VI. Die relationale Datenbanksprache SQL

SQL als Datendefinitionssprache

Datentypen

Die wichtigsten Datentypen für die Wertebereiche von Attributen sind Zahlen, Zeichenketten und Datumsangaben.

Typ	Beschreibung
char(n)	Zeichenstring mit fester Länge n. Synonym: character(n)
varchar(n)	Zeichenstring mit variabler aber maximaler Länge n. Synonyme: char varying(n), character varying(n)
int	Wert einer maschinenabhängigen, endlichen Teilmenge der ganzen Zahlen. Synonym: integer
smallint	eine maschinenabhängige Teilmenge des int-Wertebereichs
numeric(z, n)	Fixpunktzahl (Dezimalzahl) mit spezifizierbarer Genauigkeit, z = Anzahl aller Stellen, n = Anzahl der Nachkommastellen. Synonym: decimal(z, n)
real	Fließkommazahl mit maschinenabhängiger Genauigkeit

SQL als Datendefinitionssprache

Datentypen (Fortsetzung)

Typ	Beschreibung
double precision	Fließkommazahl mit doppelter, maschinenabhängiger Genauigkeit.
float(n)	Fließkommazahl mit spezifizierbarer Genauigkeit von $\geq$ n Stellen.
bit(n)	Bitstring mit fester Länge n.
bit varying(n)	Bitstring mit variabler aber maximaler Länge n.
blob	Binary Large Object. Variabel lange Byte-Sequenz von maximal 4 GB. Zum Speichern von Videosequenzen, Bilder, Audio-Dateien, etc.
date	Kalenderdatum mit Jahr (4 Stellen), Monat (2 Stellen), Tag (2 Stellen). Format: JJJJ-MM-TT
time	Tageszeit in Stunden, Minuten und Sekunden, Format: HH:MM:SS
time with time zone	Zeitunterschied zu GMT (6 Stellen), Bereich: +13:00 bis -12:59
timestamp	Wert, der Datum und Tageszeit enthält.
interval	Wert, um den ein absoluter Wert vom Typ date, time oder timestamp in/dekrementiert wird. Dient zur Formulierung von Zeitintervallen.

SQL als Datendefinitionssprache

Domains (SQL-92)

```
create {domain | datatype} [as] <domain> <datatype>
  [[not] null]
  [default <value>]
  [check (<condition>)]
```

- `<datatype>` bezeichnet einen integrierten Datentyp einschließlich eventueller Stellenanzahl.
- Die Deklaration von Domains ist vergleichbar mit einfachen, nicht-geschachtelten Typ-Vereinbarungen in einer Programmiersprache.
- Domains können in Create-Table-Anweisungen für Attributdeklarationen verwendet werden.

SQL als Datendefinitionssprache

Domains (SQL-92)

```
create {domain | datatype} [as] <domain> <datatype>
  [[not] null]
  [default <value>]
  [check (<condition>)]
```

- `<datatype>` bezeichnet einen integrierten Datentyp einschließlich eventueller Stellenanzahl.
- Die Deklaration von Domains ist vergleichbar mit einfachen, nicht-geschachtelten Typ-Vereinbarungen in einer Programmiersprache.
- Domains können in Create-Table-Anweisungen für Attributdeklarationen verwendet werden.

Beispiele:

```
create domain address char(35) null

create domain Stadtstaaten varchar(10) default 'Berlin'
```

SQL als Datendefinitionssprache

Check-Klausel

```
check (<condition>)
```

- Die Check-Klausel dient zur Festlegung lokaler Integritätsbedingungen für Domains, Attribute und Relationenschemata.
- Die Check-Klausel erlaubt die Formulierung von Prädikaten, wie sie in der Where-Klausel bei Anfrageoperationen spezifiziert werden können.

