Teil 12: Sichten (Views, Virtuelle Tabellen)

Literatur:

- Elmasri/Navathe:Fundamentals of Database Systems, 3rd Edition, 1999.
  Section. 8.5, "Views (Virtual Tables) in SQL"
- Silberschatz/Korth/Sudarshan: Database System Concepts, 3rd Edition, McGraw-Hill, 1999.
  Section 4.8, "Views" (plus 4.9.4, "Update of a View")
- Kemper/Eickler: Datenbanksysteme (in German), Ch. 4, Oldenbourg, 1997.
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.
- Date/Darwen: A Guide to the SQL Standard, Fourth Edition, Addison-Wesley, 1997.
- Oracle8 SQL Reference, Oracle Corporation, 1997, Part No. A58225-01.
- Oracle8 Concepts, Release 8.0, Oracle Corporation, 1997, Part No. A58227-01.
- Don Chamberlin: A Complete Guide to DB2 Universal Database. Morgan Kaufmann, 1998.
- Microsoft SQL Server Books Online: Accessing and Changing Data, Administering SQL Server.

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- das Konzept von Sichten erklären.

  Z.B. was ist der Unterschied zwischen der Deklaration einer Sicht und der Abspeicherung des Anfrageergebnisses in einer neuen Tabelle? Warum heißen Sichten virtuelle Tabellen?

- einige Anwendungen von Sichten aufzählen.
- Sichten in SQL deklarieren.

  Sie sollten auch wissen, wann "`WITH CHECK OPTION`" wichtig ist.

- erklären, wie Updates von Sichten ausgeführt werden, und warum nicht alle Sichten updatebar sind.
- entscheiden, ob eine Sicht updatebar ist.

Inhalt

Beispiel-Datenbank

STUDENTEN

SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	···
102	Michael	Grau	NULL
103	Daniel	Sommer	···
104	Iris	Winter	···

BEWERTUNGEN

SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
102	H	2	9
102	Z	1	10
103	H	1	5
103	Z	1	7

AUFGABEN

ATYP	ANR	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
Z	1	SQL	14

Sichten (1)

- Sichten erlauben es, eine Anfrage in der Datenbank abzuspeichern, und ihr einen Namen zu geben (man kann auch die Ergebnisspalten umbenennen):

```
CREATE VIEW HA(VORNAME, NACHNAME, GP)
AS   SELECT VORNAME, NACHNAME, SUM(PUNKTE)
     FROM   STUDENTEN S, BEWERTUNGEN B
     WHERE  S.SID = B.SID AND B.ATYP = 'H'
     GROUP  BY VORNAME, NACHNAME, S.SID
```

- Das Ergebnis einer SQL-Anfrage ist eine Tabelle.
- Man kann Sichten ("virtuelle Tabellen") in Anfragen wie normale Tabellen ("Basistabellen") nutzen.

Sichten (2)

- Z.B. kann man diese Anfrage an die Sicht stellen:

```
SELECT X.VORNAME, X.NACHNAME
FROM   HA X
WHERE  X.GP > 15
```

- Das DBMS kann intern den Namen der Sicht einfach durch die definierende Anfrage ersetzen:

```
SELECT X.VORNAME, X.NACHNAME
FROM  (SELECT VORNAME, NACHNAME, SUM(PUNKTE) GP
       FROM   STUDENTEN S, BEWERTUNGEN B
       WHERE  S.SID = B.SID AND B.ATYP = 'H'
       GROUP  BY VORNAME, NACHNAME, S.SID) X
WHERE  X.GP > 15
```


Sichten (3)

- Sichten waren schon im SQL-86 Standard enthalten, aber mit gewissen Einschränkungen.
- Da SQL-86 keine Unteranfragen unter `FROM` hatte, war es schwierig (und manchmal unmöglich) die gegebene Anfrage in eine zu übersetzen, die sich nur auf die Basistabellen bezieht:

```
SELECT VORNAME, NACHNAME
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID AND B.ATYP = 'H'
GROUP  BY VORNAME, NACHNAME, S.SID
HAVING SUM(PUNKTE) > 15
```


Sichten (4)

- Z.B. kann man auf diese Art keine Aggregation in einer Anfrage an eine Sicht zu verwenden, die selbst über eine Aggregation definiert ist.

  Das hätte eine geschachtelte Aggregation benötigt, die in SQL nicht erlaubt ist. Man brauchte eine Unteranfrage unter `FROM`.

