Teil 9: Tabellendefinition

Literatur:

- Elmasri/Navathe:Fundamentals of Database Systems, 3. Auflage, 1999. Kap. 8, “SQL — The Relational Database Standard”
- Kemper/Eickler: Datenbanksysteme, 4. Auflage, Oldenbourg, 1997. Kapitel 4: Relationale Anfragesprachen.
- Lipeck: Skript zur Vorlesung Datenbanksysteme, Univ. Hannover, 1996.
- Date/Darwen: A Guide to the SQL Standard, 4. Auflage, Addison-Wesley, 1997.
- van der Lans: SQL, Der ISO-Standard, Hanser, 1990.
- Melton/Simon: Understanding the New SQL. Morgan Kaufman, 1993.
- Oracle 8i SQL Reference, Release 2 (8.1.6), Dez. 1999, Part No. A76989-01.
- Oracle 8i Concepts, Release 2 (8.1.6), Dez. 1999, Part No. 76965-01. Kapitel 12: Built-in Datatypes.
- Chamberlin: A Complete Guide to DB2 Universal Database. Morgan Kaufmann, 1998.
- Microsoft SQL Server Books Online: Accessing and Changing Data.
- Microsoft Jet Database Engine Programmer’s Guide, 2. Auflage (Teil der MSDN Library Visual Studio 6.0). Microsoft Access 2000 Online-Hilfe.
- DuBois: MySQL. New Riders Publishing, 2000, ISBN 0-7357-0921-1, 756 Seiten.
- MySQL-Handbuch für Version 3.23.53.

Lernziele

Nach diesem Kapitel sollten Sie Folgendes können:

- `CREATE TABLE`-Anweisungen in SQL schreiben.
- Integritätsbedingungen in SQL definieren.

  `NOT NULL`, Schlüssel, Fremdschlüssel und `CHECK`.
- für ein Attribut einen Datentyp wählen.

  Sie sollten die üblichen Datentypen, die es in jedem DBMS geben sollte, aufzählen und erklären können. Sie sollten die Parameter von `NUMERIC`, `CHAR` und `VARCHAR` erklären können. Sie sollten auch wissen, welche weiteren Datentypen es noch geben könnte (für die Einzelheiten können Sie dann in der Anleitung Ihres DBMS nachschauen).
- einige Datentyp-Funktionen aufzählen (Beispiele).

Inhalt

1. Schlüssel
2. Fremdschlüssel
3. CREATE TABLE-Syntax
4. CREATE SCHEMA, DROP TABLE
5. ALTER TABLE

Integritätsbedingungen (1)

- Integritätsbedingungen (IBen) sind Bedingungen, die jeder DB-Zustand erfüllen muss.
- Z.B. können im `CREATE TABLE`-Statement in SQL folgende Arten von IBen festgelegt werden:
  - ◇ `NOT NULL`: Verbot von Nullwerten in bestimmter Spalte.
  - ◇ Schlüssel: Jeder Schlüsselwert darf nur 1× vorkommen.
  - ◇ Fremdschlüssel: Werte einer Spalte müssen auch als Schlüsselwert in einer anderen Tabelle auftauchen.
  - ◇ `CHECK`: Spaltenwerte müssen eine Bedingung erfüllen.

    Bedingung kann sich auch auf mehrere Spalten beziehen.

Integritätsbedingungen (2)

- Der SQL-92-Standard enthält eine Anweisung `CREATE ASSERTION`, die aber in den heutigen DBMS nicht implementiert ist.
- Man kann Integritätsbedingungen auch durch SQL-Anfragen formalisieren, die Verletzungen der Bedingungen ausgeben (oder als logische Formeln).

  Oder man kann die Integritätsbedingungen in natürlicher Sprache angeben. Das DBMS versteht dies zwar nicht und kann daher die Bedingung nicht erzwingen. Aber es ist dennoch eine nützliche Dokumentation für die Entwicklung von Anwendungsprogrammen (die Erfüllung der IBen muß in den Programmen zur Dateneingabe geprüft werden). Sind IBen als SQL-Anfragen formuliert (s.o.), so kann man sie von Zeit zu Zeit ausführen und ggf. Verletzungen finden.

Eindeutige Identifikation (1)

- Ein Schlüssel einer Relation R ist eine Spalte A, die die Tupel/Zeilen in R eindeutig identifiziert.

  Die Schlüsselbedingung ist in einem DB-Zustand $\mathcal{I}$ genau dann erfüllt, wenn für alle Tupel $t, u \in \mathcal{I}[R]$ gilt: wenn $t.A = u.A$, dann $t = u$.

- Wenn z.B. `SID` als Schlüssel von `STUDENTEN` deklariert wurde, ist dieser DB-Zustand verboten:

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

Eindeutige Identifikation (2)

- Wurde `SID` als Schlüssel von `STUDENTEN` deklariert, lehnt das DBMS ab, eine Zeile mit dem gleichen Wert für `SID` wie eine existierende Zeile einzufügen.
- Schlüssel sind Integritätsbedingungen: Sie müssen für alle DB-Zustände gelten, nicht nur für den derzeitigen Zustand.
- Obwohl im obigen DB-Zustand (mit nur 4 Studenten) der Nachname (`NACHNAME`) eindeutig ist, würde dies allgemein zu einschränkend sein.

  Z.B. wäre das zukünftige Einfügen von “`Nina Weiss`” unmöglich.

Eindeutige Identifikation (3)

- Ein Schlüssel kann auch aus mehreren Attributen bestehen (“zusammengesetzter Schlüssel”).

  Wenn A und B zusammen einen Schlüssel bilden, ist es verboten, dass es zwei Zeilen t und u gibt, die in beiden Attributen übereinstimmen (d.h. $t.A = u.A$ und $t.B = u.B$). Zwei Zeilen können in einem Attribut übereinstimmen, aber nicht in beiden.

- Der Schlüssel “`VORNAME, NACHNAME`” ist hier erfüllt:

STUDENTEN			
SID	VORNAME	NACHNAME	EMAIL
101	Lisa	Weiss	···
102	Michael	Weiss	···
103	Michael	Grau	···

Eindeutige Identifikation (4)

- Formel im Tupelkalkül, die ausdrückt, daß `VORNAME` und `NACHNAME` ein Schlüssel von `STUDENTEN` ist (in zweiwertiger Logik, ohne Nullwerte für `EMAIL`):

$$\begin{array}{l} \forall\ \mathtt{studenten}\ \mathtt{X},\ \mathtt{studenten}\ \mathtt{Y}: \\ \quad \mathtt{X.vorname} = \mathtt{Y.vorname} \land \mathtt{X.nachname} = \mathtt{Y.nachname} \\ \qquad \rightarrow\ \mathtt{X.sid} = \mathtt{Y.sid} \land \mathtt{X.email} = \mathtt{Y.email} \end{array}$$

- Wenn man Nullwerte für `EMAIL` in dreiwertiger Logik korrekt behandeln will, muß man es so schreiben:

$$\begin{array}{l} \forall\ \mathtt{studenten}\ \mathtt{X},\ \mathtt{studenten}\ \mathtt{Y}: \\ \quad \mathtt{X.vorname} = \mathtt{Y.vorname} \land \mathtt{X.nachname} = \mathtt{Y.nachname} \\ \qquad \rightarrow\ \mathtt{X.sid} = \mathtt{Y.sid} \land (\mathtt{X.email} = \mathtt{Y.email} \lor \\ \qquad\qquad \mathtt{X.email\ is\ null} \land \mathtt{Y.email\ is\ null}) \end{array}$$

Schlüssel: Minimalität (1)

- Sei F_1 eine Formel, die "`NACHNAME`" als Schlüssel festlegt, und F_2 eine Formel, die dem zusammengesetzten Schlüssel "`VORNAME, NACHNAME`" entspricht.
- Dann gilt $F_1 \vdash F_2$, d.h. jeder DB-Zustand $\mathcal{I}$, der den Schlüssel "`NACHNAME`" erfüllt ($\mathcal{I} \models F_1$), erfüllt auch den Schlüssel "`VORNAME, NACHNAME`" ($\mathcal{I} \models F_2$).

  Allgemein macht das Hinzufügen von Attributen Schlüssel schwächer (die Bedingung wird von mehr Zuständen erfüllt).

- Wenn also `NACHNAME` als Schlüssel deklariert wurde, ist es nicht mehr interessant, dass auch "`VORNAME, NACHNAME`" eindeutig identifizierend ist.

Schlüssel: Minimalität (2)

- Man wird nie zwei Schlüssel deklarieren, so dass einer eine Teilmenge des anderen ist.

  Nur minimale Schlüssel (in Bezug auf "$\subseteq$") sind interessant. Viele Autoren beziehen sogar die Minimalitätsbedingung in die Definition des Schlüsselbegriffs ein.

- Der Schlüssel "`NACHNAME`" ist jedoch im Beispiel-Zustand auf Folie 9-8 nicht erfüllt.
- Möchte der DB-Entwerfer diesen Zustand zulassen, so ist die Schlüsselbedingung "`NACHNAME`" zu streng.

  Das ist eigentlich keine freie Entscheidung: Der DB-Entwerfer muss Situationen in der realen Welt betrachten, um dies zu entscheiden.

Schlüssel: Minimalität (3)

- Weil der Schlüssel "`NACHNAME`" ausgeschlossen ist, kommt der zusammengesetzte Schlüssel "`VORNAME, NACHNAME`" wieder in Frage.

  Man muß natürlich auch prüfen, ob der Schlüssel "`VORNAME`" möglich ist, aber dieser ist im Beispielzustand ebenfalls nicht erfüllt.

- Der DB-Entwerfer muss nun herausfinden, ob es jemals zwei Studenten in der Vorlesung geben kann, die den gleichen Vor- und Nachnamen haben.

  Im Beispiel-Zustand gibt es solche Studenten nicht, aber die Integritätsbedingung muss für alle Zustände gelten.

Schlüssel: Minimalität (4)

- Natürliche Schlüssel können fast immer Ausnahmen haben. Sind diese Ausnahmen sehr selten, könnte man solche Schlüssel dennoch in Erwägung ziehen:
  - ◇ Nachteil: Tritt eine Ausnahme auf, muss man den Namen von einem der beiden Studenten in der DB ändern und alle Dokumente, die von der DB gedruckt werden, muss man wieder ändern.

    Nachdem ich 7 Jahre gelehrt hatte, trat das auf (in einer Vorlesung mit über 150 Studenten).
  - ◇ Vorteil: Man kann Studenten in Programmen durch ihren Vor- und Nachnamen identifizieren.

