshalb entsprechende Konzepte

Security-Aspekte in der SQL-Server-Database-Engine SSDE

Überblick

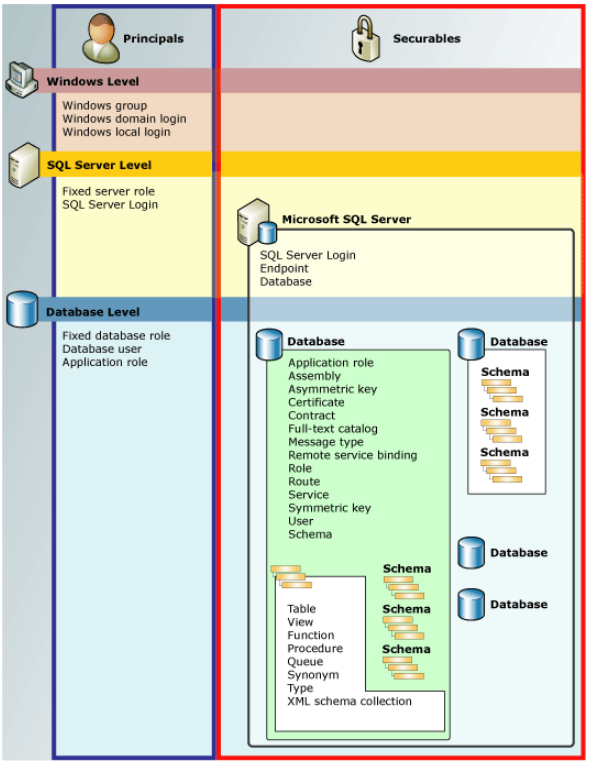


Im Zusammenhang mit der SSDE existieren diverse security-relevante Aspekte

- Dies beginnt bei dem umgebenden Betriebssystem, den dort definierten Service-Konten usw.
- Die nächste Ebene sind die eigentlichen SQL-Server-Instanzen...
  - ...innerhalb derer die SQL-Server-Datenbanken logieren.
    - In den Datenbanken findet man dann diverse Objekte wie Schemas, Tabellen, Views, Prozeduren usw.
  - Außerdem existieren diverse instanzbezogene Verwaltungsaspekte, bei denen die Security eine Rolle spielt
    - SQL-Server-Agent
    - Database Mail
    - Replikationen, Linked Server usw.

Begriffsklärung: Principal, Securable und Permission

Die Haupt-Security-Kategorien in der SSDE sind die Begriffe **Principal**, **Securable** und **Permission**, hier ein grober - und nicht vollständiger - Überblick über diese drei Kategorien:



Hier zunächst eine kurze Begriffsklärung:

- Permission
  - die Erlaubnis, etwas zu tun
  - Beispiele:
    - CREATE DATABASE, VIEW SERVER STATE, ALTER ANY LINKED SERVER, TAKE OWNERSHIP, EXECUTE, VIEW DEFINITION...
- Principal
  - Anforderer von Ressourcen
  - Beispiele:
    - SQL-Server-Login, Database-User, Application-Role
- Securable
  - Ressource, für die eine Permission erforderlich ist
  - Beispiele:
    - DATABASE, SCHEMA, ROLE...

Also:

- Ein Principal bekommt Permissions, um auf Securables zugreifen zu dürfen (oder eben auch nicht).

Übrigens:

- Principals können auch Securables sein – und umgekehrt:
  - Eine Datenbank-Rolle ist ein Principal, ihr werden Permissions bzgl. bestimmter Datenbank-Objekte eingeräumt.
  - Andererseits ist eine Datenbank-Rolle auch ein Securable, denn ein Datenbank-User kann die Permission besitzen, diese Datenbank-Rolle zu administrieren.

Permissions

Überblick

Die Permissions lassen sich in verschiedene Kategorien einteilen:

- daten-bezogen
  - Diese Permissions wirken sich auf den Zugriff auf die eigentlichen Daten in den relationalen Tabellen aus.
  - Permission-Beispiele: SELECT, UPDATE, INSERT, DELETE
- programmmodul-bezogen
  - Hier geht es um Rechte auf StoredProcedures u.ä.Programm-Module.
  - Permission-Beispiele: EXECUTE
- allgemeines Objekt-Management
  - Für die Verwaltung diverser Arten von Objekten gibt es diverse allgemeine Permissions.
  - Permission-Beispiele: VIEW DEFINITION, ALTER, TAKE OWNERSHIP, CONTROL
- für spezielle Objekte
  - Bei speziellen Objekt-Arten gibt es Permissions, die nur für diese eine Art relevant sind.
  - Permission-Beispiele: VIEW CHANGE TRACKING, REFERENCES, RECEIVE

Die Permission-Hierarchie

Neben der Zuordnung von Permissions zu Securables durch Rollen-Mitgliedschaften oder explizite Permission-Zuweisungen (siehe weiter unten) ist außerdem eine fest definierte mehrfache Permission-Hierarchie relevant.

