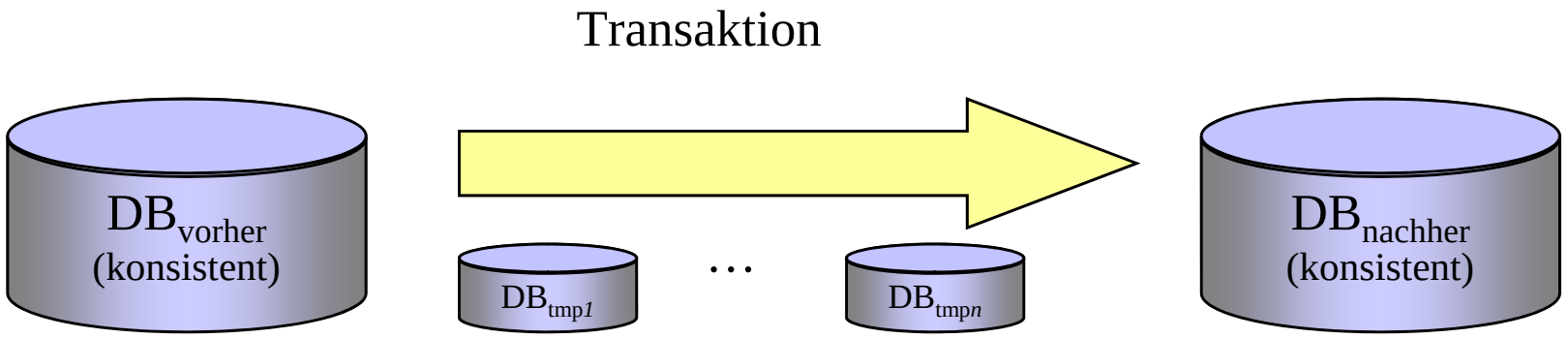
7. Datenintegrität/-Sicherheit

7.2 Datensicherheit

- Zur Erinnerung:
  - Eine Transaktion führt die Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand über.
  - Was bedeutet „konsistent“?
    - Fehlertoleranz gegenüber unbeabsichtigten Schäden (=> Recovery)
    - Isoliertheit der TAs (Synchronisation)
    - Datenintegrität

    und jetzt noch:
    - ***Datenschutz*** vor beabsichtigten Schäden



7. Datenintegrität/-Sicherheit

7.2.1 Einleitung

- Allgemeine Aspekte zum Datenschutz
  - Juristische und ethische Faktoren

    z.B. Bundes-Datenschutz-Gesetz, …
  - Organisations-(z.B. Firmen-)spezifische Regelungen

    z.B. Kreditkartenauskünfte, versch. Sicherheitsebenen für Abteilungen beim Geheimdienst, …
  - Technische Faktoren

    HW-Ebene, Betriebssystem-Ebene, DBMS-Ebene, …

7.2.1 Einleitung

- Datenschutzmechanismen
  - Identifikation und Authentisierung

    Benutzermanagement, …
  - Autorisierung und Zugriffskontrolle

    Regeln legen erlaubte Zugriffsarten von ***Sicherheitssubjekten*** auf ***Sicherheitsobjekten*** fest
    - Sicherheitssubjekt: aktive Entität, die Informationsfluss bewirkt, z.B. Benutzer(-gruppen), Anwendungsprogramme, Trigger, …
    - Sicherheitsobjekt: passive Entität mit Informationsinhalt(en), z.B. ein Tupel, ein Attribut, …
  - Auditing

    Buchführen über sicherheitsrelevante DB-Operationen

7. Datenintegrität/-Sicherheit

7.2.2 Einfache Zugriffskontrolle in SQL

- Discretionary Access Control (DAC)
  - Spezifiziert Regeln zum Zugriff auf Objekte
  - Eine Regel besteht aus
    - Einem Objekt (z.B. Relationen, Tupel, Attribute, …)
    - Einem Subjekt (z.B. Benutzer, Prozesse, …)
    - Einem Zugriffsrecht (z.B. „lesen“, „schreiben“, „löschen“, …)
    - Einem Prädikat, das eine Art Zugriffsfenster auf dem Objekt festlegt
    - Einem Booleschen Wert, der angibt, ob das Recht vom Subjekt an andere Subjekte weitergeben darf