Schlüssel: Minimalität (5)

- Wenn der Entwerfer entscheidet, dass der Nachteil des Schlüssels "`VORNAME, NACHNAME`" größer als der Vorteil ist, könnte er versuchen, weitere Attribute hinzuzufügen.
- Aber die Kombination "`SID, VORNAME, NACHNAME`" ist uninteressant, weil "`SID`" schon ein Schlüssel ist.
- Entscheidet der Entwerfer jedoch, dass "`VORNAME, NACHNAME`" "eindeutig genug" ist, wäre dies minimal, auch wenn "`SID`" schon ein Schlüssel ist.

  Anzahl der Spalten eines Schlüssels ist für die Minimalität unwichtig.

Mehrere Schlüssel

- Eine Relation kann mehr als einen Schlüssel haben.
- Z.B. ist `SID` ein Schlüssel von `STUDENTEN` und "`VORNAME, NACHNAME`" ggf. ein weiterer Schlüssel.
- Ein Schlüssel wird zum "Primärschlüssel" ernannt.

  Der Primärschlüssel sollte aus einem kurzen Attribut bestehen, das möglichst nie verändert wird (durch Updates). Der Primärschlüssel wird in anderen Tabellen verwendet, die sich auf Zeilen dieser Tabelle beziehen. In manchen Systemen ist Zugriff über Primärschlüssel besonders schnell. Ansonsten ist die Wahl des Primärschlüssels egal.
- Die anderen sind "Alternativ-/Sekundär-Schlüssel".

  SQL verwendet den Begriff `UNIQUE` für alternative Schlüssel.

Schlüssel: Notation (1)

- Die Primärschlüssel-Attribute werden oft markiert, indem man sie unterstreicht:

$$R(\underline{A_1\colon D_1}, \ldots, \underline{A_k\colon D_k}, A_{k+1}\colon D_{k+1}, \ldots, A_n\colon D_n).$$

STUDENTEN

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

- Normalerweise werden die Attribute so angeordnet, daß der Primärschlüssel am Anfang steht.

Schlüssel: Notation (2)

- In SQL können Schlüssel folgendermaßen definiert werden:

```
CREATE TABLE STUDENTEN(
            SID         NUMERIC(3)      NOT NULL,
            VORNAME     VARCHAR(20)     NOT NULL,
            NACHNAME    VARCHAR(20)     NOT NULL,
            EMAIL       VARCHAR(80),
            PRIMARY KEY(SID),
            UNIQUE(VORNAME, NACHNAME))
```

- Die genaue Syntax wird in Kapitel 9 behandelt.

Schlüssel und Nullwerte

- Der Primärschlüssel darf nicht Null sein, andere Schlüssel sollten nicht Null sein.

  In SQL-89 und DB2 muss `NOT NULL` für jedes Attribut einer `PRIMARY KEY`- oder `UNIQUE`-Bedingung festgelegt werden. In SQL-92 und Oracle impliziert die "`PRIMARY KEY`"-Deklaration automatisch "`NOT NULL`", aber "`UNIQUE`" (für alternative Schlüssel) tut dies nicht. In Oracle kann es mehrere Zeilen mit einem Nullwert in einem `UNIQUE`-Attribut geben. In SQL Server darf nur eine Zeile Null sein.
  SQL-92 definiert drei verschiedene Semantiken für zusammengesetzte Schlüssel, die nur in manchen Attributen Nullwerte haben. Man sollte dies jedoch alles vermeiden.

- Es ist nicht akzeptabel, wenn schon die "Objektidentität" des Tupels unbekannt ist.

Schlüssel und Updates

- Es wird als schlechter Stil angesehen, wenn Schlüsselattribute geändert werden (mit Updates).

  Das würde die "Objektidentität" ändern. Besser: Tupel zuerst löschen und dann das Tupel mit neuen Werten einfügen.

- Aber SQL verbietet dies nicht.

  Der Standard enthält sogar Klauseln, die festlegen, was mit Fremdschlüsseln passieren soll, wenn sich der referenzierte Schlüsselwert ändert.

Der schwächste Schlüssel

- Ein Schlüssel bestehend aus allen Spalten der Tabelle fordert nur, dass es nie zwei verschiedene Zeilen gibt, die in allen Spaltenwerten übereinstimmen.

  Theoretisch sind Relationen Mengen: Dann wäre dieser Schlüssel keine Einschränkung. In der Praxis sind Relationen jedoch zunächst Multimengen. Dieser Schlüssel verbietet nun doppelte Zeilen (und bringt damit Theorie und Praxis wieder zusammen).

- Empfehlung: Um Duplikate auszuschließen, sollte man immer mindestens einen Schlüssel für jede Relation festlegen.

  Gibt es keinen anderen Schlüssel, sollte der Schlüssel gewählt werden, der aus allen Attributen der Relation besteht.

Schlüssel: Zusammenfassung

- Bestimmte Spalten als Schlüssel zu deklarieren ist etwas einschränkender als die eindeutige Identifikations-Eigenschaft:
  - ◇ Nullwerte sind zumindest im Primärschlüssel ausgeschlossen.
  - ◇ Man sollte Updates vermeiden, zumindest beim Primärschlüssel.
- Die Eindeutigkeit ist jedoch die Hauptaufgabe eines Schlüssels. Alles andere ist sekundär.

Übungen (1)

- Wählen Sie einen Schlüssel aus:

REZEPT		
NAME	ZUTAT	MENGE
Lebkuchen	Eier	2
Lebkuchen	Zucker	200g
Mürbeteig	Butter	250g
Mürbeteig	Zucker	100g

- Geben Sie ein Beispiel für eine Einfügung an, die den Schlüssel verletzten würde:

- Könnte “MENGE” auch als Schlüssel dienen?

Übungen (2)

- Betrachten Sie meinen Terminkalender:

TERMINE				
DATUM	START	ENDE	RAUM	AUFGABE
22.11.05	10:15	11:45	307	DB I halten
22.11.05	14:00	15:00	313	Prüfung Herr Meier
22.11.05	16:00	18:00	507	Forschungstreffen

- Was wären korrekte Schlüssel?
- Beispiel für einen nicht-minimalen Schlüssel?
- Werden weitere Integritätsbedingungen benötigt?

  Kann es ungültige Zustände geben, auch wenn Schlüsselbed. erfüllt?

Inhalt

Fremdschlüssel (1)

- Das relationale Modell hat keine expliziten Relationships, Verknüpfungen oder Zeiger.
- Schlüsselattributwerte identifizieren ein Tupel.

  Sie sind "logische Adressen" der Tupel.
- Um sich in einer Relation S auf Tupel von R zu beziehen, fügt man den Primärschlüssel von R zu den Attributen von S hinzu.

  Solche Attributwerte sind "logische Zeiger" auf Tupel in R.
- Z.B. hat die Tabelle `BEWERTUNGEN` das Attribut `SID`, welches Primärschlüsselwerte von `STUDENTEN` enthält.

Fremdschlüssel (2)

`SID` in `BEWERTUNGEN` ist ein Fremdschlüssel, der `STUDENTEN` referenziert:

STUDENTEN

<u>SID</u>	VORNAME	NACHNAME	···
101	Lisa	Weiss	···
102	Michael	Grau	···
103	Daniel	Sommer	···
104	Iris	Winter	···

BEWERTUNGEN

<u>SID</u>	<u>ATYP</u>	<u>ANR</u>	PUNKTE
101	H	1	10
101	H	2	8
102	H	1	9
102	H	2	9
103	H	1	5
105	H	1	7

?
Fehler

Die hier benötigte Bedingung ist, dass jeder `SID`-Wert in `BEWERTUNGEN` auch in `STUDENTEN` auftaucht.

Fremdschlüssel (3)

- Die Deklaration von `SID` in `BEWERTUNGEN` als Fremdschlüssel, der auf `STUDENTEN` verweist, bedeutet in der Logik (Tupelkalkül):

  $$\forall\ \texttt{bewertungen B}:\ \exists\ \texttt{studenten S}:\ \texttt{S.sid} = \texttt{B.sid}$$
- Das DBMS weist dann jeden Versuch zurück, eine Bewertung für einen nicht existierenden Studenten einzufügen.

  Man bekommt eine Fehlermeldung und der Datenbankzustand bleibt unverändert. D.h. der auf der letzten Folie gezeigte Zustand kann dann nicht vorkommen: Es gibt keine Möglichkeit, die Daten so einzugeben.

Fremdschlüssel (4)

- Die Fremdschlüsselbedingung

  "`BEWERTUNGEN.SID` $\longrightarrow$ `STUDENTEN`"

  fordert, daß die Menge der Werte in der Spalte `SID` der Tabelle `BEWERTUNGEN` immer eine Teilmenge der Primärschlüsselwerte in `STUDENTEN` ist.

  Somit ist die Menge der `SID`-Werte in `STUDENTEN` eine Art "dynamische Domain" für `SID` in `BEWERTUNGEN`.
- In relationaler Algebra liefert die Projektion $\pi_{\texttt{SID}}$ die Werte der Spalte `SID`. Dann lautet die Fremdschlüsselbedingung:

  $$\pi_{\texttt{SID}}(\texttt{BEWERTUNGEN}) \subseteq \pi_{\texttt{SID}}(\texttt{STUDENTEN}).$$

Fremdschlüssel (5)

- Die Fremdschlüsselbedingung stellt sicher, dass es für jedes Tupel t in BEWERTUNGEN ein Tupel u in STUDENTEN gibt, so dass t.SID = u.SID.

  Paare solcher Tupel t und u kann man durch eine Operation der relationalen Algebra ("Join") zusammenbringen. Das entspricht der Dereferenzierung von Zeigern in anderen Modellen. Ohne Fremdschlüsselbedingung könnte es "Zeiger" geben, die ins Nichts zeigen. SQL-Anfragen stürzen in diesem Fall jedoch nicht ab: Tupel ohne "Join-Partner" werden in einer Anfrage mit einem Join eliminiert.

- Die Schlüsselbedingung in STUDENTEN stellt sicher, dass es maximal ein solches Tupel u gibt.

  Zusammen folgt, dass jedes Tupel t in BEWERTUNGEN genau ein Tupel u in STUDENTEN referenziert.

Fremdschlüssel (6)

- Die Erzwingung von Fremdschlüsselbedingungen sichert die "referentielle Integrität" der Datenbank.

