

## 5 Views, Datenunabhängigkeit, Sicherheit

### 5.1 Sichten [Views]

Eine Sicht ist einfach eine Relation, wo wir die *Definition* anstatt eine Menge Einträge speichern.

Sichten werden genau wie normale (basis-) Tabellen benutzt.

Konzeptuell: wenn "JungeStudenten" gebraucht wird, die View wird zuerst evaluiert, und das Resultat wird gebraucht für weitere Evaluation:

```
CREATE VIEW JungeStudenten(Name, Note)
AS SELECT S.Name, E.Note
FROM Studenten S, Eingeschrieben E
WHERE S.sid = E.sid and S.Alter<21
```

Sichten können mit `DROP VIEW` gelöscht werden.

Wie soll man `DROP TABLE` behandeln wenn eine Sicht diese Tabelle benutzt? `DROP TABLE` hat Optionen, die den Benutzern erlauben, das zu spezifizieren.

Zur Erinnerung: physisches, konzeptuelles und externes Schema, siehe Figur 1.

Views erlauben, das konzeptuelle Schema zu "adaptieren."

Obwohl "echte" Tabellen aus dem konzeptuellen Schema auch im externen Schema sichtbar sein können, garantieren die Views die logische Datenunabhängigkeit.

D.h. Views können benutzt werden, um Änderungen im konzeptuellen Schema zu verstecken, sodass Anwendungen nicht gestört werden.

Views können benutzt werden, um notwendige Information (oder eine Zusammenfassung davon) zu zeigen, während andere Attribute versteckt bleiben.

Z.B. "AktiveStudenten", aber nicht "Studenten" oder "Eingeschriebene."

Wir können Studenten finden, die in "Kurse" eingeschrieben sind, aber der *cid* von "Kurse" bleibt versteckt.

Views spielen auch eine zentrale Rolle in Data Warehouses

### 5.2 Sicherheit und Autorisierung

#### 5.2.1 Einführung in DB-Sicherheit

**Geheimnis:** Anwendern sollten nicht sehen können, was sie nicht dürfen.

Z.B., Eine Student kann nicht die Noten von den anderen Studenten sehen.

**Integrität:** Anwendern sollten nicht ändern können, was sie nicht dürfen.

Z.B. Nur Dozierenden dürfen Noten setzen.

**Zugänglichkeit:** Anwendern sollen sehen und modifizieren können, was sie dürfen.

### 5.2.2 Zugriffskontrolle

Eine Sicherheitspolitik [security policy] spezifiziert, was von wem zu tun erlaubt ist.

Ein Sicherheitsmechanismus [security mechanism] erlaubt uns, eine bestimmte Sicherheitspolitik zu realisieren.

Zwei Hauptmechanismen stehen auf der Stufe des DBMS zur Auswahl:

- Discretionary access control
- Mandatory access control

## 5.3 Discretionary Access Control

Konzept: Zugriffsberechtigungen oder Privilegien [priviledges] auf Objekte (Tabellen und Sichten), und Mechanismen, um den Benutzer und Benutzerinnen diese Privilegien zu geben [grant] oder zu entfernen [revoke].

Eine Person, die eine Tabelle oder eine Sicht kreiert, bekommt auf ihr automatisch alle Privilegien.

Der DMBS verfolgt, wer Privilegien auf diesem Objekt bekommt oder verliert, und überprüft, dass nur Abfragen von Benutzerinnen und Benutzer, die die nötigen Privilegien haben (wenn die Anfrage gestellt wird), tatsächlich erlaubt sind.

### 5.3.1 GRANT Befehl

**GRANT** privileges ON object TO users [WITH GRANT OPTION]

Die folgenden Privilegien können spezifiziert werden:

- **SELECT**: Alle Kolonnen können gelesen werden, sogar diejenigen, die mit einem `ALTER TABLE`-Befehl nach der Erstellung der Tabelle addiert worden sind.
- **INSERT (col-name)**: Datensätze können mit nicht-NULL oder nicht-default Werte in col-name eingefügt werden.
- **INSERT (ohne col-name)** bedeutet gleiche Berechtigung für alle Kolonnen.
- **DELETE**: Datensätze können gelöscht werden.
- **REFERENCES (col-name)**: Fremdschlüssel können (in anderen Tabellen) definiert werden, die col-name referenzieren.