- In SQL-92, Oracle, DB2, SQL Server und Access gibt es (soweit mir bekannt) keine Einschränkungen an die Verwendung von Sichten in Anfragen.

  Außer Fulltext Anfragen in SQL Server. MySQL hatte früher keine Sichten, inzwischen soll auch das implementiert sein. In Access kann man Anfragen unter einem Namen abspeichern, und dann auch in Anfragen wie Tabellen benutzen: Das sind Sichten.

Sichten (5)

- Sichten sind abgeleitete, virtuelle Tabellen, die aus den (tatsächlich abgespeicherten) Basistabellen berechnet werden (z.B. Alter aus Geburtsdatum).

  Bei Sichten wird die definierende Anfrage gespeichert (intensionale Definition), bei Basistabellen die Tupel (extensionale Definition).

- Sichten können also nie Informationen enthalten, die nicht schon in den Basistabellen enthalten ist.

  Man könnte allerdings eventuell die Definition der Sicht (die Berechnungsvorschrift) als zusätzliche Information ansehen.

- Sichten können aber die in den Basistabellen enthaltene Information anders strukturiert anzeigen.

Sichten (6)

- Die Sicht unterscheidet sich wesentlich von einer mit dem Anfrageergebnis gefüllten Tabelle:

```
CREATE TABLE HA2(...);
INSERT INTO HA2(VORNAME, NACHNAME, GP)
       SELECT VORNAME, NACHNAME, SUM(POINTS)
       FROM STUDENTEN S, BEWERTUNGEN B
       WHERE S.SID = B.SID AND B.ATYP = 'H'
       GROUP BY VORNAME, NACHNAME, S.SID
```

- Dies wertet die Anfrage nur einmal aus und speichert das Ergebnis dauerhaft in eine neue Tabelle.

Sichten (7)

- Im Unterschied dazu wird die Anfrage bei der Sicht jedesmal ausgewertet, wenn die Sicht benutzt wird.

  Der Anfrageoptimierer wird natürlich versuchen, nicht jedesmal die vollständige Sicht zu berechnen, sondern nur den Teil, der für die gegebene Anfrage relevant ist.

- Wenn die Basistabellen sich ändern,
  - ◇ wird die Sicht `HA` automatisch diese Änderung wiederspiegeln,

    Da sie ohnehin bei jeder Anfrage neu ausgewertet wird.
  - ◇ während die Tabelle `HA2` manuell aktualisiert werden muß (oder auf dem alten Stand bleibt).

Sichten (8)

- Man kann die Aktualisierung der Tabelle `HA2` auch mit Triggern automatisch durchführen lassen.

  Ein Trigger ist eine Prozedur, die in der Datenbank gespeichert ist, und automatisch ausgeführt wird, wenn bestimmte Ereignisse auftreten. In diesem Fall sind die Ereignisse Änderungen auf den Basistabellen `STUDENTEN` und `BEWERTUNGEN`. Die Programmierung ist aber nicht einfach, da ziemlich viele verschiedene Änderungen auf diesen Tabellen das Anfrageergebnis beeinflussen können.

- Neuere DBMS bieten "materialisierte Sichten":
  - ◇ Tupel der Sicht explizit abgespeichert (effizient)
  - ◇ werden bei Änderungen der Basistabellen automatisch aktualisiert (teilweise eingeschränkt).

Sichten (9)

- Man kann Sichten definieren, die für verschiedene Benutzer unterschiedliche Ergebnisse liefern:

```
CREATE VIEW MEINE_PUNKTE(ATYP, ANR, PUNKTE) AS
SELECT B.ATYP, B.ANR, B.PUNKTE
FROM   STUDENTEN S, BEWERTUNGEN B
WHERE  S.SID = B.SID AND S.NACHNAME = USER
```

- "USER" liefert den DB-Login-Namen des aktuellen Nutzers (der die Anfrage stellt).

  Damit das funktioniert, müßten die Studierenden jeweils ihren Nachnamen als DB-Account haben. Man könnte in der STUDENTEN-Tabelle aber auch eine extra Spalte für die Benutzerkennung vorsehen.

- Sichten können auch vom Datum abhängen.

Sichten (10)

- Sichten können auch in der Definition anderer Sichten verwendet werden.
- Auf diese Art können komplexe Anfragen Schritt für Schritt aufgebaut werden.
- Rekursive Sichten waren in SQL-92 verboten.