  Statt "Fremdschlüsselbedingung" kann man auch "referentielle Integritätesbedingung" sagen.

- Fremdschlüssel implementiert "eins-zu-viele"-Relationship: Ein Student hat viele Aufgaben gelöst.
- Die Tabelle BEWERTUNGEN, die den Fremdschlüssel enthält, wird "Kindtabelle" der referentiellen Integritätsbedingung genannt und die referenzierte Tabelle STUDENTEN ist die "Elterntabelle".

Fremdschlüssel (7)

- Die Tabelle BEWERTUNGEN enthält noch einen Fremdschlüssel, der die gelöste Aufgabe referenziert.
- Aufgaben werden durch eine Kategorie und eine Nummer (ATYP und ANR) identifiziert:

BEWERTUNGEN

SID	ATYP	ANR	PUNKTE
101	H	1	10
101	H	2	8
101	Z	1	12
102	H	1	9
⋮	⋮	⋮	⋮

AUFGABEN

<u>ATYP</u>	<u>ANR</u>	···	MAXPT
H	1	···	10
H	2	···	10
Z	1	···	14

Fremdschlüssel (8)

- Eine Tabelle mit zusammengesetztem Schlüssel (wie AUFGABEN) muss mit einem Fremdschlüssel referenziert werden, der die gleiche Spaltenanzahl hat.
- Die zugehörigen Spalten müssen den gleichen Datentyp haben.
- Es ist nicht nötig, dass die zugehörigen Spalten den gleichen Namen haben.
- Im Beispiel erfordert der Fremdschlüssel, dass jede Kombination von ATYP und ANR, die in BEWERTUNGEN vorkommt, auch in AUFGABEN existiert.

Fremdschlüssel (9)

- Spalten werden nach der Position in der Deklaration zugeordnet: Ist (`VORNAME, NACHNAME`) der Schlüssel und (`NACHNAME, VORNAME`) der Fremdschlüssel, werden Einfügungen meist Fehler geben.

  Sind die Datentypen von `VORNAME` und `NACHNAME` sehr verschieden, kann der Fehler schon bei der Deklaration des Fremdschlüssels erkannt werden. Aber manche Systeme erfordern nur "kompatible" Datentypen und das ist bereits mit `VARCHAR`-Typen verschiedener Länge erfüllt.

- Nur (ganze) Schlüssel können referenziert werden.

  Nicht beliebige Attribute, und auch nicht nur Teile eines zusammengesetzten Schlüssels. Normalerweise sollte man nur den Primärschlüssel referenzieren, aber SQL erlaubt auch alternative Schlüssel.

Fremdschlüssel: Notation (1)

- In der Attributlisten-Notation können Fremdschlüssel durch einen Pfeil und den Namen der referenzierten Tabelle markiert werden. Bei zusammengesetzten Fremdschlüsseln braucht man Klammern:

```
BEWERTUNGEN(SID → STUDENTEN,
            (ATYP, ANR) → AUFGABEN, PUNKTE)
STUDENTEN(SID, VORNAME, NACHNAME, EMAIL)
AUFGABEN(ATYP, ANR, THEMA, MAXPT)
```

- Da normalerweise nur Primärschlüssel referenziert werden, ist es nicht nötig, die zugehörigen Attribute der referenzierten Tabelle anzugeben.

Fremdschlüssel: Notation (2)

- Obiges Beispiel ist untypisch, weil alle Fremdschlüssel Teil eines Schlüssels sind. Das ist nicht nötig:

```
MODULKATALOG(MNR, TITEL, BESCHREIBUNG)
MODULANGEBOT(ANR, MNR → MODULKATALOG, SEM,
             (DOZ_VORNAME, DOZ_NACHNAME)→DOZENT)
DOZENT(VORNAME, NACHNAME, BUERO, TEL)
```

  In diesem Beispiel sind auch die Namen von Fremdschlüssel-Attributen und referenzierten Attributen verschieden. Das ist möglich.

- Manche markieren Fremdschlüssel durch gestricheltes Unterstreichen oder einen Strich oben. Dann ist aber die referenzierte Tabelle nicht klar.

Fremdschlüssel: Notation (3)

- In SQL können Fremdschlüssel wie folgt deklariert werden:

```
CREATE TABLE BEWERTUNGEN(
             SID         NUMERIC(3)   NOT NULL,
             ATYP        CHAR(1)      NOT NULL,
             ANR         NUMERIC(2)   NOT NULL,
             PUNKTE      NUMERIC(4,1) NOT NULL,
             PRIMARY KEY(SID, ATYP, ANR),
             FOREIGN KEY(SID)
                         REFERENCES STUDENTEN,
             FOREIGN KEY(ATYP,ANR)
                         REFERENCES AUFGABEN)
```

Fremdschlüssel: Notation (4)

- In der Tabellen-Notation können Fremdschlüssel z.B. folgendermaßen deklariert werden:

BEWERTUNGEN	SID	ATYP	ANR	PUNKTE
Typ	NUMERIC(3)	CHAR(1)	NUMERIC(2)	NUMERIC(2)
Null	N	N	N	N
Referenz	STUDENTEN	AUFGABEN	AUFGABEN	

- Zusammengesetzte Fremdschlüssel sind wieder ein Problem.

  Sollte die obige Notation unklar sein, gibt man die Namen der referenzierten Spalten mit an oder verteilt die Information über Fremdschlüssel auf mehrere Zeilen. In seltenen Fällen können sich Fremdschlüssel auch überlappen. Dann sind immer mehrere Zeilen nötig.

Fremdschlüssel: Notation (5)

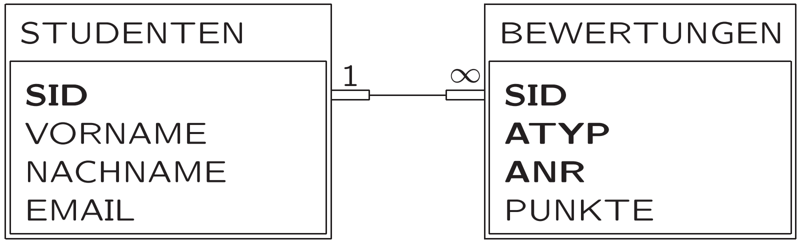
- In der Oracle-DBA-Prüfung wird folgende Notation für relationale Schemata verwendet, z.B. für MUSIC_PIECE(<u>PNO</u>, PNAME, CNO→COMPOSER):

Instance Chart for Table MUSIC_PIECE			
Column Name:	PNO	PNAME	CNO
Key Type:	PK		FK
Nulls/Unique:	NN, U	NN	
FK Table:			COMPOSER
FK Column:			CNO
Datatype:	NUMBER	VARCHAR	NUMBER
Length:	4	40	2

FK: “foreign key”, NN: “not null”, PK: “primary key”, U: “unique”.

Fremdschlüssel: Notation (6)

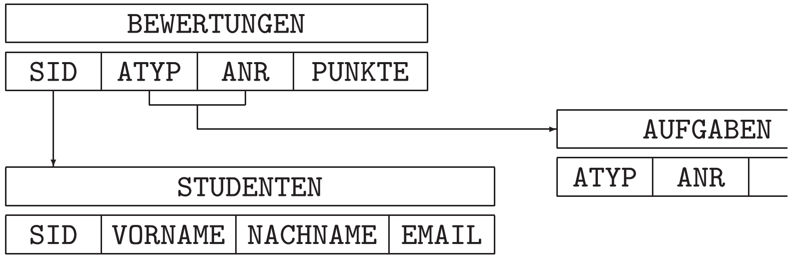
- MS Access stellt Fremdschlüssel in “Relationships” dar (Primärschlüsselattribute fettgedruckt):



- “1” / “∞” symbolisieren das eins-zu-viele-Relationship.

Fremdschlüssel: Notation (7)

- Natürlich kann man auch Pfeile verwenden:



- Achtung: Manche Leute zeichnen den Pfeil auch in die entgegengesetzte Richtung!

  Z.B. im Oracle-DBA-Examen. Man muss genau auf den gegebenen DB-Zustand achten.

Mehr über Fremdschlüssel (1)

Fremdschlüssel und Nullwerte:

- Solange kein "Not Null"-Constraint spezifiziert ist, können Fremdschlüssel Null sein.
- Die Fremdschlüsselbedingung ist sogar dann erfüllt, wenn die referenzierenden Attribute "Null" sind. Das entspricht einem "nil"-Zeiger.
- Wenn ein Fremdschlüssel (FS) mehrere Attribute hat, sollten entweder alle oder keines Null sein.

  Aber Oracle und SQL-92 erlauben teilweise definierte Fremdschlüssel. In Oracle ist die Bedingung erfüllt, wenn mindestens ein FS-Attribut Null ist. Der SQL-92-Standard definiert 3 verschiedene Semantiken.

Mehr über Fremdschlüssel (2)

Gegenseitige Referenzierung:

- Es ist möglich, dass Eltern- und Kindtabelle die gleiche sind, z.B.

  ANG(<u>ANGNR</u>, ANAME, JOB, CHEF^{o}→ANG, ABTNR→ABT)
  PERSON(<u>NAME</u>, MUTTER^{o}→PERSON, VATER^{o}→PERSON)

- Zwei Relationen können sich gegenseitig referenzieren, z.B.

  ANGESTELLTE(<u>ANGNR</u>, ..., ABT→ABTEILUNGEN)
  ABTEILUNGEN(<u>ABTNR</u>, ..., CHEF^{o}→ANGESTELLTE).

- Übung/Rätsel: Wie kann man Tupel einfügen?

Bitte merken:

- Fremdschlüssel (FS) sind selbst keine Schlüssel!

  Die Attribute eines Fremdschlüssels können Teil eines Schlüssels sein, aber das ist eher die Ausnahme. Die FS-Bedingung hat nichts mit einer Schlüsselbedingung zu tun. Für manche Autoren ist jedoch jedes Attribut, das Tupel identifiziert (nicht unbedingt in der gleichen Tabelle), ein Schlüssel. Dann wären FS Schlüssel, aber normale Schlüssel brauchen dann immer einen Zusatz ("Primär-/Alternativ-").

- Nur Schlüssel einer Relation können referenziert werden, keine beliebigen Attribute.
- Enthält die referenzierte Relation zwei Attribute, muss der FS auch aus zwei Attributen bestehen (gleiche Datentypen und gleiche Reihenfolge).