WITH GRANT OPTION

Wenn eine Person ein Privileg mit `GRANT OPTION` besitzt, kann das Privileg an anderen Personen weitergegeben werden (mit oder ohne `GRANT OPTION`).

Nur die Besitzer können `CREATE`, `ALTER`, und `DROP` ausführen.

### 5.3.2 GRANT und REVOKE von Privilegien

- `GRANT INSERT, SELECT ON Sailors TO Horatio`  
Horatio kann Sailors abfragen oder Datensätze in Sailors einfügen.
- `GRANT DELETE ON Sailors TO Yuppy WITH GRANT OPTION`  
Yuppy kann Datensätze löschen, und auch andere erlauben, so zu machen.
- `GRANT UPDATE (rating) ON Sailors TO Dustin`

Dustin kann (nur) das Feld *rating* von den Sailors-Datensätze aktualisieren (modifizieren).

- `GRANT SELECT ON OldSailors TO Guppy, Yuppy`

Das erlaubt NICHT, Guppy und Yuppy die Sailors-Tabelle abzufragen

REVOKE: Wenn ein Privileg von X weggenommen wird, ist es auch von allen Anwendern weggenommen, die es *nur* von X bekommen haben.

### 5.3.3 GRANT/REVOKE auf Sichten

Wenn der Besitzer einer Sicht das `SELECT` Privileg auf der assoziierten Tabelle verliert, wird die Sicht nicht mehr zugreifbar.

Falls die Besitzerin einer Sicht ein Privileg `WITH GRANT OPTION` auf einer unterlegenen Tabelle verliert, verliert sie auch das entsprechende Privileg auf der Sicht; auch die Benutzer, die das gleiche Privileg auf der Sicht erhalten hatten, verlieren dieses Privileg.

### 5.3.4 Sichten und Sicherheit

Sichten werden benutzt, um nötigen Informationen zu zeigen (oder eine Zusammenfassung davon), während die Einzelheiten von der(den) untenstehenden Relation(en) versteckt werden.

Z.B. Gegeben sei `OldSailors`, aber nicht `Sailors`, wir können nicht finden welche `Sailors` jünger als 40 sind.

Die Person, die eine Sicht erstellt, hat ein Privileg auf der Sicht wenn sie oder er das Privileg auf alle unterstehenden Tabellen besitzt.

Zusammen mit `GRANT/REVOKE`, sind Sichten ein sehr leistungsfähiges Zugriffskontrollmechanismus

### 5.3.5 Rollen-Basierte Autorisierung

In SQL-92, Privilegien sind mit `authorization ids` zugewiesen, die eine einzige Person oder eine Gruppe von Personen bezeichnen können.

In SQL:1999 (und in vielen heutigen Systemen) werden Privilegien an Rollen zugewiesen.

- Rollen können an Personen oder an anderen Rollen zugewiesen werden
- Reflektieren besser, wie reelle Organisationen arbeiten.
- Es zeigt, wie „echte Standards“ oft „de facto Standards“, die in populäre Systeme sich etabliert haben, nachholen müssen.

### 5.3.6 Sicherheit bis zur Stufe eines Feldes!

Man kann eine Sicht erstellen, die nur ein Feld von einem Datensatz zurückgibt (wie?)

Denn das Privileg mit dem `GRANT` auf dieser Sicht entsprechend geben

Es erlaubt also, eine arbiträre Granularität der Zugriffskontrolle, aber:

- Verwirrend zu spezifizieren (obwohl das kann man hinter eine guten UI verstecken)
- Performanz ist nicht akzeptabel wenn wir es öfters machen müssen (zu viele Sichterstellungen und lookups)

## 5.4 Internet-Orientierte Sicherheit

Grösste Herausforderung: Benutzerauthentifizierung und -Vertrauen [trust].

Wenn eine DB aus einem sicheren Ort zugegriffen wird, können Passwort-basierten Strategien normalerweise eingesetzt werden

Für Zugriff über einem externen Netzwerk ist Vertrauen aber schwierig zu erreichen

Wenn jemand mit der Kreditkarte von Sam etwas von Ihnen kaufen will, wie können Sie sicher sein, es ist nicht jemandem, der oder die seine Karte gestohlen hat?

Wie kann Sam sicher sein, dass das GUI, dass er benutzt um seine Kreditkarteninformation zu eingeben, ist tatsächlich Ihre, und nicht irgendein Site, dass Ihnen „spoof“ (um die Informationen zu stehlen)? Wie kann er sicher sein, dass die sensitive Information nicht „gesniff“ wird, während sie über den Netzwerk gesendet wird?