Beispiele: [Relationen definieren]

```
create domain Stadtstaaten varchar(10) default 'Berlin'
  check (value in ('Berlin', 'Hamburg', 'Bremen'))

check(Alter between 18 and 27)

check(Dauer > 3)
```

SQL als Datendefinitionssprache

Check-Klausel

```
check (<condition>)
```

- Die Check-Klausel dient zur Festlegung lokaler Integritätsbedingungen für Domains, Attribute und Relationenschemata.
- Die Check-Klausel erlaubt die Formulierung von Prädikaten, wie sie in der Where-Klausel bei Anfrageoperationen spezifiziert werden können.

Beispiele: [Relationen definieren]

```
create domain Stadtstaaten varchar(10) default 'Berlin'
  check (value in ('Berlin', 'Hamburg', 'Bremen'))


check(Alter between 18 and 27)

check(Dauer > 3)


check ((select avg(Alter) from Reiseleiter) <
       (select sum(TeilnAnz) from Reisen where Stadt='Paris'))
```

SQL als Datendefinitionssprache

Datenbanken

```
create database <database>
  [with {[buffered] log | log mode ansi}]
```

- Erstellt eine Datenbank und macht diese zur aktuellen Datenbank. Mit dem Erzeugen einer Datenbank werden auch die Systemrelationen *(Data Dictionary)* angelegt.
- Durch Angabe der `with`-Option wird die Protokollierung von Transaktionen aktiviert. Diese wird im Falle eines Datenbankfehlers zum Wiederherstellen der Daten benötigt.

```
close database <database>
```

- Schließt die aktuell geöffnete Datenbank. Eine geschlossene Datenbank kann gelöscht werden.

```
drop database <database>
```

- Löscht eine Datenbank einschließlich aller Daten, Indizes und Systemrelationen. Es kann weder die aktuelle Datenbank noch eine Datenbank, die gerade von anderen Benutzern verwendet wird, gelöscht werden.

```
use <database>
```

- Auswahl einer Datenbank aus einer Menge von existierenden Datenbanken als Default für nachfolgende SQL-Befehle.

SQL als Datendefinitionssprache

Relationen definieren

```
create table <table>
(
 <attribute1> <datatype> [default <value>] [not null] [<attr_int_cond>],
 <attribute2> <datatype> [default <value>] [not null] [<attr_int_cond>],
 ...
 {<rel_int_cond>}*
)
```

SQL als Datendefinitionssprache

Relationen definieren

```
create table <table>
(
 <attribute1> <datatype> [default <value>] [not null] [<attr_int_cond>],
 <attribute2> <datatype> [default <value>] [not null] [<attr_int_cond>],
 ...
 {<rel_int_cond>}*
)
```

```
<attr_int_cond> ::= {unique |
                     primary key |
                     references <table>[(<attribute>)] |
                     check (<condition>)}
```

SQL als Datendefinitionssprache

Relationen definieren

```
create table <table>
(
 <attribute1> <datatype> [default <value>] [not null] [<attr_int_cond>],
 <attribute2> <datatype> [default <value>] [not null] [<attr_int_cond>],
 ...
 {<rel_int_cond>}*
)
```

```
<attr_int_cond> ::= {unique |
                     primary key |
                     references <table>[(<attribute>)] |
                     check (<condition>)}
```

```
<rel_int_cond> ::= {unique (<attribute1>, ...) |
                    primary key (<attribute1>, ...) |
                    foreign key (<attribute1>, ...)
                      references <table>[(<attribute1>,...)][<action>] |
                    check (<condition>)}
```

SQL als Datendefinitionssprache

Relationen definieren

```
create table <table>
(
 <attribute1> <datatype> [default <value>] [not null] [<attr_int_cond>],
 <attribute2> <datatype> [default <value>] [not null] [<attr_int_cond>],
 ...
 {<rel_int_cond>}*
)
```

```
<attr_int_cond> ::= {unique |
                     primary key |
                     references <table>[(<attribute>)] |
                     check (<condition>)}
```

```
<rel_int_cond> ::= {unique (<attribute1>, ...) |
                    primary key (<attribute1>, ...) |
                    foreign key (<attribute1>, ...)
                      references <table>[(<attribute1>,...)][<action>] |
                    check (<condition>)}
```

```
<action> ::= [on update <type>] [on delete <type>]
```

