

5 Views, Datenunabhängigkeit, Sicherheit

5.1 Sichten [Views]

5.1.1 Views erstellen

Wir betrachten folgendes schema:

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

```
CREATE VIEW Used_Parts( name, type, provider, country )
  AS SELECT  p.name, p.type, p.provider, prov.country
  FROM  parts p, provider prov, part_part_rel pp
  WHERE  p.provider = prov.name and p.id = pp.childid
```

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

Konzeptuell: wenn `part_info` gebraucht wird, wird die View zuerst evaluiert, und das Resultat wird für weitere Evaluation gebraucht.

Sichten können mit `DROP VIEW` gelöscht werden.

Wie soll man `DROP TABLE` behandeln wenn eine Sicht diese Tabelle benutzt? In manche Systeme hat `DROP TABLE` Optionen, die den Benutzern erlauben, das Verhalten zu spezifizieren. In Oracle wird die Table unabhängig von der View gelöscht, und die View wird damit unbrauchbar (Fehlermeldung)

5.1.2 Konzeptuelle Rolle der Views

Sichten sind mehr als ein „convenience tool.“ Sie spielen eine konzeptuelle Rolle, indem sie erlauben, das konzeptuelle Schema zu „adaptieren.“ (Zur Erinnerung schaut man die Schicht-Architektur der Datenbank mit physisches, konzeptuelles und externes Schema, siehe Abbildung 4 oben.)

Obwohl „echte“ Tabellen aus dem konzeptuellen Schema auch im externen Schema sichtbar sein können, garantieren die Views die logische Datenunabhängigkeit. In anderen Worten können Views benutzt werden, um Änderungen im konzeptuellen Schema zu verstecken, sodass Anwendungen von den Änderungen nicht gestört werden.

5.1.3 Views als Mittel, um Information zu verstecken

Dazu können Views auch benutzt werden, um gewisse Information zu zeigen, während andere Attribute versteckt bleiben. Diese Eigenschaft ist notwendig, weil wir in der Datenbank keine Redundanz wollen, deshalb kann man keine Daten von einer Tabelle auf einer anderen verschieben.

Zum Beispiel kann `Swiss_Used_Parts` gewisse Eigenschaften von Part oder Provider spezifizieren:

```
CREATE VIEW Swiss_Used_Parts( name, type, provider )
  AS SELECT  u.name, u.type, u.provider
  FROM  used_parts u
  WHERE  u.country <> 'Switzerland'
```

sodass nur Teile aus der Schweiz sichtbar sind. Auch zu bemerken ist, dass die *id* von "Parts" bleibt versteckt.

Eine andere Möglichkeit ist, die Information als Zusammenfassung zu präsentieren:

```
CREATE VIEW Swiss_Used_Parts( num, provider )
  AS SELECT count(*), u.provider
  FROM swiss_used_parts u
  group by u.provider
```

Views spielen auch eine zentrale Rolle in Data Warehouses wo sie in gewisse Fälle auch materialisiert werden, d.h. die Table wird physisch als solche gespeichert.

5.2 Sicherheit und Autorisierung

5.2.1 Einführung in DB-Sicherheit

Wieso brauchen wir überhaupt Sicherheit? Obwohl das selbstverständlich ist, muss man es zuerst formalisieren, bevor man ein robustes Sicherheitssystem einführen kann.

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 **Sicherheitsverfahren** [security policy] spezifiziert, was von wem zu tun erlaubt ist.

Ein **Sicherheitsmechanismus** [security mechanism] erlaubt uns, eine bestimmter Sicherheitsverfahren 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 [privileges] 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.

Hier einige Beispiele:

- `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.

5.3.2 GRANT/REVOKE auf Views

Wenn der Besitzer einer View 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, ein Privileg auf der Sicht erhalten hatten, verlieren dieses Privileg.

5.3.3 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

5.3.4 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.

5.3.5 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 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 meiste kommerzielle Systeme unterstützen MAC nicht. Es wird in Versionen für spezialisierte Anwendungen (z.B. Militär) unterstützt.

5.4.1 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.

5.4.2 Bell-LaPadula Modell

Objekte (e.g., Tabellen, Views, Datensätze, Kolonnen, Zeilen)

Subjekte (e.g., Personen, Anwendungsprogramme)

Sicherheitsklassen sind mit einer Ordnung organisiert

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

wobei „A > B“ bedeutet, dass Klasse A ist empfindlicher als Klasse B. (Es kann selbstverständlich mehr als 4 Klassen geben).

Jedem Objekt wird einer Sicherheitsklasse zugewiesen, und jedes Subjekt wird einer Genehmigung für eine bestimmte Klasse gegeben.

- Subjekt S kann Objekt O **lesen**, nur wenn $\text{class}(S) \geq \text{class}(O)$ (Simple Security Property). Das heisst, wenn S eine TS berechtigung hat, kann es eine Tabelle mit C-Klasse lesen, aber nicht das Gegenteil
- Subjekt S kann Objekt O **schreiben**, nur wenn $\text{class}(S) \leq \text{class}(O)$ (*-Property). Das heisst ein Objekt mit S Genehmigung kann nur Tabellen mit S oder TS Klassen schreiben.

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

Jetzt können wir das besprochene trojanische Pferd verhindern: Falls Peter Sicherheitsklasse C hat, Sabine Klasse S hat:

-Die Tabelle von Peter, `salär`, hat die Zulassung von Peter, C. Die Anwendung von Sabine hat ihre Zulassung S

-Deshalb kann die Anwendung in `salär` NICHT schreiben.

Die MAC regeln können beliebig mit den DAC Regeln kombiniert werden

5.4.3 Multistufige Relationen und Polyinstantiation

Um MAC in einer RDBMS umzusetzen, eine Sicherheitsklasse muss an jeder Objekt zugewiesen werden. Wie schon angesprochen kann diese Zuweisung entweder auf der Stufe der Tabellen, Zeilen, oder Kolonnen sein. Nehmen wir an hier, dass jede Zeile einer Klasse zugewiesen ist. Um das zu implementieren, fügen wir das Konzept von multistufige Tabellen. Abbildung 11 zeigt eine Solche Tabelle.

<u>bid</u>	bname	color	class
101	Salsa	Red	S
102	Pinto	Brown	C

Abbildung 11: Eine Multistufige Relation

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 <101,Pasta,Blue,C> inserieren will:

-Insertion kann wegen Schlüsseleinschränkung (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.