Dies soll am Beispiel der nachfolgenden – nicht vollständigen – Liste der Permissions-Hierarchie erläutert werden.

	class_desc	**permission_name**	**type**	**covering_ permission _ name**	**parent_ class_desc**	**parent_covering_ permission_name**
	...					
	...	...	...	...	...	...
1	DATABASE	ALTER	AL	CONTROL	SERVER	ALTER ANY DATABASE
2	DATABASE	CONTROL	CL		SERVER	CONTROL SERVER
3	DATABASE	ALTER ANY DATABASE AUDIT	ALDA	ALTER	SERVER	ALTER ANY SERVER AUDIT
4	DATABASE	VIEW DATABASE STATE	VWDS	CONTROL	SERVER	VIEW SERVER STATE
5	DATABASE	VIEW DEFINITION	VW	CONTROL	SERVER	VIEW ANY DEFINITION
6	DATABASE	TAKE OWNERSHIP	TO	CONTROL	SERVER	CONTROL SERVER
7	DATABASE	ALTER	AL	CONTROL	SERVER	ALTER ANY DATABASE
8	DATABASE	CONTROL	CL		SERVER	CONTROL SERVER
	...	...	...	...	...	...
10	SERVER	ALTER ANY SERVER AUDIT	ALAA	CONTROL SERVER		
11	SERVER	CONTROL SERVER	CL			

	class_desc	permission_name	type	covering_ permission _ name	parent_ class_desc	parent_covering_ permission_name
33	...					

Hier nun eine kurze Beispiel-Erläuterung der Hierarchie A (Vererbung von Permission zu Permission):

- Die Permission in Zeile 3 beinhaltet das "ALTER ANY DATABASE AUDIT"-Recht auf eine Datenbank:

	class_desc	permission_ name	type
3	DATABASE	ALTER ANY DATABASE AUDIT	ALDA

- Die Eintragung "ALTER" in der Spalte "covering_ permission_ name" bedeutet:
  Wenn jemand das "ALTER"-Recht auf einer Datenbank hat, dann ist das "ALTER ANY DATABASE AUDIT"-Recht mit eingeschlossen:

	class_desc	permission_ name	type	covering_ permission _ name
3	DATABASE	ALTER ANY DATABASE AUDIT	ALDA	ALTER

- Das "ALTER"-Recht wiederum findet sich in Zeile 1, und von dort wird mit dem "CONTROL"-Recht auf Zeile 2 verwiesen:

	class_desc	permission_ name	type	covering_ permission _ name
1	DATABASE	ALTER	AL	CONTROL
2	DATABASE	CONTROL	CL	
3	DATABASE	ALTER ANY DATABASE AUDIT	ALDA	ALTER

- Das bedeutet: Wenn jemand auf einer Datenbank das Recht "CONTROL" oder "ALTER" oder "ALTER ANY DATABASE AUDIT", dann kann er Database-Audits verändern.

Desweiteren gibt es noch eine Hierarchie B (Vererbung über Objekt-Hierarchien), auch hierzu ein kurzes Beispiel:

- Die Permission in Zeile 3 beinhaltet das "ALTER ANY DATABASE AUDIT"-Recht auf eine Datenbank:

	class_desc	permission_ name	type
3	DATABASE	ALTER ANY DATABASE AUDIT	ALDA

- Die Eintragungen "SERVER" in der Spalte "parent_class_desc" sowie "ALTER ANY SERVER AUDIT" in der Spalte "parent_covering_ permission_name" bedeutet:

Wenn jemand das "ALTER ANY SERVER AUDIT"-Recht auf Ebene des "SERVER" hat, dann ist das "ALTER ANY DATABASE AUDIT"-Recht mit eingeschlossen:

	class_desc	permission_name	type		parent_class_desc	parent_covering_permission_name
3	DATABASE	ALTER ANY DATABASE AUDIT	ALDA		SERVER	ALTER ANY SERVER AUDIT

- Auch dieses SERVER-Recht seinerseits kann wieder einer Hierarchie entstammen: Das "ALTER ANY SERVER AUDIT"-Recht hat man entweder explizit (Zeile 11) oder wenn man das "CONTROL SERVER"-Recht hat (Zeile 11)

	class_desc	permission_name	type		parent_class_desc	parent_covering_permission_name
3	DATABASE	ALTER ANY DATABASE AUDIT	ALDA		SERVER	ALTER ANY SERVER AUDIT
	...	...	...	...	...	...
10	SERVER	ALTER ANY SERVER AUDIT	ALAA	CONTROL SERVER		
11	SERVER	CONTROL SERVER	CL			

Auflösen der impliziten Permissions-Hierarchie:

- Bei Bedarf kann man mit der Funktion "ImplyingPermissions" die impliziten Permissions-Hierarchien (siehe weiter oben im Kapitel "Die Permission-Hierarchie") auflösen und auf die zugrundeliegenden Basis-Permissions zurückführen.
- Die Funktion "ImplyingPermissions" existiert standardmäßig nicht als aufrufbereite Funktion im MS SQL Server, sondern nur als Quelltext in BooksOnline:
  "SQL Server 2008 Books Online > Database Engine > Security and Protection (Database Engine) > Identity and Access Control (Database Engine) > Permissions (Database Engine) > Covering/Implied Permissions (Database Engine)"
- Beispiel 1
  - ```
    SELECT * FROM
       dbo.ImplyingPermissions ( 'database', 'view definition' )
    ```

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

  - Dies bedeutet für eine mögliche Herkunft des "VIEW DEFINITION"-Rechts:
    - explizite Vergabe auf Datenbank-Ebene
    - implizite Vergabe durch das "CONTROL"-Recht auf Datenbank-Ebene
    - implizite Vergabe durch das "VIEW ANY DEFINITION"-Recht auf Server-Ebene
    - implizite Vergabe durch das "CONTROL SERVER"-Recht auf Server-Ebene
    ```

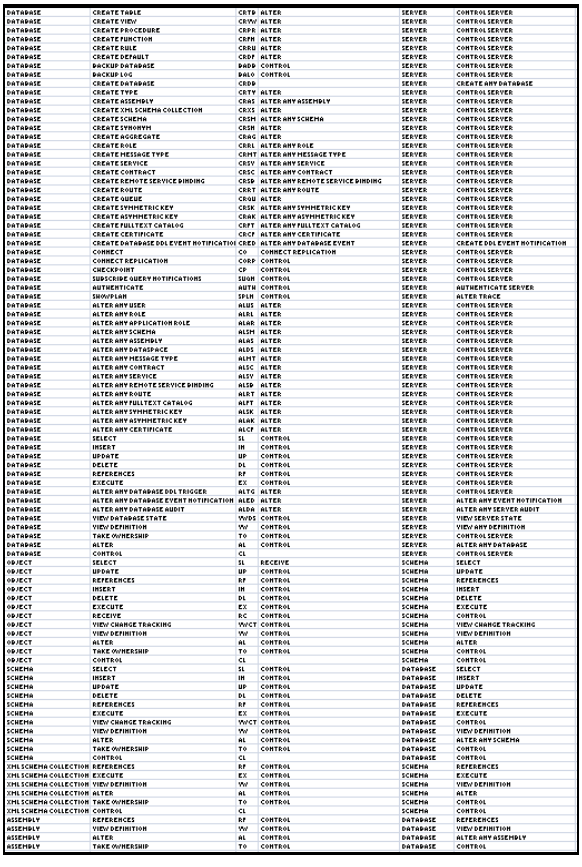
- ```
  SELECT * FROM
      dbo.ImplyingPermissions ( 'object', 'update' )
  ```

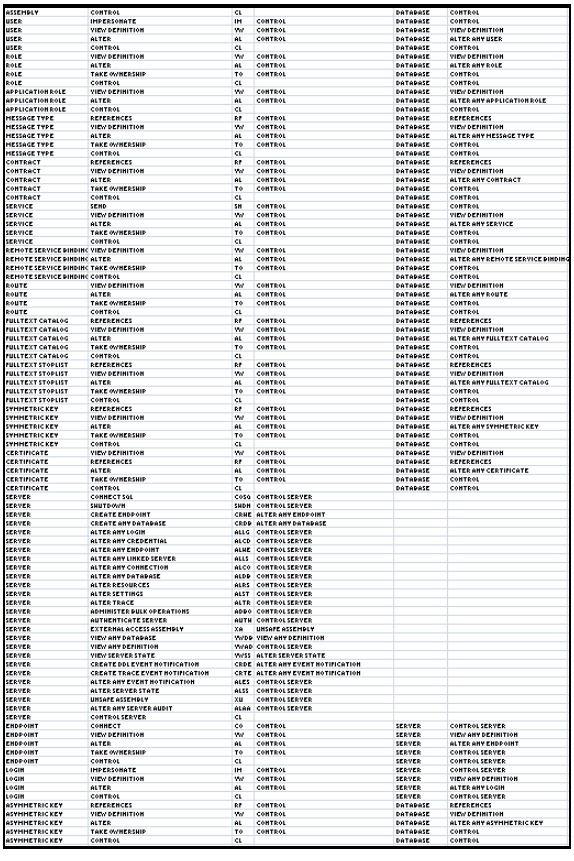
permname	class	height	rank
UPDATE	OBJECT	0	0
CONTROL	OBJECT	0	1
UPDATE	SCHEMA	1	1
CONTROL	SCHEMA	1	2
UPDATE	DATAB...	2	2
CONTROL	DATAB...	2	3
CONTROL SERVER	SERVER	3	4

- Dies bedeutet für eine mögliche Herkunft des "UPDATE"-Rechts:
  - explizite Vergabe auf Objekt-Ebene
  - implizite Vergabe durch das "CONTROL"-Recht auf Objekt-Ebene
  - implizite Vergabe durch das "UPDATE"-Recht auf Schema-Ebene
  - implizite Vergabe durch das "CONTROL"-Recht auf Schema-Ebene
  - implizite Vergabe durch das "UPDATE"-Recht auf Datenbank-Ebene
  - implizite Vergabe durch das "CONTROL"-Recht auf Datenbank-Ebene
  - implizite Vergabe durch das "CONTROL SERVER"-Recht auf Server-Ebene

NB: Die vollständige Liste der Permission-Hierarchie kann wie folgt abgerufen werden:

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





Security auf Ebene der SQL-Server-Instanzen

Securables

Auf Instanz-Ebene existieren folgende Securables:
```

- Datenbanken
  - `SELECT * FROM sys.databases`
- Logins
  - `SELECT * FROM sys.server_principals`
- der eigene Server
- (Endpoints)
  - `SELECT * FROM sys.endpoints`
  - Wird nur der Vollständigkeit halber erwähnt und nicht hier weiter betrachtet.

Principals

In jeder SQL-Server-Instanz existieren diese frei definierbaren Principals:

```
SELECT * FROM sys.server_principals
```

- Logins
  - `SELECT * FROM sys.server_principals`
    `WHERE [type] IN ( 'S', 'U', 'G', 'C', 'K' )`
  - Logins gewähren den grundsätzlichen Zugriff auf die SQL-Server-Instanzen und existieren als unterschiedliche Typen
    - SQL-Server-Login
      - Diese Logins werden inklusive Paßwort innerhalb der SQL-Server-Instanz definiert (Stichwort "SQL Server Authentication mode")
      - `SELECT * FROM sys.server_principals`
        `WHERE [type] = 'S'`
    - Windows-User-Login
      - Hierbei erfolgt die Definition und Authentifizierung gegen das Windows-Betriebssystem (Active Directory o.ä., Stichwort "Windows Authentication mode")
      - `SELECT * FROM sys.server_principals`
        `WHERE [type] = 'U'`
    - Windows-Gruppen-Login
      - Analog dem Windows-User-Login erfolgt die Definition und Authentifizierung gegen das Windows-Betriebssystem, allerdings indirekt über eine entsprechende Gruppenzugehörigkeit im Active Directory o.ä.
      - Wichtig: Alle SQL-Server-internen Protokollierungen, neu vergebene Ownerships usw. agieren nicht mit der authentifizierenden Windows-Gruppe, sondern mit dem verwendeten Windows-User-Login – man kann sich also in diesem Fall nicht hinter einem anonymen Gruppen-Login verstecken, sondern man hinterläßt trotzdem personalisierte Spuren.
      - `SELECT * FROM sys.server_principals`
        `WHERE [type] = 'G'`
    - (Login mapped to a certificate) / (Login mapped to an asymmetric key)
      - Wird nur der Vollständigkeit halber erwähnt und nicht hier weiter betrachtet.
      - `SELECT * FROM sys.server_principals`
        `WHERE [type] IN ( 'C', 'K' )`