FS und Updates (1)

Diese Operationen können Fremdschlüssel verletzen:

- Einfügen in Kindtabelle `BEWERTUNGEN` ohne passendes Tupel in Elterntabelle `STUDENTEN`.
- Löschen aus Elterntabelle `STUDENTEN`, wenn das gelöschte Tupel noch referenziert wird.
- Änderung des FS `SID` in der Kindtabelle `BEWERTUNGEN` in einen Wert, der nicht in `STUDENTEN` vorkommt.

  Wird normalerweise wie Einfügen behandelt.

- Änderung des Schlüssels `SID` in `STUDENTEN`, wenn der alte Wert noch referenziert wird.

FS und Updates (2)

Man beachte:

- Löschungen aus der Kindtabelle BEWERTUNGEN und Einfügungen in die Elterntabelle STUDENTEN können nie zu Verletzungen der Fremdschlüsselbedingung führen.

  Daher muß das DBMS bei diesen Operationen die Bedingung nicht überprüfen.

Reaktionen auf Einfügung, die FS-Bedingung verletzt:

- Das Einfügen wird abgelehnt (Fehlermeldung). Der DB-Zustand bleibt unverändert.

FS und Updates (3)

Reaktionen auf Löschen referenzierter Schlüsselwerte:

- Die Löschung wird abgelehnt. Zustand unverändert.
- Kaskadierendes (rekursives) Löschen: Alle Tupel in BEWERTUNGEN, die das gelöschte Tupel in STUDENTEN referenzierten, werden mit gelöscht.
- Der Fremdschlüssel wird auf Null gesetzt.

  In SQL-92 enthalten, in DB2 unterstützt, aber nicht in Oracle.
- Der Fremdschlüssel wird auf einen vorher definierten Default-Wert gesetzt.

  In SQL-92 enthalten, aber nicht in Oracle oder DB2.

FS und Updates (4)

Reaktionen auf Updates referenzierter Schlüsselwerte:

- Änderung wird abgelehnt. Der DB-Zustand bleibt unverändert.

  DB2 und Oracle unterstützen nur diese Alternative des SQL-92-Standards. Auf jeden Fall ist die Änderung von Schlüsselattributen schlechter Stil.
- Kaskadierender Update.

  D.h. das Attribut SID in BEWERTUNGEN wird genauso geändert, wie das Attribut SID in STUDENTEN geändert wurde.
- Fremdschlüssel wird Null gesetzt.
- Fremdschlüssel wird auf Default-Wert gesetzt.

FS und Updates (5)

- Bei der Definition eines Fremdschlüssels muss entschieden werden, welche Reaktion die beste ist.
- Default ist die erste Alternative ("Keine Aktion").
- Für das Löschen aus der Elterntabelle sollten alle Systeme kaskadierendes Löschen unterstützen.

  Das ist eine Art aktive Integritätserzwingung: Das System lehnt die Änderung nicht ab, sondern macht andere Änderungen, um den DB-Zustand zu reparieren.
- Andere Alternativen gibt es bis jetzt nur in wenigen Systemen.

Inhalt

Beispiel (1)

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

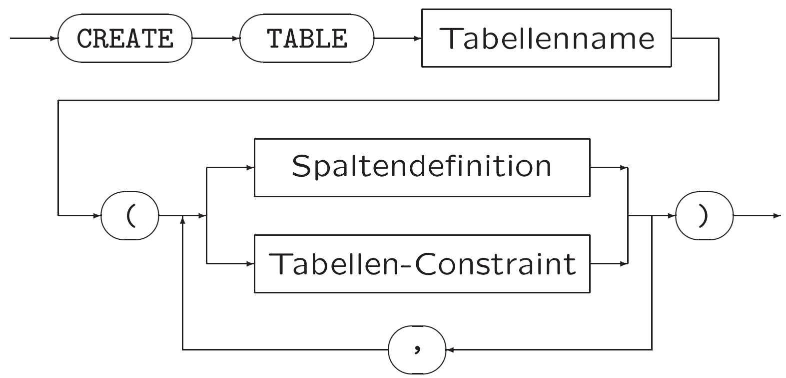
Beispiel (2)

- Das CREATE TABLE-Statement in SQL definiert:
  - Tabellenname
  - Spalten und ihre Datentypen
  - Constraints (NOT NULL, (Fremd-)Schlüssel, CHECK)
- Z.B. STUDENTEN(SID, VORNAME, NACHNAME, EMAIL^o):

```
CREATE TABLE STUDENTEN (
       SID      NUMERIC(3)  NOT NULL PRIMARY KEY
                            CHECK(SID > 0),
       VORNAME  VARCHAR(20) NOT NULL,
       NACHNAME VARCHAR(20) NOT NULL,
       EMAIL    VARCHAR(128),
       UNIQUE(VORNAME, NACHNAME) )
```

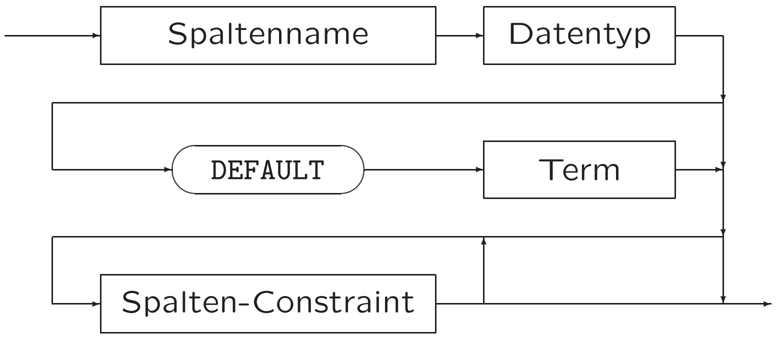

CREATE TABLE: Syntax (1)

CREATE TABLE-Statement:



CREATE TABLE: Syntax (2)

Spaltendefinition:



Constraints: Überblick (1)

- In SQL kann man Constraints als Spalten-Constraints oder Tabellen-Constraints definieren.
- Spalten-Constraints sind Bedingungen, die sich nur auf eine Spalte beziehen. Tabellen-Constraints können sich auf mehrere Spalten beziehen.
- Spalten-Constraints (außer vielleicht `NOT NULL`) können auch als "Tabellen-Constraints" formuliert werden, aber die Syntax von Spalten-Constraints ist etwas einfacher.

Intern werden Spalten-Constraints in Tabellen-Constraints übersetzt.

Constraints: Überblick (2)

- Spalten-Constraints werden in der Spaltendefinition festgelegt (nur durch Leerzeichen getrennt).
- Spalten-Constraints im Beispiel:

```
CREATE TABLE STUDENTEN (
        SID      NUMERIC(3)   NOT NULL PRIMARY KEY
                              CHECK(SID>0),
        VORNAME  VARCHAR(20)  NOT NULL,
        NACHNAME VARCHAR(20)  NOT NULL,
        EMAIL    VARCHAR(128),
        UNIQUE(VORNAME, NACHNAME) )
```


Constraints: Überblick (3)

- Tabellen-Constraints werden durch ein Komma von den Spaltendefinitionen und voneinander getrennt:
- Tabellen-Constraint im Beispiel:

```
CREATE TABLE STUDENTEN (
        SID      NUMERIC(3)   NOT NULL PRIMARY KEY
                              CHECK(SID>0),
        VORNAME  VARCHAR(20)  NOT NULL,
        NACHNAME VARCHAR(20)  NOT NULL,
        EMAIL    VARCHAR(128),
        UNIQUE(VORNAME, NACHNAME) )
```

Constraints: Überblick (4)

- Gleiches Beispiel, aber Spalten-Constraints (außer `NOT NULL`) durch Tabellen-Constraints ersetzt:

```
CREATE TABLE STUDENTEN (
        SID      NUMERIC(3)  NOT NULL,
        VORNAME  VARCHAR(20) NOT NULL,
        NACHNAME VARCHAR(20) NOT NULL,
        EMAIL    VARCHAR(128),
        PRIMARY KEY (SID),
        CHECK(SID > 0),
        UNIQUE(VORNAME, NACHNAME) )
```


Spalten-Constraints (1)

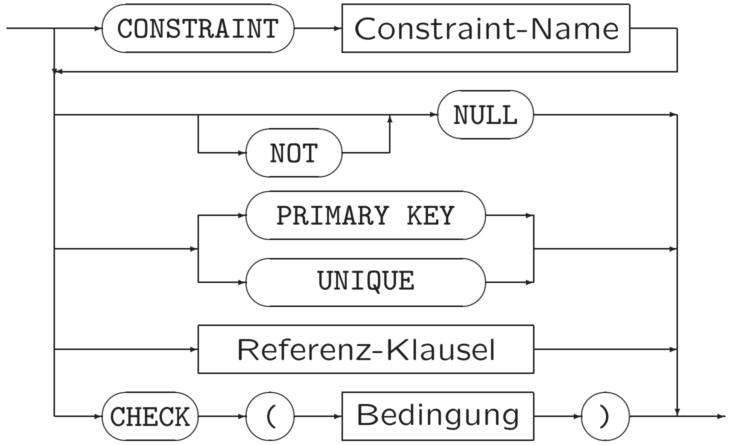
- Es gibt fünf Arten von Spalten-Constraints:
  - ◇ `NOT NULL`: Keine Nullwerte in dieser Spalte.
  - ◇ `PRIMARY KEY`: Diese Spalte ist der Primärschlüssel der Tabelle.
  - ◇ `UNIQUE`: Diese Spalte ist ein Alternativschlüssel.
  - ◇ `REFERENCES` T: Die Spalte ist ein Fremdschlüssel.

    D.h. Werte dieser Spalte müssen in der Primärschlüssel-Spalte der Tabelle T auftauchen.
  - ◇ `CHECK` (C): Werte der Spalte müssen C erfüllen.

    C ist eine Bedingung wie in der `WHERE`-Klausel, nur mit gewissen Einschränkungen (siehe unten).

Spalten-Constraints (2)

Spalten-Constraint:



Spalten-Constraints (3)

- Man kann "`NULL`" als Spalten-Constraint schreiben, um zu betonen, dass Nullwerte erlaubt sind.

  Das ist jedoch keine richtige Bedingung und ist sowieso der Default.
- `PRIMARY KEY` impliziert `NOT NULL`.

  DB2 verlangt jedoch, dass `NOT NULL` zusätzlich festgelegt wird. In MySQL muss `NOT NULL` explizit deklariert werden, wenn der Primärschlüssel als Tabellen-Constraint definiert wurde.
- Es darf nur einen Primärschlüssel je Tabelle geben.
- `UNIQUE` impliziert `NOT NULL` nicht.

