Einstieg in das SQL- und Datenbanktuning

PHP-User-Group Stuttgart

14.01.2009

Warum Datenbanken einen „Hals“ bekommen und was sich dagegen tun lässt. Tuning und Performancesteigerung ohne zusätzliche Hardware.

Ein Einstieg in das SQL- und Datenbanktuning.

Loblied auf den Tabellen-Index!

Einstieg in das SQL- und Datenbanktuning

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

Monitoring- und Tuning-Tipps

Analyse: MySQL

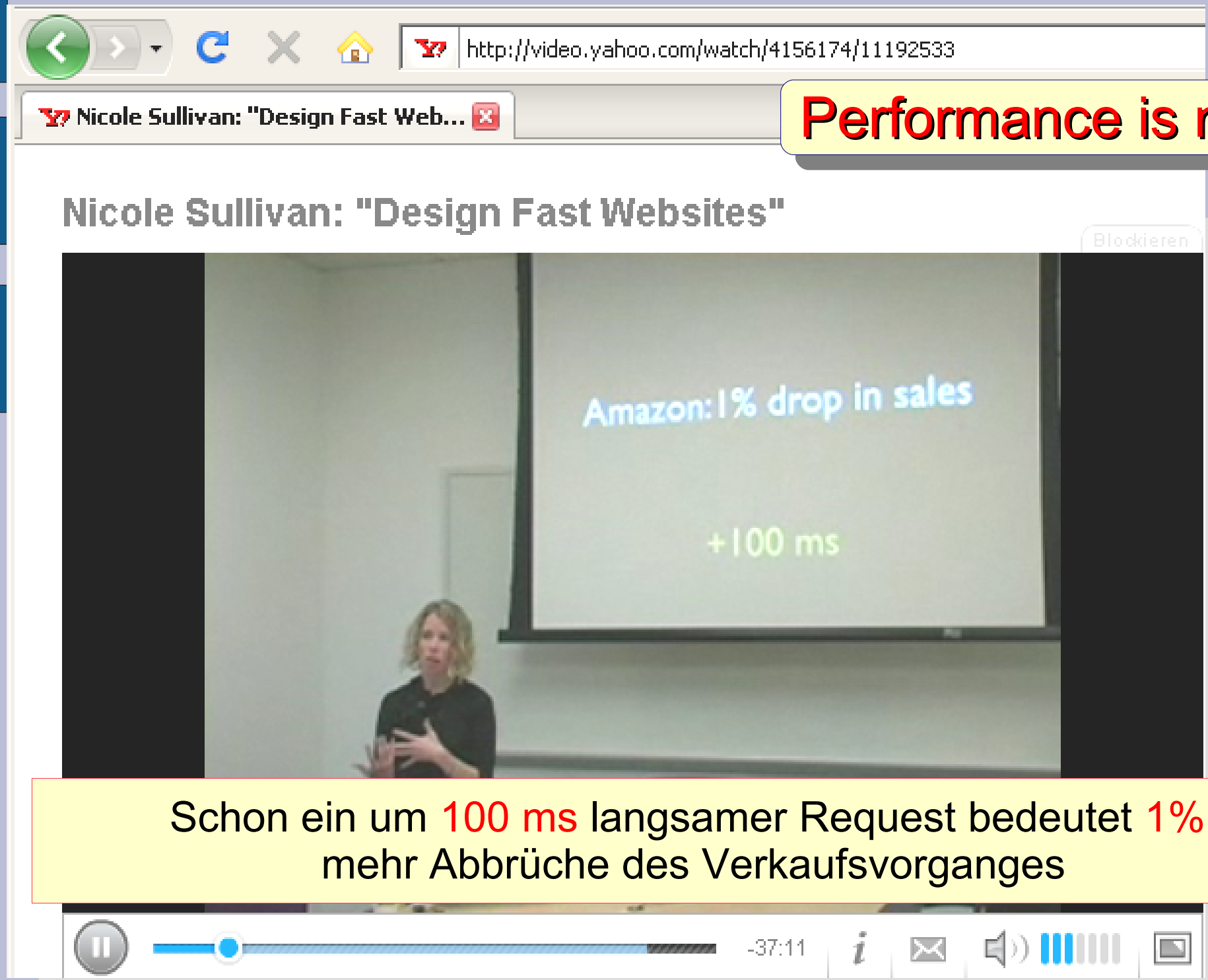
Jens Giessmann
jg@handcode.de
http://www.handcode.de/

In Webapplikationen ist oft (meistens?) die Datenbank der "Flaschenhals".

Möglichkeiten zur Analyse:

- top
- SHOW PROCESSLIST
- SHOW VARIABLES
- SHOW STATUS
- Slow_log aktivieren
  -> mysqldumpslow zur Auswertung
- EXPLAIN nutzen um Queries zu analysieren

Performance is money



Schon ein um 100 ms langsamer Request bedeutet 1% mehr Abbrüche des Verkaufsvorganges

Einstieg in das SQL- und Datenbanktuning

Tabellen und Indices

Welche Indices existieren für die Tabelle KUNDE?

```
CREATE TABLE kunde (

 kunde_id INT NOT NULL,
 name VARCHAR(50) NOT NULL,
 vorname VARCHAR(30) NOT NULL,
 gebdatum DATE,
 plz INT,
 PRIMARY KEY (kunde_id)
);
```

Die meisten Datenbanken (aber nicht alle) erzeugen automatisch auf den PRIMARY KEY einen Index. Ohne PK-Index gilt so eine Tabelle als „incomplete“.

Einstieg in das SQL- und Datenbanktuning

Primary Key

Was bedeutet, was kann ein Primary Key?

- Sichert die Eindeutigkeit einer Zeile in dieser Tabelle.
- Es kann nur einen pro Tabelle geben.
- Er darf keine NULL Werte enthalten.
- Wird benötigt für Fremdschlüssel – Beziehungen (Foreign-Key)
- Beschreibt einen physischen Zugriffspfad zur Basistabelle,
  dies gilt natürlich auch für alle anderen Indices.

Einstieg in das SQL- und Datenbanktuning

Index und Basistabelle

Die Datenbanktabelle KUNDE besteht eigentlich aus einem Index und einer Basistabelle!