7.2.2 Einfache Zugriffskontrolle in SQL

- Die Regeln werden typischerweise in einer eigenen Tabelle oder in einer Matrix (Spalten: Objekte, Zeilen: Subjekte) gespeichert
- Zugriff eines Subjekts auf ein Objekt nur, wenn entsprechender Eintrag in Tabelle/Matrix
- Umsetzung
  - Als View (mit den entsprechenden Update-Problematiken)
  - Abänderung der Anfrage entsprechend den Bedingungen
    - `select`-Klausel darf nur Attribute enthalten, auf die der entspr. Benutzer Zugriff hat
    - Zugriffsprädikat kann konjunktiv an die `where`-Bedingung angefügt werden
    - ...

7.2.2 Einfache Zugriffskontrolle in SQL

- Nachteile von DAC
  - Performanz: Abhängig von der Granularität der Autorisierung können diese Tabellen/Matrizen sehr groß werden
  - Beruht auf der Annahme, dass Erzeuger der Daten deren Eigner und damit für die Sicherheit verantwortlich ist
    - Erzeuger können Zugriffsrechte damit beliebig weitergeben
    - Beispiel Firma: Angestellte erzeugen Daten und sind dann in der Verantwortung für die Sicherheit dieser Daten
  - Weitergabe von Rechten kann zu Problemen führen
    - S1 gibt Recht an S2
    - S1 gibt Recht an S3
    - S2 gibt Recht an S3
    - S1 will Recht S3 wieder entziehen ???

7.2.2 Einfache Zugriffskontrolle in SQL

- Trotzdem:
  - DAC ist einfach umzusetzen und daher sehr gebräuchlich
  - Zugriffskontrolle im SQL-92 Standard basiert auf DAC-Modell
- Apropos: SQL Standard
  - Stellt keine Normen für Authentisierung oder Auditing auf
  - Einfache Zugriffskontrolle nach DAC-Modell
    - `grant` – vergibt Rechte
    - `revoke` – entzieht Rechte
    - Intial liegen alle Rechte beim Administrator (DBA)
- Manche DBMS stellen Zugriffskontroll-Mechanismen nach mächtigeren Modellen (z.B. dem MAC-Modell, siehe später) zur Verfügung

7.2.2 Einfache Zugriffskontrolle in SQL

- Autorisierung mir `grant`
  - Typische Form:

    ```
    grant <OPERATION> on <TABLE> to <USER>
    ```

  - Dabei ist `<OPERATION>`:
    - `select` Lesezugriff
    - `delete` Löschen
    - `insert (<Attribute>)` Einügen der spezifizierten Attribute
    - `update (<Attribute>)` Verändern der spezifizierten Attribute
    - `references (<Attribut>)` Fremdschlüssel auf das Attribut

      ACHTUNG: hier gilt es natürlich referentielle Integrität einzuhalten, daher könnte man dadurch die Schlüsselwerte der anderen Relation herausbekommen:
      - » Es gibt Relation `Agenten` mit Schlüssel = geheime Kennung
      - » Wir haben keine Zugriffsrechte auf diesen Schlüssel kennen aber das Schema von `Agenten`
      - » Mit `create table at(Kennung char(4) references Agenten);` können wir durch Einfügen einiger Zeilen prüfen ob entsprechende Werte in `Agenten` existieren

7.2.2 Einfache Zugriffskontrolle in SQL

- Recht zur Weitergabe von Rechten durch Anhängen von

  `with grant option`

  am Ende eines `grant`-Befehls
- Entziehen eines Rechts mit `revoke`
  - Bei Privileg mit Weitergaberecht:
    - `restrict` falls Weitergabe erfolgt, bricht DBMS mit Fehlermeldung ab
    - `cascade` löscht auch die Rechte, die durch Weitergabe entstanden sind
- Umsetzungen von bedingten Rechten in SQL durch Sichten

  Beispiel: Tutoren für EIP sollen nur die Daten der Studenten im ersten Semester lesen können

```
create view ErstSemester as
    select * from Studenten where Semester = 1;

grant select on ErstSemester to tutor
```