  Alle in einer Sichtdefinition verwendeten Sichten müssen vorher schon vollständig definiert sein.

- SQL-99 erlaubt sie (Beispiel s.u.), sie werden aber erst in wenigen Systemen unterstützt (z.B. DB2).

  Rekursive Sichten sind eine Spezialität von Deduktiven DBen.

Ausblick: Rekursion (1)

- Rekursion ist wichtig für Baum-strukturierte Daten (Hierarchien) und "Transitive Hülle" Anfragen:
  - ◇ Z.B. enthält die Oracle-Beispiel-Tabelle EMP für jeden Angestellten die Angestellten-Nummer des direkten Vorgesetzten (Spalte MGR).
  - ◇ Angenommen, A hat den Vorgesetzten B, B wiederum den Vorgesetzten C, und C den Chef D.
  - ◇ C und D sind dann indirekte Vorgesetzte von A.
  - ◇ Wenn man alle indirekten Vorgesetzten berechnen will, braucht man Rekursion.

Ausblick: Rekursion (2)

Weiteres Beispiel: "Bill of Materials" Anfragen:

- Eine Firma setzt elementare Teile zu Baugruppen zusammen, Baugruppen zu größeren Baugruppen, bis zu den Produkten, die sie verkauft (z.B. Autos).
- Man bestimme die Kosten der fertigen Produkte, wenn folgende Informationen in der DB vorliegen:
  - ◇ Die Einkaufspreise der elementaren Teile.
  - ◇ Die Stückliste jeder Baugruppe/jedes Produktes (enthält elementare Teile und Baugruppen).
  - ◇ Die notwendige Arbeitszeit (Stundenlohn) für die Montage jeder Baugruppe/jedes Produktes.

Ausblick: Rekursion (3)

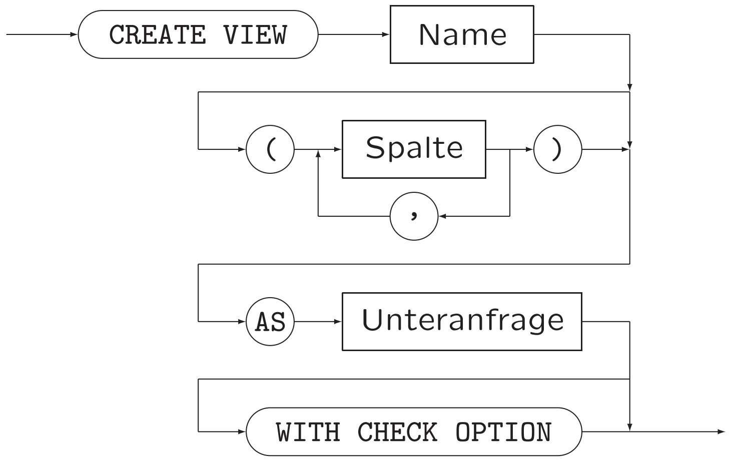
- SQL-99 erlaubt rekursive Sichten:

```
WITH
    RECURSIVE INDIRECT_MGR(EMPNO, BOSS) AS
        SELECT EMPNO, MGR FROM EMP
        UNION
        SELECT A.EMPNO, B.BOSS
        FROM   EMP A, INDIRECT_MGR B
        WHERE  A.MGR = B.EMPNO
SELECT BOSS FROM INDIRECT_MGR
WHERE  EMPNO = 7369
```

- Dies funktioniert so in DB2.

  Aber es gibt eine Warnung über mögliche Endlosschleifen.

Syntax (1)



Syntax (2)

- `ORDER BY` ist nicht in View-Definitionen erlaubt.

  Normalerweise bewirkt es auch nur ganz am Ende einer Anfrage etwas, während Sichten als Teilanfragen einer größeren Anfrage genutzt werden. Oracle erlaubt `ORDER BY` aber in Sichtdefinitionen. Im Zusammenhang mit `ROWNUM`-Bedingungen würde es auch in Unteranfragen Sinn machen.

- Sichtdefinitionen können gelöscht werden mit:

  `DROP VIEW ⟨NAME⟩`

- In Oracle kann man folgendes schreiben:

  `CREATE OR REPLACE VIEW ⟨NAME⟩ ...`

  Überschreibt ggf. bereits existierende Definition.

