Teil II

Erweiterte Konzepte innerhalb SQL

8 REFERENTIELLE INTEGRITÄT

Referentielle Integritätsbedingungen entstehen aus dem Zusammenhang zwischen Primär- und Fremdschlüsseln. Eine referentielle Integritätsbedingung

```
FOREIGN KEY (<attr-list>) REFERENCES <table'>(<attr-list'>);
```

definiert eine Inklusionsabhängigkeit: Zu jedem (Fremdschlüssel)wert der Attribute (<attr-list>) der *referenzierenden* Tabelle (*C- (Child-) Table*) <table> muss ein entsprechender Schlüsselwert von (<attr-list>) in der *referenzierten* Tabelle <table> (*P- (Parent-) Table*) existieren.

Dabei muss (<attr-list'>) ein Candidate Key der referenzierten Tabelle sein. In ORACLE kann nur der deklarierte PRIMARY KEY referenziert werden, damit ist die Angabe der Spalten optional.

Referentielle Integritätsbedingungen treten immer auf, wenn bei der Umsetzung vom ER-Modell zum relationalen Modell Schlüsselattribute der beteiligten Entities in Beziehungstypen eingehen:

```
CREATE TABLE Country
 (Name  VARCHAR2(32),
  Code  VARCHAR2(4) PRIMARY KEY,
  ...);

CREATE TABLE Continent
 (Name     VARCHAR2(10) PRIMARY KEY,
  Area     NUMBER(2));

CREATE TABLE encompasses
 (Continent    VARCHAR2(10) REFERENCES Continent(Name),
  Country      VARCHAR2(4) REFERENCES Country(Code),
  Percentage   NUMBER);
```

Aufgabe *referentieller Aktionen* bzgl. einer C-Tabelle ist, bei Veränderungen am Inhalt der P-Tabelle Aktionen auf der C-Tabelle auszuführen, um die referentielle Integrität der Datenbasis zu erhalten. Ist dies nicht möglich, so werden die gewünschten DELETE/UPDATE-Operationen nicht ausgeführt, bzw. zurückgesetzt.

8.1 Referentielle Aktionen im SQL-2 Standard

Nach dem *SQL-2-Standard* werden referentielle Integritätsbedingungen werden bei CREATE TABLE und ALTER TABLE als <columnConstraint> (für einzelne Spalten)

```
CONSTRAINT <name>
  REFERENCES <table'> (<attr'>)
```

```
[ ON DELETE { NO ACTION | RESTRICT | CASCADE | SET DEFAULT | SET NULL } ]
[ ON UPDATE { NO ACTION | RESTRICT | CASCADE | SET DEFAULT | SET NULL } ]
```

oder `<tableConstraint>` (für mehrere Spalten) angegeben:

```
CONSTRAINT <name>
  FOREIGN KEY [ (<attr-list>)]
  REFERENCES <table'> (<attr-list'>)
  [ ON DELETE { NO ACTION | RESTRICT | CASCADE | SET DEFAULT | SET NULL } ]
  [ ON UPDATE { NO ACTION | RESTRICT | CASCADE | SET DEFAULT | SET NULL } ]
```

Die Klauseln `ON DELETE` und `ON UPDATE` geben an, welche referentiellen Aktionen bei einem `DELETE` bzw. `UPDATE` auf die referenzierte Tabelle ausgeführt werden sollen, um die referentielle Integrität der Datenbasis zu gewährleisten.

1. Ein `INSERT` bzgl. der referenzierten Tabelle oder ein `DELETE` bzgl. der referenzierenden Tabelle ist immer unkritisch.
   `INSERT INTO Country VALUES ('Lummerland,'LU',...);`
   `DELETE FROM is_member ('D','EU');`
2. Ein `INSERT` oder `UPDATE` bzgl. der referenzierenden Tabelle, das einen Fremdschlüsselwert erzeugt, zu dem kein Schlüssel in der referenzierten Tabelle existiert, ist immer unzulässig:
   `INSERT INTO City VALUES ('Karl-Marx-Stadt','DDR',...);`
   anderenfalls ist es unkritisch:
   `UPDATE City SET Country='A' WHERE Name='Munich';`
3. Notwendig sind damit nur referentielle Aktionen für `DELETE` und `UPDATE` bzgl. der referenzierten Tabelle:
   `UPDATE Country SET Code='UK' WHERE Code='GB';` oder
   `DELETE FROM Country WHERE Code='I';`