  In DB2 kann `UNIQUE` nur zusammen mit `NOT NULL` verwendet werden.

Spalten-Constraints (4)

- SQL-86 unterstützte nur [`NOT NULL` [`UNIQUE`]].
- Außerdem war `UNIQUE` in vielen Systemen nicht implementiert.
- In älterem Code wurde oft "`CREATE UNIQUE INDEX`" verwendet, um Schlüsselbedingungen zu erzwingen.

  Index: Datenstruktur zum schnellen Zugriff auf Zeilen mit gegebenem Spaltenwert. Ein "`UNIQUE INDEX`" kann nur eine Zeile pro Spaltenwert aufnehmen (intern zur Überwachung vom Schlüsseln eingesetzt).
- 1989 wurde der Standard um ein optionales "Integrity Enhancement Feature" erweitert (SQL-89).

  Dies enthielt die obigen Konstrukte.

CHECK-Constraints (1)

- Die Bedingung C eines `CHECK`-Constraints sieht wie eine `WHERE`-Bedingung ohne Unteranfragen aus. Dabei sind Funktionen wie z.B. `SYSDATE`, die sich später ändern können, ausgeschlossen.

  In Spalten-Constraints kann nur diese eine Spalte verwendet werden. Ansonsten verwendet man Tabellen-Constraints (siehe unten).

CHECK-Constraints (2)

- Grundidee effizienter Constraint-Überprüfung: Bedingung war vor dem Update erfüllt.
- Das DBMS prüft nur geänderte/eingefügte Zeilen.

  Da solche Constraints im leeren DB-Zustand gelten und das System sicherstellt, dass Updates deren Gültigkeit nicht zerstören, folgt per Induktion, dass sie in jedem DB-Zustand gelten. Das erklärt auch, warum `SYSDATE` verboten ist: Constraints könnten ohne Update ungültig werden.
- `CHECK`-Constraints werden von MySQL und Access nicht unterstützt.

CHECK-Constraints (3)

- Oracle, SQL Server und DB2 schließen Unteranfragen in `CHECK`-Constraints aus.

  Somit ist eine grundlegende Einschränkung in `CHECK`-Constraints, dass sie für jedes einzelne Tupel getrennt auswertbar sein müssen.
- SQL-92 erlaubt Unteranfragen innerhalb von `CHECK`-Constraints, aber dann ist die Integritätsprüfung schwierig/ineffizient.

  Wird eine in der Unteranfrage verwendete Tabelle geändert, ist es im allgemeinen schwierig herauszufinden, für welche der Zeilen die `CHECK`-Bedingung erneut überprüft werden muss. Eine einfache Lösung wäre, alle Zeilen zu überprüfen, aber dann führt jede kleine Änderung zu einer langen Integritätsprüfung.

CHECK-Constraints (4)

- Wären Unteranfragen unter `CHECK` implementiert, könnte ein Fremdschlüssel so formuliert werden:

```
CREATE TABLE BEWERTUNGEN( Nicht implementiert!
   SID NUMERIC(3)
       CHECK(SID IN (SELECT SID FROM STUDENTEN)),
   ... )
```

- Wenn ein `STUDENTEN`-Tupel t gelöscht oder seine `SID` geändert wird, betrifft dies nur Tupel in `BEWERTUNGEN`, die die (alte) SID von t haben.

  Es wäre dann also nicht nötig, die `CHECK`-Bedingung in `BEWERTUNGEN` für alle Tupel zu überprüfen.

Constraints und Nullwerte

- Integritätsbedingungen gelten als erfüllt, wenn das Ergebnis den Wahrheitswert "unbekannt" hat.

  Die Definitionen für Schlüssel und Fremdschlüssel, die aus mehreren Spalten bestehen, von denen nur einige Null sind, sind kompliziert und systemabhängig.

- Z.B. kann bei dieser Deklaration die E-Mail-Adresse Null sein. Ist sie es nicht, muss sie "`@`" enthalten:

```
CREATE TABLE STUDENTEN(
   ...,
   EMAIL VARCHAR(128)
         CHECK(EMAIL LIKE '%@%'))
```


Constraint-Namen (1)

- Einem Constraint kann ein Name gegeben werden, indem man "`CONSTRAINT` ⟨`Name`⟩" voranstellt.

  MySQL erlaubt Constraint-Namen nur für Tabellen-Constraints. Es scheint sie jedoch sowieso gleich wieder zu vergessen.

- Constraint-Namen müssen innerhalb eines Schemas eindeutig sein, wohingegen Spaltennamen nur eindeutig für eine Tabelle sein müssen.

  In DB2 müssen Constraint-Namen nur für eine Tabelle eindeutig sein. In DB2 können `NOT NULL`-Constraints nicht benannt werden.

- Definieren Sie immer Namen (außer für `NOT NULL`). Sonst wählt das System Namen wie "`SYS_C036`".

Constraint-Namen (2)

- Wenn ein Constraint verletzt ist, wird der Name ausgegeben: System-generierte Namen ergeben unverständliche Fehlermeldungen.

  Die Fehlermeldung für Not Null-Constraints ist meist klar:
  `ORA-01400: cannot insert NULL into (USER.TABLE.COLUMN)`
  Gibt es mehrere Schlüssel, ist dies schon unklar:
  `ORA-00001: unique constraint (BRASS.SYS_C007916) violated`
  Für einen Fremdschlüssel bekommt man diese Fehlermeldung:
  `ORA-02291: integrity constraint (BRASS.SYS_C007914) violated - parent key not found.`
  Für Check-Constraints erhält man diese Nachricht:
  `ORA-02290: check constraint (BRASS.SYS_C007915) violated.`

- Namen erleichtern das Löschen von Constraints.

Constraint-Namen (3)

- Beispiel mit Constraint-Namen:

```
CREATE TABLE STUDENTEN (
       SID      NUMERIC(3) NOT NULL
                CONSTRAINT SID_MUSS_POSITIV_SEIN
                           CHECK(SID > 0)
                CONSTRAINT STUDENTEN_SCHLUESSEL
                           PRIMARY KEY,
       VORNAME  VARCHAR(20) NOT NULL,
       NACHNAME VARCHAR(20) NOT NULL,
       EMAIL    VARCHAR(128),
       CONSTRAINT STUDENTENNAMEN_EINDEUTIG
                  UNIQUE(VORNAME, NACHNAME) )
```


Tabellen-Constraints (1)

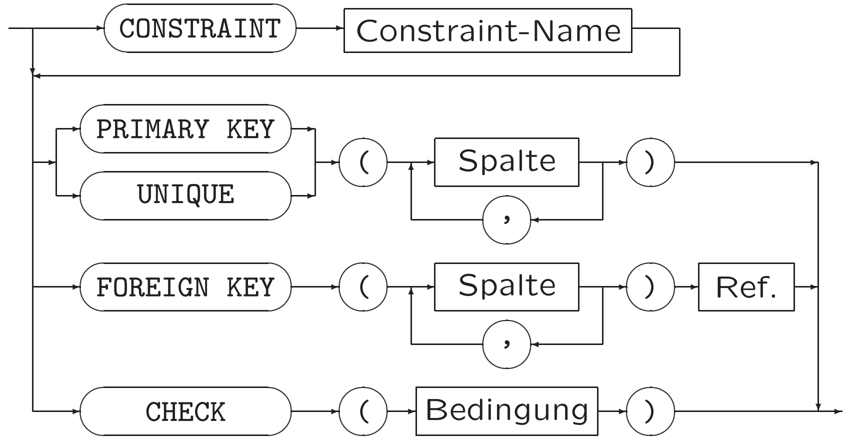
- Tabellen-Constraints werden benötigt, wenn ein (Fremd-)Schlüssel mehrere Spalten hat oder sich eine CHECK-Bedingung auf mehrere Spalten bezieht.

  Constraints, die sich nur auf eine Spalte beziehen, können als Spalten- oder Tabellen-Constraint definiert werden.

- Die vier Arten von Tabellen-Constraints sind:
  - ◇ PRIMARY KEY(A_1, ..., A_2)
  - ◇ UNIQUE(A_1, ..., A_2)
  - ◇ FOREIGN KEY(A_1, ..., A_2) REFERENCES R
  - ◇ CHECK(C)

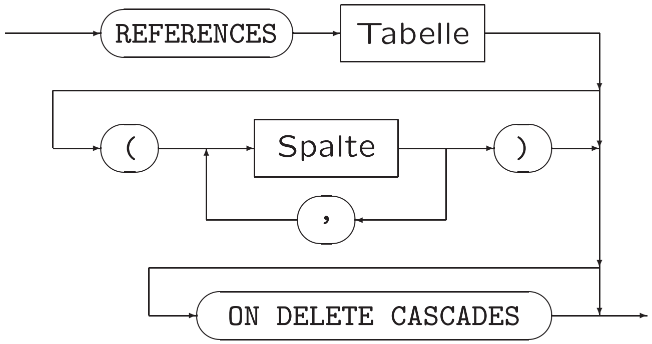
Tabellen-Constraints (2)

Tabellen-Constraint:



Fremdschlüssel (1)

Referenz-Klausel (Ref.):



Fremdschlüssel (2)

- Die Referenz-Klausel steht in Spalten-Constraints hinter der Spalte und in Tabellen-Constraints hinter der `FOREIGN KEY`-Deklaration.
- Es ist möglich, die referenzierten Spaltennamen anzugeben, z.B. `REFERENCES STUDENTEN(SID)`.

  Es können jedoch nur Schlüssel (`PRIMARY KEY` oder `UNIQUE`) referenziert werden. Werden keine Spalten genannt, wird der Primärschlüssel referenziert. Es macht selten Sinn, Alternativschlüssel zu referenzieren.
- Fremdschlüssel und referenzierter Schlüssel müssen die gleiche Spaltenanzahl und zusammengehörige Spalten den gleichen Datentyp haben.

Fremdschlüssel (3)

- MySQL prüft keine Fremdschlüssel, erlaubt aber die Deklaration von Fremdschlüsseln in `CREATE TABLE`.
- Beispiel: `BEWERTUNGEN(SID→STUDENTEN, ...)`
  - ◇ In SQL-92, Oracle, DB2 (nicht in SQL Server, Access) kann man verlangen, dass die Bewertungen eines Studenten automatisch mitgelöscht werden, wenn man ihn löscht:

    `REFERENCES STUDENTEN ON DELETE CASCADES`
  - ◇ Sonst lehnt das DBMS die Löschung eines Studenten ab, von dem es noch Bewertungen gibt.