Bemerkungen:

- Mittels `default` lässt sich ein Standardwert für Attribute vorgeben.
- Integritätsbedingungen beziehen sich auf ein Tupel, eine Relation oder mehrere Relationen. Besteht ein Schlüssel aus einem einzigen Attribut, so kann die entsprechende Integritätsbedingung gemäß `<attr_int_cond>` direkt innerhalb der Attributdeklaration vereinbart werden.
- Integritätsbedingungen für Schlüssel, die aus mehreren Attributen bestehen, werden gemäß `<rel_int_cond>` vereinbart.
- In SQL-92 sind Primary-Key-Attribute implizit „unique“ und „not null“.

SQL als Datendefinitionssprache

Relationen definieren: Beispiele

```
create table Buch
(ISBN char(10) not null,
 Titel varchar(200),
 Verlagsname varchar(30) not null)
```

SQL als Datendefinitionssprache

Relationen definieren: Beispiele

```
create table Buch
(ISBN char(10) not null,
 Titel varchar(200),
 Verlagsname varchar(30) not null)
```

Verwendung der Form `<attr_int_cond>`:

```
create table Buch
(ISBN char(10) primary key,
 Titel varchar(200),
 Verlagsname varchar(30) not null references Verlage(Verlagsname))
```

Verwendung der Form `<rel_int_cond>`:

```
create table Buch
(ISBN char(10),
 Titel varchar(200),
 Verlagsname varchar(30) not null,
 primary key (ISBN),
 foreign key (Verlagsname) references Verlage(Verlagsname))
```

SQL als Datendefinitionssprache

Relationen definieren: Beispiele [Check-Klausel]

```
create table Buch_Versionen
(ISBN char(20),
 Auflage smallint check (Auflage > 0),
 Jahr integer check (Jahr between 1800 and 2020),
 Seitenzahl integer check (Seiten > 0),
 Preis decimal(8,2) check (Preis ≤ 250),

 primary key (ISBN, Auflage),
 foreign key (ISBN) references Buch (ISBN),


 check ((select sum(Preis) from Buch_Versionen) <
        (select sum(Budget) from Arbeitsgruppen))

)
```

SQL als Datendefinitionssprache

Relationen definieren: On-Klausel

```
<action> ::= [on update <type>] [on delete <type>]
```

Wird ein Primärschlüsselwert geändert oder gelöscht, kann es Fremdschlüsselwerte geben, die anzupassen sind. Hierfür lässt sich eine `on update`-Klausel und eine `on delete`-Klausel angeben, gefolgt von einem der folgenden Aktionstypen:

- `cascade`
- `set null`
- `set default`
- `restrict` (Default)

SQL als Datendefinitionssprache

Relationen definieren: On-Klausel

```
<action> ::= [on update <type>] [on delete <type>]
```

Wird ein Primärschlüsselwert geändert oder gelöscht, kann es Fremdschlüsselwerte geben, die anzupassen sind. Hierfür lässt sich eine `on update`-Klausel und eine `on delete`-Klausel angeben, gefolgt von einem der folgenden Aktionstypen:

- `cascade`
  Zusammen mit `on update` werden die Fremdschlüsselwerte aktualisiert, damit sie den neuen Primärschlüsselwert referenzieren. Zusammen mit `on delete` werden die Tupel gelöscht, die den gelöschten Primärschlüsselwert referenzierten.
- `set null`
- `set default`
- `restrict` (Default)

SQL als Datendefinitionssprache

Relationen definieren: On-Klausel

```
<action> ::= [on update <type>] [on delete <type>]
```

Wird ein Primärschlüsselwert geändert oder gelöscht, kann es Fremdschlüsselwerte geben, die anzupassen sind. Hierfür lässt sich eine `on update`-Klausel und eine `on delete`-Klausel angeben, gefolgt von einem der folgenden Aktionstypen:

- `cascade`
  Zusammen mit `on update` werden die Fremdschlüsselwerte aktualisiert, damit sie den neuen Primärschlüsselwert referenzieren. Zusammen mit `on delete` werden die Tupel gelöscht, die den gelöschten Primärschlüsselwert referenzierten.
- `set null`
  Setzt die Fremdschlüsselwerte auf Null, falls der referenzierte Primärschlüsselwert geändert oder gelöscht wurde.
- `set default`
  Setzt die Fremdschlüsselwerte auf den Wert, der in der Default-Klausel des Fremdschlüsselattributes angegeben ist, falls der referenzierte Primärschlüsselwert geändert oder gelöscht wurde.
- `restrict` (Default)

SQL als Datendefinitionssprache

Relationen definieren: On-Klausel

```
<action> ::= [on update <type>] [on delete <type>]
```

Wird ein Primärschlüsselwert geändert oder gelöscht, kann es Fremdschlüsselwerte geben, die anzupassen sind. Hierfür lässt sich eine `on update`-Klausel und eine `on delete`-Klausel angeben, gefolgt von einem der folgenden Aktionstypen:

- `cascade`
  Zusammen mit `on update` werden die Fremdschlüsselwerte aktualisiert, damit sie den neuen Primärschlüsselwert referenzieren. Zusammen mit `on delete` werden die Tupel gelöscht, die den gelöschten Primärschlüsselwert referenzierten.

- `set null`
  Setzt die Fremdschlüsselwerte auf Null, falls der referenzierte Primärschlüsselwert geändert oder gelöscht wurde.

- `set default`
  Setzt die Fremdschlüsselwerte auf den Wert, der in der Default-Klausel des Fremdschlüsselattributes angegeben ist, falls der referenzierte Primärschlüsselwert geändert oder gelöscht wurde.

- `restrict` (Default)
  Erzeugt eine Fehlermeldung bei dem Versuch, einen Primärschlüsselwert zu ändern oder zu löschen, wenn dieser als Fremdschlüsselwert referenziert wird.

SQL als Datendefinitionssprache

Relationen ändern

```
alter table <table>
{
 add <attribute> <datatype> ...[before <attribute>] |
 modify <attribute> <datatype> |
 drop <attribute> |
 <add_cons>
}
```

- Modifiziert eine bestehende Relation. Es können Attribute eingefügt oder gelöscht, Constraints hinzugefügt, verändert oder gelöscht werden.

SQL als Datendefinitionssprache

Relationen ändern

```
alter table <table>
{
 add <attribute> <datatype> ...[before <attribute>] |
 modify <attribute> <datatype> |
 drop <attribute> |
 <add_cons>
}
```

- Modifiziert eine bestehende Relation. Es können Attribute eingefügt oder gelöscht, Constraints hinzugefügt, verändert oder gelöscht werden.

```
<add_cons> ::= {add constraint unique (<attribute1>, ...) |
               add constraint primary key (<attribute1>, ...) |
               add constraint foreign key (<attribute1>, ...)
                                 references <table> (<attribute1>, ...) |
               add constraint check (<condition>)}
```

SQL als Datendefinitionssprache

Relationen löschen

```
drop table <table>
```

- Löscht alle in einer Relation enthaltenen Daten, die Indizes und Constraints auf den Attributen inklusive aller Referenz-Constraints, alle mit der Relation verbundenen Synonyme sowie alle für diese Relation vergebenen Berechtigungen.
- Es können nur Relationen der aktuellen Datenbank gelöscht werden.
- Das Löschen von Relationen mit Systeminformation ist mit dieser Anweisung nicht möglich.

SQL als Datendefinitionssprache

Datendefinitionsbefehle im Überblick

	database	table	index	view	synonym	schema
create	•	•	•	•	•	•
alter		•	•			
close	•					
drop	•	•	•	•	•	
use	•					

SQL als Datenmanipulationssprache

Einfügen von Tupeln

```
insert into <table>
{
 [(<attribute1>, ...)] values (<expression1>, ...) {, (...)}*0   |
 [(<attribute1>, ...)] values <SFW-Block>
}
```

- Einfügen von vollständigen oder unvollständigen Tupeln. Die Tupel können explizit oder mittels eines Select-From-Where-Blocks spezifiziert werden.
- Alle einzufügenden Werte müssen die Integritätsbedingungen erfüllen.