`NO ACTION`:

Die `DELETE/UPDATE`-Operation auf der P-Tabelle wird zunächst ausgeführt; *Nach der Operation* wird überprüft, ob "dangling references" in der C-Tabelle entstanden sind. Falls ja, war die Operation verboten und wird zurückgenommen.

```
CREATE TABLE River
(Name VARCHAR2(20) CONSTRAINT RiverKey PRIMARY KEY,
 River VARCHAR2(20) REFERENCES River(Name),
 Lake VARCHAR2(20) REFERENCES Lake(Name),
 Sea VARCHAR2(25) REFERENCES Sea(Name),
 Length NUMBER);

DELETE FROM River;
```

Unter der Annahme, dass es keine weiteren Referenzen auf *River* gäbe (was in MONDIAL nicht der Fall ist, da *located River* referenziert), würden dabei alle Flüsse gelöscht. Referenzen durch Nebenflüsse, die ebenfalls gelöscht werden, sind dabei kein Hindernis (da der Datenbankzustand *nach* der kompletten Operation betrachtet wird).

`RESTRICT`:

Die `DELETE/UPDATE`-Operation auf der P-Tabelle wird nur dann ausgeführt, wenn keine "dangling references" in der C-Tabelle entstehen können:

```
DELETE FROM Organization;
```

wird abgebrochen, falls es eine Organisation gibt, die Mitglieder hat (Referenz von *is_ member* auf *Organization*).

`CASCADE:`

Die `DELETE/UPDATE`-Operation auf der P-Tabelle wird ausgeführt. Die referenzierenden Tupel der C-Tabelle werden ebenfalls mittels `DELETE` entfernt, bzw. mittels `UPDATE` geändert. Ist die C-Tabelle selbst P-Tabelle bzgl. einer anderen Bedingung, so wird das `DELETE/UPDATE` bzgl. der dort festgelegten Lösch/Änderungs-Regel weiter behandelt:

```
UPDATE Country SET Code='UK' WHERE Code='GB';
```

wird am sinnvollsten dadurch ausgeführt, dass die Ersetzung für referenzierende Tupel ebenfalls durchgeführt wird:

Country:	(United Kingdom,GB,...)	$\leadsto$	(United Kingdom,UK,...)
Province:	(Yorkshire,GB,...)	$\leadsto$	(Yorkshire,UK,...)
City:	(London,GB,Greater London,...)	$\leadsto$	(London,UK,Greater London,...)

`SET DEFAULT:`

Die `DELETE/UPDATE`-Operation auf der P-Tabelle wird ausgeführt und bei den referenzierenden Tupeln der C-Tabelle wird der entsprechende Fremdschlüsselwert auf die für die betroffenen Spalten festgelegten `DEFAULT`-Werte gesetzt. Dafür muss dann wiederum ein entsprechendes Tupel in der referenzierten Relation existieren. Beispiel: Eine Firmendatenbank, in denen jedes Projekt einem Mitarbeiter zugeordnet ist. Fällt dieser aus, werden seine Projekte zur Chefsache gemacht.

`SET NULL:`

Die `DELETE/UPDATE`-Operation auf der P-Tabelle wird ausgeführt. In der C-Tabelle wird der entsprechende Fremdschlüsselwert durch `NULL` ersetzt. Voraussetzung ist hier, dass `NULL`s zulässig sind.

Beispiel: Die Relation *located* gibt an, an welchem Fluss/See/Meer eine Stadt liegt, z. B.

located(Bremerhaven,Nds.,D,Weser,`NULL`,Nordsee)

Wird nun mit `DELETE * FROM River WHERE Name='Weser';`

das Tupel 'Weser' in der Relation River gelöscht, sollte die Information, dass Bremerhaven an der Nordsee liegt, erhalten bleiben:

located(Bremerhaven,Nds.,D,`NULL`,`NULL`,Nordsee)

8.2 Referentielle Aktionen in ORACLE

In ORACLE 9 sind nur `ON DELETE/UPDATE NO ACTION`, `ON DELETE CASCADE` und `ON DELETE SET NULL` implementiert `:-(`. Als Default wird `NO ACTION` angenommen; damit ist nur optional anzugeben, falls `ON DELETE CASCADE` oder `ON DELETE SET NULL` verwendet werden soll:

```
CONSTRAINT <name>
  REFERENCES <table'> (<attr'>) [ON DELETE CASCADE]
```

für `<columnConstraint>` bzw.

```
CONSTRAINT <name>
  FOREIGN KEY [ (<attr-list>)]
  REFERENCES <table'> (<attr-list'>)
  [ON DELETE CASCADE]
```

für `<tableConstraint>` (für mehrere Spalten).