Außerdem gibt es folgende fest definierte Principals:

- fixed Server Roles
  - `EXECUTE sp_helpsrvrole`

	ServerRole	Description
1	sysadmin	System Administrators
2	securityadmin	Security Administrators
3	serveradmin	Server Administrators
4	setupadmin	Setup Administrators
5	processadmin	Process Administrators
6	diskadmin	Disk Administrators
7	dbcreator	Database Creators
8	bulkadmin	Bulk Insert Administrators

- Oder auch
  - ```
    SELECT * FROM sys.server_principals
                  WHERE [type] = 'R'
    ```
- Diese Server-Rollen beinhalten feste Kombinationen von Securables und Permissions, die im Wesentlichen für administrative Zwecke vorgesehen sind.
- Organisatorisch kann man sie als eine Art Gruppe verstehen, denn den Server-Rollen können Mitglieder zugewiesen werden.
- Server-Rollen können nicht ineinander geschachtelt werden.
    ```

Einzel-Permissions

Im MS-SQL-Server können zwar keine selbstdefinierten Server-Rollen angelegt werden, allerdings kann man Principals schon unabhängig von den fest definierten Server-Rollen weitere Permissions gewähren (oder auch entziehen).

Die möglichen Kombinationen von Securables und Permissions auf Instanz-Ebene können wie folgt abgefragt werden:

```
SELECT          DISTINCT          class_desc,          permission_name
      FROM                              sys.fn_builtin_permissions('')
      WHERE     parent_class_desc     IN     (     '',     'SERVER'     )
      ORDER BY 1, 2
```

Securable	Permission
DATABASE	ALTER
DATABASE	ALTER ANY APPLICATION ROLE
DATABASE	ALTER ANY ASSEMBLY
DATABASE	ALTER ANY ASYMMETRIC KEY
DATABASE	ALTER ANY CERTIFICATE
DATABASE	ALTER ANY CONTRACT
DATABASE	ALTER ANY DATABASE AUDIT
DATABASE	ALTER ANY DATABASE DDL TRIGGER
DATABASE	ALTER ANY DATABASE EVENT NOTIFICATION
DATABASE	ALTER ANY DATASPACE
DATABASE	ALTER ANY FULLTEXT CATALOG
DATABASE	ALTER ANY MESSAGE TYPE
DATABASE	ALTER ANY REMOTE SERVICE BINDING
DATABASE	ALTER ANY ROLE
DATABASE	ALTER ANY ROUTE
DATABASE	ALTER ANY SCHEMA
DATABASE	ALTER ANY SERVICE
DATABASE	ALTER ANY SYMMETRIC KEY
DATABASE	ALTER ANY USER
DATABASE	AUTHENTICATE
DATABASE	BACKUP DATABASE
DATABASE	BACKUP LOG
DATABASE	CHECKPOINT
DATABASE	CONNECT
DATABASE	CONNECT REPLICATION
DATABASE	CONTROL
DATABASE	CREATE AGGREGATE
DATABASE	CREATE ASSEMBLY
DATABASE	CREATE ASYMMETRIC KEY
DATABASE	CREATE CERTIFICATE
DATABASE	CREATE CONTRACT
DATABASE	CREATE DATABASE

Securable	Permission
DATABASE	CREATE DATABASE DDL EVENT NOTIFICATION
DATABASE	CREATE DEFAULT
DATABASE	CREATE FULLTEXT CATALOG
DATABASE	CREATE FUNCTION
DATABASE	CREATE MESSAGE TYPE
DATABASE	CREATE PROCEDURE
DATABASE	CREATE QUEUE
DATABASE	CREATE REMOTE SERVICE BINDING
DATABASE	CREATE ROLE
DATABASE	CREATE ROUTE
DATABASE	CREATE RULE
DATABASE	CREATE SCHEMA
DATABASE	CREATE SERVICE
DATABASE	CREATE SYMMETRIC KEY
DATABASE	CREATE SYNONYM
DATABASE	CREATE TABLE
DATABASE	CREATE TYPE
DATABASE	CREATE VIEW
DATABASE	REFERENCES
DATABASE	SELECT
DATABASE	SHOWPLAN
DATABASE	SUBSCRIBE QUERY NOTIFICATIONS
DATABASE	TAKE OWNERSHIP
DATABASE	UPDATE
DATABASE	VIEW DATABASE STATE
DATABASE	VIEW DEFINITION
ENDPOINT	ALTER
ENDPOINT	CONNECT
ENDPOINT	CONTROL
ENDPOINT	TAKE OWNERSHIP
ENDPOINT	VIEW DEFINITION

Securable	Permission
LOGIN	ALTER
LOGIN	CONTROL
LOGIN	IMPERSONATE
LOGIN	VIEW DEFINITION
SERVER	ADMINISTER BULK OPERATIONS
SERVER	ALTER ANY CONNECTION
SERVER	ALTER ANY CREDENTIAL
SERVER	ALTER ANY DATABASE
SERVER	ALTER ANY ENDPOINT
SERVER	ALTER ANY EVENT NOTIFICATION
SERVER	ALTER ANY LINKED SERVER
SERVER	ALTER ANY LOGIN
SERVER	ALTER ANY SERVER AUDIT
SERVER	ALTER RESOURCES
SERVER	ALTER SERVER STATE
SERVER	ALTER SETTINGS
SERVER	ALTER TRACE
SERVER	AUTHENTICATE SERVER
SERVER	CONNECT SQL
SERVER	CONTROL SERVER
SERVER	CREATE ANY DATABASE
SERVER	CREATE DDL EVENT NOTIFICATION
SERVER	CREATE ENDPOINT
SERVER	CREATE TRACE EVENT NOTIFICATION
SERVER	EXTERNAL ACCESS ASSEMBLY
SERVER	SHUTDOWN
SERVER	UNSAFE ASSEMBLY
SERVER	VIEW ANY DATABASE
SERVER	VIEW ANY DEFINITION
SERVER	VIEW SERVER STATE

Security auf Ebene der SQL-Server-Datenbanken

Principals

Folgende Principals stehen auf Datenbank-Ebene zur Verfügung:

```
SELECT * FROM sys.database_principals
```

- Database User
  - `SELECT * FROM sys.database_principals WHERE [type] IN ( 'S', 'U', 'G', 'C', 'K' )`
  - basierend auf diesen Login-Typen:
    - SQL-Server-Login
      - `SELECT * FROM sys.database_principals WHERE [type] = 'S'`
    - Windows-User-Login
      - `SELECT * FROM sys.database_principals WHERE [type] = 'U'`
    - Windows-Gruppen-Login
      - `SELECT * FROM sys.database_principals WHERE [type] = 'G'`
    - (Login mapped to a certificate)
      - `SELECT * FROM sys.database_principals WHERE [type] = 'C'`
    - (Login mapped to an asymmetric key)
      - `SELECT * FROM sys.database_principals WHERE [type] = 'K'`

Außerdem gibt es folgende fest definierte Database-Principals:

- Database Roles

```
SELECT * FROM sys.database_principals
                    WHERE [type] IN ( 'R', 'A' )
```

  - Fixed Database Roles
    - `EXECUTE sp_helpdbfixedrole`

	DbFixedRole	Description
1	db_owner	DB Owners
2	db_accessadmin	DB Access Administrators
3	db_securityadmin	DB Security Administrators
4	db_ddladmin	DB DDL Administrators
5	db_backupoperator	DB Backup Operator
6	db_datareader	DB Data Reader
7	db_datawriter	DB Data Writer
8	db_denydatareader	DB Deny Data Reader
9	db_denydatawriter	DB Deny Data Writer

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

    - Diese Datenbank-Rollen beinhalten feste Kombinationen von Securables und Permissions, die im Wesentlichen für administrative Zwecke vorgesehen sind.
    - Organisatorisch kann man sie als eine Art Gruppe verstehen, denn den Datenbank-Rollen können Mitglieder zugewiesen werden.

> Fixed Database Roles können nicht ineinander geschachtelt werden!

  - (non fixed) Database Roles

- ```
  EXECUTE sp_helprole
  SELECT * FROM sys.database_principals
        WHERE [type] = 'R' AND is_fixed_role = 0
  ```
- Im Gegensatz zu den festen Datenbank-Rollen sind die (non fixed) Database Roles frei definierbar:
  - bzgl. der zugeordneten Permissions
  - bzgl. der Mitglieder
- NB: (non fixed) Database Roles können ineinander geschachtelt werden und auch Mitglied in fixed Database Roles sein.

- Application Roles
  - ```
    EXECUTE sp_helprole
    SELECT * FROM sys.database_principals
          WHERE [type] = 'A'
    ```
  - Application Roles haben einen Sonderstatus – siehe weiter unten zum Thema "Impersonation"!

Securables

Innerhalb von Datenbanken stehen folgende Securables zur Verfügung:

- Schemas
  - `SELECT * FROM sys.schemas`
  - Innerhalb der Schemas befinden sich die Objects:
    - `SELECT * FROM sys.all_objects`
    - Objekte können von verschiedenem Typ sein

Typ	Bezeichnung
AF	Aggregate function (CLR)
C	CHECK constraint
D	DEFAULT (constraint or stand-alone)
F	FOREIGN KEY constraint
FN	SQL scalar function
FS	Assembly (CLR) scalar-function
FT	Assembly (CLR) table-valued function
IF	SQL inline table-valued function
IT	Internal table
P	SQL Stored Procedure
PC	Assembly (CLR) stored-procedure
PG	Plan guide
PK	PRIMARY KEY constraint

Typ	Bezeichnung
R	Rule (old-style, stand-alone)
RF	Replication-filter-procedure
S	System base table
SN	Synonym
SQ	Service queue
TA	Assembly (CLR) DML trigger
TF	SQL table-valued-function
TR	SQL DML trigger
U	Table (user-defined)
UQ	UNIQUE constraint
V	View
X	Extended stored procedure

    - User-Objekte findet man unter
      - `SELECT * FROM sys.objects`
    - System-Objekte findet man unter
      - `SELECT * FROM sys.system_objects`
    - Tables
      - User-Tables