Tabelle KUNDE

kunde_id	RID
1	5
2	1
3	2
4	3
5	4

PK-Index

RID	kunde_id	name	vorname	gebdatum	plz
1	2	Name-2	Vorname-2	21.01.1960	70500
2	3	Name-3	Vorname-3	22.01.1960	70501
3	4	Name-4	Vorname-4	23.01.1960	70502
4	5	Name-5	Vorname-5	24.01.1960	70503
5	1	Name-1	Vorname-1	25.01.1960	70504

Basistabelle

B-Baum
sortierte Daten

Basistabelle = unsortierter „Haufen“

Einstieg in das SQL- und Datenbanktuning

Indices?

Wann sind Indices sinnvoll?

- Sichern der Datenkonsistenz
- Kann eine erforderliche Sortierung ersetzen
- Sorgen für einen schnellen lesenden Zugriff auf die Daten

Was spricht gegen Indices?

- Etwas höherer Speicherbedarf, eventuell mehr Administration
- Schreibender Zugriff (Insert, Update, Delete) wird langsamer
- Unnötige und redundante Indices bremsen das System

Einstieg in das SQL- und Datenbanktuning

PHP-Testprogramm

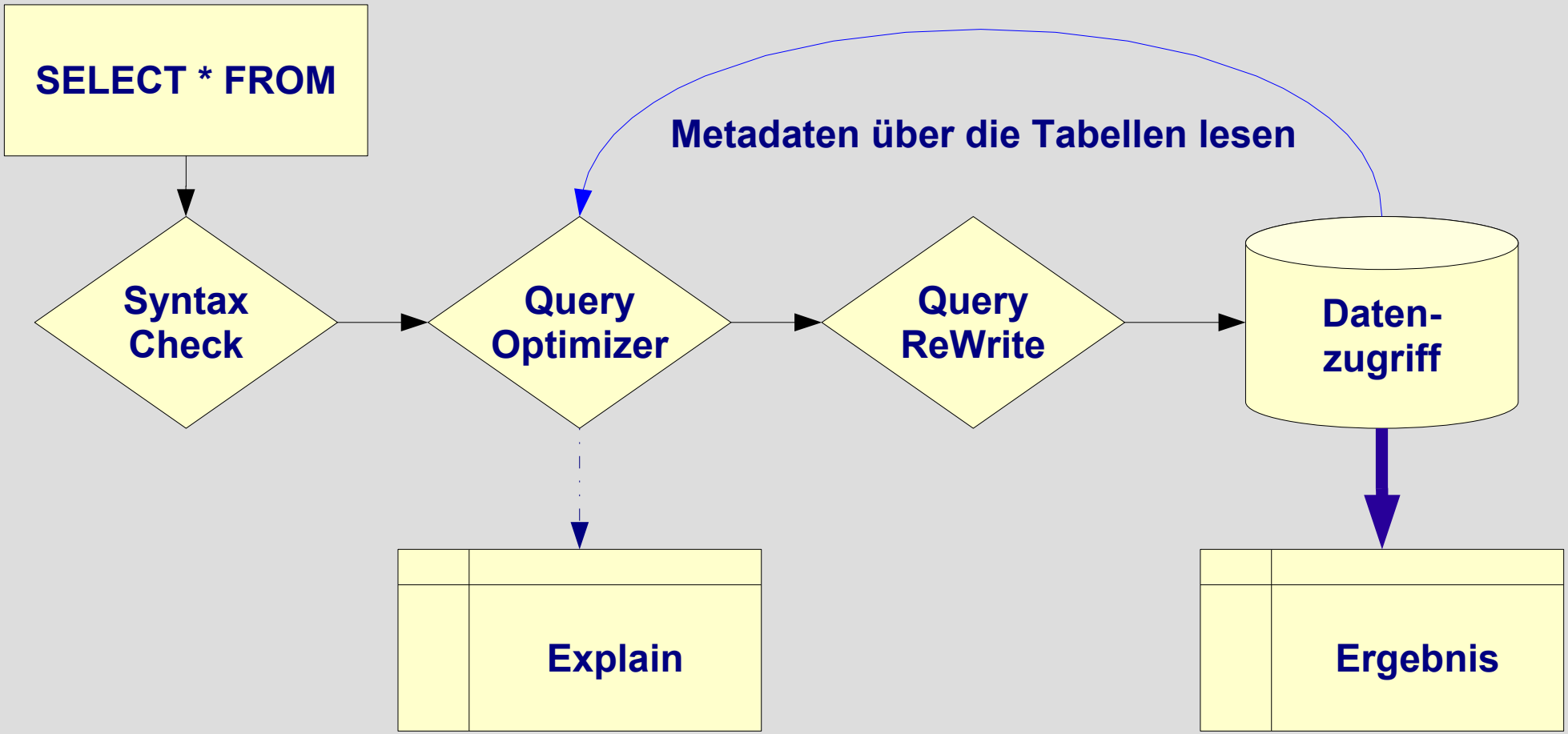
Das PHP-Programm für den Performance-Test in den folgenden Beispielen nutzt PDO sowie ***prepare, bind*** und ***execute***.

```
<?php
 [...]
 $database = new database();
 $dbh = $database->connect_pdo_mysql();
 if ($dbh) {
  $nStart = microtime(true);
  $sQuery = 'SELECT ...
              WHERE warenkorb_id = :nWarenkorb_id
                AND verkauf_id > :nVerkauf_id';
  $stmt = $dbh->prepare($sQuery, array(PDO::ATTR_CURSOR, PDO::CURSOR_FWDONLY));
  $stmt->bindParam('nWarenkorb_id',$nWarenkorb_id);
  $stmt->bindParam('nVerkauf_id',$nVerkauf_id);
  if ($stmt->execute()) {
   $result = $stmt->fetchAll();
  }
  $nEnde = microtime(true);
  $nTime = $nEnde - $nStart;
  echo ' Zeit : '.$nTime;
 }
?>
```

Einstieg in das SQL- und Datenbanktuning

SQL Abfrage

Wenn eine SQL-Abfrage an die Datenbank geschickt wird, passiert dort (vereinfacht) folgendes:



Einstieg in das SQL- und Datenbanktuning

Praxis I

In einem Warenkorb wird der aktuelle Artikel angezeigt. Es sind mehrere Artikel im Warenkorb und es soll dem Kunden möglich sein, zum vorherigen oder nächsten Artikel in seinem Warenkorb zu blättern (navigieren) und dann diesen anzuzeigen.

Die verkauf_id (PK) ist nicht lückenlos, deshalb ist -1 und +1 nicht möglich. (Ziel: Ein reine SQL Lösung)

Warenkorb

<< aktueller Artikel >>

Einstieg in das SQL- und Datenbanktuning

Versuch 1

MySQL 5.0.27-community-nt

```
CREATE TABLE verkauf (

 verkauf_id INT NOT NULL,
 warenkorb_id INT NOT NULL,
 kunde_id INT NOT NULL,
 artikel_id INT NOT NULL,
 datum DATE NOT NULL,
 preis DEC(10,2) NOT NULL,
 menge INT NOT NULL,
 PRIMARY KEY (verkauf_id)
) ENGINE=InnoDB

203350 Datensätze
Verkauf-Id:   1 - 203350
Warenkorb-Id: 1 - 4067
```

Vorwärts blättern:

```
SELECT MIN(verkauf_id)
  FROM verkauf
 WHERE warenkorb_id = :nWK_ID
   AND verkauf_id > :nVk_ID
```

Testergebnisse:

bei **warenkorb_id** = 0001 : 0,066 Sek.
bei **warenkorb_id** = 2000 : 0,035 Sek.
bei **warenkorb_id** = 4000 : 0,001 Sek.

Einstieg in das SQL- und Datenbanktuning

Versuch 2

MySQL 5.0.27-community-nt

```
CREATE TABLE verkauf (

 verkauf_id INT NOT NULL,
 warenkorb_id INT NOT NULL,
 kunde_id INT NOT NULL,
 artikel_id INT NOT NULL,
 datum DATE NOT NULL,
 preis DEC(10,2) NOT NULL,
 menge INT NOT NULL,
 PRIMARY KEY (verkauf_id)
) ENGINE=InnoDB

203350 Datensätze
Verkauf-Id:   1 - 203350
Warenkorb-Id: 1 - 4067
```

Vorwärts blättern:

```
SELECT verkauf_id
  FROM verkauf
 WHERE warenkorb_id = :nWK_ID
   AND verkauf_id > :nVk_ID
 ORDER BY verkauf_id ASC
 LIMIT 1
```

Testergebnisse:

bei **warenkorb_id** = 0001 : 0,00034 S.
bei **warenkorb_id** = 2000 : 0,00034 S.
bei **warenkorb_id** = 4000 : 0,00034 S.

Einstieg in das SQL- und Datenbanktuning

Versuch 3

IBM DB2/NT 9.1 Express-C

```
CREATE TABLE verkauf (

 verkauf_id INT NOT NULL,
 warenkorb_id INT NOT NULL,
 kunde_id INT NOT NULL,
 artikel_id INT NOT NULL,
 datum DATE NOT NULL,
 preis DEC(10,2) NOT NULL,
 menge INT NOT NULL,
 PRIMARY KEY (verkauf_id)
)

203350 Datensätze
Verkauf-Id:   1 - 203350
Warenkorb-Id: 1 - 4067
```

Vorwärts blättern:

```
SELECT MIN(verkauf_id)
  FROM verkauf
 WHERE warenkorb_id = :nWK_ID
   AND verkauf_id > :nVk_ID
```

Testergebnisse:

bei **warenkorb_id** = 0001 : 0,120 Sek.
bei **warenkorb_id** = 2000 : 0,062 Sek.
bei **warenkorb_id** = 4000 : 0,002 Sek.

Einstieg in das SQL- und Datenbanktuning

Versuch 4

IBM DB2/NT 9.1 Express-C

```
CREATE TABLE verkauf (

 verkauf_id INT NOT NULL,
 warenkorb_id INT NOT NULL,
 kunde_id INT NOT NULL,
 artikel_id INT NOT NULL,
 datum DATE NOT NULL,
 preis DEC(10,2) NOT NULL,
 menge INT NOT NULL,
 PRIMARY KEY (verkauf_id)
)

203350 Datensätze
Verkauf-Id:   1 - 203350
Warenkorb-Id: 1 - 4067
```

Vorwärts blättern:

```
SELECT verkauf_id
  FROM verkauf
 WHERE warenkorb_id = :nWK_ID
   AND verkauf_id > :nVk_ID
 ORDER BY verkauf_id ASC
 FETCH FIRST 1 ROW ONLY
```

Testergebnisse:

bei **warenkorb_id** = 0001 : 0,120 Sek.
bei **warenkorb_id** = 2000 : 0,061 Sek.
bei **warenkorb_id** = 4000 : 0,002 Sek.

Einstieg in das SQL- und Datenbanktuning

Versuch 5

IBM DB2 – zusätzlichen Index

```
a)

CREATE INDEX sx_verkauf_01
  ON verkauf (warenkorb_id)

b)

CREATE INDEX sx_verkauf_02
  ON verkauf
  (warenkorb_id,verkauf_id)

203350 Datensätze
Verkauf-Id:    1 - 203350
Warenkorb-Id: 1 - 4067
```

Vorwärts blättern:

```
SELECT MIN(verkauf_id)
  FROM verkauf
 WHERE warenkorb_id = :nWK_ID
   AND verkauf_id > :nVk_ID
```

Testergebnisse:

bei **warenkorb_id** = 0001 : 0,0005 Sek.
bei **warenkorb_id** = 2000 : 0,0005 Sek.
bei **warenkorb_id** = 4000 : 0,0005 Sek.

Einstieg in das SQL- und Datenbanktuning

Versuch 5a

IBM DB2 - Zugriffspfad

a)

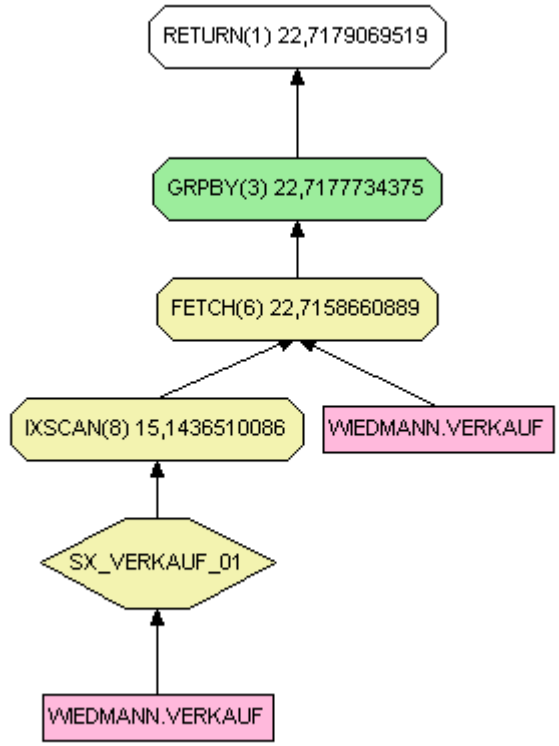
```
CREATE INDEX sx_verkauf_01
  ON verkauf (warenkorb_id)

b)

CREATE INDEX sx_verkauf_02
  ON verkauf
   (warenkorb_id,verkauf_id)
```

203350 Datensätze
Verkauf-Id: 1 - 203350
Warenkorb-Id: 1 - 4067

a)



Einstieg in das SQL- und Datenbanktuning

Versuch 5b

IBM DB2 - Zugriffspfad

a)

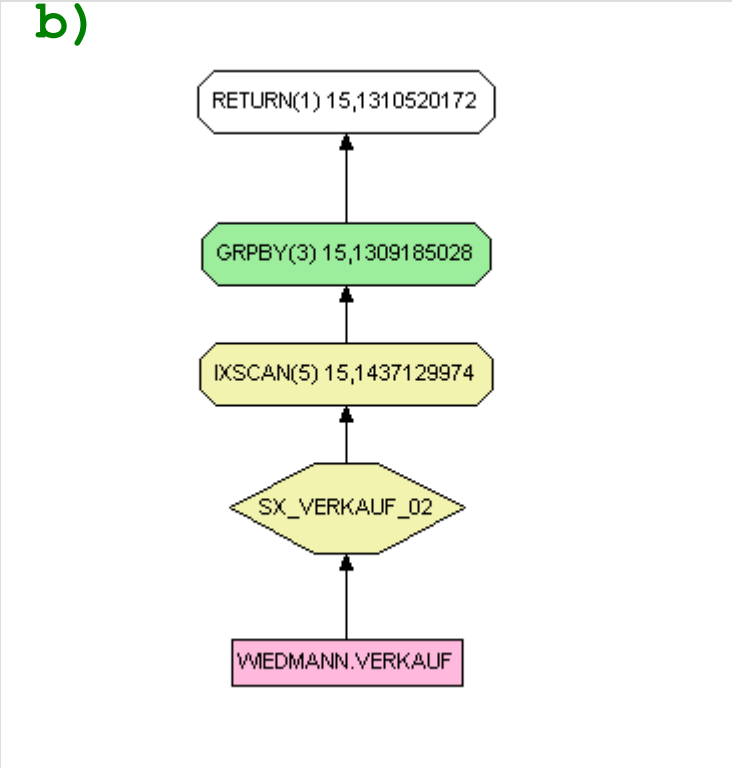
```
CREATE INDEX sx_verkauf_01
  ON verkauf (warenkorb_id)
```

b)

```
CREATE INDEX sx_verkauf_02
  ON verkauf
   (warenkorb_id,verkauf_id)
```

```
203350 Datensätze
Verkauf-Id:    1 - 203350
Warenkorb-Id: 1 - 4067
```

b)



Index-Only-Zugriff
bzw. Indexscan

Einstieg in das SQL- und Datenbanktuning

Versuch 6

MySQL 5.0.27-community-nt

Vorwärts blättern:

```
SELECT MIN(verkauf_id)
  FROM verkauf
 WHERE warenkorb_id = :nWK_ID
   AND verkauf_id > :nVk_ID
```

plus zusätzlicher Index:

```
CREATE INDEX sx_verkauf_02
  ON verkauf
   (warenkorb_id,verkauf_id)
```

Testergebnisse (Versuch 1)

bei **warenkorb_id** = 0001 : 0,066 Sek.
bei **warenkorb_id** = 2000 : 0,035 Sek.
bei **warenkorb_id** = 4000 : 0,001 Sek.

Testergebnisse (Versuch 6)

bei **warenkorb_id** = 0001 : 0,00029 Sek.
bei **warenkorb_id** = 2000 : 0,00029 Sek.
bei **warenkorb_id** = 4000 : 0,00029 Sek.

Einstieg in das SQL- und Datenbanktuning

Versuch 6

MySQL 5.0.27-community-nt

Vorwärts blättern

```
SELECT MIN(verkauf_id)
  FROM verkauf
 WHERE warenkorb_id = :nWK_ID
   AND verkauf_id > :nVk_ID
```

plus zusätzlicher Index:

```
CREATE INDEX sx_verkauf_02
    ON verkauf
    (warenkorb_id,verkauf_id)
```

Testergebnisse (Versuch 1)

bei **warenkorb_id** = 0001 : 0,066 Sek.
bei **warenkorb_id** = 2000 : 0,035 Sek.
bei **warenkorb_id** = 4000 : 0,001 Sek.

Testergebnisse (Versuch 6)

bei **warenkorb_id** = 0001 : 0,00029 Sek.
bei **warenkorb_id** = 2000 : 0,00029 Sek.
bei **warenkorb_id** = 4000 : 0,00029 Sek.

Einstieg in das SQL- und Datenbanktuning

Praxis II

Auswertung welche **zehn Artikel** im Postleitzahlengebiet zwischen **70000 und 70500** bisher den **größten Umsatz** gemacht haben.

Also eine typische Top-10-Frage des Vertriebs!

```
CREATE TABLE verkauf (

 verkauf_id INT NOT NULL,
 warenkorb_id INT NOT NULL,
 kunde_id INT NOT NULL,
 artikel_id INT NOT NULL,
 datum DATE NOT NULL,
 preis DEC(10,2) NOT NULL,
 menge INT NOT NULL,
 PRIMARY KEY (verkauf_id)
);
```

```
203350 Datensätze
Verkauf-Id:    1 - 203350
Warenkorb-Id: 1 - 4067
```

```
CREATE TABLE kunde (

 kunde_id INT NOT NULL,
 name VARCHAR(50) NOT NULL,
 vorname VARCHAR(30) NOT NULL,
 gebdatum DATE,
 plz INT,
 PRIMARY KEY (kunde_id)
);
```

```
9999 Datensätze
Kunde-Id:       1 - 9999
Plz:        01003 - 99990
```

```
CREATE TABLE artikel (

 artikel_id INT NOT NULL,
 artikel_nr VARCHAR(20) NOT NULL,
 bezeichnung VARCHAR(100) NOT NULL,
 preis DEC(10,2),
 PRIMARY KEY (artikel_id)
);
```

```
10000 Datensätze
Artikel-Id: 10000 - 19999
```

Einstieg in das SQL- und Datenbanktuning

vorhandene Indices

MySQL 5.0.27-community-nt

```
mysql> show index from verkauf;
+---------+------------+---------------+--------------+-------------+-----------+-------------+-...+
| Table   | Non_unique | Key_name      | Seq_in_index | Column_name | Collation | Cardinality | ...+
+---------+------------+---------------+--------------+-------------+-----------+-------------+-...+
| verkauf |          0 | PRIMARY       |            1 | verkauf_id  | A         |      203729 | ...+
| verkauf |          1 | sx_verkauf_02 |            1 | warenkorb_id| A         |        8149 | ...+
| verkauf |          1 | sx_verkauf_02 |            2 | verkauf_id  | A         |      203729 | ...+
+---------+------------+---------------+--------------+-------------+-----------+-------------+-...+
3 rows in set (0.00 sec)


mysql> show index from kunde;
+-------+------------+----------+--------------+-------------+-----------+-------------+-...+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | ...+
+-------+------------+----------+--------------+-------------+-----------+-------------+-...+
| kunde |          0 | PRIMARY  |            1 | kunde_id    | A         |        9617 | ...+
+-------+------------+----------+--------------+-------------+-----------+-------------+-...+
1 row in set (0.06 sec)


mysql> show index from artikel;
+---------+------------+----------+--------------+-------------+-----------+-------------+-...+
| Table   | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | ...+
+---------+------------+----------+--------------+-------------+-----------+-------------+-...+
| artikel |          0 | PRIMARY  |            1 | artikel_id  | A         |       10144 | ...+
+---------+------------+----------+--------------+-------------+-----------+-------------+-...+
1 row in set (0.00 sec)

mysql>
```

Einstieg in das SQL- und Datenbanktuning

Versuch 1 – Theta Join Style

MySQL 5.0.27-community-nt

```
SELECT a.artikel_nr, SUM(v.menge*v.preis) AS umsatz
  FROM verkauf v,
       artikel a,
       kunde k
 WHERE k.plz BETWEEN 70000 AND 70500
   AND k.kunde_id = v.kunde_id
   AND a.artikel_id = v.artikel_id
 GROUP BY a.artikel_nr
 ORDER BY umsatz DESC
 LIMIT 10;
```

Testergebnisse: 0,094 Sek.

Einstieg in das SQL- und Datenbanktuning

Versuch 2 – ANSI Join Style

MySQL 5.0.27-community-nt

```
SELECT a.artikel_nr, SUM(v.menge*v.preis) AS umsatz
  FROM verkauf v
  JOIN kunde k
    ON k.kunde_id = v.kunde_id
  JOIN artikel a
    ON a.artikel_id = v.artikel_id
 WHERE k.plz BETWEEN 70000 AND 70500
 GROUP BY a.artikel_nr
 ORDER BY umsatz DESC
 LIMIT 10;
```

Testergebnisse: 0,094 Sek.

Einstieg in das SQL- und Datenbanktuning

Versuch 3 – ANSI Join + *correlated* Subselect

MySQL 5.0.27-community-nt

```
SELECT a.artikel_nr, SUM(v.menge*v.preis) AS umsatz
  FROM verkauf v
  JOIN artikel a
    ON a.artikel_id = v.artikel_id
 WHERE EXISTS ( SELECT k.kunde_id FROM kunde k
                  WHERE k.plz BETWEEN 70000 AND 70500
                    AND k.kunde_id = v.kunde_id)
 GROUP BY a.artikel_nr
 ORDER BY umsatz DESC
 LIMIT 10;
```

Testergebnisse: 0,159 Sek.

Einstieg in das SQL- und Datenbanktuning

Versuch 4 – Theta Join + Subselect

MySQL 5.0.27-community-nt

```
SELECT a.artikel_nr, SUM(v.menge*v.preis) AS umsatz
  FROM verkauf v,
       artikel a
 WHERE v.kunde_id IN ( SELECT k.kunde_id FROM kunde k
                        WHERE k.plz BETWEEN 70000 AND 70500 )
   AND a.artikel_id = v.artikel_id
 GROUP BY a.artikel_nr
 ORDER BY umsatz DESC
 LIMIT 10;
```

Testergebnisse: 0,202 Sek.

Einstieg in das SQL- und Datenbanktuning

Versuch 5 – ANSI Join + Subselect

MySQL 5.0.27-community-nt

```
SELECT a.artikel_nr, SUM(v.menge*v.preis) AS umsatz
  FROM verkauf v
  JOIN artikel a
  ON a.artikel_id = v.artikel_id
  JOIN ( SELECT k2.kunde_id FROM kunde k2
           WHERE k2.plz BETWEEN 70000 AND 70500 ) k
    ON k.kunde_id = v.kunde_id
 GROUP BY a.artikel_nr
 ORDER BY umsatz DESC
 LIMIT 10;
```

Testergebnisse: 0,542 Sek.

Einstieg in das SQL- und Datenbanktuning

Versuch 5 – ANSI Join + Subselect

MySQL 5.0.27-community-nt

```
SELECT a.artikel_nr, SUM(v.menge*v.preis) AS umsatz
  FROM verkauf v
  JOIN artikel a
  ON a.artikel_id = v.artikel_id
  JOIN ( SELECT k2.kunde_id FROM kunde k2
          WHERE k2.plz BETWEEN 70000 AND 70500 ) k
    ON k.kunde_id = v.kunde_id
 GROUP BY a.artikel_nr
 ORDER BY umsatz DESC
 LIMIT 10;
```

Aber wer hat hier mal wieder nicht zuerst mit **EXPLAIN** geprüft?!?

Einstieg in das SQL- und Datenbanktuning

EXPLAIN Versuch 1

MySQL 5.0.27-community-nt

```
mysql> EXPLAIN
    -> SELECT a.artikel_nr, SUM(v.menge*v.preis) AS umsatz
    ->   FROM verkauf v,
    ->        artikel a,
    ->        kunde k
    ->  WHERE k.plz BETWEEN 70000 AND 70500
    ->    AND k.kunde_id = v.kunde_id
    ->    AND a.artikel_id = v.artikel_id
    ->  GROUP BY a.artikel_nr
    ->  ORDER BY umsatz DESC
    ->  LIMIT 10;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	v	ALL	NULL	NULL	NULL	NULL	203729	Using temporary; Using filesort
1	SIMPLE	k	eq_ref	PRIMARY	PRIMARY	4	sample.v.kunde_id	1	Using where
1	SIMPLE	a	eq_ref	PRIMARY	PRIMARY	4	sample.v.artikel_id	1	

3 rows in set (0.02 sec)

mysql>

Testergebnisse: 0,094 Sek.

Einstieg in das SQL- und Datenbanktuning

EXPLAIN Versuch 5

MySQL 5.0.27-community-nt

```
mysql> EXPLAIN
    -> SELECT a.artikel_nr, SUM(v.menge*v.preis) AS umsatz
    ->   FROM verkauf v
    ->   JOIN artikel a
    ->     ON a.artikel_id = v.artikel_id
    ->   JOIN ( SELECT k2.kunde_id FROM kunde k2
    ->            WHERE k2.plz BETWEEN 70000 AND 70500 ) k
    ->     ON k.kunde_id = v.kunde_id
    ->   GROUP BY a.artikel_nr
    ->   ORDER BY umsatz DESC
    ->   LIMIT 10;
```

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	PRIMARY	<derived2>	ALL	NULL	NULL	NULL	NULL	54	Using temporary; Using filesort
1	PRIMARY	v	ALL	NULL	NULL	NULL	NULL	203729	Using where
1	PRIMARY	a	eq_ref	PRIMARY	PRIMARY	4	sample.v.artikel_id	1	
2	DERIVED	k2	ALL	NULL	NULL	NULL	NULL	10225	Using where

```
4 rows in set (0.00 sec)

mysql>
```

Testergebnisse: 0,542 Sek.

Einstieg in das SQL- und Datenbanktuning

Lösung zu 2

Generell sollte auf jeden Fremdschlüssel einer Tabelle ein Index angelegt werden!

```
CREATE TABLE verkauf (

 verkauf_id INT NOT NULL,
 warenkorb_id INT NOT NULL,
 kunde_id INT NOT NULL,
 artikel_id INT NOT NULL,
 datum DATE NOT NULL,
 preis DEC(10,2) NOT NULL,
 menge INT NOT NULL,
 PRIMARY KEY (verkauf_id),

 CONSTRAINT fk_kunde
  FOREIGN KEY (kunde_id)
   REFERENCES kunde (kunde_id)
    ON DELETE RESTRICT,

 CONSTRAINT fk_artikel
  FOREIGN KEY (artikel_id)
   REFERENCES artikel (artikel_id)
    ON DELETE RESTRICT
);
```

```
CREATE TABLE kunde (

 kunde_id INT NOT NULL,
 name VARCHAR(50) NOT NULL,
 vorname VARCHAR(30) NOT NULL,
 gebdatu DATE,
 plz INT
 PRIMARY
);
```

```
CREATE TABLE artikel (

 artikel_id INT NOT NULL,
 artikel_nr VARCHAR(20) NOT NULL,
 bezeichnung VARCHAR(100) NOT NULL,
 preis DEC(10,2),
 PRIMARY KEY (artikel_id)
```

Bei MySQL ab Version 5.0 werden auf einen Fremdschlüssel automatisch Indices erzeugt.

Einstieg in das SQL- und Datenbanktuning

vorhandene Indices II

MySQL 5.0.27-community-nt

```
mysql> show index from verkauf;
+---------+------------+---------------+--------------+-------------+-----------+-------------+...+
| Table   | Non_unique | Key_name      | Seq_in_index | Column_name | Collation | Cardinality | ...+
+---------+------------+---------------+--------------+-------------+-----------+-------------+...+
| verkauf |          0 | PRIMARY       |            1 | verkauf_id  | A         |      203729 | ...+
| verkauf |          1 | sx_verkauf_02 |            1 | warenkorb_id| A         |        8149 | ...+
| verkauf |          1 | sx_verkauf_02 |            2 | verkauf_id  | A         |      203729 | ...+
| verkauf |          1 | fk_kunde      |            1 | kunde_id    | A         |        6571 | ...+
| verkauf |          1 | fk_artikel    |            1 | artikel_id  | A         |       20372 | ...+
+---------+------------+---------------+--------------+-------------+-----------+-------------+...+
5 rows in set (0.00 sec)

mysql> show index from kunde;
+-------+------------+----------+--------------+-------------+-----------+-------------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality |
+-------+------------+----------+--------------+-------------+-----------+-------------+
| kunde |          0 | PRIMARY  |            1 | kunde_id    | A         |        9617 |
+-------+------------+----------+--------------+-------------+-----------+-------------+
1 row in set (0.06 sec)

mysql> show index from artikel;
+---------+------------+----------+--------------+-------------+-----------+-----
| Table   | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Card
+---------+------------+----------+--------------+-------------+-----------+-----
| artikel |          0 | PRIMARY  |            1 | artikel_id  | A         |
+---------+------------+----------+--------------+-------------+-----------+-----
1 row in set (0.00 sec)

mysql>
```

Tipp am Rande:
Gelegentlich an eine Optimierung der Tabellen und deren Statistiken denken.

- **OPTIMIZE TABLE tabname**
- **ANALYZE TABLE tabname**

Einstieg in das SQL- und Datenbanktuning

EXPLAIN Versuch 5

MySQL 5.0.27-community-nt

```
mysql> EXPLAIN
    -> SELECT a.artikel_nr, SUM(v.menge*v.preis) AS umsatz
    ->   FROM verkauf v
    ->   JOIN artikel a
    ->     ON a.artikel_id = v.artikel_id
    ->   JOIN ( SELECT k2.kunde_id FROM kunde k2
    ->           WHERE k2.plz BETWEEN 70000 AND 70500 ) k
    ->     ON k.kunde_id = v.kunde_id
    ->  GROUP BY a.artikel_nr
    ->  ORDER BY umsatz DESC
    ->  LIMIT 10;

+----+-------------+------------+--------+---------------------+----------+---------+---------------------+-------+---------------------------------+
| id | select_type | table      | type   | possible_keys       | key      | key_len | ref                 | rows  | Extra                           |
+----+-------------+------------+--------+---------------------+----------+---------+---------------------+-------+---------------------------------+
|  1 | PRIMARY     | <derived2> | ALL    | NULL                | NULL     | NULL    | NULL                |    54 | Using temporary; Using filesort |
|  1 | PRIMARY     | v          | ref    | fk_kunde,fk_artikel | fk_kunde | 4       | k.kunde_id          |    31 |                                 |
|  1 | PRIMARY     | a          | eq_ref | PRIMARY             | PRIMARY  | 4       | sample.v.artikel_id |     1 |                                 |
|  2 | DERIVED     | k2         | ALL    | NULL                | NULL     | NULL    | NULL                | 10225 | Using where                     |
+----+-------------+------------+--------+---------------------+----------+---------+---------------------+-------+---------------------------------+
4 rows in set (0.00 sec)
```

Alt

Testergebnisse: 0,542 Sek.

Tuning!!!

Testergebnisse: 0,008 Sek.

Versuch 5 ist mit diesen Indices jetzt schneller als Versuch 1– 4 ebenfalls mit diesen Indices.

Einstieg in das SQL- und Datenbanktuning

Versuch 1 – Theta Join Style

IBM DB2/NT 9.1 Express-C

```
SELECT a.artikel_nr, SUM(v.menge*v.preis) AS umsatz
  FROM verkauf v,
       artikel a,
       kunde k
 WHERE k.plz BETWEEN 70000 AND 70500
   AND k.kunde_id = v.kunde_id
   AND a.artikel_id = v.artikel_id
 GROUP BY a.artikel_nr
 ORDER BY umsatz DESC
 FETCH FIRST 10 ROWS ONLY;
```

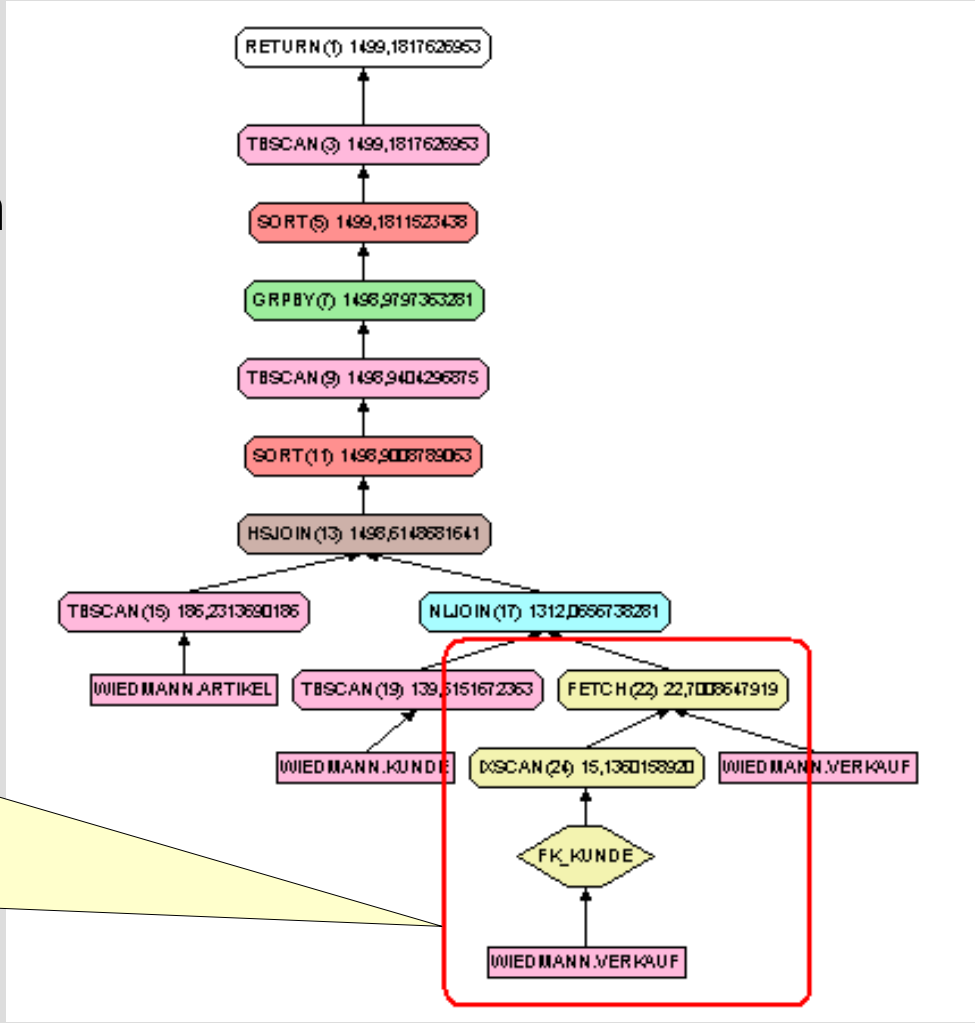
Testergebnisse: 0,071 Sek. Ebenso bei Versuch 2 - 5!

Einstieg in das SQL- und Datenbanktuning

Visual Explain nach Query ReWrite

IBM DB2/NT 9.1 Express-C

Neuer Versuch nachdem die beiden Indices auf die FK-Spalten kunde_id und artikel_id der Tabelle VERKAUF erzeugt sind.



Testergebnisse: 0,008 Sek.

Im Zugriffsplan wird nur ein Index genutzt und 2x TBSCAN.
=> Noch nicht optimal.

Einstieg in das SQL- und Datenbanktuning

Wo kann hier noch optimiert werden ?

IBM DB2/NT 9.1 Express-C

```
SELECT a.artikel_nr, SUM(v.menge*v.preis) AS umsatz
  FROM verkauf v,
       artikel a,
       kunde k
 WHERE k.plz BETWEEN 70000 AND 70500 /* (1)-Bereichsabfrage
   AND k.kunde_id = v.kunde_id
   AND a.artikel_id = v.artikel_id
 GROUP BY a.artikel_nr /* (2) Gruppierung = Sortierung
 ORDER BY umsatz DESC /* (3) Sortieren nach
 FETCH FIRST 10 ROWS ONLY;
```

Weitere Indices sind sinnvoll :

(1) bei „häufigen“ Bereichsabfragen und Feldkombinationen (plz + kunde_id)
(2) bei „häufigen“ Sort- und Group-by-Anforderungen (artikel_nr + artikel_id)
(3) berechneter Wert entsteht (materialisiert) erst bei der Ausführung (kein Index)

Einstieg in das SQL- und Datenbanktuning

Wo kann hier noch optimiert werden ?

IBM DB2/NT 9.1 Express-C

```
(1) CREATE INDEX sx_kunde_01 ON kunde (plz, kunde_id);

(2) CREATE INDEX sx_artikel_01 ON artikel (artikel_nr, artikel_id);
```

Diese neuen Indices werden von der Datenbank verwendet:

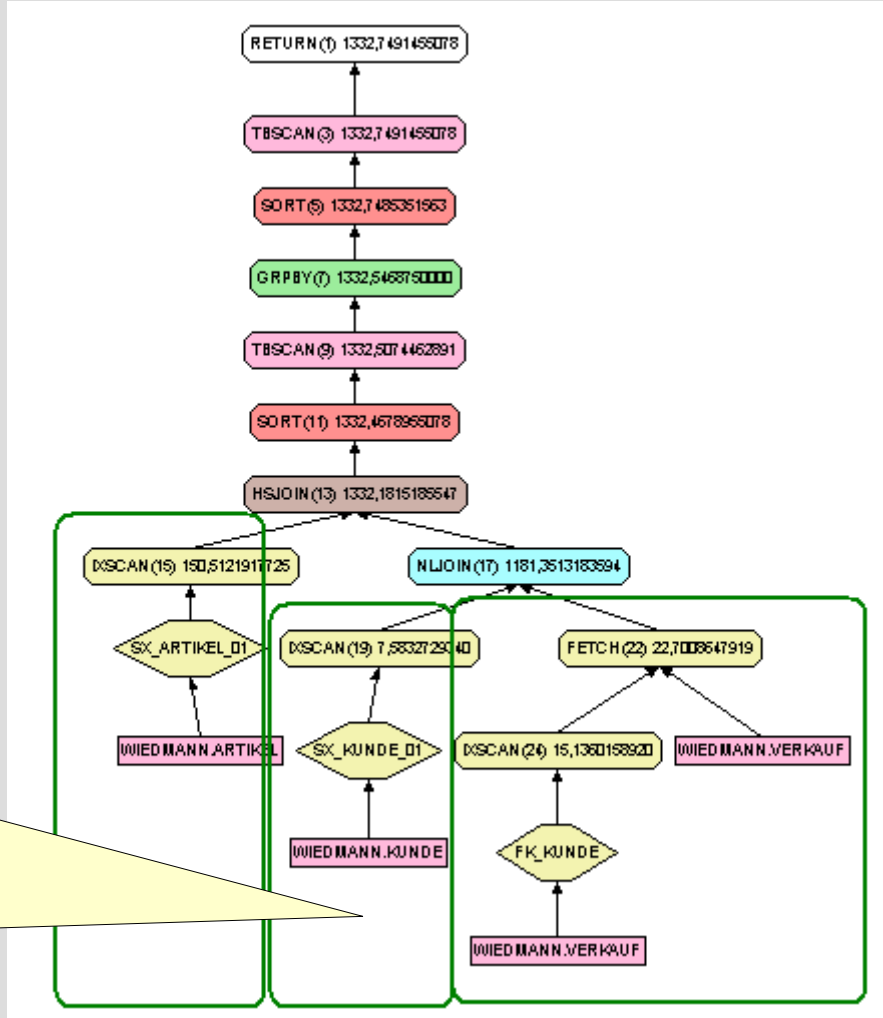
(1) bei „häufigen“ Bereichsabfragen und Feldkombinationen (plz + kunde_id)
(2) bei „häufigen“ Sort- und Group-by-Anforderungen (artikel_nr + artikel_id)

Einstieg in das SQL- und Datenbanktuning

Visual Explain nach Query ReWrite

IBM DB2/NT 9.1 Express-C

Neuer Versuch nachdem die beiden Indices auf die Tabellen artikel und kunde vorhanden sind.



Testergebnisse: 0,006 Sek.

Im Zugriffsplan werden jetzt pro Tabelle ein Index genutzt und keine „teuren“ TBSCAN mehr.

Einstieg in das SQL- und Datenbanktuning

Zusammenfassung und Grundregeln

- Jede SQL-Abfrage mit EXPLAIN überprüfen
  für MySQL http://dev.mysql.com/doc/refman/5.1/de/explain.html
- Für jede Tabelle einen PRIMARY KEY anlegen.
- Für jeden Fremdschlüssel einen INDEX definieren.
- Für häufige Abfragen oder Sortierungen spezielle
  (auch mehrteilige!) Indices anlegen (Ziel: Index-Only-Zugriff)
- und ....

Performance ist dynamisch,
tägliches Tuning ist deshalb wichtig und notwendig.

Einstieg in das SQL- und Datenbanktuning

The End!