Insbesondere die Tatsache, dass `ON UPDATE CASCADE` fehlt, ist beim Durchführen von Updates ziemlich lästig:

Beispiel 9 (Umbenennung eines Landes) Für die Tabelle *Country* ist das Kürzel *Code* als `PRIMARY KEY` definiert. *Code* wird u. a. in *Province* referenziert, d.h. ist dort *Fremdschlüssel*:

```
CREATE TABLE Country
  ( Name  VARCHAR2(32) NOT NULL UNIQUE,
    Code  VARCHAR2(4) CONSTRAINT CountryKey PRIMARY KEY);

CREATE TABLE Province
  ( Name    VARCHAR2(32)
    Country VARCHAR2(4) CONSTRAINT ProvRefsCountry
                    REFERENCES Country(Code));
```

Die beiden Tabellen enthalten unter anderem die Tupel ('United Kingdom','GB') und ('Yorkshire','GB'). Nun soll das Landeskürzel von 'GB' nach 'UK' geändert werden.

- `UPDATE Country SET Code='UK' WHERE Code='GB'` führt zu einer "dangling reference" des Tupels ('Yorkshire','GB').
- will man zuerst `UPDATE Province SET Code='UK' WHERE Code='GB'` ändern, gibt es kein zu referenzierendes Tupel für ('Yorkshire','UK').

Damit muss man zuerst die referentielle Integritätsbedingung außer Kraft setzen, dann die Updates vornehmen, und danach die referentielle Integritätsbedingung wieder aktivieren:

```
ALTER TABLE Province DISABLE CONSTRAINT ProvRefsCountry;
UPDATE Country SET Code='UK' WHERE Code='GB';
UPDATE Province SET Country='UK' WHERE Country='GB';
ALTER TABLE Province ENABLE CONSTRAINT ProvRefsCountry;
```

□

Man kann ein Constraint auch bei der Tabellendefinition mitdefinieren, und sofort disablen:

```
CREATE TABLE <table>
   ( <col> <datatype> [DEFAULT <value>]
         [<colConstraint> ... <colConstraint>],
     ⋮
     <col> <datatype> [DEFAULT <value>]
         [<colConstraint> ... <colConstraint>],
     [<tableConstraint>],
     ⋮
     [<tableConstraint>])
     DISABLE ...
     ⋮
     DISABLE ...
     ENABLE ...
     ⋮
```

```
        ENABLE ...;
```

In dem oben beschriebenen Fall stellt “nur” das Ändern von Daten ein Problem dar – das mit `ON UPDATE CASCADE` gelöst werden könnte. Ein analoges Problem ergibt sich aus gegenseitigen und zyklischen Referenzen zwischen verschiedenen Tabellen:

```
CREATE TABLE Country
   ( Name VARCHAR2(32),
     Code VARCHAR2(4) PRIMARY KEY,
     Capital VARCHAR2(35),
     Province VARCHAR2(32),
     ⋮
     CONSTRAINT CountryCapRefsCity
        FOREIGN KEY (Capital,Code,Province)
        REFERENCES City(Name,Country,Province));
CREATE TABLE Province
   ( Name VARCHAR2(32),
     Country VARCHAR2(4),
     Capital VARCHAR2(35),
     ⋮
     PRIMARY KEY (Name, Country),
     CONSTRAINT ProvRefsCountry
        FOREIGN KEY (Country)
        REFERENCES Country(Name)),
     CONSTRAINT ProvCapRefsCity
        FOREIGN KEY (Capital,Country,Name)
        REFERENCES City(Name,Country,Prov));
CREATE TABLE City
   ( Name VARCHAR2(35),
     Country VARCHAR2(4),
     Province VARCHAR2(32),
     ⋮
     PRIMARY KEY (Name, Country, Province),
     CONSTRAINT CityRefsProv
        FOREIGN KEY (Country, Province)
        REFERENCES Province(Country, Name));
```

Ein Einfügen von Daten in diese Tabellen ist so nicht möglich, da in der jeweils anderen Tabelle noch nichts existiert. In diesem Fall wird man z. B. `CityRefsProv` bei der Tabellendefinition disablen, dann die Relationen *City*, *Province* und *Country* “von unten her” füllen, und danach das Constraint durch

```
ALTER TABLE Country
  ENABLE CONSTRAINT CityRefsProv;
```

aktivieren.