Fremdschlüssel (4)

- Gegeben sei eine DB mit Vorlesungsdaten:
  `VORLESUNGEN(..., DOZENT`^{o}`→PROFESSOREN, ...)`.
- In SQL-92 und DB2 (nicht in Oracle, SQL Server, Access) kann man festlegen, dass `DOZENT` auf Null gesetzt wird, wenn der Professor gelöscht wird. Die Vorlesungen bleiben dann in der DB.
- Dazu schreibt man: "`ON DELETE SET NULL`".
- In SQL-92 kann man auch "`SET DEFAULT`" wählen. Das wird in keinem der fünf Systeme unterstützt.

Fremdschlüssel (5)

- In SQL-92 kann man auch die Reaktion auf Änderungen des Schlüssels von `PROFESSOREN` festlegen (Namensänderung).
- Dazu schreibt man "`ON UPDATE ...`".
  Es wird von keinem der fünf Systeme unterstützt.

  DB2 versteht "`ON UPDATE`" mit den Parametern "`NO ACTION`" und "`RESTRICT`", aber das Ablehnen der Updates von referenzierten Schlüsselwerten ist der Default.

Default-Spaltenwerte (1)

- Im Befehl zum Erstellen neuer Zeilen (INSERT) muss man nicht für alle Spalten Werte definieren.

  INSERT wird in Kapitel 7 erläutert. Wird eine Zeile über eine Sicht eingefügt, kann es sein, dass man gar nicht für alle Spalten Werte angeben kann.

- Normalerweise wird in fehlenden Spalten ein Nullwert eingesetzt.
- Es ist jedoch möglich, einen Default-Wert festzulegen, der in Spalten, für die kein Wert gegeben ist, gespeichert wird.

Default-Spaltenwerte (2)

- Z.B. sollen alle Aufgaben MAXPT = 10 haben, wenn der INSERT-Befehl keinen Wert für MAXPT enthält:

```
CREATE TABLE AUFGABEN(
        ...
        MAXPT NUMERIC(2) DEFAULT 10
                         NOT NULL
                         CHECK(MAXPT >= 0))
```

- "DEFAULT SYSDATE" speichert das aktuelle Datum in der Spalte (Datum der Zeilenerstellung).

  "SYSDATE" funktioniert nur in Oracle. In SQL Server und SQL-92: "CURRENT_TIMESTAMP". In DB2: "CURRENT TIMESTAMP". MySQL verwendet für die erste Spalte des Typs "TIMESTAMP" automatisch das aktuelle Datum als Defaultwert (man kann dies nicht explizit festlegen).

Default-Spaltenwerte (3)

- Manchmal können verschiedene Nutzer neue Zeilen in einer Tabelle einfügen. Mit "DEFAULT USER" wird der Name des aktuellen Nutzers in einer Spalte gespeichert (z.B. EINGEGEBEN_VON).
- Auf diese Weise kennt man den Nutzer, der für das Einfügen einer bestimmten Zeile verantwortlich ist.

  "USER" war schon im SQL-86-Standard enthalten und müsste sehr portabel sein. Aber MySQL und Access unterstützen "USER" nicht.

Default-Spaltenwerte (4)

- INSERT-Rechte können selektiv für bestimmte Spalten an Datenbank-Benutzer vergeben werden.
- Der Benutzer kann dann für die anderen Spalten bei der Tupel-Einfügung keine Werte angeben.
- Daher kann er/sie dann den DEFAULT-Wert nicht überschreiben.

  Natürlich muss für die UPDATE-Rechte die gleiche Einschränkung gelten. Der Benutzername oder das aktuelle Datum werden dann immer so gespeichert, wie es im Default-Wert definiert ist.

Default-Spaltenwerte (5)

- `DEFAULT` war in SQL-86 nicht enthalten und Access unterstützt es nicht.
- In SQL-92 darf der Default-Wert nur sein:
  - ◇ eine Konstante,
  - ◇ eine Funktion ohne Argumente oder

    Genauer: `USER`, `CURRENT_USER`, `SESSION_USER`, `SYSTEM_USER`, `CURRENT_DATE`, `CURRENT_TIME[(...)]`, `CURRENT_TIMESTAMP[(...)]`
  - ◇ `NULL`.
- Oracle akzeptiert jeden Term ohne Spaltennamen.
- MySQL erlaubt nur Konstanten als Default-Werte.

Default-Spaltenwerte (6)

- "`DEFAULT NULL`" wird verwendet, wenn kein Default-Wert explizit definiert ist.
- Somit ist es unmöglich, eine Zeile ohne festgelegten Wert für eine Spalte, die `NOT NULL` ist und keinen definierten Default-Wert hat, einzufügen.
- MySQL vermeidet dies, indem z.B. der leere String oder `0` als Default-Wert für `NOT NULL`-Spalten verwendet wird, wenn kein anderer definiert wurde.

  Dies verletzt den Standard und macht Fehlererkennung schwieriger. Man kann auch nicht explizit "`DEFAULT NULL`" für eine `NOT NULL`-Spalte definieren.

Eindeutige Zahlen (1)

- Oft muss man eindeutige Zahlen generieren, z.B. die `SID` für Studenten (künstlicher Primärschlüssel).
- SQL-92 hat keinen Mechanismus dafür.
- Theoretisch könnte man das aktuelle Maximum in der Tabelle abfragen, und eins dazu addieren.

  Alternativ könnte man auch eine Tabelle mit einer Zeile und Spalte erstellen, die das Maximum enthält. Das ist evtl. etwas schneller, obwohl man das Maximum auch mit einem B-Baum-Index schnell findet.
- Aber bei gleichzeitigen Nutzern wird dies schwierig, siehe Teil 7. Daher haben viele DBMS einen Mechanismus zum Generieren von eindeutigen Zahlen.

Eindeutige Zahlen (2)

Oracle:

- Oracle hat ein DB-Objekt "sequence" (Sequenz):

  ```
  CREATE SEQUENCE STUD_IDS START WITH 101
  ```
- Man kann folgendermaßen eine Zahl abrufen und den in der Sequenz gespeicherten Wert erhöhen:

  ```
  SELECT STUD_IDS.NEXTVAL FROM DUAL
  ```

  (`DUAL` ist eine Dummy-Tabelle mit einer Zeile.)
- Man kann `STUD_IDS.NEXTVAL` nicht als `DEFAULT` festlegen, aber man kann es im `INSERT`-Befehl verwenden.

Eindeutige Zahlen (3)

SQL Server:

- In SQL Server kann man das Wort `IDENTITY` zur Deklaration einer Integer-Spalte hinzufügen:

```
SID NUMERIC(3) IDENTITY NOT NULL PRIMARY KEY
```

- Werte dieser Spalte kann man nicht festlegen, der nächste Wert wird automatisch eingefügt.

  Sogar wenn man keine Spalten-Liste unter `INSERT` verwendet, wird die Spalte in der `VALUES`-Liste übersprungen. Besser: Spalten festlegen. `IDENTITY` kann nicht zusammen mit `DEFAULT` verwendet werden.

- Man kann Startwert und Schrittweite festlegen:

```
SID NUMERIC(3) IDENTITY(100,1) PRIMARY KEY
```


Eindeutige Zahlen (4)

DB2:

- In DB2 kann man festlegen, dass ein Spaltenwert z.B. von einem "Generator eindeutiger Zahlen" (unique number generator) berechnet wird:

```
SID NUMERIC(3)
    GENERATED ALWAYS AS IDENTITY(START WITH 101)
    NOT NULL PRIMARY KEY
```

- "`GENERATED ALWAYS`" heißt, dass man keinen Wert für diese Spalte im `INSERT`-Befehl festlegen kann.

  Die Alternative ist `GENERATED BY DEFAULT`. Beide Fälle schließen ein `DEFAULT`-Statement aus. Es gibt weitere Parameter für den Generator eindeutiger Zahlen, z.B. `IDENTITY(START WITH 101, INCREMENT BY 1)`.

Eindeutige Zahlen (5)

MySQL:

- In MySQL kann man das Wort `AUTO_INCREMENT` zur Deklaration einer Integer-Spalte hinzufügen:

```
SID INT AUTO_INCREMENT PRIMARY KEY
```

- Dies funktioniert nur mit binären Integer-Typen.

  Z.B. funktioniert es nicht mit `NUMERIC(3)`. Man kann `UNSIGNED`-Typen verwenden. Die Spalte muss als Schlüssel deklariert sein (`PRIMARY KEY` oder `UNIQUE`).

- Wird mit `INSERT` für diese Spalte `NULL` oder `0` festgelegt, wird stattdessen die nächste Zahl eingefügt.

  Ein deklarierter Default-Wert wird einfach ignoriert.

Eindeutige Zahlen (6)

Access:

- Access hat den Datentyp `COUNTER`, der eindeutige Zahlen erzeugt:

```
SID COUNTER NOT NULL PRIMARY KEY
```

- Man kann Startwert und Schrittweite mit Parametern dieses Datentyps festlegen:

```
SID COUNTER(100,1) NOT NULL PRIMARY KEY
```

- Um den nächsten Wert des Zählers einzufügen, muss man die Spalten unter `INSERT` explizit angeben und die Spalte `SID` weglassen.

Temporäre Tabellen

- SQL-92 und einige DBMS ermöglichen die Deklaration temporärer Tabellen, die
  - ◇ automatisch am Ende der Transaktion oder Sitzung gelöscht werden,

    Je nach System kann es sein, dass die Tabelle selbst nicht gelöscht wird, sondern nur all ihre Zeilen.
  - ◇ unsichtbar für andere (parallele) Sitzungen sind.

    Tabellen anderer Benutzer sind normalerweise unsichtbar (außer der Nutzer hat Zugriffsrechte erteilt). Bei temporären Tabellen haben aber auch parallele Sitzungen unter der gleichen Nutzerkennung verschiedene Kopien.
- Für Details siehe Handbuch des jeweiligen DBMS.

Speicherparameter

- In den meisten DBMS hat das "`CREATE TABLE`"-Statement viele Speicherparameter, die verändert werden können.
- Diese Parameter gehören zum physischen Schema, nicht zum konzeptuellen Schema.

  Es ist etwas unglücklich, dass hier beide Dinge vermischt werden.
- Der SQL-Standard enthält keine Definition, die zu dem physischen Schema gehört.
- Ist die Performance wichtig, sollte man die Bedeutung der Parameter verstehen und diese verändern.