7. Datenintegrität/-Sicherheit

7.2.3 Verfeinerte Zugriffskontrolle

- Bisher nur
  - Explizite Autorisierung
    - Bei vielen Objekten viele Regeln => großer Aufwand
    - Schöner wäre, wenn wir uns durch ***implizite Autorisierung*** etwas sparen könnten
  - Positive Autorisierung
    - Darf ein Subjekt 4 der 5 möglichen Operationen auf einem Objekt, müssen alle 4 (explizit) erlaubt werden (analog: z.B. 1 aus einer Gruppe von 10 Subjekten hat als einziges Subjekt ein spez. Recht nicht, …)
    - Schöner wäre, z.B. *per default* alle zu erlauben und nur die eine Operation zu verbieten (***negative Autorisierung***)
    - Dazu nötig: Unterschied zwischen ***starker*** und ***schwacher Autorisierung:***

      Schwache Autorisierung wird als Standardeinstellung (z.B. für alle Subjekte) verwendet und gilt daher immer, falls keine andere (starke) Autorisierung erfolgt

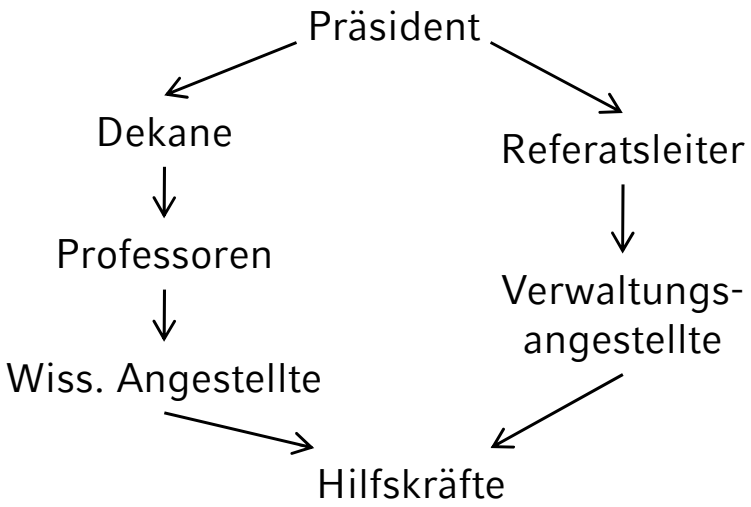
7.2.3 Verfeinerte Zugriffskontrolle

- Kernidee der Erweiterung
  - Subjekte, Objekte und Operationen werden hierarchisch angeordnet
  - Explizite Autorisierung auf einer bestimmten Stufe der Hierarchie bewirkt implizite Autorisierung auf anderen Stufen der Hierarchie
  - Unterscheidung in
    - Positive Autorisierung — schreibe (Objekt, Subjekt, Operation)
    - Negative Autorisierung — schreibe (Objekt, Subjekt, ¬Operation)
  - Unterscheidung zwischen
    - Starker Autorisierung — schreibe (...)
    - Schwacher Autorisierung — schreibe [...]

7.2.3 Verfeinerte Zugriffskontrolle

- Implizite Autorisierung von Subjekten
  - Rolle
    - Funktion einer Menge von Benutzer im System
    - Beinhaltet die Rechte, die zur Umsetzung notwendig sind)
  - Rollenhierarchie enthält mind.
    - Eine ausgezeichnete Rolle mit der maximalen Menge an Rechten (z.B. DBA, Firmenleitung, ...) als Wurzel der Hierarchie
    - Eine eindeutige grundlegende Rolle (z.B. alle Angestellten)

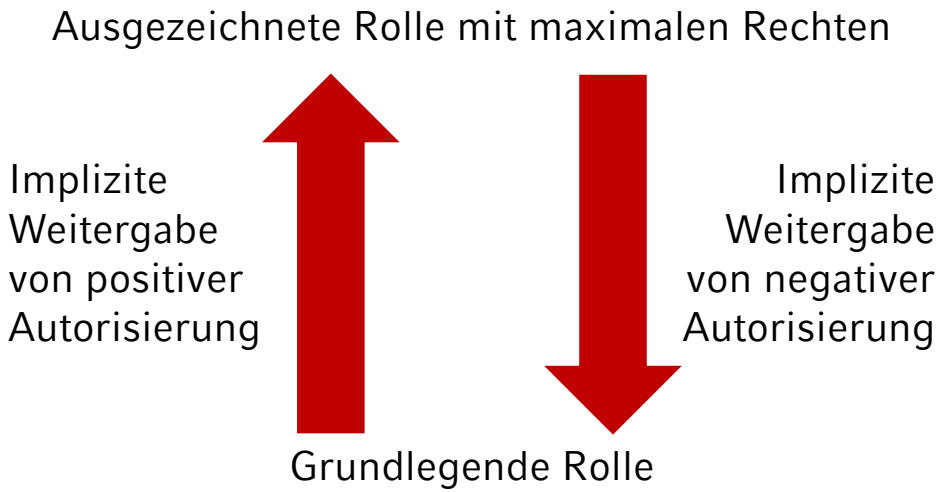
Beispiel:



7.2.3 Verfeinerte Zugriffskontrolle

– Regeln zur impliziten Autorisierung

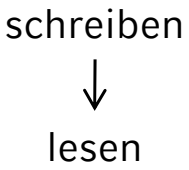
1. Eine explizite positive Autorisierung auf einer Stufe resultiert in einer impliziten positiven Autorisierung auf allen höheren Stufen (z.B. besitzen Dekane implizit alle Zugriffsrechte die explizit oder implizit für Professoren gelten)
2. Eine explizite negative Autorisierung auf einer Stufe resultiert in einer impliziten negativen Autorisierung auf allen niedrigeren Stufen (z.B. gilt der explizite Zugriffsverbot auf ein Objekt für den Referatsleiter implizit auch für den Verwaltungsangestellten)



7.2.3 Verfeinerte Zugriffskontrolle

- Implizite Autorisierung von Operationen
  - Analog: Operationshierarchien

    Beispiel

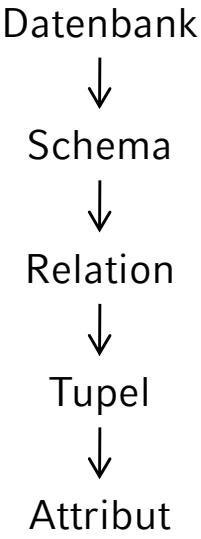


  - Weitergabe der Rechte nun umgekehrt:
    - Positive Autorisierung wird nach unten weitergegeben (Schreibberechtigung impliziert Leseberechtigung)
    - Negative Autorisierung wird nach oben weitergegeben (Leseverbot impliziert auch Schreibverbot)

7.2.3 Verfeinerte Zugriffskontrolle

- Implizite Autorisierung von Objekten
  - Granularitätshierarchien für Objekte

    Bsp.: Leserecht für eine Relation R sollte Leserecht für die einzelnen Tupel von R implizieren
  - Regeln hängen meist von der auszuführenden Operation ab, z.B.
    - Explizites Lese- und Schreibrecht auf einer Relation impliziert (nur) Leserecht auf deren Schema
    - Leserechte müssen immer auch nach unten implizit weiter geleitet werden
    - Definition einer neuen Relation hat keine Implikation auf andere Ebenen
    - …



7.2.3 Verfeinerte Zugriffskontrolle

- Typhierarchien
  - Bieten eine weitere Dimension für implizite Autorisierung
  - Werden durch is-a-Beziehungen (Generalisierung/Spezialisierung) zwischen Entities definiert (vgl. oo Programmierung)
    - Zugriffsrecht auf einen Objekttypen O impliziert Zugriffsrecht auf von O vererbte Attribute im Untertypen
    - Attribut eines Untertypen ist nicht vom Obertypen erreichbar
    - Zugriff auf Objekttypen O impliziert Zugriff auf vom Obertypen ererbte Attribute in O
  - Problem:
    - Vererbung wird im relationalen Modell nicht unterstützt sondern nur simuliert
    - Daher wird eine implizite Autorisierung entlang einer Typhierarchie in relationalen DBMS meist nicht unterstützt

7. Datenintegrität/-Sicherheit

7.2.4 MAC und Multilevel DBs

- MAC = Mandatory Access Control
  - Einführung einer Sicherheitshierarchie

    z.B. „streng geheim“, „geheim“, „vertraulich“, „unklassifiziert“
  - Sicherheitseinstufung für
    - Subjekte s: clear(s) spezifiziert die Vertrauenswürdigkeit von s
    - Objekte o: class(o) spezifiziert die Sensitivität von o
  - Typische Zugriffsregeln:
    1. Subjekt s darf Objekt o nur lesen, wenn class(o) ≤ clear(s)
    2. Objekt o wird mit mindestens der Einstufen des schreibenden Subjekts s versehen, d.h. clear(s) ≤ class(o)

    - Bemerkungen
      - Die zweite Regel stellt sicher, dass ein Benutzer der Klasse „streng geheim“ auch nur „streng geheime“ Objekte schreibt, insbesondere v.a. keine „unklassifizierten“ Objekte (Write Down , Leak)
      - Niedrig eingestufte Objekte können „hochklassifiziert“ werden (und sehen dadurch möglicherweise ungewollt vertrauenswürdiger /fundierter aus)

7.2.4 MAC und Multilevel DBs

- Bewertung von MAC
  - Potentiell größere Sicherheit durch mächtigeren (ausdrucksstärkeren) Kontrollmechanismus
  - Typischerweise Organisationsproblem
    - Benutzer unterschiedlicher Klassifikationsstufen können nicht zusammen arbeiten
    - Alle Objekte der Datenbank müssen eingestuft sein
  - Problem des Abgriffs nicht freigegebener Daten besteht immer noch:

    Relation Agenten

Class (Tupel)	Kennung	class	Name	class	Drink	class
sg sg	007 008	g sg	James Bond Harry Potter	g sg	Wodka Limo	sg sg

    Benutzer mit clear = g sieht

Kennung	Name	Drink
007	James Bond	---

    und möchte Tupel mit Kennung 008 eingeben, was verweigert wird (womit klar ist, dass diese Kennung schon existiert, das entspr. Tupel aber höher klassifiziert ist)

7.2.4 MAC und Multilevel DBs

- Multilevel DBs
  - Lösung des Problems durch ***Polyinstanziierung***
    - Ein Tupel darf mehrfach mit unterschiedlichen Sicherheitseinstufungen vorkommen (alle Einstufungen müssen aber tatsächlich paarweise verschieden sein)
    - Die DB stellt sich damit Nutzern unterschiedlicher Einstufungen unterschiedlich dar

      (Im Beispiel von vorher gäbe es nun zwei Einträge mit Schlüssel 008 mit unterschiedlicher Klassifizierung)
  - Damit können nun auch Benutzer unterschiedlicher Klassifikationen auf „den gleichen“ Daten arbeiten, da eine Bearbeitung nicht sofort zu einer Höherklassifizierung (nach Regel 2 des MAC-Modells) führt

7.2.4 MAC und Multilevel DBs

- Umsetzung
  - Schema einer Multilevel Relation *R* besteht aus
    - *n* Attributen A_i mit ihren Domänen D_i (wie gehabt)
    - Für jedes Attribut A_i eine Klassifizierung C_i
    - Eine Klassifizierung TC des gesamten Tupels
  - Für jede Zugriffsklasse c gibt es dann eine ***Relationeninstanz*** R_c
  - In R_c sind alle Tupel $(a_1,c_1, a_2,c_2, \ldots a_n,c_n, tc)$ mit $c \geq c_i$
  - Der Wert a_1 eines Attribut A_1 ist sichtbar (d.h. $a_1 \in D_i$) falls $c \geq c_i$ ansonsten `null`
- Integritätsbedingungen
  - Fundamental im normalen relationalen Modell:
    - Eindeutigkeit des Schlüssels
    - Referentielle Integrität
  - In Multilevel DBs Erweiterung nötig

7.2.4 MAC und Multilevel DBs

- Schlüssel:

  in Multilevel Relationen heißt der benutzerdefinierte Schlüssel ***sichtbarer Schlüssel***

  Sei im folgenden K der sichtbare Schlüssel von Relation *R*

- Entity Integrität

  für alle Instanzen R_c von *R* und alle Tupel $r \in R_c$ gilt

  1. $A_i \in K \Rightarrow r.A_i \neq \texttt{null}$

     d.h. kein Schlüsselattribute besitzt `null`-Werte

  2. $A_i, A_j \in K \Rightarrow r.C_i = r.C_j$

     d.h. alle Schlüssel haben die gleiche Klassifizierung (sonst kann Möglichkeit des Zugriffs auf Tupel nicht eindeutig geklärt werden)

  3. $A_i, \notin K \Rightarrow r.C_i \geq r.C_K$ (wobei C_K Zugriffsklasse des Schlüssels)

     d.h. Nicht-Schlüsselatribute haben mindestens die Zugriffsklasse des Schlüssels

7.2.4 MAC und Multilevel DBs

- Null-Integrität

  für alle Instanzen R_c von *R* gilt

  1. Für alle Tupel $r \in R_c$ gilt: $r.A_i = \texttt{null} \Rightarrow r.C_i = r.C_K$

     d.h. `null`-Werte erhalten die Klassifizierung des Schlüssels

  2. R_c ist subsumierungsfrei, d.h. es existieren keine zwei Tupel r und s in R_c, bei denen für alle Attribute A_i entweder

     – $r.A_i = s.A_i$ und $r.C_i = s.C_i$

     oder

     – $r.A_i \neq \texttt{null}$ und $s.A_i = \texttt{null}$

     gilt

     d.h. Tupel, über die schon mehr bekannt ist, werden „verschluckt“

7.2.4 MAC und Multilevel DBs

Beispiel für Subsumtionsfreiheit:

Class (Tupel)	Kennung	class	Name	class	Drink	class
g sg	007 008	g sg	James Bond Harry Potter	g sg	--- Limo	g sg

Ein streng geheimer Benutzer fügt Drink von 007 ein

⇒ Er erwartet dann

Class (Tupel)	Kennung	class	Name	class	Drink	class
sg sg	007 008	g sg	James Bond Harry Potter	g sg	Wodka Limo	sg sg

⇒ Ohne Subsumtionsfreiheit

Class (Tupel)	Kennung	class	Name	class	Drink	class
g sg sg	007 007 008	g g sg	James Bond James Bond Harry Potter	g g sg	--- Wodka Limo	g sg sg

7.2.4 MAC und Multilevel DBs

- Interinstanz-Integrität

  Die Konsistenz ziwschen den einzelnen Instanzen der Multilevel-Relation muss gewährleistet sein.

  Daher: für alle Instanzen R_c und $R_{c'}$ von *R* mit $c < c'$ gilt

  $$R_{c'} = f(R_c, c')$$

  wobei die Filterfunktion f wie folgt arbeitet:

  1. Für jedes $r \in R_c$ mit $r.C_K \leq c'$ muss ein Tupel $s \in R_{c'}$ existieren, mit

  $$s.A_i = \begin{cases} r.A_i & \text{wenn } r.C_i \leq c' \\ \text{null} & \text{sonst} \end{cases} \quad \text{und} \quad s.C_i = \begin{cases} r.C_i & \text{wenn } r.C_i \leq c' \\ r.C_K & \text{sonst} \end{cases}$$

  2. $R_{c'}$ enthält außer diesen keine weiteren Tupel
  3. Subsumierte Tupel werden eliminiert

7.2.4 MAC und Multilevel DBs

- Polyinstanziierungs-Integrität

  für alle Instanzen R_c von *R* und alle A_i gilt die funktionale Abhängigkeit:

  $$\{K, C_K, C_i\} \rightarrow A_i$$

  d.h. ein Tupel ist eindeutig bestimmt durch den Schlüssel und die Klassifizierung aller Attribute

  (diese Bedingung entspricht der „normalen“ Schlüsselintegrität)

- Umsetzung
  - Zerlegung einer Multilevel-Relation in mehrere normale Relationen (für jede Klassifizierungsebene), die bei Benutzeranfragen entsprechend zusammengesetzt werden können
  - Sicherstellung der hier genannten Integritätsbedingungen meist durch Trigger