SQL als Datenmanipulationsssprache

Einfügen von Tupeln

```
insert into <table>
{
 [(<attribute1>, ...)] values (<expression1>, ...) {, (...)}*0   |
 [(<attribute1>, ...)] values <SFW-Block>
}
```

- Einfügen von vollständigen oder unvollständigen Tupeln. Die Tupel können explizit oder mittels eines Select-From-Where-Blocks spezifiziert werden.
- Alle einzufügenden Werte müssen die Integritätsbedingungen erfüllen.

Zwei Verwendungsformen:

1. Ohne Attributnamen. Anzahl, Reihenfolge und Datentyp der Werte müssen der Definition der Relation entsprechen. Die Reihenfolge der Attribute ist in der Relation „Syscolumns“ definiert.
2. Mit Attributnamen. Einfügen der Werte gemäß den Attributnamen. Fehlende Attribute erhalten den Nullwert.

SQL als Datenmanipulationsssprache

Einfügen von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Kursgebuehr = {AngNr, KursNr, TnNr, Gebuehr}
- Angebot = {AngNr, KursNr, Datum, Ort}
- nimmt_teil = {AngNr, KursNr, TnNr}

„Füge einen neuen Teilnehmer mit TnNr 200 für Kurs G08 und AngNr 1 in die Kursgebührrelation ein. Die Teilnamegebühr sei noch nicht bekannt.“

```
insert into Kursgebuehr values (1, G08, 200, null)
```
bzw.
```
insert into Kursgebuehr (AngNr, KursNr, TnNr) values (1, G08, 200)
```

SQL als Datenmanipulationssprache

Einfügen von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Kursgebuehr = {AngNr, KursNr, TnNr, Gebuehr}
- Angebot = {AngNr, KursNr, Datum, Ort}
- nimmt_teil = {AngNr, KursNr, TnNr}

„Füge einen neuen Teilnehmer mit TnNr 200 für Kurs G08 und AngNr 1 in die Kursgebührrelation ein. Die Teilnamegebühr sei noch nicht bekannt.“

```
insert into Kursgebuehr values (1, G08, 200, null)
```
bzw.
```
insert into Kursgebuehr (AngNr, KursNr, TnNr) values (1, G08, 200)
```

„Füge ein neues Kursangebot mit AngNr 3 für G08 für den 15. März 1991 in Ulm ein.“

```
insert into Angebot values (3, G08, 15-03-1991, ULM)
```

SQL als Datenmanipulationsssprache

Einfügen von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Kursgebuehr = {AngNr, KursNr, TnNr, Gebuehr}
- Angebot = {AngNr, KursNr, Datum, Ort}
- nimmt_teil = {AngNr, KursNr, TnNr}

„Füge einen neuen Teilnehmer mit TnNr 200 für Kurs G08 und AngNr 1 in die Kursgebührrelation ein. Die Teilnamegebühr sei noch nicht bekannt.“

```
insert into Kursgebuehr values (1, G08, 200, null)   bzw.
insert into Kursgebuehr (AngNr, KursNr, TnNr) values (1, G08, 200)
```

„Füge ein neues Kursangebot mit AngNr 3 für G08 für den 15. März 1991 in Ulm ein.“

```
insert into Angebot values (3, G08, 15-03-1991, ULM)
```

„Die Relation Kursgebuehr sei leer. Fülle die Attribute AngNr, KursNr und TnNr mit den Einträgen der Relation Nimmt_teil. Das Attribut Gebuehr soll ohne Wert bleiben.“

```
insert into Kursgebuehr (AngNr, KursNr, TnNr)
select *
from nimmt_teil
```

SQL als Datenmanipulationsssprache

Löschen von Tupeln

```
delete from <table>
[where <condition>]
```

- Löscht alle Tupel, die `<condition>` erfüllen, aus `<table>`. Die leere Relation bleibt als Eintrag im Katalog erhalten.