Anwendungen von Sichten (1)

- Bequemlichkeit / Wiederverwendung: Wiederkehrende Muster in Anfragen sind bereits vordefiniert.
- Die Basisrelationen sollten in BCNF sein, aber es ist u.U. bequemer, nicht normalisierte Relationen in Anfragen zu verwenden.

  Redundante Daten in Sichten sind kein Problem, weil diese Daten ja nicht abgespeichert werden, sondern nach Bedarf berechnet werden.

- Anpassung des DB-Schemas an die Wünsche verschiedener Benutzer / Benutzer-Gruppen.

Anwendungen von Sichten (2)

- Sicherheit: Manche Benutzer sollten nur einen Teil einer Tabelle sehen können (bestimmte Zeilen oder Spalten), oder nur aggregierte/anonymisierte Information.

  Ohne Sichten ist die Granularität für Zugriffsrechte im wesentlichen die Tabelle. Nur die Änderungsrechte können üblicherweise auch für einzelne Spalten vergeben werden.

- Logische Datenunabhängigkeit: Man kann neue Attribute zu einer Tabelle hinzufügen, und die alte Version noch als Sicht zur Verfügung stellen.

Inhalt

1. Konzept, Benutzung von Sichten in Anfragen
2. Updates von Sichten
3. Sichten und Zugriffsschutz (Sicherheit)

View Update Problem (1)

- Da die Sichten nicht explizit abgespeichert sind, müssen Updates auf den Sichten in Updates auf den Basistabellen übersetzt werden.
- Dies ist nicht immer möglich:

```
CREATE VIEW GUTE_HA(SID, ANR, PUNKTE) AS
SELECT SID, ANR, PUNKTE
FROM   BEWERTUNGEN
WHERE  ATYP = 'H' AND PUNKTE > 8
```

- Nun sei folgender Update betrachtet:

```
INSERT INTO GUTE_HA
VALUES (103, 2, 5);
```


View Update Problem (2)

- Man könnte natürlich View Updates verbieten.

  Es scheint anfangs ja auch merkwürdig, daß man Anfrage-Ergebnisse modifizieren will.

- Sie sind aber für viele Anwendungen wichtig:
  - ◇ Logische Datenunabhängigkeit: Es war früher ja eine Tabelle, jetzt ist es eine Sicht.
  - ◇ Zugriffsrechte: Ein Nutzer soll Zugriff nur auf bestimmte Spalten bekommen, bzw. Zeilen, die eine bestimmte Bedingung erfüllen.

    Diese “Teiltabelle” soll er auch ändern können.

View Update Problem (3)

- Lösung:
  - ◇ Relativ einfache Sichten (z.B. Projektion und Selektion auf einer Tabelle) sind (mit gewissen Einschränkungen) updatebar.
  - ◇ Komplexe Sichten sind "read only".
  - ◇ Manche Systeme (z.B. Oracle) bieten "Instead-of" Trigger: Man kann selbst programmieren, wie bestimmte View Updates durch Updates auf den Basistabellen implementiert werden sollen.

View Update Problem (4)

- Beispiel:

```
CREATE VIEW HA_BEW(SID, NR, PUNKTE) AS
SELECT SID, ANR, PUNKTE
FROM   BEWERTUNGEN
WHERE  ATYP = 'H'
```

- Nun sei folgender Update betrachtet:

```
UPDATE HA_BEW
SET    PUNKTE = PUNKE + 1
WHERE  NR = 2
```

- Dies wird übersetzt in:

```
UPDATE BEWERTUNGEN
SET    PUNKTE = PUNKE + 1
WHERE  ATYP = 'H' AND ANR = 2
```


View Update Problem (5)

- Entsprechend sind auch Löschungen möglich:

```
DELETE FROM HA_BEW
WHERE  SID = 101
```

- Dies wird übersetzt in:

```
DELETE FROM BEWERTUNGEN
WHERE  ATYP = 'H' AND SID = 101
```

- Übersetzung von Updates auf einfachen Sichten:
  - ◇ Name der Sicht → Name der Basistabelle,
  - ◇ Selektionsbedingung der Sicht wird an Bedingung des Update-Befehls angehängt,
  - ◇ ggf. umbenannte Attribute → Originalnamen.