Wie bereits in Abschnitt 2.3 gesagt, können Tabellen, auf die noch eine referentielle Integritätsbedingung zeigt, mit dem einfachen `DROP TABLE`-Befehl nicht gelöscht werden. Mit

```
DROP TABLE <table> CASCADE CONSTRAINTS;
```

wird eine Tabelle mit allen auf sie zeigenden referentielle Integritätsbedingungen gelöscht.

9 VIEWS – TEIL 2

9.1 View Updates

Views werden häufig (in Kombination mit der Vergabe von Zugriffsrechten, siehe Abschnitt10) dazu benutzt, den realen Datenbestand für Benutzer in einer veränderten Form darzustellen. Damit ein Benutzer in seiner Sicht(weise) Updates ausführen kann, müssen diese auf die Basisrelationen abgebildet werden. ORACLE verwendet Heuristiken, um aufgrund des Schemas festzustellen, ob eine solche Abbildung eindeutig möglich ist.

Über die Tabelle `USER_UPDATABLE_COLUMNS` des Data Dictionary kann der Benutzer abfragen, welche (View-)Spalten updatable sind (in dieser Abfrage ist es wichtig, den Tabellennamen in Großbuchstaben anzugeben!). In dem folgenden View können alle Spalten außer *Density* verändert werden. Da *Density* ein abgeleiteter Wert ist, kann diese Spalte natürlich nicht direkt verändert werden.

```
CREATE OR REPLACE VIEW temp AS
SELECT Name, Code, Area, Population, Population/Area AS Density
FROM Country;

SELECT * FROM USER_UPDATABLE_COLUMNS
WHERE Table_Name = 'TEMP';
```

Table_Name	Column_Name	UPD	INS	DEL
temp	Name	yes	yes	yes
temp	Code	yes	yes	yes
temp	Area	yes	yes	yes
temp	Population	yes	yes	yes
temp	Density	no	no	no

```
INSERT INTO temp (Name, Code, Area, Population)
   VALUES ('Lummerland','LU',1,4)
SELECT * FROM temp where Code = 'LU';
```

Name	Code	Area	population	Density
Lummerland	LU	1	4	4

Da das View bei der Ausgabe aus der (durch das `INSERT` veränderten) aktuellen Basistabelle *Country* neu berechnet wird, enthält es auch den Wert für *Density*.

Werte, die durch Aggregatfunktionen berechnet wurden, sind in Views ebenfalls nicht veränderbar. Diese Fälle sind relativ einfach damit zu begründen sind, dass berechnete Werte nicht geändert werden können.

```
CREATE VIEW CityCountry (City, Country) AS
     SELECT City.Name, Country.Name
     FROM City, Country
     WHERE City.Country = Country.Code;

SELECT * FROM USER_UPDATABLE_COLUMNS
WHERE Table_Name = 'CITYCOUNTRY';
```

Table_Name	Column_Name	UPD	INS	DEL
CityCountry	City	yes	yes	yes
CityCountry	Country	no	no	no

Über diese Sicht können also Städte(namen) verändert werden. das Einfügen von Tupeln ist nicht möglich, da das Attribut *Country* nicht verändert/inserted werden darf, andererseits *Country* aber als Teil des Schlüssels einer neu eingefügten Stadt angegeben werden müsste.

```
UPDATE CityCountry
SET City = 'Wien'
WHERE City = 'Vienna';

SELECT * FROM City WHERE Country = 'A';
```

Name	Country	Province	...
Wien	A	Vienna	...
⋮	⋮	⋮	⋮

Aber entgegen der oben angegebenen Werte können Daten in *CityCountry* gelöscht werden:

```
SQL> delete from CityCountry where country='Austria';
9 Zeilen wurden gelöscht.
```

Dieser Befehl löscht die betroffenen Städte aus *City*, löscht jedoch *Austria* nicht aus *Country*.

Ein spezielles Problem stellen Join-Views dar, bei denen mehrere Basistabellen verknüpft werden. Generell erlaubt Oracle 8 nicht, dass ein View Update *mehrere* Basistabellen gleichzeitig verändert. Außerdem kommt es auch häufig vor, dass Werte zwar unverändert aus Basistabellen übernommen werden, und es trotzdem nicht möglich ist, eine eindeutige Abbildung der Änderungen auf die Basistabelle zu garantieren. Die Oracle-Heuristiken basieren *nur* auf Schema-Informationen, betrachten also nicht, ob in der *gegebenen Datenbankinstanz* eine eindeutige Umsetzung möglich ist. Dabei spielen Schlüsseleigenschaften eine wichtige Rolle:

- Ist der Schlüssel einer Basistabelle auch gleichzeitig Schlüssel des Views, ist eine eindeutige Umsetzung möglich.
- umfasst der Schlüssel einer Basistabelle einen Schlüssel des Views, ist eine Umsetzung möglich (wobei eine Veränderung/Löschung an einem Tupel des Views möglicherweise mehrere Tupel der Basistabelle beeinflusst):

```
CREATE OR REPLACE VIEW temp AS
SELECT country, name, population
FROM Province A
WHERE population = (SELECT MAX(population)
                    FROM Province B
                    WHERE A.Country = B.Country);
```

```
SELECT * FROM temp WHERE Country = 'D';
```

Country	**Name**	**Population**
D	Nordrhein-Westfalen	17816079

```
UPDATE temp
SET population = 0 where Country = 'D';
SELECT * FROM Province WHERE Country = 'D';
```

Ergebnis: die Bevölkerung der bevölkerungsreichsten Provinz Deutschlands wird auf 0 gesetzt. Damit ändert sich auch die Auswahl der Provinzen für das View !

```
SELECT * FROM temp WHERE Country = 'D';
```

Country	**Name**	**Population**
D	Bayern	11921944

- Ein Einfügen in das folgende View ist nicht möglich, da der Schlüssel der Basisrelation (*Province.Name* und *Province,Country* nicht komplett in den Attributen des Views enthalten ist.

  ```
  CREATE OR REPLACE VIEW CountryProvPop AS
  SELECT country, population
  FROM Province A;
  ```

- Umfasst der Schlüssel einer Basistabelle keinen Schlüssel des Views komplett, ist keine eindeutige Umsetzung mehr möglich (siehe Aufgaben).

Bei Join-Views hat man allgemein das Problem, das man meistens einen Equi-Join über Schlüsselattribute bildet, wobei nur eines der Attribute dann in dem View auftritt – das andere (das zwar mit dem anderen übereinstimmt, aber eben formal nicht dasselbe Attribut ist wird dann nicht auf die darunterliegende Basistabelle umgesetzt.

In dem obigen Beispiel wurde ein Tupel eines Views so modifiziert, dass es aus dem Wertebereich des Views hinausfiel. Da Views häufig verwendet werden, um den "Aktionsradius" eines Benutzers einzuschränken, ist dies in vielen Kontexten unerwünscht und kann durch `WITH CHECK OPTION` verhindert werden:

Beispiel 10 Ein Benutzer soll *nur* mit US-amerikanischen Städte arbeiten.

```
CREATE OR REPLACE VIEW UScities AS
SELECT *
FROM City
WHERE Country = 'USA'
WITH CHECK OPTION;

UPDATE UScities
```

```
 SET Country = 'D' WHERE Name = 'Miami';
```

liefert die Fehlermeldung

```
FEHLER in Zeile 1:
ORA-01402: Verletzung der WHERE-Klausel einer View WITH CHECK OPTION
```

Es ist übrigens erlaubt, Tupel aus dem View zu *löschen*. □

9.2 Materialized Views; View Maintenance

Views werden bei jeder Anfrage neu berechnet. Dies hat den Vorteil, dass sie immer den aktuellen Datenbankzustand repräsentieren. Bei der Verwendung großer Views über teilweise nur selten veränderten Daten – wie sie im realen Leben häufig vorkommen – ist eine ständige komplette Neuberechnung jedoch ineffizient. Zu diesem Zweck können *Materialized Views* eingesetzt werden, die bei jeder Datenänderung automatisch aktualisiert werden (dies kann u. a. durch Trigger (vgl. Abschnitt 14 geschehen). Materialized Views werden im Praktikum nicht behandelt. Die Probleme, die sich aus der Aktualisierung ergeben werden unter dem Stichwort *View Maintenance* zusammengefasst.

Sowohl *View Updates* als auch *View Maintenance* sind aktuelle Forschungsthemen (Theorie und Implementierung).