- `SELECT * FROM sys.tables`
- Columns
  - ```
    SELECT
        O.name, C.*
        FROM sys.all_columns C
        INNER JOIN sys.all_objects O
              ON C.object_id = O.object_id
        INNER JOIN sys.tables T
              ON C.object_id = T.object_id
        ORDER BY O.name, C.column_id
    ```
- System-Tables
  - `SELECT * FROM sys.all_objects WHERE [type] = 'S'`
  - Columns
    - ```
      SELECT
          O.name, C.*
          FROM sys.all_columns C
          INNER JOIN sys.all_objects O
                ON C.object_id = O.object_id
          WHERE [type] = 'S'
          ORDER BY O.name, C.column_id
      ```
- internal Tables
  - `SELECT * FROM sys.internal_tables`
  - Columns
    - ```
      SELECT
          O.name, C.*
          FROM sys.all_columns C
          INNER JOIN sys.all_objects O
                ON C.object_id = O.object_id
          INNER JOIN sys.internal_tables T
                ON C.object_id = T.object_id
          ORDER BY O.name, C.column_id
      ```
- Views
  - User-Views
    - `SELECT * FROM sys.views`
    - Columns
      - ```
        SELECT
            O.name, C.*
            FROM sys.all_columns C
            INNER JOIN sys.all_objects O
                  ON C.object_id = O.object_id
            INNER JOIN sys.views V
                  ON C.object_id = V.object_id
            ORDER BY O.name, C.column_id
        ```
  - System-Views
    - `SELECT * FROM sys.system_views`
    - Columns
      - ```
        SELECT
            O.name, C.*
            FROM sys.all_columns C
            INNER JOIN sys.all_objects O
                  ON C.object_id = O.object_id
            INNER JOIN sys.system_views V
                  ON C.object_id = V.object_id
            ORDER BY O.name, C.column_id
        ```
  - alle Views
    - `SELECT * FROM sys.all_views`
        ```

- Columns
  - ```
    SELECT
        O.name, C.*
        FROM sys.all_columns C
        INNER JOIN sys.all_objects O
              ON C.object_id = O.object_id
        INNER JOIN sys.all_views V
              ON C.object_id = V.object_id
        ORDER BY O.name, C.column_id
    ```

- SQL-Module
  - SQL-Module sind quelltextbasierte Objekte (wie z.B. Stored Procedures)
  - SQL-Module, die ein Tabelle als Ergebnis liefern

Typ	Bezeichnung
IF	SQL inline table-valued function
TF	SQL table-valued-function

  - ```
    SELECT
        O.name, O.[type], SM.*
        FROM sys.all_objects O
        INNER JOIN sys.all_sql_modules SM
              ON O.object_id = SM.object_id
        WHERE [type] IN ( 'IF', 'TF' )
        ORDER BY O.name
    ```

  - Columns
    - ```
      SELECT
          O.name, C.*
          FROM sys.all_columns C
          INNER JOIN sys.all_objects O
                ON C.object_id = O.object_id
          WHERE [type] IN ( 'IF', 'TF' )
          ORDER BY O.name, C.column_id
      ```

  - sonstige SQL-Module

Typ	Bezeichnung
D	DEFAULT (constraint or stand-alone)
FN	SQL scalar function
P	SQL Stored Procedure
PC	Assembly (CLR) stored-procedure
R	Rule (old-style, stand-alone)
RF	Replication-filter-procedure
TR	SQL DML trigger

    - ```
      SELECT
          O.name, O.[type], SM.*
          FROM sys.all_objects O
          INNER JOIN sys.all_sql_modules SM
                ON O.object_id = SM.object_id
          WHERE [type] IN
                ( 'D', 'FN', 'P', 'PC', 'R', 'RF', 'TR' )
          ORDER BY O.name
      ```

- sonstige Objekte
  - sonstige Objekte, die ein Tabelle als Ergebnis liefern

Typ	Bezeichnung
FT	Assembly (CLR) table-valued

Typ	Bezeichnung
	function

- ```
  SELECT
        O.*
        FROM sys.all_objects O
        WHERE [type] = 'FT'
        ORDER BY O.name
  ```
- Columns
  - ```
    SELECT
        O.name, C.*
        FROM sys.all_columns C
        INNER JOIN sys.all_objects O
              ON C.object_id = O.object_id
        WHERE [type] = 'FT'
        ORDER BY O.name, C.column_id
    ```
- andere sonstige Objekte

Typ	Bezeichnung
AF	Aggregate function (CLR)
C	CHECK constraint
F	FOREIGN KEY constraint
FS	Assembly (CLR) scalar-function
PG	Plan guide
PK	PRIMARY KEY constraint
SN	Synonym
SQ	Service queue
TA	Assembly (CLR) DML trigger
UQ	UNIQUE constraint
X	Extended stored procedure

  - ```
    SELECT
        O.*
        FROM sys.all_objects O
        WHERE [type] IN
              ( 'AF', 'C', 'F', 'FS', 'PG', 'PK',
                'SN', 'SQ', 'TA', 'UQ', 'X' )
        ORDER BY O.name
    ```
  - Zu diesen Objekten ist Folgendes zu bemerken:
    - Nicht alle dieser Objekte sind Securables, für die explizit Permissions vergeben werden können.
    - Die Detaildefinitionen dieser Objekte wird an dieser Stelle nicht näher betrachtet.
- Innerhalb der Schemas befinden sich außerdem die "Types" und die "XML Schema Collections":
  - Diese Securables werden hier nicht vertiefend betrachtet.
- Auch Database Principals sind Securables (siehe weiter oben).
- Sonstige Securables
  - Assembly, Message Type, Route, Service, Remote Service Binding, Fulltext Catalog, Certificate, Asymmetric Key, Symmetric Key, Contract
    ```

Einzel-Permissions

Innerhalb von Datenbanken kann man den Principals unabhängig von den fest definierten Datenbank-Rollen weitere Permissions gewähren (oder auch entziehen).

Die möglichen Kombinationen von Securables und Permissions auf Instanz-Ebene können wie folgt abgefragt werden:

```
SELECT DISTINCT class_desc, permission_name
    FROM sys.fn_builtin_permissions('')
    WHERE parent_class_desc NOT IN ( '', 'SERVER' )
    ORDER BY 1, 2
```

Securable	Permission
APPLICATION ROLE	ALTER
APPLICATION ROLE	CONTROL
APPLICATION ROLE	VIEW DEFINITION
ASSEMBLY	ALTER
ASSEMBLY	CONTROL
ASSEMBLY	REFERENCES
ASSEMBLY	TAKE OWNERSHIP
ASSEMBLY	VIEW DEFINITION
ASYMMETRIC KEY	ALTER
ASYMMETRIC KEY	CONTROL
ASYMMETRIC KEY	REFERENCES
ASYMMETRIC KEY	TAKE OWNERSHIP
ASYMMETRIC KEY	VIEW DEFINITION
CERTIFICATE	ALTER
CERTIFICATE	CONTROL
CERTIFICATE	REFERENCES
CERTIFICATE	TAKE OWNERSHIP
CERTIFICATE	VIEW DEFINITION
CONTRACT	ALTER
CONTRACT	CONTROL
CONTRACT	REFERENCES
CONTRACT	TAKE OWNERSHIP
CONTRACT	VIEW DEFINITION
FULLTEXT CATALOG	ALTER
FULLTEXT CATALOG	CONTROL
FULLTEXT CATALOG	REFERENCES
FULLTEXT CATALOG	TAKE OWNERSHIP
FULLTEXT CATALOG	VIEW DEFINITION
FULLTEXT STOPLIST	ALTER
FULLTEXT STOPLIST	CONTROL
FULLTEXT STOPLIST	REFERENCES

Securable	Permission
MESSAGE TYPE	ALTER
MESSAGE TYPE	CONTROL
MESSAGE TYPE	REFERENCES
MESSAGE TYPE	TAKE OWNERSHIP
MESSAGE TYPE	VIEW DEFINITION
OBJECT	ALTER
OBJECT	CONTROL
OBJECT	DELETE
OBJECT	EXECUTE
OBJECT	INSERT
OBJECT	RECEIVE
OBJECT	REFERENCES
OBJECT	SELECT
OBJECT	TAKE OWNERSHIP
OBJECT	UPDATE
OBJECT	VIEW CHANGE TRACKING
OBJECT	VIEW DEFINITION
REMOTE SERVICE BINDING	ALTER
REMOTE SERVICE BINDING	CONTROL
REMOTE SERVICE BINDING	TAKE OWNERSHIP
REMOTE SERVICE BINDING	VIEW DEFINITION
ROLE	ALTER
ROLE	CONTROL
ROLE	TAKE OWNERSHIP
ROLE	VIEW DEFINITION
ROUTE	ALTER
ROUTE	CONTROL
ROUTE	TAKE OWNERSHIP
ROUTE	VIEW DEFINITION
SCHEMA	ALTER
SCHEMA	CONTROL

Securable	Permission
SCHEMA	INSERT
SCHEMA	REFERENCES
SCHEMA	SELECT
SCHEMA	TAKE OWNERSHIP
SCHEMA	UPDATE
SCHEMA	VIEW CHANGE TRACKING
SCHEMA	VIEW DEFINITION
SERVICE	ALTER
SERVICE	CONTROL
SERVICE	SEND
SERVICE	TAKE OWNERSHIP
SERVICE	VIEW DEFINITION
SYMMETRIC KEY	ALTER
SYMMETRIC KEY	CONTROL
SYMMETRIC KEY	REFERENCES
SYMMETRIC KEY	TAKE OWNERSHIP
SYMMETRIC KEY	VIEW DEFINITION
TYPE	CONTROL
TYPE	EXECUTE
TYPE	REFERENCES
TYPE	TAKE OWNERSHIP
TYPE	VIEW DEFINITION
USER	ALTER
USER	CONTROL
USER	IMPERSONATE
USER	VIEW DEFINITION
XML SCHEMA COLLECTION	ALTER
XML SCHEMA COLLECTION	CONTROL
XML SCHEMA COLLECTION	EXECUTE
XML SCHEMA COLLECTION	REFERENCES
XML SCHEMA	TAKE OWNERSHIP

Securable	Permission
FULLTEXT STOPLIST	TAKE OWNERSHIP
FULLTEXT STOPLIST	VIEW DEFINITION

Securable	Permission
SCHEMA	DELETE
SCHEMA	EXECUTE

Securable	Permission
COLLECTION	
XML SCHEMA COLLECTION	VIEW DEFINITION

Welche konkreten Rechte sind vergeben?

Vergebene Rechte auf Server-Ebene

Zum Abrufen der vergebenen Rechte existiert die folgende – etwas wilde – 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 müssen Relationen zu folgenden Tabellen hergestellt werden:
  - sys.server_principals
  - sys.servers
  - sys.endpoints

Vergebene Rechte auf Datenbank-Ebene

Zum Abrufen der vergebenen Rechte existiert die folgende – etwas wilde – 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 müssen Relationen zu folgenden Tabellen hergestellt werden:
  - sys.all_objects
  - sys.all_columns
  - ...oder diversen anderen Tabellen
  - Von dort sind dann je nach Art der Securables ggf. weitere JOINS auf diverse weitere Tabellen erforderlich, die hier nicht im Detail beschrieben werden.

Effektive Berechtigungen

Überblick

Effektive Berechtigungen entstehen durch Kombination von

- verschiedenen Logins (durch Gruppenmitgliedschaften auf Windows-Ebene 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)

GRANT, REVOKE und DENY

- Mit "GRANT" wird ein Recht explizit vergeben.
- Durch "GRANT ... WITH GRANT OPTION" wird neben dem eigentlichen Recht auch das Recht vergeben, dieses Recht vergeben zu dürfen.
- Der "REVOKE"-Befehl ist das Gegenteil von "GRANT", er entzieht eine vorher "GRANT" vergebene Berechtigung.
- Außerdem gibt es noch den "DENY"-Befehl. Hiermit wird ein explizit oder implizit vergebenes Recht sozusagen unwirksam gemacht.
  - Hat also z.B. ein Datenbank-User auf Grund einer Gruppenmitgliedschaft neben anderen Rechten das Recht "ALTER ANY DATABASE AUDIT", so kann ihm dieses mit dem Befehl
    ```
    DENY ALTER ANY DATABASE AUDIT TO <SomeDatabaseUser>
    ```
    explizit versagt werden.
- DENY ist stärker als GRANT – fast immer ☺
  - Ausnahme im Original-Ton von BooksOnline:
    "A table-level DENY does not take precedence over a column-level GRANT. This inconsistency in the permissions hierarchy has been preserved for the sake of backward compatibility."

Ermitteln effektiver Rechte

Vorbemerkung: Alle Funktionen in diesem Kapitel arbeiten immer mit dem aktuellen Security-Kontext! Will man entsprechende Berechtigungs-Abfragen auf andere als den eigenen Principal abfragen, muß man mit IMPERSONATION arbeiten (siehe weiter unten)!

Ermitteln der Permissions eines Principals auf ein Securable?

- Oder anders formuliert: "Welche Rechte bestehen alles auf ein bestimmtes 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

Hat ein Principal eine bestimmte Berechtigung auf ein konkretes Securable?

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

Impersonation

Überblick

Mit Hilfe von Impersonation kann man den Security-Kontext wechseln und erreichen, daß bestimmte Programmteile mit anderen Permissions arbeiten.

Explizite Impersonation

Durch die "EXECUTE AS"-Anweisung kann man explizit in einen anderen Security-Kontext wechseln, z. B.:

```
EXECUTE AS LOGIN = 'SomeSpecialLogin'
```

Voraussetzung dafür ist die Vergabe einer entsprechenden "IMPERSONATION"-Berechtigung des aktuellen Logins auf das Login "SomeSpecialLogin" – sonst könnte ja jeder als jeder beliebige andere Principal agieren!

Relevante Befehle und Funktionen in diesem Umfeld sind u.a.:

- EXECUTE AS
- GRANT IMPERSONATE
- REVERT
- SUSER_NAME()
- USER_NAME()
- CURRENT_USER()
- SESSION_USER()
- SYSTEM_USER()

Implizite Impersonation durch EXECUTE-AS-Klausel

Mit der "EXECUTE AS"-Klausel kann man für StoredProcedures und andere Programmier-Elemente festlegen, in welchem Security-Kontext sie ausgeführt werden sollen:

- CREATE PROCEDURE dbo.usp_Demo
  WITH EXECUTE AS 'SqlUser1'
  AS
  SELECT user_name(); -- Shows execution context is set to SqlUser1.
  -- now do something useful...
  GO
- Wer immer diese Prozedur aufruft, sie wird in jedem Fall im Security-Kontext von "SqlUser1" und dadurch mit dessen Rechten ausgeführt.

Implizite Impersonation durch Application Roles

Application Roles sind eine weitere Möglichkeit der impliziten Impersonation – siehe Books Online:

- A user executes a client application.
- The client application connects to an instance of SQL Server as the user.
- The application then executes the sp_setapprole stored procedure with a password known only to the application.
- If the application role name and password are valid, the application role is enabled.
- At this point the connection loses the permissions of the user and assumes the permissions of the application role.

Tipps & Tricks

"A-G-DL-P" – Empfehlung für ein Authentifizierungs-Konzept

Dieses Konzept basiert auf einer wohl durchdachten Gruppenhierarchie auf Windows-/ AD-Ebene:

- Windows-**A**ccounts...
- ...werden in **g**lobale Windows-Gruppen gesteckt,
- diese wiederum in **d**omänen-**l**okalen Gruppen zusammengefaßt
- und diese dann letztendlich im SQL-Server mit **P**ermissions versehen
- NB: Also kein Einsatz von reinen SQL-Server-Logins!

Der Vorteil dieses Konzepts liegt darin, daß das Rechte-Management im laufenden Betrieb ganz simpel durch die Pflege von Gruppenmitgliedschaften erfolgt und damit zum Teil schon automatisiert erfolgt.
Demgegenüber steht ein erhöhter einmaliger Einrichtungsaufwand, den man aber in Kauf nehmen sollte!

Entsprechende Namenskonventionen über alle relevanten Ebenen sind rein organisatorisch zwingend erforderlich, sonst ist ein gewisses Chaos vorprogrammiert!

Für AD-Insider: Außerdem ist zu beachten, daß die beteiligten DomainController bzgl. "Domain functional level" und "Forest functional level" korrekt definiert sein müssen!

Kleinkram

- Fest definierte Server- bzw. Datenbank-Rollen können nicht ineinander verschachtelt werden.
- Das Reconnect-Feature "sp_change_users_login" steht nur für sql-server-basierte Logins zur Verfügung, nicht für windows-authentifizierte logins.
- Beim Detach von Datenbank-Dateien werden die AccessControlLists der betroffenen MDF-, NDF- und LDF-Dateien verändert – lediglich der User, der das Detach ausgeführt hat, hat noch Zugriff!
- 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*"

Zusammenfassung

Die Security-Konfiguration der Datenbank-Engine ist ein sehr komplexes Gebilde, mit dem man sich ausführlich befassen muß, um keine unliebsame Überraschungen zu erleben.