View Update Problem (6)

- Nun sei folgende Einfügung betrachtet:

```
INSERT INTO HA_BEW
VALUES (103, 2, 10)
```

- Die übliche Übersetzung ist:

```
INSERT INTO BEWERTUNGEN(SID, ANR, PUNKTE)
VALUES (103, 2, 10)
```

- Bei der Ausführung dieses Updates kommt es zum Fehler, weil für die ausgeblendete Spalte `ATYP` kein Nullwert erlaubt ist.

  Nach dem SQL-Standard zählt die Sicht als updatebar. D.h. es gibt überhaupt eine definierte Übersetzung auf die Basistabellen. Jeder Update könnte eventuell einer Integritätsbedingung widersprechen.

View Update Problem (7)

- Analyse der Sichtdefinition hätte gezeigt, daß es für die Spalte ATYP nur einen möglichen Wert gibt:

```
CREATE VIEW HA_BEW(SID, NR, PUNKTE) AS
SELECT SID, ANR, PUNKTE
FROM   BEWERTUNGEN
WHERE  ATYP = 'H'
```

- Heutige kommerzielle DBMS wie Oracle führen eine solche Analyse aber nicht durch.

  Zum "View Update Problem" gibt es sehr viele Forschungsarbeiten, die natürlich auch solche Analysen beinhalten.

- Falls HA_BEW die einzige solche Sicht ist, kann man 'H' als Default-Wert für die Spalte ATYP deklarieren.

View Update Problem (8)

- Korrekte Übersetzung von View Updates auf die Basistabellen (Forderungen):
  - ◇ Wenn die Sicht im neuen DB-Zustand ausgewertet wird, soll sie genau den angegebenen Update widerspiegeln (als wäre sie eine Basistabelle).

    Das bedeutet insbesondere auch, daß es nicht noch irgendwelche zusätzlichen Änderungen geben darf.
  - ◇ Minimale Änderung der Basistabellen

    Es darf nicht mehr geändert werden, als für den Update nötig ist.
  - ◇ Eventuell: Die Anzahl geänderter Tupel in Sicht und Basistabelle sollte übereinstimmen.

View Update Problem (9)

- Selbst wenn eine Übersetzung eines View Updates auf die Basistabellen möglich ist, ist das Ergebnis nicht immer eindeutig. Beispiel folgt.
- Die Aufgaben-Tabelle sei um eine dritte Hausaufgabe über relationale Algebra erweitert:

AUFGABEN

<u>ATYP</u>	<u>ANR</u>	THEMA	MAXPT
H	1	ER	10
H	2	SQL	10
H	3	RA	10
Z	1	SQL	14

View Update Problem (10)

- Folgende Verbund-Sicht sei betrachtet:

```
CREATE VIEW THEMA_ERG(THEMA, SID, PUNKTE) AS
SELECT A.THEMA, B.SID, B.PUNKTE
FROM   AUFGABEN A, BEWERTUNGEN B
WHERE  A.ATYP = 'H' AND B.ATYP = 'H'
AND    A.ANR = B.ANR
```

THEMA_ERG

THEMA	SID	PUNKTE
ER	101	10
ER	102	9
ER	103	5
SQL	101	8
SQL	102	9

View Update Problem (11)

- Man möchte jetzt das Thema SQL in relationale Algebra ändern:

```
UPDATE THEMA_ERG
SET    THEMA = 'RA'
WHERE  THEMA = 'SQL'
```

- Es gibt zwei sinnvolle Übersetzungen des Updates:
  - Man kann in der Tabelle AUFGABEN das Thema der Aufgabe 2 von SQL in RA ändern.
  - Man kann in der Tabelle BEWERTUNGEN die beiden Einträge für die Hausaufgabe 2 in Hausaufgabe 3 ändern (die hat schon das Thema RA).

View Update Problem (12)

- Mehrdeutigkeiten ergeben sich auch, weil die meisten Datentyp-Funktionen nicht eindeutig invertierbar sind.

```
CREATE VIEW HA_SUM(SID, PUNKTE) AS
SELECT H1.SID, H1.PUNKTE+H2.PUNKTE
FROM   BEWERTUNGEN H1, BEWERTUNGEN H2
WHERE  H1.ATYP = 'H' AND H1.ANR = 1
AND    H2.ATYP = 'H' AND H2.ANR = 2
AND    H1.SID = H2.SID
```

- Wenn man jetzt die Punktzahl von Student 102 von 18 auf 19 erhöhen möchte, ist nicht klar, wie sich die Punkte auf die beiden Aufgaben verteilen.

Updatebare Sichten (1)

Einschränkungen in SQL-92:

- Die FROM-Klausel enthält nur eine Tabelle.
- Die SELECT-Klausel enthält kein DISTINCT und keine Datentyp-Operationen (wie z.B.+).
- Keine Aggregationsfunktionen (COUNT, SUM, etc.).

  Auch kein GROUP BY, HAVING.
- Unteranfragen in der WHERE-Klausel sind erlaubt, sie dürfen aber keine Tupelvariablen über der Tabelle der Hauptanfrage deklarieren.
- Kein UNION etc. (nur einzelner SELECT-Ausdruck).

Updatebare Sichten (2)

- Die unter FROM genannte Tabelle ist die Tabelle, die durch den View Update geändert wird.
- Selbst wenn eine Anfrage die obigen Bedingungen erfüllt, verletzt der erzeugte Update auf der Basistabelle möglicherweise Integritätsbedingungen wie NOT NULL (Beispiel s.o.).
- Selbst wenn der erzeugte Update fehlerfrei ausgeführt wird, erscheint das eingefügte/geänderte Tupel möglicherweise nicht in der Sicht.

  Siehe "WITH CHECK OPTION" unten.

Updatebare Sichten (3)

- Semijoins sind in updatebaren Sichten grundsätzlich möglich (Formulierung als Unteranfrage).
- Durch die Einschränkung, daß Unteranfragen nicht die zu ändernde Tabelle selbst enthalten dürfen, ist garantiert, daß sie in altem und neuem Zustand gleich ausgewertet werden.
- Beispiel siehe nächste Folie.

Updatebare Sichten (4)

- Beispiel (nicht updatebar):

```
CREATE VIEW BESTE(ATYP, ANR, SID, PUNKTE) AS
SELECT X.ATYP, X.ANR, X.SID, X.PUNKTE
FROM   BEWERTUNGEN X
WHERE  NOT EXISTS
       (SELECT *
        FROM   BEWERTUNGEN Y
        WHERE  X.ATYP=Y.ATYP AND X.ANR=Y.ANR
        AND    X.PUNKTE < Y.PUNKTE)
```

- Man könnte theoretisch ('H', 2, 103, 10) einfügen, aber dies würde gleichzeitig ein Tupel löschen.

Updatebare Sichten (5)

- DB2 unterscheidet zwischen
  - ◇ Sichten, bei denen `DELETE` möglich ist ("deletable views"): wie SQL-92-Forderungen, aber `UNION ALL` und Datentyp-Operationen sind erlaubt.
  - ◇ Updatebaren Spalten in Sichten: Die Sicht muß "deletable" sein, und die Spaltendefinition in der Sicht muß eine einzelne Eingabespalte sein.

    Keine Datentyp-Operationen.
  - ◇ Sichten, bei denen `INSERT` möglich ist ("insertable views"): Alle Spalten sind updatebar und es ist kein `UNION ALL` verwendet.

Updatebare Sichten (6)

Oracle:

- Oracle erlaubt bestimmte Updates auf Sichten, die durch Joins definiert sind.

  Außerdem kann man das Ergebnis des View Updates prozedural über "`INSTEAD OF`" Trigger definieren.
- Auf Sichten sind keine IBen (wie Schlüssel) definiert, aber sie können logisch impliziert sein (von der Sichtdefinition und IBen auf Basistabellen).
- Eine Basistabelle, deren Schlüssel auch Schlüssel der Sicht ist, heißt "key preserved table" ("Tabelle mit erhaltenem Schlüssel") der Sicht.

Updatebare Sichten (7)

- Updates auf der Sicht werden in Updates auf der “key preserved table” übersetzt. Beispiel:

```
CREATE VIEW BEW(ATYP, ANR, SID, PUNKTE, MAXPT) AS
SELECT B.ATYP, B.ANR, B.SID, B.PUNKTE, A.MAXPT
FROM   BEWERTUNGEN B, AUFGABEN A
WHERE  B.ATYP = A.ATYP AND B.ANR = A.ANR
```

- `BEWERTUNGEN` ist die “key preserved table”.
- Spalten `SID` und `PUNKTE` sind updatebar.
- Löschungen aus `BEW` → Löschungen aus `BEWERTUNGEN`.
- Einfügungen in `BEW` → Einfügungen in `BEWERTUNGEN`. Dabei kann kein Wert für `MAXPT` angegeben werden.