Einschränkungen (1)

SQL Server:

- max. 1024 Spalten pro Tabelle
- max. 8060 Bytes pro Zeile

  Die Daten von `TEXT`-Spalten etc. werden separat gespeichert.
- Schlüssel können max. 16 Spalten enthalten.
- Schlüsselwerte können max. 900 Bytes haben.

  Das schränkt die Konkatenation von Spaltenwerten für alle Spalten ein.

Einschränkungen (2)

DB2:

- max. 500 Spalten pro Tabelle
- max. 4005 Bytes pro Zeile

  Einschließlich des Deskriptors von LOB-Spalten, aber nicht dessen Daten.
- Schlüssel können max. 16 Spalten enthalten.
- Schlüsselwerte können max. 255 Bytes haben.

Einschränkungen (3)

Oracle:

- max. 1000 Spalten pro Tabelle
- max. 32 Spalten pro Schlüssel
- Schlüsselwerte (oder Index-Werte) sind auf 40% der Blockgröße beschränkt.

  Die DB-Blockgröße kann innerhalb angemessener Grenzen konfiguriert werden (wenn die DB erstellt wird).
- Die Zeilenlänge ist nicht beschränkt, aber die Performance lässt nach, wenn Zeilen auf verschiedene Blöcke aufgeteilt werden müssen.

Einschränkungen (4)

Access:

- max. 255 Spalten pro Tabelle
- max. 10 Spalten pro Schlüssel
- max. 2000 Zeichen in einer Zeile

  Gilt nicht bei Memo- und OLE-Objekt-Spalten.
- Die maximale Größe einer Tabelle ist 1 GB.
- max. 2 GB pro Datenbank (.mdb-Datei)

  Man kann Verknüpfungen zu Tabellen in anderen Dateien einfügen, um die Grenzen zu überschreiten.

Inhalt

CREATE SCHEMA (1)

- In Oracle und SQL Server entsprechen sich DB-Accounts und Schemas 1:1.

  Benötigt ein Nutzer mehr als ein Schema, müssen mehrere Accounts für die gleiche Person erstellt werden.
- Tabellen werden im System durch ihren Namen und ihren Eigentümer identifiziert.

  In SQL Server: Server, Datenbank, Eigentümer, Name.
- In DB2 sind Schemas und Nutzer verschiedene Dinge, obwohl jedes Schema zu genau einem Nutzer gehört.

CREATE SCHEMA (2)

- Es gibt einen `CREATE SCHEMA`-Befehl, der jedoch nur benötigt wird, wenn sich zwei Tabellen gegenseitig referenzieren:

```
CREATE SCHEMA AUTHORIZATION ⟨Account⟩
    CREATE TABLE ABT(ABTNR NUMERIC(2) PRIMARY KEY,
        CHEF NUMERIC(4) REFERENCES ANG, ...)
    CREATE TABLE ANG(ANGNR NUMERIC(4) PRIMARY KEY,
        ABTNR NUMERIC(2) REFERENCES ABT, ...);
```

- Man beachte, dass die einzelnen `CREATE TABLE`-Statements nicht durch ";" getrennt werden.

CREATE SCHEMA (3)

- MySQL, Access unterstützen `CREATE SCHEMA` nicht.
- In Oracle, DB2, SQL Server kann `CREATE SCHEMA` die Befehle `CREATE TABLE`, `CREATE VIEW`, `GRANT` enthalten.

  In DB2 sind auch `COMMENT ON` und `CREATE INDEX` erlaubt.
- Schlägt ein Statement fehl, wird nichts ausgeführt.
- Da DB2 mehrere Schemas pro Nutzer zulässt, lautet die Syntax dort

  `CREATE SCHEMA` ⟨Name⟩ `AUTHORIZATION` ⟨Nutzer⟩

  Der Schema-Name und die Autorisierungs-Klausel sind beide optional, aber mindestens eines der beiden wird in DB2 benötigt.

Tabellen löschen (1)

- Tabellen werden mit folgendem Befehl gelöscht:

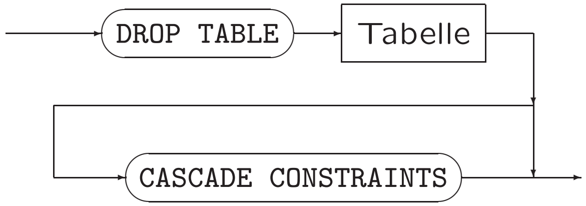
  `DROP TABLE ⟨Tabellenname⟩`
- Natürlich löscht dies auch alle Zeilen in der Tabelle.

  Man muss vorsichtig sein! In Oracle wird die aktuelle Transaktion automatisch beendet, wenn eine Tabelle gelöscht wird. Somit ist es nicht möglich, das Löschen oder vorangehende Aktionen rückgängig zu machen.

Tabellen löschen (2)

- Was passiert, wenn die Tabelle in Fremdschlüssel-Bedingungen referenziert wird?
  - ◇ In SQL Server und Access muss die referenzierende Tabelle zuerst gelöscht werden.
  - ◇ In DB2 wird der Fremdschlüssel automatisch gelöscht.
  - ◇ In Oracle kann "`CASCADE CONSTRAINTS`" angegeben werden, um Fremdschlüssel mit zu löschen.
  - ◇ In SQL-92 heißt es nur "`CASCADE`".

Tabellen löschen (3)



Beispiele:

- `DROP TABLE STUDENTEN CASCADE CONSTRAINTS`
- `DROP TABLE BEWERTUNGEN`

  Löscht man zuerst die Tabelle `BEWERTUNGEN` (enthält Fremdschlüssel) und dann `STUDENTEN`, so ist kein `CASCADE CONSTRAINTS` notwendig.

Inhalt

1. Schlüssel
2. Fremdschlüssel
3. CREATE TABLE-Syntax
4. CREATE SCHEMA, DROP TABLE
5. ALTER TABLE

ALTER TABLE (1)

- Mit dem `ALTER TABLE`-Befehl kann man das Schema einer existierenden Tabelle verändern.
- Im Prinzip könnte man die Daten temporär woanders speichern, die Tabelle löschen, sie verändert neu erstellen und die Daten zurückkopieren. Aber
  - ◇ Kopieren ist unpraktisch bei großen Tabellen.
  - ◇ Tabelleneinträge könnten von Fremdschlüsseln referenziert werden.

    Dann muss man evtl. die gesamte DB neu erstellen. Auch Indexe, Grants, Sichten, Trigger etc. referenzieren Tabellen. Manches davon wird verlorengehen, wenn die Tabelle gelöscht wird.

ALTER TABLE (2)

- Beispiele für Änderungen eines Tabellen-Schemas:
  - ◇ Neue Spalten können hinzugefügt werden.

    Daher ist es sicherer, in Anwendungsprogrammen statt `SELECT *` Spaltennamen anzugeben.
  - ◇ Die Breite von Spalten kann erhöht werden.

    Z.B. von `VARCHAR(20)` zu `VARCHAR(30)`.
    Das ist eigentlich im SQL-92-Standard nicht möglich, aber in allen fünf DBMS (Oracle, DB2, SQL Server, Access, MySQL).
  - ◇ Constraints kann man hinzufügen/entfernen.

    In manchen Systemen kann man eine Bedingung (Constraint) auch deaktivieren, so dass sie nicht mehr überprüft wird, aber noch im System gespeichert ist. Später kann man sie wieder aktivieren.

ALTER TABLE (3)

- `ALTER TABLE` war nicht in SQL-86 enthalten.
- Es ist im SQL-92-Standard enthalten, aber DBMS-Implementierungen unterscheiden sich stark in der Syntax und darin, was geändert werden kann.
- Der SQL-92-Standard hat folgende Möglichkeiten:
  - ◇ Spalten können hinzugefügt oder entfernt werden.
  - ◇ Der Default-Wert einer Spalte kann geändert werden, aber der Datentyp nicht.
  - ◇ Constraints kann man hinzufügen/entfernen.

ALTER TABLE (4)

SQL-92 (Spalten hinzufügen):

- Z.B. Spalte "`ZUSATZPKT`" zu "`STUDENTEN`" hinzufügen:

```
ALTER TABLE STUDENTEN
      ADD COLUMN ZUSATZPKT NUMERIC(4,1)
                           CHECK(ZUSATZPKT >= 0)
```

- Das Schlüsselwort "`COLUMN`" ist optional.
- Die neue Spalte hat zunächst in allen Zeilen einen Nullwert.
- Der Wert kann anschließend verändert werden.

  Es kann jedoch zu Effizienzproblemen kommen, wenn die Zeilen viel länger werden, als sie beim Einfügen waren.

ALTER TABLE (5)

SQL-92 (Spalten hinzufügen, fortgesetzt):

- Wird ein Default-Wert festgelegt, kann der neue Wert `NOT NULL` sein:

```
ALTER TABLE STUDENTEN
  ADD ZUSATZPKT NUMERIC(4,1) DEFAULT 0 NOT NULL
```

- Die neue Spalte wird die letzte (rechts) in der Tabelle.

  Es ist nicht möglich, sie woanders einzufügen.

- Sichten (gespeicherte Anfragen) werden nicht beeinflusst, weil `SELECT *` durch eine explizite Spaltenliste ersetzt wird, wenn die Sicht gespeichert wird.

ALTER TABLE (6)

SQL-92 (Spalten entfernen):

- Spalten können aus Tabellen entfernt werden:

```
ALTER TABLE STUDENTEN
      DROP COLUMN ZUSATZPKT RESTRICT
```

- `RESTRICT` heißt, dass `ALTER TABLE` fehlschlägt, wenn die Spalte in Constraints/Sichten referenziert wird.

  Constraints, die sich nur auf diese Spalte beziehen, zählen nicht: Sie werden automatisch gelöscht.

- Die Alternative ist `CASCADE`: Das referenzierende DB-Objekt wird mit gelöscht.

  Es gibt keinen Default: Man muss `RESTRICT` oder `CASCADE` angeben.

ALTER TABLE (7)

SQL-92 (Spalten verändern):

- Die einzige erlaubte Veränderung einer Spalte ist die Änderung des Default-Wertes:
```
ALTER TABLE AUFGABEN
ALTER COLUMN MAXPT SET DEFAULT 12
```
- Der Default kann mit dieser Syntax auch auf Null gesetzt werden:
```
ALTER TABLE AUFGABEN
ALTER COLUMN MAXPT DROP DEFAULT
```
- In SQL-92 ist es nicht möglich, den Datentyp einer Spalte zu verändern.

