Advanced SQL verstehen und einsetzen

PHP-User-Group Stuttgart

10.06.2009

- Aktuelle Datenbank-Features verstehen und nutzen
- **SQL-Power** aktivieren anstatt *Arrays sortieren*
- Valide Daten garantieren und performante DB-Zugriffe sicherstellen
- Ein Blick diesseits und jenseits des SQL-Standards

SQL-Implementierungen kennen und bewerten

Advanced SQL verstehen und einsetzen

Wer bin ich ?

Thomas Wiedmann

- > 20 Jahre Problemlösungen in der Softwareentwicklung
- Seit fünf Jahren Projekte mit PHP und Oracle PL/SQL bzw. DB2/NT
- Zend Certified PHP Engineer (ZCE)
- IBM Certified Solution Expert - DB2 UDB v7.1 Database Administration
- Autor diverser Fachartikel in der „Toolbox“ und im PHP-Magazin
- Autor des Buches „DB2 – SQL, Programmierung, Tuning“ © 2001
- Plus diverse Tutorials auf meiner Homepage http://www.twiedmann.de

Advanced SQL verstehen und einsetzen

Datenbanksysteme

Für meine Beispiele, habe ich die folgenden freien bzw. kostenlosen Datenbanksysteme (jeweils in der Windows-Version) verwendet.

- IBM DB2 9.1 Express-C
  http://www-01.ibm.com/software/data/db2/9/edition-express-c.html
- MySQL 5.1.30 (InnoDB)
  http://dev.mysql.com/downloads/mysql/5.1.html
- Oracle 10g Express Edition Release 10.2.
  http://www.oracle.com/technology/software/products/database/xe/index.html

Advanced SQL verstehen und einsetzen

Themenüberblick

Überblick

- SQL Standard? (Was'n das..)
- CHECK Constraint
- INSERT mehrzeilig
- CASE Konstrukte
- Trigger (INSERT, UPDATE, DELETE)
- Zahlengeneratoren um einen eindeutigen Primary Key zu erzeugen (Autoincrement, Generated Always, Sequence etc.)

Advanced SQL verstehen und einsetzen

SQL Standard

- **S**tructured **E**nglish **QUE**ry **L**anguage = **SEQUEL** (IBM 1974, Dr. E.F. Codd)

- SQL-86 (1986-ANSI) (Level 1, Level 2)
- SQL-89 (1989-ANSI) (Level 1, Level 2, Integritätserweiterungen)

Advanced SQL verstehen und einsetzen

SQL Standard

- **S**tructured **E**nglish **QUE**ry **L**anguage = **SEQUEL** (IBM 1974, Dr. E.F. Codd)

- SQL-86 (1986-ANSI) (Level 1, Level 2)
- SQL-89 (1989-ANSI) (Level 1, Level 2, Integritätserweiterungen)
- SQL-87 (1987-ISO)
- SQL-89 (1989-ISO)

- **SQL-92 auch SQL2 (1992 – ISO/ANSI)**
  (Entry Level, Intermediate Level, Full Level)

Advanced SQL verstehen und einsetzen

SQL Standard

- **S**tructured **E**nglish **QUE**ry **L**anguage = **SEQUEL** (IBM 1974, Dr. E.F. Codd)

- SQL-86 (1986-ANSI) (Level 1, Level 2)
- SQL-89 (1989-ANSI) (Level 1, Level 2, Integritätserweiterungen)
- SQL-87 (1987-ISO)
- SQL-89 (1989-ISO)

- **SQL-92 auch SQL2 (1992 – ISO/ANSI)**
  (Entry Level, Intermediate Level, Full Level)

- SQL:1999 auch SQL3 auch SQL-99 (unterteilt in Parts u. a. SQL/XML, ...)
- SQL:2003 auch SQL4 (unterteilt in Parts u. a. SQL/XML, SQL/OLAP, SQL/MED)
- SQL:2008 auch SQL5 (unterteilt in Parts u. a. SQL/XML, SQL/OLAP, SQL/MED)

Quelle: Norbert Denne, DB2 Theorie und Praxis, DGD Verlag, ISBN 3-929187-04-3
Quelle: Can Türcker, SQL:1999 & SQL:2003, dpunkt.verlag, ISBN 3-89864-219-4

Advanced SQL verstehen und einsetzen

CHECK Constraint

Innerhalb dem CREATE TABLE Befehl können für die einzelne Spalten zusätzliche Feldvalidierungen festgelegt werden.

Die Checkbedingung muss **TRUE** sein.

Die Checkbedingungen werden bei INSERT, UPDATE und nachträglich bei ALTER TABLE geprüft. Im Fehlerfall wird ein SQL-Fehler ausgegeben. CHECK gehört zur **D**ata **D**efinition **L**anguage (DDL).

```
CREATE TABLE personal
[...]
 geschlecht CHAR(1) NOT NULL,

 CONSTRAINT person_geschlecht
  CHECK (geschlecht IN ( 'M', 'W' )),
[...]
```

Advanced SQL verstehen und einsetzen

IBM DB2

```
CREATE TABLE personal (

 person_id INT NOT NULL,
 vorname VARCHAR(50) NOT NULL,
 nachname VARCHAR(50) NOT NULL,
 geschlecht CHAR(1) NOT NULL,

 CONSTRAINT pers_geschlecht
  CHECK (geschlecht IN ( 'M', 'W' )),

 PRIMARY KEY (person_id)
);

INSERT INTO personal
( person_id, vorname, nachname, geschlecht ) VALUES
( 1, 'Vorname', 'Nachname', 'm' );
```

=> SQL0545N Die angeforderte Operation ist nicht zulässig, da eine Zeile gegen die Prüfung auf Integritätsbedingung "WIEDMANN.PERSONAL.PERS_GESCHLECHT" verstößt. SQLSTATE=23513 ✓

Advanced SQL verstehen und einsetzen

MySQL

```
CREATE TABLE personal (

 person_id INT NOT NULL,
 vorname VARCHAR(50) NOT NULL,
 nachname VARCHAR(50) NOT NULL,
 geschlecht CHAR(1) NOT NULL,

 CONSTRAINT personal_geschlecht
  CHECK (geschlecht IN ( 'M', 'W' )),

 PRIMARY KEY (person_id)
);

INSERT INTO personal
( person_id, vorname, nachname, geschlecht ) VALUES
( 1, 'Vorname', 'Nachname', 'U' );
```

=> Query OK, 1 row affected (0.02 sec) ☹

RTFM: ..The CHECK clause is parsed but ignored by all storage engines.

Advanced SQL verstehen und einsetzen

ORACLE

```
CREATE TABLE personal (

 person_id INT NOT NULL,
 vorname VARCHAR2(50) NOT NULL,
 nachname VARCHAR2(50) NOT NULL,
 geschlecht CHAR(1) NOT NULL,

 CONSTRAINT personal_geschlecht
  CHECK (geschlecht IN ( 'M', 'W' )),

 PRIMARY KEY (person_id)
);

INSERT INTO personal
( person_id, vorname, nachname, geschlecht ) VALUES
( 1, 'Vorname', 'Nachname', 'm' );
```

=> ORA-02290: CHECK-Constraint (SYS.PERSONAL_GESCHLECHT) verletzt ✓

Advanced SQL verstehen und einsetzen

CHECK Constraint

Eignet sich sehr gut für einfache Datenvalidierungen (Assertions) und erhöht die Datenqualität.

```
ALTER TABLE personal
 ADD CONSTRAINT pers_vorname_min
  CHECK (LENGTH (vorname) > 2);
```

Überprüfungen auf andere Tabellen sind mit CHECK derzeit nicht möglich.

```
ALTER TABLE personal
 ADD CONSTRAINT pers_kunde
  CHECK ((SELECT COUNT(*) FROM kunde) > 1 );
```

Hierfür eignen sich eher Foreign Key Definitionen oder Trigger.

DB2 ✓	MySQL ☹	ORACLE ✓

Advanced SQL verstehen und einsetzen

INSERT mehrzeilig

Neben dem typischen Einfügen einer neuen Datenzeile, können auch gleichzeitig mehrere Rows auf einmal eingefügt werden.

Der Befehl wird als ein Block abgearbeitet, spart „Zeit“ und ist performant.

```
INSERT INTO tabelle
(spalte1, spalte2, ...) VALUES
(1      , 'eins' , ...),
(2      , 'zwei' , ...),
(3      , 'drei' , ...);
```

Advanced SQL verstehen und einsetzen

IBM DB2

```
INSERT INTO personal
( person_id, vorname, nachname, geschlecht ) VALUES
( 1, 'Vorname-1', 'Nachname-1', 'M' ),
( 2, 'Vorname-2', 'Nachname-2', 'W' ),
( 3, 'Vorname-3', 'Nachname-3', 'W' );
```

=> DB20000I Der Befehl SQL wurde erfolgreich ausgeführt.

Advanced SQL verstehen und einsetzen

MySQL

```
INSERT INTO personal
( person_id, vorname, nachname, geschlecht ) VALUES
( 1, 'Vorname-1', 'Nachname-1', 'M' ),
( 2, 'Vorname-2', 'Nachname-2', 'W' ),
( 3, 'Vorname-3', 'Nachname-3', 'W' );
```

=> Query OK, 3 rows affected (0.03 sec)
=> Records: 3 Duplicates: 0 Warnings: 0

Advanced SQL verstehen und einsetzen

ORACLE

```
INSERT INTO personal
( person_id, vorname, nachname, geschlecht ) VALUES
( 1, 'Vorname-1', 'Nachname-1', 'M' ),
( 2, 'Vorname-2', 'Nachname-2', 'W' ),
( 3, 'Vorname-3', 'Nachname-3', 'W' );
```

```
=> ORA-00933: SQL-Befehl wurde nicht korrekt beendet
```

```
INSERT INTO personal
( person_id, vorname, nachname, geschlecht )
SELECT 1, 'Vorname-1', 'Nachname-1', 'M' FROM dual UNION ALL
SELECT 2, 'Vorname-2', 'Nachname-2', 'W' FROM dual UNION ALL
SELECT 3, 'Vorname-3', 'Nachname-3', 'W' FROM dual;
```

```
=> 3 row(s) inserted.
```

Advanced SQL verstehen und einsetzen

INSERT mehrzeilig

Vorteil:

Der mehrzeilige INSERT wird von der Datenbank als ein Befehl verarbeitet und ist deshalb performanter als einzelne INSERT Befehle.

Nachteil:

Ist irgendwo ein Fehler, wird der komplette Befehl nicht ausgeführt.

DB2 ✓	MySQL ✓	ORACLE 😐

Advanced SQL verstehen und einsetzen

CASE Befehl

Mit dem CASE können innerhalb einem SELECT Befehl einfache Entscheidungen zu bestimmten Spalteninhalten getroffen werden. CASE kennt zwei Methoden:

- einfacher Vergleich ähnlich dem PHP <?php ... switch(spalte1) { … } ?>
- Suche ähnlich dem WHERE (search-condition)

```
CASE spalte1
 WHEN 1 THEN 'eins'
END AS einfacher_vergleich

-----------------------------------------------

CASE
 WHEN spalte1 = 1 THEN 'eins'
 WHEN spalte1 = 2 THEN 'zwei'
END AS search_condition
```

CASE Beispiel

PERSON_ID	VORNAME	NACHNAME	GESCHLECHT
1	Vorname-1	Nachname-1	M
2	Vorname-2	Nachname-2	W
3	Vorname-3	Nachname-3	W

```
SELECT COUNT(*) AS personen,

       COUNT (
        CASE geschlecht
         WHEN 'M' THEN 1
         ELSE NULL
        END
       ) AS mann,

       COUNT(
        CASE
         WHEN geschlecht = 'W' THEN 1
         ELSE NULL
        END
       ) AS frau

 FROM personal;
```

PERSONEN	MANN	FRAU
3	1	2

Advanced SQL verstehen und einsetzen

CASE Konstrukt

Vorteil:

Mit CASE lassen sich einfache Entscheidungsstrukturen ähnlich dem switch() Befehl oder einer IF THEN ELSE Konstruktion zusammen bauen. Auch „sortieren“ ist möglich, dies kann bei speziellen Reportanforderungen hilfreich sein. Ein Beispiel ...

```
SELECT vorname, nachname
  FROM personal
ORDER BY
 geschlecht,
 CASE geschlecht
  WHEN 'M' THEN vorname || nachname
  WHEN 'W' THEN nachname || vorname
 END ASC;
```

DB2 ✓ MySQL ✓ ORACLE ✓

Advanced SQL verstehen und einsetzen

TRIGGER Definition

Mit TRIGGER werden kleine Datenbankprogramme bezeichnet, die auf bestimmte Ereignisse (zumeist Datenveränderungen) automatisch von der Datenbank ausgelöst werden. Die Trigger-Syntax unterscheidet sich leider sehr stark bei den verschiedenen Datenbanksystemen.

Bevor ein INSERT, UPDATE oder DELETE ausgeführt wird, startet der TRIGGER..

- BEFORE INSERT ..
- BEFORE UPDATE ..
- BEFORE DELETE ..

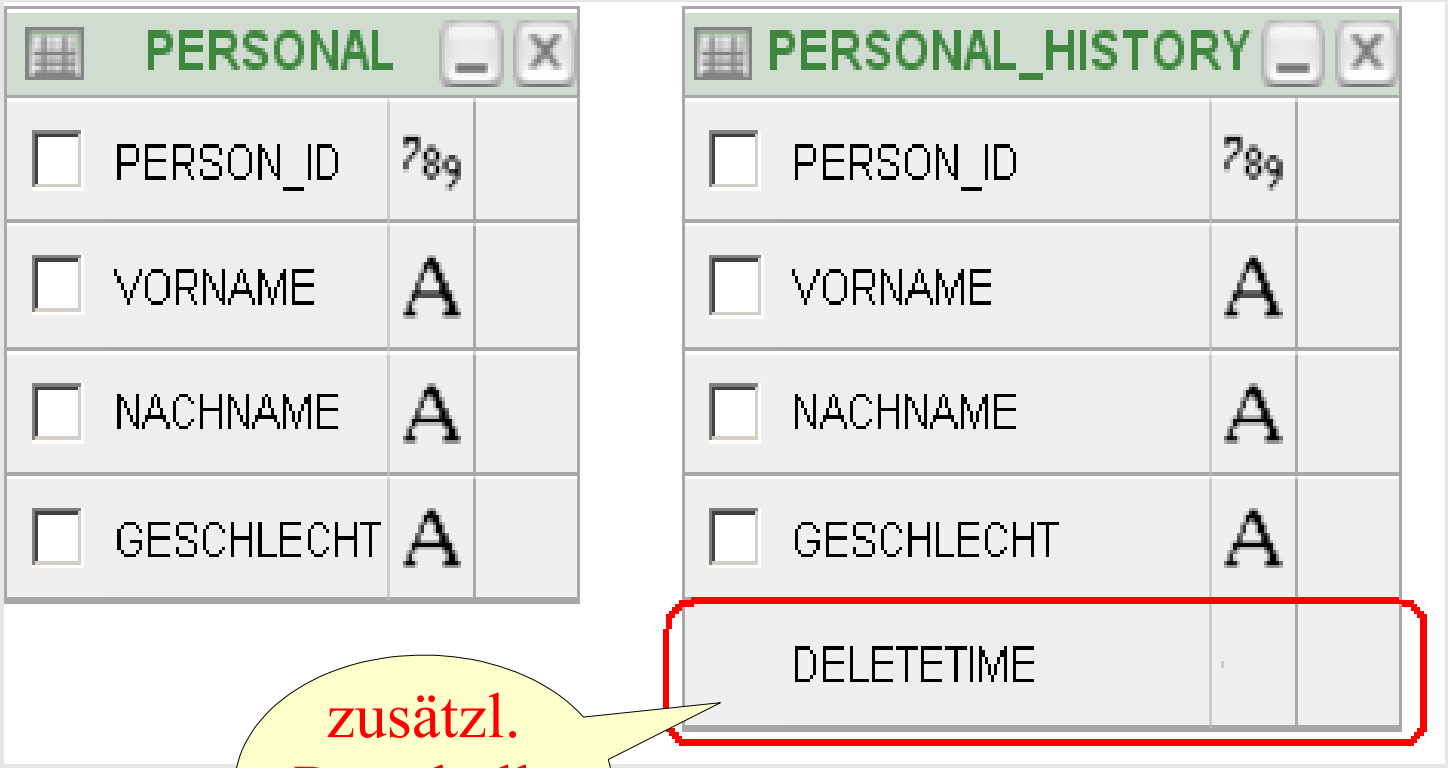
Nachdem ein INSERT, UPDATE oder DELETE ausgeführt wurde, aber noch vor dem COMMIT, startet der TRIGGER..

- AFTER INSERT ..
- AFTER UPDATE ..
- AFTER DELETE ..

Advanced SQL verstehen und einsetzen

Löschprotokoll

Aus der bestehenden Tabelle PERSONAL sollen Datensätze gelöscht und diese Datensätze in eine Protokoll-Tabelle PERSONAL_HISTORY verschoben werden.



Advanced SQL verstehen und einsetzen

Protokoll Variante 1

Dazu sind folgende SQL-Befehle (Schema) notwendig.

```
1) Start Transaktion

2) SELECT person_id, vorname, ... FROM personal
   WHERE person_id = 1;

3) INSERT INTO personal_history
   ( person_id, vorname, ...) VALUES
   ( 1, 'Vorname-1'...);

4) DELETE FROM personal WHERE person_id = 1;

5) Ende Transaktion
```

Advanced SQL verstehen und einsetzen

Protokoll Variante 2

Dazu sind folgende SQL-Befehle (Schema) notwendig.

```
1) Start Transaktion

2) INSERT INTO personal_history
   SELECT person_id, vorname, ... FROM personal
   WHERE person_id = 1;

3) DELETE FROM personal WHERE person_id = 1;

4) Ende Transaktion
```

Advanced SQL verstehen und einsetzen

Protokoll Variante 3

Dazu sind folgende SQL-Befehle (Schema) notwendig.

```
1) Start Transaktion

2) DELETE FROM personal WHERE person_id = 1;

   [AFTER DELETE TRIGGER ...]

3) Ende Transaktion (COMMIT)
```

```
=> SELECT * FROM personal_history;
```

PERSON_ID	VORNAME	NACHNAME	GESCHLECHT	DELETETIME
1	Vorname-1	Nachname-1	M	15.04.2009 19:22:30 281000

Advanced SQL verstehen und einsetzen

DB2 TRIGGER

Mit Hilfe folgendem AFTER DELETE TRIGGERs wird der gelöschte Datensatz aus der Tabelle PERSONAL in der Tabelle PERSONAL_HISTORY archiviert. Zusätzlich wird mit CURRENT_TIMESTAMP der Zeitpunkt des Löschvorgangs protokolliert.

```
CREATE TRIGGER personal_ad_trig
  AFTER DELETE ON personal
  REFERENCING OLD AS old_row
  FOR EACH ROW

  INSERT INTO personal_history
  VALUES (
   old_row.person_id,
   old_row.vorname,
   old_row.nachname,
   old_row.geschlecht,
   CURRENT_TIMESTAMP
  );
```

Advanced SQL verstehen und einsetzen

MySQL TRIGGER

Drei Schritte - braucht der Admin - um einen Trigger in der MySQL Console anzulegen. Das Befehlsende ; muss auf z.B. $$ umgesetzt werden.

```
mysql> DELIMITER $$

mysql> CREATE TRIGGER personal_ad_trig
    ->   AFTER DELETE ON personal
    ->   FOR EACH ROW
    -> BEGIN
    ->  INSERT INTO personal_history
    ->  VALUES (
    ->   OLD.person_id,
    ->   OLD.vorname,
    ->   OLD.nachname,
    ->   OLD.geschlecht,
    ->   NOW()
    ->  );
    -> END$$

mysql> DELIMITER ;
```

Advanced SQL verstehen und einsetzen

ORACLE TRIGGER

Mit Hilfe folgendem AFTER DELETE TRIGGERs wird der gelöschte Datensatz aus der Tabelle PERSONAL in der Tabelle PERSONAL_HISTORY archiviert. Zusätzlich wird mit SYSTIMESTAMP der Zeitpunkt des Löschvorgangs protokolliert.

```
CREATE TRIGGER personal_ad_trig
  AFTER DELETE ON personal
  FOR EACH ROW
BEGIN
  INSERT INTO personal_history
  VALUES (
   :old.person_id,
   :old.vorname,
   :old.nachname,
   :old.geschlecht,
   SYSTIMESTAMP
  );
END;
```

Advanced SQL verstehen und einsetzen

TRIGGER

Vorteil:

Mit Triggern lassen sich sehr gut Feldvalidierungen und Business-regeln definieren und hinterlegen. Da die Trigger automatisch gestartet werden, können diese nicht „umgangen“ werden.

Nachteil:

Der SQL-Standard hilft hier nicht wirklich. Die Trigger-Syntax unterscheidet sich deutlich zwischen den verschiedenen Datenbanksystemen.
Auch Trigger „kosten“ Verarbeitungszeit. Trigger-Cascade-Problematik.

DB2 ✓	MySQL ✓	ORACLE ✓

Advanced SQL verstehen und einsetzen

Zahlengeneratoren für PK

Mit Zahlengeneratoren wie AUTOINCREMENT, SEQUENCE, GENERATED ALWAYS oder mit Triggern bzw. individuellen Zahlenpool-Tabellen lassen sich Werte für den Primary Key (PK) erzeugen.

Ein **Primary Key** ist **eindeutig, NOT NULL** und für gewöhnlich

- numerisch
- ganzzahlig
- zumeist aufsteigend (beginnend ab 1, dann immer +1)
- sollte schnell zu ermitteln/erzeugen sein
- (lückenlos) – z. B. die Rechnungsnummer

Advanced SQL verstehen und einsetzen

MySQL Autoincrement

Unser © **Änterpreis** Warenwirtschaftssystem in der MySQL Portierung enthält diese beiden Tabellen..

```
CREATE TABLE rechnung (
 rechnung_nr INT NOT NULL AUTO_INCREMENT,
 betrag DEC(10,2) NOT NULL,
 PRIMARY KEY (rechnung_nr)
);

CREATE TABLE buchhaltung (
 rechnung_nr INT NOT NULL,
 betrag DEC(10,2),
 bezahlt DATE,
 CONSTRAINT fk_buchhaltung
  FOREIGN KEY (rechnung_nr)
   REFERENCES rechnung (rechnung_nr)
    ON DELETE RESTRICT
);
```

Advanced SQL verstehen und einsetzen

MySQL Autoincrement

Wir dürfen eine Rechnung schreiben und speichern die Daten in unserer Datenbank. Weitergabe des generierten Primary Key der Tabelle RECHNUNG an die abhängige Tabelle BUCHHALTUNG.

```
mysql> INSERT INTO rechnung VALUES
    -> (NULL, 105.99);

mysql> INSERT INTO buchhaltung VALUE
    -> (LAST_INSERT_ID(), 105.99, NULL);

mysql> SELECT * FROM buchhaltung;
+-------------+--------+----------+
| rechnung_nr | betrag | bezahlt  |
+-------------+--------+----------+
|           1 | 105.99 | NULL     |
+-------------+--------+----------+
```

Advanced SQL verstehen und einsetzen

MySQL Autoincrement

```
mysql> INSERT INTO rechnung VALUES
    -> (NULL, 205.99),
    -> (NULL, 305.99);

mysql> INSERT INTO buchhaltung VALUES
    -> (LAST_INSERT_ID(), 205.99, NULL),
    -> (LAST_INSERT_ID(), 305.99, NULL);

mysql> select * from buchhaltung;
+--------------+--------+----------+
| rechnung_nr | betrag | bezahlt |
+--------------+--------+----------+
|            1 | 105.99 | NULL    |
|            2 | 205.99 | NULL    |
|            2 | 305.99 | NULL    |
+--------------+--------+----------+
```

```
mysql> select * from rechnung;
+--------------+--------+
| rechnung_nr | betrag |
+--------------+--------+
|            1 | 105.99 |
|            2 | 205.99 |
|            3 | 305.99 |
+--------------+--------+
```

Advanced SQL verstehen und einsetzen

MySQL LAST_INSERT_ID

RTFM: ..LAST_INSERT_ID() liefert bei Multi-Insert immer den ersten generierten Wert zurück. In diesem Fall also die Ziffer 2.

```
mysql> INSERT INTO rechnung VALUES
    -> (NULL, 205.99),
    -> (NULL, 305.99);

mysql> INSERT INTO buchhaltung VALUES
    -> (LAST_INSERT_ID(), 205.99, NULL),
    -> (LAST_INSERT_ID()+1, 305.99, NULL);
```

```
mysql> select * from buchhaltung;
+--------------+--------+----------+
| rechnung_nr | betrag | bezahlt |
+--------------+--------+----------+
|            1 | 105.99 | NULL    |
|            2 | 205.99 | NULL    |
|            3 | 305.99 | NULL    |
+--------------+--------+----------+
```

```
mysql> select * from rechnung;
+--------------+---------+
| rechnung_nr | betrag |
+--------------+---------+
|            1 | 105.99 |
|            2 | 205.99 |
|            3 | 305.99 |
+--------------+---------+
```

Advanced SQL verstehen und einsetzen

DB2 Generated Always

Unser © **Änterpreis** Warenwirtschaftssystem in der DB2 Portierung enthält diese beiden Tabellen..

```
CREATE TABLE rechnung (
 rechnung_nr INT NOT NULL
  GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
 betrag DEC(10,2) NOT NULL,
 PRIMARY KEY (rechnung_nr)
);

CREATE TABLE buchhaltung (
 rechnung_nr INT NOT NULL,
 betrag DEC(10,2),
 bezahlt DATE,
 CONSTRAINT fk_buchhaltung
  FOREIGN KEY (rechnung_nr)
   REFERENCES rechnung (rechnung_nr)
    ON DELETE RESTRICT
);
```

Advanced SQL verstehen und einsetzen

DB2 Generated Always

Wir dürfen eine Rechnung schreiben und speichern die Daten in unserer Datenbank.

```
INSERT INTO rechnung
(betrag) VALUES (105.99);

INSERT INTO buchhaltung
(rechnung_nr, betrag, bezahlt ) VALUES
(SYSIBM.IDENTITY_VAL_LOCAL(), 105.99, NULL);

SELECT * FROM buchhaltung;
+-------------+--------+----------+
| rechnung_nr | betrag | bezahlt  |
+-------------+--------+----------+
|           1 | 105.99 | NULL     |
+-------------+--------+----------+
```

Advanced SQL verstehen und einsetzen

Autoincrement

Vorteil:

Mit AUTOINCREMENT oder GENERATED ALWAYS Spalten in der Tabellen-definition läßt sich einfach ein eindeutiger Primary Key Wert erzeugen.

Nachteil:

Bei Master-Detail Tabellen unterscheidet sich die Übergabetechnik des PK-Wertes. ORACLE kennt (meines Wissens / derzeit) kein Äquivalent zu AUTOINCREMENT oder GENERATED ALWAYS. ORACLE bietet hierfür die SEQUENCE Funktionalität an.

=> Bei jedem INSERT erhöht sich der Zahlenwert und kann durch ein ROLLBACK nicht zurückgesetzt werden. Kein lückenloser PK garantiert.

DB2 ✓	MySQL ✓	ORACLE ☹

Advanced SQL verstehen und einsetzen

SEQUENCE Definition

Eine Sequence ist ein eigenständiges Datenbankobjekt und wird mit CREATE SEQUENCE … erzeugt. Sequence gehört zu den Zahlengeneratoren wie AUTOINCEMENT, ist allerdings von keiner Tabelle abhängig.

```
CREATE SEQUENCE auftrag_seq
 START WITH 1
 INCREMENT BY 1
 NOCACHE
 NOCYCLE
 ORDER;
```

Bei jedem „NEXTVAL“Aufruf einer SEQUENCE wird der interne Zähler gemäß seiner „INCREMENT BY“ Definition erhöht.

- NEXTVAL holt die nächste neue Nummer vom Zahlengenerator
- PREVVAL / CURRVAL liest die letzte vergebene Nummer vom Zahlengenerator

Advanced SQL verstehen und einsetzen

SEQUENCE Einsatz

SEQUENCE eignet sich sehr gut für einen typischen Primary Key.

DB2

INSERT INTO auftrag VALUES (**NEXTVAL FOR auftrag_seq**, feld2, feld3, ...);
INSERT INTO position VALUES (**PREVVAL FOR auftrag_seq**, feldA, feldB, ...);

ORACLE

INSERT INTO auftrag VALUES (**auftrag_seq.nextval**, feld2, feld3, ...);
INSERT INTO position VALUES (**auftrag_seq.currval**, feldA, feldB, ...);

=> SEQUENCE eignet sich nicht für eine zwingend lückenlose Zahlenfolge. Nach einem SQL-Fehler oder ROLLBACK der Transaktion fehlen die vergebenen Nummern.

DB2 ✓	MySQL ☹	ORACLE ✓

Advanced SQL verstehen und einsetzen

Lückenlose Nummer

Ein Vorschlag, um eine lückenlose Vergabe einer Rechnungsnummer zu garantieren.

Zahlenpool – Tabelle

```
CREATE TABLE zahlenpool (
 jahr INT NOT NULL,
 maxnr INT NOT NULL
);
```

plus vorbelegter Inhalt und Nummerkreise (letzte vergebene Nummer)

JAHR	MAXNR
2007	101
2008	141
2009	42
2010	0

Advanced SQL verstehen und einsetzen

Lückenlose Nummer

Dazu ist folgender Ablauf notwendig.

```
1) Start Transaktion

2) INSERT INTO rechnung VALUES (0, 3200 );

   [BEFORE INSERT TRIGGER …]

3) Ende Transaktion (COMMIT)
```

```
=> SELECT * FROM rechnung;
```

RECHNUNG_NR	BETRAG
43	3200

Advanced SQL verstehen und einsetzen

IBM DB2 - Lückenlose Nummer

```
CREATE TRIGGER rechnung_bi_trig
  NO CASCADE BEFORE INSERT ON rechnung
  REFERENCING NEW AS new_row
  FOR EACH ROW MODE DB2SQL

  SET new_row.rechnung_nr =
  (SELECT maxnr+1 FROM zahlenpool
    WHERE jahr = CAST( CHAR(CHAR(CURRENT DATE,ISO),4) AS INT)
  );

CREATE TRIGGER rechnung_ai_trig
  AFTER INSERT ON rechnung
  REFERENCING NEW AS new_row
  FOR EACH ROW MODE DB2SQL

  UPDATE zahlenpool
     SET maxnr = new_row.rechnung_nr
   WHERE jahr = CAST( CHAR(CHAR(CURRENT DATE,ISO),4) AS INT);
```

Advanced SQL verstehen und einsetzen

MySQL - Lückenlose Nummer

```
DELIMITER $$

CREATE TRIGGER rechnung_bi_trig
  BEFORE INSERT ON rechnung
  FOR EACH ROW
 BEGIN
  DECLARE new_rechnung_nr INTEGER;

  SELECT maxnr+1
    INTO new_rechnung_nr
    FROM zahlenpool
   WHERE jahr = DATE_FORMAT(CURRENT_DATE,'%Y') FOR UPDATE;

  SET NEW.rechnung_nr = new_rechnung_nr;

  UPDATE zahlenpool
     SET maxnr = NEW.rechnung_nr
   WHERE jahr = DATE_FORMAT(CURRENT_DATE,'%Y');

END$$
```

Advanced SQL verstehen und einsetzen

ORACLE - Lückenlose Nummer

```
CREATE TRIGGER rechnung_bi_trig
  BEFORE INSERT ON rechnung
  FOR EACH ROW

BEGIN

  SELECT maxnr+1
    INTO :NEW.rechnung_nr
    FROM zahlenpool
   WHERE jahr = TO_NUMBER(TO_CHAR(SYSDATE,'YYYY'))
   FOR UPDATE;

  UPDATE zahlenpool
     SET maxnr = :NEW.rechnung_nr
   WHERE jahr = TO_NUMBER(TO_CHAR(SYSDATE,'YYYY'));

END;
```

Advanced SQL verstehen und einsetzen

Lückenlose Nummer

Mit Triggern lassen sich sehr gut Konzepte aufbauen um eine lückenlose Zahlenfolge – wie z. B. Die Rechnungsnummer - zu garantieren.

Zu beachten ist hier die Problematik eines Transaktionsabbruchs, bei dem keine Ziffern „verloren“ gehen dürfen, damit keine Lücken entstehen.

DB2 benötigt bei dem gezeigten Konzept zwei Trigger. Ein Multi-Row-Insert wird nicht korrekt abgearbeitet.

DB2 😐	MySQL ✓	ORACLE ✓

Advanced SQL verstehen und einsetzen

Zusammenfassung

	IBM DB2 9.1 Express-C	MySQL 5.1.30	Oracle 10g Express Edition Release Version 10.2.
CHECK	✓	☹	✓
Multi-Row-Insert	✓	✓	😐
CASE	✓	✓	✓
TRIGGER (Delete-History)	✓	✓	✓
Autoincrement (oder ähnliches)	✓	✓	☹
SEQUENCE	✓	☹	✓
Lückenlose Nummer	😐	✓	✓

Advanced SQL verstehen und einsetzen

Zusammenfassung

	IBM DB2 9.1 Express-C	MySQL 5.1.30	Oracle 10g Express Edition Release Version 10.2.
CHECK	✓	☹	✓
Multi-Row-Insert	✓	✓	😐
(Delete-Hist…)	✓	✓	✓
Autoincrement (oder ähnliches)	✓	✓	✓
SEQUEN…	☹	☹	✓
…Nummer	😐	✓	✓

The End!

..Fragen ? Fragen ? Fragen ? Frag… …agen ? …e… ?..

CHECK, CASE, TRIGGER, SEQUENCE, STANDAR…, SQL-POWER