Updatebare Sichten (8)

SQL Server:

- SQL Server scheint sehr großzügig mit der Updatebarkeit von Sichten zu sein.
- Ich habe im Handbuch aber keine genauen Angaben darüber gefunden, was erlaubt ist.

Inhalt

Zugriffsrechte für Sichten (1)

- Für Sichten kann man mit dem `GRANT`-Befehl anderen Benutzern Zugriffsrechte geben (genau wie für normale Tabellen).

  Wenn ein Benutzer lesend auf die Sicht zugreifen will, braucht er das `SELECT`-Recht für die Sicht. Wenn er Tupel löschen will, das `DELETE`-Recht, zum Einfügen das `INSERT`-Recht, und zum Ändern einer Spalte das `UPDATE`-Recht für die Sicht oder für die betreffende Spalte der Sicht. Das ist alles genau wie bei normalen Tabellen. Natürlich machen `INSERT`, `UPDATE`, `DELETE`-Rechte nur Sinn, wenn die Sicht updatebar ist.

- Der andere Benutzer braucht keine Rechte an den Basistabellen, die der Sicht zugrunde liegen.

  Sichten werden ja gerade verwendet, um Zugriffe auf die Basistabellen einzuschränken.

Zugriffsrechte für Sichten (2)

- Es ist also ganz typisch, daß ein Benutzer z.B.
  - ◇ das `SELECT`-Recht für die Sicht hat,
  - ◇ aber nicht für die Basistabellen, die von der Sicht abgefragt werden.
- Dann kann er auf die Basistabellen nur über die Sicht zugreifen, und entsprechend
  - ◇ nur einen Teil der Tabellen sehen (bestimmte Zeilen oder Spalten), bzw.
  - ◇ nur aggregierte oder anonymisierte Daten.

Zugriffsrechte für Sichten (3)

- Aggregierte Daten lassen manchmal doch Rückschlüsse auf einzelne Personen zu:
  - ◇ Möglicherweise ist in dieser Vorlesung nur eine Person, die Physik studiert. Die Durchschnittsnote nach Studiengang wäre dann problematisch.
  - ◇ Die Punkte-DB zeigt die Anzahl der Teilnehmer mit höherer Punktzahl an. Ist sie 0, weiß man, daß jeder andere Teilnehmer, der diese Aufgabe abgegeben hat, nicht mehr Punkte hat.

Zugriffsrechte für Sichten (4)

- Man muß nicht unbedingt selbst Besitzer der verwendeten Basistabellen sein, um eine Sicht definieren zu können.
- Man braucht aber wenigstens das `SELECT`-Recht für die Basistabellen.
- Verliert man dieses Recht später (durch `REVOKE`), kann man die Sicht nicht mehr verwenden.

  Die Definition der Sicht wird aber nicht gelöscht. Die definierende Anfrage kann man immer noch im Systemkatalog nachschauen. Es kann ja auch sein, daß das `REVOKE` ein Fehler war, der gleich wieder durch einen `GRANT`-Befehl rückgängig gemacht wird. Es wäre ja schlimm, wenn dabei die Sicht gelöscht werden würde.

Zugriffsrechte für Sichten (5)

- Für den Zugriff auf eine Sicht
  - ◇ wird zunächst geprüft, ob der aktuelle Benutzer (der auf die Sicht zugreifen möchte) entsprechende Rechte an der Sicht hat.
  - ◇ Dann wird geprüft, ob der Benutzer, der die Sicht definiert hat, die notwendigen Rechte an den Basistabellen hat, und zwar "`WITH GRANT OPTION`".

    Definition einer Sicht und Weitergabe der Rechte daran hat ja dieselbe Wirkung wie die `GRANT OPTION`.
  - ◇ Falls der aktuelle Benutzer die Sicht selbst definiert hat, ist die `GRANT OPTION` nicht nötig.

Zugriffsrechte für Sichten (6)

- Obige Regeln gelten so in DB2 und Oracle.
- In SQL Server kann man Rechte an einer Sicht nur weitergeben, wenn man die zugrundeliegenden Tabellen besitzt.

  Genauer ist der `GRANT`-Befehl erfolgreich, aber die Zugriffe auf die Basistabellen werden in diesem Fall mit den mit den Rechten des Benutzers geprüft, der die Anfrage an die Sicht stellt, nicht des Benutzers, der die Sicht definiert hat (auch wenn dieser Benutzer die `GRANT OPTION` hat).
- In DB2 bekommt man das `CONTROL`-Recht an der Sicht nur, wenn man es für die Basistabellen besitzt.

Check Option (1)

- Angenommen, ein Tutor darf Punkte für alle Aufgabenarten außer der Endklausur eintragen:

```
CREATE VIEW TUTOR_BEW(SID, ATYP, ANR, PUNKTE) AS
SELECT SID, ATYP, ANR, PUNKTE
FROM   BEWERTUNGEN
WHERE  ATYP <> 'E'
```

- Überraschenderweise ist bei dieser Sichtdefinition möglich, Punkte für die Endklausur einzutragen:

```
INSERT INTO TUTOR_BEW VALUES (101, 'E', 1, 1000)
```

  Natürlich muß die Aufgabe in `AUFGABEN` eingetragen sein, sonst gibt es eine Fremdschlüssel-Verletzung. Das ist aber ein anderes Problem.

Check Option (2)

- Dieser Update wird übersetzt in:

```
INSERT INTO BEWERTUNGEN(SID, ATYP, ANR, PUNKTE)
            VALUES (101, 'E', 1, 1000)
```

  und anschließend ausgeführt.
- Das eingefügte Tupel wird in der Sicht nicht angezeigt, da es der `WHERE`-Bedingung aus der Sichtdefinition widerspricht.

  Es steht aber in der Basistabelle `BEWERTUNGEN`.
- Das Tupel kann über die Sicht nicht mehr geändert oder gelöscht werden (die Bedingung der Sicht wird ja jedem solchen Update angehängt).

Check Option (3)

- In der gleichen Weise ist möglich, daß ein Tupel nach einem `UPDATE` die Bedingung der Sicht verletzt:

```
UPDATE TUTOR_BEW
SET    ATYP = 'E'
WHERE  SID = 101 AND ATYP = 'H' AND ANR = 1
```

- Der Update bewirkt effektiv eine Löschung des Tupels aus der Sicht (es wird nicht mehr angezeigt).
- Auch hier wird der beabsichtigte Zugriffsschutz umgangen: Der Tutor kann beliebige Tupel einfügen.
- Lösung: `WITH CHECK OPTION` in Sichtdefinition.

Check Option (4)

- Die Sichtdefinition sieht jetzt so aus:

```
CREATE VIEW TUTOR_BEW(SID, ATYP, ANR, PUNKTE) AS
SELECT SID, ATYP, ANR, PUNKTE
FROM   BEWERTUNGEN
WHERE  ATYP <> 'E'
WITH   CHECK OPTION
```

- Jetzt wird geprüft, daß eingefügte oder geänderte Tupel auch die Bedingung der Sicht erfüllen.

  Also hinterher wirklich in der Sicht erscheinen.

- Verletzen sie die Bedingung, so wird die Einfügung / der Update nicht durchgeführt (Fehlermeldung).

Check Option (5)

- Warum ist die `CHECK OPTION` nicht Standard?
  - ◇ Performance-Nachteil: Zusätzlicher Test nötig.

    Oft nur minimale Verlangsamung. Je nach Sichtdefinition kann es aber auch nötig sein, dabei eine komplexe Unteranfrage auszuwerten. Immerhin kann man es vor dem Update auswerten, da die Unteranfrage sich nicht auf diese Tabelle beziehen kann.
  - ◇ Jetzt kann ein Update auf der Sicht nicht einfach in einen Update auf der Basistabelle übersetzt werden: Man bekommt einen Update mit vorgeschaltetem Test.
  - ◇ Nicht immer nötig (siehe nächste Folie).

Check Option (6)

- Falls der Tutor nur Zugriff auf Hausaufgaben haben soll, sieht die Situation ganz anders aus:

```
CREATE VIEW TUTOR_HA(SID, ANR, PUNKTE) AS
SELECT SID, ATYP, ANR, PUNKTE
FROM   BEWERTUNGEN
WHERE  ATYP = 'H'
```

  Einfügungen in die Sicht sind nur möglich, wenn `'H'` Default-Wert für die Spalte `ATYP` in `BEWERTUNGEN` ist. Updates sind auf jeden Fall möglich.

- Die eingefügten bzw. geänderten Tupel können die Bedingung der Sicht nicht verletzen, da die verwendete Spalte `ATYP` in der Sicht ausgeblendet ist.