*Encryption* ist eine Methode um diese Probleme zu verhindern.

### 5.4.1 Encryption

Daten werden für sichere Übermittlung oder Speichern “maskiert”

Encrypt(data, encryption key) = encrypted data

Decrypt(encrypted data, decryption key) = original data

Ohne decryption key, sind die Daten sinnloses schrott.

Symmetric Encryption:

Encryption key = decryption key; alle autorisierte Personen kennen die decryption key (eine Schwäche).

DES, used since 1977, has 56-bit key; AES has 128-bit (optionally, 192-bit or 256-bit) key

Public-Key Encryption: Alle Personen haben zwei Schlüssel :

Encryption Key: allen bekannt

Decryption Key: nur Sie kennen ihre Decryption Key.

Benutzt in RSA (Turing Award!)

### 5.4.2 RSA Public-Key Encryption

Let the data be an integer  $I$

Choose a large ( $\gg I$ ) integer  $L = p * q$

$p, q$  are large, say 1024-bit, distinct prime numbers

Encryption: Choose a random number  $1 < e < L$  that is relatively prime to  $(p-1) * (q-1)$

Encrypted data  $S = I^e \text{ mod } L$

Decryption key  $d$ : Chosen so that

$d * e = 1 \text{ mod } ((p-1) * (q-1))$

We can then show that  $I = S^d \text{ mod } L$

It turns out that the roles of  $e$  and  $d$  can be reversed; so they are simply called the public and private keys

### 5.4.3 Certifying Servers: SSL, SET

If Amazon distributes their public key, Sam's browser will encrypt his order using it.

So, only Amazon can decipher the order, since no one else has Amazon's private key.

But how can Sam (or his browser) know that the public key for Amazon is genuine? The SSL protocol covers this.

Amazon contracts with, say, Verisign, to issue a certificate <Verisign, Amazon, amazon.com, public-key>

This certificate is stored in encrypted form, encrypted with Verisign's *private* key, known only to Verisign.

Verisign's public key is known to all browsers, which can therefore decrypt the certificate and obtain Amazon's public key, and be confident that it is genuine.

The browser then generates a temporary *session key*, encodes it using Amazon's public key, and sends it to Amazon.

All subsequent msgs between the browser and Amazon are encoded using symmetric encryption (e.g., DES), which is more efficient than public-key encryption.

What if Sam doesn't trust Amazon with his credit card information?

Secure Electronic Transaction protocol: 3-way communication between Amazon, Sam, and a trusted server, e.g., Visa.

### 5.4.4 Authenticating Users

Amazon can simply use password authentication, i.e., ask Sam to log into his Amazon account.

Done after SSL is used to establish a session key, so that the transmission of the password is secure!

Amazon is still at risk if Sam's card is stolen and his password is hacked. Business risk ...

Digital Signatures:

Sam encrypts the order using his *private* key, then encrypts the result using Amazon's public key.

Amazon decrypts the msg with their private key, and then decrypts the result using Sam's public key, which yields the original order!

Exploits interchangeability of public/private keys for encryption/decryption

Now, no one can forge Sam's order, and Sam cannot claim that someone else forged the order.

## 5.5 Mandatory Access Control [Obligatorische Zugriffskontrolle]

Systemweite Regeln, die nicht von einzelnen Personen geändert werden können.

Jedem DB-Objekt ist einer Sicherheitsklasse [security class] zugewiesen.

Jedem Subjekt (Person oder Anwendung) ist einer Zulassungsklasse [clearance] für eine Sicherheitsklasse zugewiesen.

Regeln basieren auf Sicherheitsklasse und Zulassung, um zu bestimmen, wer die Objekte lesen oder schreiben kann.

Die meisten kommerziellen Systeme unterstützen MAC nicht. Es wird in Versionen für spezialisierte Anwendungen (z.B. Militär) unterstützt.

### 5.5.1 Wieso MAC?

Discretionary Control hat einige Schwächen, z.B. das Problem des *Trojanischen Pferdes*:

Peter kreiert eine Tabelle „Salär“ und gibt INSERT-Privilegien an Sabine (die es nicht weiss)

Peter modifiziert den Code einer Anwendung, die von Sabine benutzt wird, um zusätzlich geheime Informationen in der Tabelle Salär zu schreiben

Jetzt kann Peter die geheime Informationen sehen.