SQL als Datenmanipulationsssprache

Löschen von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Angebot = {AngNr, KursNr, Datum, Ort}
- nimmt_teil = {AngNr, KursNr, TnNr}

„Lösche alle Tupel aus der nimmt_teil-Relation.“

```
delete
from nimmt_teil
```

SQL als Datenmanipulationssprache

Löschen von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Angebot = {AngNr, KursNr, Datum, Ort}
- nimmt_teil = {AngNr, KursNr, TnNr}

„Lösche alle Tupel aus der nimmt_teil-Relation.“

```
delete
from nimmt_teil
```

„Lösche in der nimmt_teil-Relation alle Kurse, die vor dem 1. März 1990 stattgefunden haben.“

```
delete
from nimmt_teil
where (AngNr, KursNr) in
    (select AngNr, KursNr
     from Angebot
     where Datum < 01-03-1990)
```

SQL als Datenmanipulationsssprache

Ändern von Tupeln

```
update <table> [[as] <alias>]
set <attribute1> = <expression1>
 [, <attribute2> = <expression2>, ...]
[where <condition>]
```

SQL als Datenmanipulationsssprache

Ändern von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Kursgebuehr = {AngNr, KursNr, TnNr, Gebuehr}
- Std_Gebuehr = {KursNr, Gebuehr}

„Erhöhe alle Gebühren um 10%.“

```
update Kursgebuehr
set Gebuehr = Gebuehr*1.1
```

SQL als Datenmanipulationsssprache

Ändern von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Kursgebuehr = {AngNr, KursNr, TnNr, Gebuehr}
- Std_Gebuehr = {KursNr, Gebuehr}

„Erhöhe alle Gebühren um 10%.“

```
update Kursgebuehr
set Gebuehr = Gebuehr*1.1
```

„Erhöhe die Gebühren der Teilnehmer mit TnNR > 150 um 10%“

```
update Kursgebuehr
set Gebuehr = Gebuehr*1.1
where TnNr > 150
```

SQL als Datenmanipulationsssprache

Ändern von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Kursgebuehr = {AngNr, KursNr, TnNr, Gebuehr}
- Std_Gebuehr = {KursNr, Gebuehr}

„Erhöhe alle Gebühren um 10%.“

```
update Kursgebuehr
set Gebuehr = Gebuehr*1.1
```

„Erhöhe die Gebühren der Teilnehmer mit TnNR > 150 um 10%“

```
update Kursgebuehr
set Gebuehr = Gebuehr*1.1
where TnNr > 150
```

„Setze alle Gebühren, für die noch kein Wert spezifiziert wurde, auf die Standardgebühr.“

```
update Kursgebuehr g
set g.Gebuehr =
    (select s.Gebuehr from Std_Gebuehr s where g.KursNr = s.KursNr)
where g.Gebuehr is null
```

SQL als Datenmanipulationssprache

Ändern von Tupeln: Beispiele

Gegeben seien folgende Relationenschemata:

- Kursgebuehr = {AngNr, KursNr, TnNr, Gebuehr}
- Std_Gebuehr = {KursNr, Gebuehr}

„Erhöhe alle Gebühren um 10%.“

```
update Kursgebuehr
set Gebuehr = Gebuehr*1.1
```

„Erhöhe die Gebühren der Teilnehmer mit TnNR > 150 um 10%“

```
update Kursgebuehr
set Gebuehr = Gebuehr*1.1
where TnNr > 150
```

„Setze alle Gebühren, für die noch kein Wert spezifiziert wurde, auf die Standardgebühr.“

```
update Kursgebuehr g
set g.Gebuehr =
    (select s.Gebuehr from Std_Gebuehr s where g.KursNr = s.KursNr)
where g.Gebuehr is null
```

Bemerkungen:

- Bildet ein SFW-Block den rechten Operand einer Gleichung, so darf der SFW-Block nur *ein* Tupel als Return-Wert haben. Siehe Beispiel: `g.Gebuehr = (select ...)`