SQL
Constraints & Integrität

Foliensatz 3

DI(FH) Gerald Aistleitner, 2015/16

Datenintegrität

- Bezeichnet die Korrektheit der Daten aus Anwendersicht (= logische Korrektheit)
- Einhaltung sollte vom System geprüft werden
- Definition von Integritätsbedingungen mittels ***CONSTRAINTS***

Constraints

- Können benannt werden, andernfalls generiert ORACLE einen Namen mit folgendem Format: SYS_Cnnnnn

- Erstellen der Constraints
  - beim Erstellen einer Tabelle
  - nachträglich (zB erst nach Datenimport)

- Definition auf Spalten – oder Tabellenebene

- Abfrage vorhandener Constraints im DD

```
SELECT * FROM all_constraints;
```

Erstellen von Constraints

- Syntax

```
CREATE TABLE table_name (
  column_name data_type [DEFAULT expression]
                        [column_constraint]
  [, column_name …]
  [, table_constraint, …]
)
```

- Nachträgliches Erstellen

```
ALTER TABLE table_name
ADD (table_constraint)
```

- table_constraint:

```
[CONSTRAINT constraint_name]
constraint_type (column_name1
                 [,column_name2, ..]))
```

NOT NULL Integrität

- Syntax – column integrity:

```
column_name …
[CONSTRAINT constraint_name] {NULL |
                              NOT NULL}
```

- Stellt sicher, dass der Feldinhalt nicht NULL wird

Primärschlüssel-Integrität

- Syntax – column integrity:

```
column_name … [CONSTRAINT constraint_name]
PRIMARY KEY
```

- Syntax – table integrity:

```
column_name …,
[CONSTRAINT c_name]PRIMARY KEY (
   column_name1 [,column_name2, ..]);
```

- Syntax – nachträgliche Definition:

```
ALTER TABLE table_name ADD
([CONSTRAINT c_name] PRIMARY KEY (
   column_name1 [,column_name2, …]));
```

- Primärschlüssel kann bis zu 32 Spalten umfassen
- (Kombinierter) Wert muss eindeutig sein, NOT NULL

UNIQUE Integrität (Alternate Key Integrity)

- Syntax – column integrity:
  ```
  column_name … [CONSTRAINT c_name] UNIQUE
  ```
- Syntax – table integrity:
  ```
  column_name …,
  [CONSTRAINT constraint_name]
  UNIQUE (column_name1 [,column_name2, …]),
  ```
- Stellt die Eindeutigkeit des Wertes bzw. der Wertekombination bei Angabe mehrerer Spalten sicher
- Darf einmal NULL enthalten (im Gegensatz zu PRIMARY KEY)

Fremdschlüssel-Integrität

- Syntax – column integrity:

```
column_name … [CONSTRAINT constraint_name]
REFERENCES table_name[(column_name1
                       [,column_name2,...])]
[ON DELETE CASCADE | SET NULL]
```

- Syntax – table integrity:

```
column_name …,
[CONSTRAINT constraint_name]
FOREIGN KEY (column_name1 [,col_name2, ..])
REFERENCES table_name[(col_name1[,...])]
[ON DELETE CASCADE | SET NULL]
```

- Bsp.:

```
CREATE TABLE teams (
teamno NUMBER(2) PRIMARY KEY,
playerno NUMBER(4) REFERENCES players, ...)
```

Fremdschlüssel-Integrität

- Tabellen, auf die verwiesen wird, muss ein Primärschlüssel oder Alternate Key definiert sein.
- Bei Verweis auf den Primärschlüssel muss kein Spaltenname angegeben werden, sonst schon!
- Anzahl und Datentypen des Primärschlüssels und Fremdschlüssels müssen übereinstimmen.
- Auch der Primärschlüssel oder Teile davon können Fremdschlüssel sein.

Fremdschlüssel-Prüfungen

- ***ON DELETE CASCADE***

  Löschen des Elternsatzes bewirkt automatisches Löschen der Kindsätze, damit diese nicht in der Luft hängen!

- ***ON DELETE SET NULL***

  Löschen des Elternsatzes setzt Referenzen darauf auf NULL!

Check-Integrität

- Syntax – column integrity:
  ```
  column_name … [CONSTRAINT constraint_name]
  CHECK (condition)
  ```

- Syntax – table integrity:
  ```
  column_name …,
  [CONSTRAINT c_name] CHECK (condition)
  ```

- Bedingung kann für eine oder mehrere Spalten angegeben werden (mehrere → Tabellenebene)

- Bedingung darf keine Subquery oder Pseudospalten enthalten!

- Bedingung ist in Klammern zu setzen!

Löschen von Integritätsbedingungen

- Syntax:

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name
```

- Evtl. vorher Namen der Constraint abfragen mit:

```
SELECT * from USER_CONSTRAINTS;
```

- Löschen eines Primärschlüssels:
  Wird zurückgewiesen, wenn Fremdschlüssel auf den Primärschlüssel verweisen.

Nummernfolgen / Sequenzen

- Oft wird ein künstlicher Schlüssel (Surrogat) benötigt, der nichts mit der Realität zu tun hat.
  → fortlaufende Nummer

- **Variante 1: MAX()+1**
  SELECT MAX(teamno)+1 FROM teams;
  INSERT INTO teams VALUES (…);
  → Problem: Mehrfachvergabe möglich!
  → Lösung: Tabelle sperren?!

- **Variante 2: eigene (evtl. gemeinsame) Nummerntabelle**
  nur die Nummerntabelle muss gesperrt werden!
  → aufwendige, organisatorische Lösung
  → gemeinsame Tabelle: Flaschenhals

Sequenzen

- **Variante 3: Sequence**
  generiert autom. Eindeutige Integerzahlen

- Syntax:

```
CREATE SEQUENCE seq_name
[START WITH integer]
[INCREMENT BY integer]
[{MAXVALUE integer | NOMAXVALUE}]
[{MINVALUE integer | NOMINVALUE}]
[{CYCLE | NOCYCLE}]
[{ORDER | NOORDER}]
[{CACHE integer | NOCACHE}]
```

- ```
  Create SEQUENCE customers_seq
  START WITH 1000
  INCREMENT BY 1;
  ```
  ```

Sequenzen

- ```
  INSERT INTO customers (id, name)
  VALUES (customers_seq.nextval, 'Muster');
  ```

- Aktuellen Wert anzeigen:
  ```
  select customers_seq.currval from dual;
  ```

- Bearbeiten einer Sequenz / Syntax:
  ```
  ALTER SEQUENCE seq_name
  [INCREMENT BY integer]
  [{MAXVALUE integer | NOMAXVALUE}]
  [{MINVALUE integer | NOMINVALUE}]
  [{CYCLE | NOCYCLE}]
  [{ORDER | NOORDER}]
  [{CACHE integer | NOCACHE}]
  ```

- ```
  DROP SEQUENCE seq_name
  ```

VIEWS

- In der Datenbank gespeicherte Abfragen
- Stellen „virtuelle Tabellen“ dar, deren Inhalt und Struktur auf anderen Tabellen oder Views basieren.
- => Gestaltung des ***externen Schemas***
- Einsatz:
  - um den DB-Zugriff einzuschränken
  - um komplexe Abfragen zu vereinfachen (Joins)
  - um Datenunabhängigkeit zu ermöglichen
  - verschiedene Sichten auf gleiche Daten

VIEWS

- Syntax:

```
CREATE [OR REPLACE] VIEW view_name
[(column_name1 [, column_name2, …])]
AS SELECT …....
[WITH CHECK OPTION [CONSTRAINT constr_name]]
[WIDTH READ ONLY]
```

- Zugriff auf die View wie auf eine Basistabelle
- Kein ORDER BY erlaubt (beim Aufruf schon!)
- Eine andere View kann bei der Definition verwendet werden
- Neue Spaltennamen möglich, ansonsten gleich wie die Spaltennamen der Query. Alias ist möglich!

TRANSAKTIONEN

- Stellen sicher, dass die Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand überführt werden kann.

- Ist ein Übergang zu einem konsistenten Zustand nicht möglich, so muss die Transaktion vollständig zurückgerollt werden. (rollback)

- LOCK:
  Mechanismus, der konkurrierenden Zugriff gleichzeitiger Transaktionen verhindert.

Transaktionen

- Syntax:
  - COMMIT [WORK]
  - ROLLBACK [WORK]
  - LOCK table_name1 [, table_name2, …]
    IN ROW SHARE
       ROW EXCLUSIVE
       SHARE UPDATE
       SHARE
       SHARE ROW EXCLUSIVE
       EXCLUSIVE
    [NOWAIT]
  - SET TRANSACTION READ ONLY
  - SET AUTOCOMMIT {ON | OFF}

INDIZES

- Datenstruktur zur Steigerung der Anfrageoptimierung
- Definierte Spalten werden in einer Art Tabelle mit Wert und Satzadresse abgeleitet.
- Suche über hochperformante Algorithemen (zB B-Bäume)
- Index wird meist im Hauptspeicher gehalten
- Syntax:

```
CREATE [UNIQUE] INDEX index_name
ON table_name (col_name1 [,col_name2, …]);

DROP INDEX index_name;
```

INDIZES

Faustregeln für die Erstellung eines Index:

Es ist sinnvoll ...	**Es ist nicht sinnvoll ...**
... aus **Integritätsgründen** einen unique index zu erstellen.	... über ein Attribut einen Index zu definieren, das nur **wenige unterschiedliche Werte** enthält (ausgenommen Bitmap Index)
... auf **Fremdschlüssel** einen Index zu definieren, da die meisten Joins über die Beziehung Primärschlüssel <> Fremdschlüssel laufen.	... eine Abfrage auf **<>** durch einen Index zu unterstützen.
... über Attribute einen Index zu definieren, wenn nach diesen oft **abgefragt** wird.	... über ein Attribut einen Index zu definieren, das sehr **oft Null** enthält.
... über Attribute einen Index zu definieren, wenn nach diesen oft **sortiert** wird.	

INDIZES

Index sollte erstellt werden wenn:

- ✓ die Spalte häufig in der WHERE-Klausel oder in einer JOIN-Bedingung verwendet wird
- ✓ die Spalte einen großen Wertebereich enthält
- ✓ die Spalte keine hohe Anzahl von NULL-Werten enthält
- ✓ zwei oder mehr Spalten häufig zusammen in einer WHERE-Klausel oder einer JOIN-Bedingung verwendet werden
- ✓ es sich um eine große Tabelle handelt, und die meisten Abfragen rufen erwartungsgemäß weniger als 2-4% der Zeilen ab.

Index sollte nicht erstellt werden, wenn:

- x es sich um eine kleine Tabelle handelt
- x die Spalten nicht oft als Bedingung in der Abfrage verwendet werden
- x die meisten Abfragen erwartungsgemäß mehr als 2-4% der Zeilen abrufen
- x die Tabelle häufig aktualisiert wird.