Die Modifikationen des Codes ist nicht unter Kontrolle des DBMS, aber das DBMS kann probieren zu verhindern, dass es als Kanal für geheime Informationen benutzt wird.

### 5.5.2 Bell-LaPadula Modell

Objekte (e.g., Tabellen, Sichten, Datensätze)

Subjekte (e.g., Personen, Anwendungsprogramme)

Sicherheitsklassen:

Top secret (TS), secret (S), confidential (C), unclassified (U):  $TS > S > C > U$

Jedem Objekt und Subjekt wird einer Klasse zugewiesen.

- Subjekt S kann Objekt O lesen nur wenn  $class(S) \geq class(O)$  (Simple Security Property)
- Subjekt S kann Objekt O schreiben nur wenn  $class(S) \leq class(O)$  (\*-Property)

Intuition:

Die Idee ist, dass die Information nie von einer höheren zu einer tieferen Sicherheitsklasse fließen kann.

z.B., falls Peter Sicherheitsklasse C hat, Sabine Klasse S hat:

-Die Tabelle von Peter, EMP, hat die Zulassung von Peter, C Die Anwendung von Sabine hat ihre Zulassung S

-Deshalb kann die Anwendung in EMP NICHT schreiben.

Die MAC regeln können beliebig mit den DAC Regeln kombiniert werden

### 5.5.3 Multistufige Relationen

Personen mit S und TS Zulassung werden beide Zeilen sehen; eine Person mit C wird nur die zweite Zeile sehen, eine Person mit U wird keine Zeile sehen.

Falls eine Person mit C  $\langle 101, \text{Pasta}, \text{Blue}, C \rangle$  inserieren will:

-Insertion kann wegen Schlüsselbeschränkung (PK) nicht erfolgen

-Aber der Person diese Information geben ist schon ein Hinweis, dass es ein anderes Objekt gibt mit Schlüssel 101 und Klasse  $> C$

-Das Problem ist gelöst in dem das Feld „class“ auch als Teil der Schlüssel behandelt wird.

## 6 Dateiorganisationen und Indizes

### 6.1 Alternative für Dateiorganisationen

Viele Alternativen existieren, *jede ist ideal für eine bestimmte Situation, und nicht so gut für eine andere:*

- Haufen [Heap] Dateien: Gut wenn die typische durchgeführte Operation ein Durchsuchen [scan] der gesamten Datei ist, wo alle Datensätze zugegriffen werden müssen.
- Sortierte Dateien: Gut wenn Datensätze in einer bestimmten Ordnung zugegriffen werden müssen, oder wenn nur ein "Bereich" der eingetragenen Werten gebraucht wird.
- Hash Dateien: Gut für Gleichheitsabfragen.
  - Die Datei enthält eine Menge Behältern [buckets]. Behälter = *Hauptseite [primary page]* plus null oder mehrere Überfluss-Seiten [overflow pages].
  - Hash-Funktion h*:  $h(r)$  = Behälter, in welchem der Datensatz  $r$  gehört.  $h$  benutzt nur eine Untermenge der Feldern von  $r$ , diese sind dann die *Suchfelder [search fields]* genannt.

### 6.2 Dateiorganisationsvergleich

Das folgende Vergleich erlaubt uns zu sehen, in welche Situation bestimmte Organisationen gut sind und in welche nicht.

Wir untersuchen fünf verschiedene Operationen auf den drei Dateiorganisationen:

- Scan (alle Datensätze verarbeiten)
- Gleichheitsabfrage
- Bereichsabfrage
- Datensatz einfügen
- Datensatz löschen

Für die Einfachheit nehmen wir CPU-Kosten nicht in Betracht.

- B**: Anzahl Datenseite
- R**: Anzahl Datensätze per Datenseite
- D**: (Mittlere) Zeit, um eine Seite zu lesen oder zu schreiben

Die Fokussierung auf der Anzahl I/Os verursacht die Vernachlässigung des Gewinns, der mit pre-fetching gewonnen werden konnte. Das heisst, dass sogar die I/O Kosten nur angenähert sind. Es handelt sich hier um eine Mittelwert-Analyse, die auf einfache Annahmen basiert.

*Immerhin ist sie gut genug, um die allgemeinen Trends zu zeigen!*

#### Annahmen im Kostenmodell

- Bei insert und delete werden einzelne Datensätze behandelt
- Heap Dateien: