Lehrstuhl Praktische Informatik III
Universität Mannheim
Prof. Dr. Guido Moerkotte
Sven Helmer

März 2004

Hauptdiplomklausur Datenbankpraktikum Wintersemester 2003/2004

Name: .....
Vorname: .....
Matrikelnummer: .....
Studienfach: .....

Wichtige Hinweise:

1. Prüfen Sie Ihr Klausurexemplar auf Vollständigkeit (8 Seiten).
2. Es sind keine Hilfsmittel zugelassen.
3. Die Klausur dauert 66 Minuten.
4. Jede Aufgabe ist auf dem zugehörigen Aufgabenblatt (und ggf. auf separaten Lösungsblättern) zu bearbeiten.
5. Vermerken Sie Ihren Namen und Ihre Matrikelnummer auf jedem Aufgaben- (bzw. Lösungs-) blatt. Blätter ohne Angabe des Namens und der Matrikelnummer werden nicht bewertet.
6. Das Deckblatt sowie alle Aufgabenblätter (evtl. Lösungsblätter) sind abzugeben.

	maximale Anzahl Punkte	erreichte Anzahl Punkte
Aufgabe 1	6	
Aufgabe 2	6	
Aufgabe 3	14	
Aufgabe 4	10	
Aufgabe 5	6	
Aufgabe 6	12	
Aufgabe 7	12	
	66	

1. (6 Punkte)
   Gegeben folgende Meßwerttabelle:

   Messwerte

MID	Messwert	Genauigkeit
1	3.04587	1
2	15.679056	-1
3	4.87683	3
4	5.8237	2

   Welche Ausgabe produziert folgende SQL-Anfrage?

```
SELECT MID,
  CASE Genauigkeit
    WHEN 1 THEN CAST(CAST(Messwert AS DECIMAL(10,1)) AS CHAR(12))
    WHEN 2 THEN CAST(CAST(Messwert AS DECIMAL(10,2)) AS CHAR(12))
    WHEN 3 THEN CAST(CAST(Messwert AS DECIMAL(10,3)) AS CHAR(12))
    ELSE 'Messfehler!'
  END AS Wert
FROM   Messwerte
ORDER BY MID;
```

2. (6 Punkte)
   Welche Bedeutung hat folgende Warnung beim Ausführen einer SQL-Anfrage: “A recursive common table expression may contain an infinite loop” (SQLCODE +347, SQLSTATE 01605)? Was kann man gegen diese Warnung tun?

3. Gegeben die folgenden Relationen

Klausur

KID	Name	bestanden
500	DBS I	48
501	DBS II	NULL
502	DBPrakt	NULL

Teilnehmer

MatrNr	KID
333333	500
222222	501
111111	502

(a) (6 Punkte)
Was sind die Inhalte der Variablen v1 bis v6 nach Durchlaufen des folgenden Embedded-SQL Fragments?

```
EXEC SQL
DECLARE CURSOR C1 FOR
SELECT T.KID, K.Name
FROM   Klausur K, Teilnehmer T
WHERE  K.KID = T.KID
ORDER BY T.KID, T.MatrNr;

EXEC SQL OPEN C1;

EXEC SQL FETCH C1 INTO :v1, :v2;
EXEC SQL FETCH C1 INTO :v3, :v4;
EXEC SQL FETCH C1 INTO :v5, :v6;
```

v1=

v2=

v3=

v4=

v5=

v6=

(b) (4 Punkte)
Was ändert sich an der Relation Klausur nach Ausführen des folgenden Embedded-SQL Fragments?

```
EXEC SQL
UPDATE Klausur
SET bestanden = :b1
WHERE bestanden IS NULL;
```

Der Inhalt der Hostvariablen b1 sieht folgendermaßen aus:
b1 = 0

(c) (4 Punkte)
Ein Programmierer versucht die Anfrage aus Teil (b) so zu formulieren:

```
EXEC SQL
UPDATE Klausur
SET bestanden = :b1
WHERE bestanden = :b2 INDICATOR :b2ind;
```

Der Inhalt der Hostvariablen sieht folgendermaßen aus:
b1 = 0, b2 = 0, b2ind = -1
Zu seinem Erstaunen ändert sich an der Tabelle Klausur nichts. Wieso?

4. (a) (5 Punkte)
Betrachten Sie folgendes Embedded-SQL-Fragment aus einer SQC-Datei:

```
EXEC SQL
CREATE TABLE Personal (
PersNr INT   NOT NULL PRIMARY KEY,
Name   CHAR(80));

EXEC SQL
INSERT INTO Personal
VALUES(007, 'James Bond');
```

Wird sich diese SQC-Datei übersetzen lassen? Begründen Sie Ihre Antwort kurz.

(b) (5 Punkte)
Betrachten Sie nun folgendes Dynamic-SQL-Fragment (in Java) mit gleicher Funktionalität:

```
Statement stmt1 = con.createStatement();
stmt1.executeUpdate("CREATE TABLE Personal (
                     PersNr INT   NOT NULL PRIMARY KEY,
                     Name   CHAR(80))");
Statement stmt2 = con.createStatement();
stmt2.executeUpdate("INSERT INTO Personal
                     VALUES(007, 'James Bond')");
```

Wird sich diese Datei übersetzen lassen? Begründen Sie auch hier Ihre Antwort kurz.

5. (6 Punkte)
   Vergleichen Sie kurz CGI und Servlets.

6. Erläutern Sie den Unterschied der folgenden Parameter bei einem CREATE FUNCTION Aufruf:

   (a) (3 Punkte)
   EXTERNAL ACTION/NO EXTERNAL ACTION

   (b) (3 Punkte)
   SCRATCHPAD/NO SCRATCHPAD

(c) (3 Punkte)
FINAL CALL/NO FINAL CALL

(d) (3 Punkte)
FENCED/NOT FENCED

7. (12 Punkte)
Schreiben Sie eine externe Funktion *addiereWochen*, die auf ein Datum X Wochen daraufaddiert und dieses neue Datum zurückgibt. Für die Eingabewerte "12.03.2004" und "2" würde beispielsweise "26.03.2004" zurückgeliefert.

Die CREATE FUNCTION Anweisung und der Prozedurkopf sind bereits angegeben:

```
CREATE FUNCTION addiereWochen(Datum, Wochen)
   RETURNS date
   EXTERNAL NAME 'myfile!addiereWochen'
   NOT VARIANT
   NO EXTERNAL ACTION
   NULL CALL
   LANGUAGE C
   FENCED
   PARAMETER STYLE DB2SQL
   NO SQL;
```

`SQLUDF_INTEGER` entspricht in C dem Datentyp `long`. Sie dürfen außerdem folgende vordefinierten Prozeduren verwenden:

`void konvDatumJMT(SQLUDF_DATE* datum, long* jahr, long* monat, long* tag)` spaltet ein Datum in Jahr, Monat und Tag auf.

`void konvJMTDatum(long* jahr, long* monat, long* tag, SQLUDF_DATE* datum)` setzt ein Jahr, Monat und Tag zu einem Datum zusammen.

`long tageImMonat(long jahr, long monat)`
gibt die Anzahl der Tage eines Monats zurück (berücksichtigt Schaltjahre).

```
void addiereWochen(
   SQLUDF_DATE*    datumIn,
   SQLUDF_INTEGER* wochenIn,
   SQLUDF_DATE*    datumOut,
   short*          nullDatumIn,
   short*          nullWochenIn,
   short*          nullDatumOut,
   char*           sqlstate,
   char*           fnName,
   char*           specificName,
   char*           message)
```