ALTER TABLE (8)

SQL-92 (Constraints verändern):

- Eine Bedingung hinzufügen:
```
ALTER TABLE STUDENTEN
      ADD CONSTRAINT ZUSATZPKT_DEF
      CHECK(ZUSATZPKT IS NOT NULL)
```
Es können nur Tabellen-Constraints hinzugefügt werden, aber Spalten-Constraints sind sowieso nur syntaktische Abkürzungen.

- Einen benannten Constraint entfernen:
```
ALTER TABLE STUDENTEN
      DROP CONSTRAINT ZUSATZPKT_DEF RESTRICT
```
Date/Darwen behaupten, dass der Standard verlangt, RESTRICT oder CASCADE festzulegen, obwohl das nur bei Schlüsseln Sinn macht, die in Fremdschlüsseln referenziert werden.

ALTER TABLE (9)

Oracle (Spalten hinzufügen):

- Z.B. Spalte "ZUSATZPKT" zu "STUDENTEN" hinzufügen:
```
ALTER TABLE STUDENTEN ADD
  (ZUSATZPKT NUMERIC(4,1) CHECK(ZUSATZPKT >= 0))
```
- Bei existierenden Zeilen ist die neue Spalte Null.

  Natürlich kann man den Wert später verändern.
- Ist ein Default-Wert festgelegt, kann die neue Spalte NOT NULL sein:
```
ALTER TABLE STUDENTEN ADD
  (ZUSATZPKT NUMERIC(4,1) DEFAULT 0 NOT NULL
                          CHECK(ZUSATZPKT >= 0))
```


ALTER TABLE (10)

Oracle (Spalten verändern):

- Der Datentyp einer Spalte kann verändert werden:
```
ALTER TABLE AUFGABEN MODIFY (THEMA VARCHAR(100))
```
- Die Breite einer Spalte kann nicht verringert werden, außer sie enthält nur Nullwerte.
- MODIFY kann nicht verwendet werden, um Spalten-Constraints zu ändern, außer NULL/NOT NULL.

  Es gibt eine andere Syntax dafür, siehe nächste Folie.
- In Oracle können Spalten nicht gelöscht oder umbenannt werden.

ALTER TABLE (11)

Oracle (Constraints hinzufügen/entfernen):

- Einen Constraint hinzufügen:

```
ALTER TABLE STUDENTEN ADD
    (CONSTRAINT SCH2 UNIQUE(VORNAME, NACHNAME))
```

- Hier muss die Syntax für Tabellen-Constraints verwendet werden.

  Intern werden alle Constraints in Tabellen-Constraints übersetzt.

- Einen benannten Constraint entfernen:

```
ALTER TABLE STUDENTEN DROP CONSTRAINT SCH2
```

  Man benötigt zum Entfernen meist den Namen des Constraints. Man kann ihn im Data Dictionary nachsehen (USER_CONSTRAINTS).

ALTER TABLE (12)

Oracle (Constraints entfernen, fortgesetzt):

- Primär- und Alternativschlüssel kann man entfernen, ohne den Namen zu kennen:

```
ALTER TABLE STUDENTEN DROP PRIMARY KEY CASCADE
ALTER TABLE STUDENTEN DROP UNIQUE(VORNAME,NACHNAME)
```

- NOT NULL-Constraints können mit einer Spaltenmodifikation entfernt werden:

```
ALTER TABLE AUFGABEN MODIFY (THEMA NULL)
```

- Man kann Constraints auch aktivieren/deaktivieren.

  Eine deaktivierte Bedingung existiert noch im Data Dictionary, wird aber nicht überprüft/erzwungen.

ALTER TABLE (13)

Oracle (Tabellen umbenennen):

- Tabellen können in Oracle umbenannt werden. Dies geschieht mit einem speziellen "RENAME"-Befehl:

```
RENAME STUDENTEN TO STUD
```

- Dies ist im SQL-92-Standard nicht enthalten.

  Es funktioniert auch in DB2, aber nicht in SQL Server, MySQL oder Access. Oracle (aber nicht DB2) versteht auch "ALTER TABLE STUDENTEN RENAME TO STUD".

- Ich kenne keine Möglichkeit, Spalten oder DB-Nutzer in Oracle umzubenennen.

  Wenn Sie es wissen, sagen Sie es mir.

ALTER TABLE (14)

DB2 (Spalten hinzufügen):

- Das Hinzufügen von Spalten funktioniert wie im SQL-Standard:

```
ALTER TABLE STUDENTEN
    ADD COLUMN ZUSATZPKT NUMERIC(4,1)
```

- Ist ein Default-Wert festgelegt, kann die neue Spalte NOT NULL sein.
- Es gibt keine Möglichkeit, Spalten zu löschen oder umzubenennen.

ALTER TABLE (15)

DB2 (Spalten verändern):

- Die einzige erlaubte Modifikation einer Spalte ist die Änderung der Länge einer VARCHAR-Spalte:

```
ALTER TABLE AUFGABEN
      ALTER THEMA SET DATA TYPE VARCHAR(100)
```

  Es sind wieder nur Erhöhungen der Länge möglich.
- Es scheint so, dass der NULL/NOT NULL-Status nicht geändert werden kann.

  Man kann NOT NULL mit einem CHECK-Constraint schreiben, aber das System versteht die Äquivalenz nicht richtig, z.B. verlangt PRIMARY KEY die Bedingung NOT NULL.

ALTER TABLE (16)

DB2 (Constraints hinzufügen/entfernen):

- Hinzufügen funktioniert wie im SQL-Standard:

```
ALTER TABLE STUDENTEN
      ADD CHECK(ZUSATZPKT IS NOT NULL)
```

- Ein benannter Constraint kann entfernt werden (wie in SQL-92, aber ohne RESTRICT/CASCADE):

```
ALTER TABLE STUDENTEN
      DROP CONSTRAINT ZUSATZPKT_DEF
```

- Primärschlüssel kann man ohne Namen entfernen:

```
ALTER TABLE STUDENTEN DROP PRIMARY KEY
```


ALTER TABLE (17)

SQL Server (Spalten hinzufügen/löschen):

- Das Hinzufügen funktioniert wie im Standard, aber das Wort "COLUMN" nach "ADD" ist nicht erlaubt.

```
ALTER TABLE STUDENTEN ADD ZUSATZPKT NUMERIC(4,1)
```

  Ist ein Default-Wert festgelegt, kann die neue Spalte NOT NULL sein.
- Eine Spalte kann wie folgt gelöscht werden (das Schlüsselwort "COLUMN" wird verlangt, RESTRICT oder CASCADE werden nicht verstanden):

```
ALTER TABLE STUDENTEN DROP COLUMN ZUSATZPKT
```


ALTER TABLE (18)

SQL Server (Spalten verändern):

- Der Datentyp einer Spalte kann verändert werden, aber nicht, wenn die Spalte ein Schlüssel ist oder ein Check-Constraint vorliegt:

```
ALTER TABLE AUFGABEN
      ALTER COLUMN THEMA VARCHAR(100)
```

  Verringerungen der Größe sind möglich, wenn die Daten noch passen.
- Man kann auch die NULL/NOT NULL-Bedingung in Spalten ändern:

```
ALTER TABLE AUFGABEN
      ALTER COLUMN THEMA VARCHAR(100) NULL
```

ALTER TABLE (19)

SQL Server (Constraints hinzufügen/entfernen):

- Das Hinzufügen funktioniert wie im SQL-Standard:

```
ALTER TABLE STUDENTEN
    ADD CHECK(ZUSATZPKT IS NOT NULL)
```

Eine Spalte muss Not Null sein, um einen Primärschlüssel-Constraint hinzuzufügen.

- Eine benannte Bedingung kann entfernt werden:

```
ALTER TABLE STUDENTEN
    DROP CONSTRAINT ZUSATZPKT_DEF
```

(Wie in SQL-92, aber ohne RESTRICT/CASCADE).

ALTER TABLE (20)

Access:

- Das Hinzufügen und Löschen von Constraints funktioniert wie im SQL-92-Standard.

  Das Schlüsselwort COLUMN ist optional, auch wenn es im Handbuch anscheinend verlangt wird. CASCADE und RESTRICT werden im Handbuch beim Löschen von Constraints nicht erwähnt, aber sie werden akzeptiert (obwohl sie wie im Standard nicht unbedingt nötig sind).

- Der Datentyp einer Spalte kann vielseitig geändert werden. Man kann z.B. zwischen Zahlen und Zeichenketten konvertieren (wenn die Zeichenketten ein numerisches Format haben).

```
ALTER TABLE STUDENTEN ALTER COLUMN ZUSATZPKT CHAR(20)
```


ALTER TABLE (21)

MySQL:

- MySQL hat ein sehr umfangreiches ALTER TABLE-Statement mit vielen Optionen (siehe Handbuch).

  Dies liegt zum Teil daran, dass MySQL ALTER TABLE ausführt, indem eine Kopie der gesamten Tabelle mit der neuen Struktur erstellt wird. Das kann bei großen Tabellen aufwändig werden und andere Systeme versuchen, dies zu vermeiden (was zu vielen Einschränkungen führt).

- Die SQL-92-Syntax wird verstanden, außer beim Löschen von benannten Constraints.

  Sogar RESTRICT/CASCADE werden zumindest syntaktisch für DROP COLUMN akzeptiert, auch wenn sie im Handbuch nicht aufgeführt sind.

ALTER TABLE (22)

MySQL, fortgesetzt:

- Es gibt eine nicht standardisierte Syntax für das Löschen der Constraints, die MySQL unterstützt.

  Mit "ALTER TABLE T DROP PRIMARY KEY" kann man den Primärschlüssel entfernen. Bei anderen Schlüsseln muss man den Namen X herausfinden, den MySQL dem Schlüssel bzw. Index zugeordnet hat (mit "SHOW CREATE TABLE T"), dann "ALTER TABLE T DROP INDEX X". CHECK-Constraints und Fremdschlüssel werden nicht unterstützt. Der Datentyp einer Spalte und der NULL/NOT NULL-Status werden wie folgt geändert: "ALTER TABLE AUFGABEN MODIFY THEMA VARCHAR(100) NULL".

- Tabellen und Spalten können umbenannt werden.

```
ALTER TABLE STUDENTEN RENAME TO STUD
ALTER TABLE STUDENTEN CHANGE SID STUD_NR NUMERIC(3) PRIMARY KEY
```