Ludwig-Maximilians-Universität München
Institut für Informatik
PD Dr. Peer Kröger
Johannes Niedermayer

SS 2013
xx.xx.xxxx

Klausur zur Vorlesung
Einführung in die Informatik: Systeme und Anwendungen
Lösungen

Vorname:	
Name:	
Geb.-Datum:	
Matr.-Nr.:	

Die Klausur besteht aus 7 Aufgaben. Die Punkte sind bei jeder Aufgabe angegeben. Bitte überprüfen Sie, ob Sie ein vollständiges Exemplar erhalten haben.

Tragen Sie die Lösungen in den dafür vorgesehenen Raum im Anschluss an jede Aufgabe ein. Falls der Platz für Ihre Lösung nicht ausreicht, benutzen Sie bitte nur die ausgeteilten Zusatzblätter!
Tragen Sie bitte oben auf jeder ungeraden Seite Ihren Namen und Ihre Matrikelnummer ein.

Verwenden Sie keinen Rot-, Grün- oder Bleistift.

Aufgabe	mögliche Punkte	erreichte Punkte
1. Allgemeine Fragen	11	
2. Prozesse	10	
3. Speicherverwaltung	10	
4. Relationale Algebra	7	
5. SQL	10	
6. E/R-Modellierung	11	
7. Normalisierung	9	
Summe	68	
Note:		

Entwertung der Klausur
Meine Klausur soll **nicht** korrigiert und **nicht** gewertet werden.

München, den xx.xx.xxxx **Unterschrift:** _____

Aufgabe 1 Allgemeine Fragen (4+1+1+2+1+1+1 Punkte)

- Erklären Sie die Funktionsweise einer Festplatte. Erläutern Sie dazu, wie die Daten gespeichert werden. Wie ist die Platte (physisch) unterteilt? Wie werden Daten auf Platte geschrieben?

Lösungsvorschlag:
Speicherung auf magnetischen rotierenden Platten (1P)
Platte, Oberfläche, Spur, Sektor (2P)
Lesekopfpositionierung, Rotation der Platte, Übertragung der Information. (1P)

- Erklären Sie die Aufgabe eines Betriebssystems

Lösungsvorschlag:
BS bildet Schnittstelle für Anwendungsprogramme und spezielle Systemprogramme zur Hardware. BS bewahrt den Nutzer vor der Komplexität der HW.

- Nennen Sie zwei mögliche Scheduling-Verfahren

Lösungsvorschlag:
Round-Robin, Prioritäts-Scheduling, Shortest-Job-First

- Was ist der Unterschied zwischen fester und dynamischer Partitionierung?

Lösungsvorschlag:
Feste Paritionierung teilt den Hauptspeicher in gleich große Partitionen, dynamische Partitionierung teilt den Hauptspeicher in Paritionen variabler Größe ein.

- Der Equi-Join $R \bowtie_{x=y} S$ in der relationalen Algebra ist eine abgeleitete Operation. Stellen Sie diesen durch Grundoperationen dar!

Lösungsvorschlag:

$R \bowtie_{x=y} = \sigma_{x=y}(R \times S)$

- Erklären Sie den Unterschied von überwachten (supervised) und unüberwachten (unsupervised) Verfahren im Data Mining.

Lösungsvorschlag:

Supervised: Trainingsphase erforderlich, der Lernerfolg kann überwacht werden Unsupervised: Die Methode lernt nicht, sondern findet Muster, die einem bestimmten Modell entsprechen.

- Was versteht man unter Outliern im Data Mining?

Lösungsvorschlag:

Ein Outlier ist eine Beobachtung, die sich von den anderen Beobachtungen so deutlich unterscheidet, daß man denken könnte, sie sei von einem anderen Mechanismus generiert worden.

Aufgabe 2 Prozesse (7+3 Punkte)

In einer Spielwarenfabrik werden Spielzeugmotorräder hergestellt. Ein Motorrad wird dabei aus einem Rahmen und zwei Reifen zusammengebaut. Dazu gibt es zwei Erzeugerprozesse E_1 und E_2 und einen Verbraucherprozess V. E_1 stellt Rahmen her und legt sie in eine Zwischenablage. E_2 stellt die Reifen her und legt sie ebenfalls in dieselbe Zwischenablage. Die Zwischenablage hat Platz für 1 Rahmen und 2 Reifen, d.h. es können nicht mehr als 1 Rahmen und nicht mehr als 2 Reifen gelagert werden. Ist die Zwischenablage mit einem Rahmen und zwei Reifen gefüllt, kann V den Rahmen und die zwei Reifen aus der Zwischenablage nehmen und zusammenbauen.

Die Prozesse sind durch folgende *fehlerhafte* Prozessbeschreibungen gegeben (die einzelnen Aktionen sind zusätzlich mit einer Nummer versehen):

```
// dient dem wechselseitigen Ausschluss der Zugriffe auf die Zwischenablage
var mutex: binary_semaphore;
init(mutex, 1);

// zeigt an, ob der Platz für Rahmen bzw. Reifen in der Zwischenablage frei ist
var rahmen_frei, reifen_frei: semaphore;
init(rahmen_frei, 1); init(reifen_frei, 2);

// zeigt an, wieviele Rahmen bzw. Reifen in der Zwischenablage abgelegt wurden
var rahmen_abgelegt, reifen_abgelegt: semaphore;
init(rahmen_abgelegt, 0); init(reifen_abgelegt, 0);
```

Prozess E_1

```
    REPEAT {
(1)   erzeuge Reifen;
(2)   wait(reifen_frei);
(3)   wait(mutex);
(4)   lege Reifen in die Zwischenablage;
(5)   signal(mutex);
(6)   signal(reifen_abgelegt);
    }
```

Prozess E_2

```
    REPEAT {
(1)   erzeuge Rahmen;
(2)   wait(rahmen_frei);
(3)   wait(mutex);
(4)   lege Rahmen in die Zwischenablage;
(5)   signal(rahmen_abgelegt);
(6)   signal(mutex);
    }
```

Prozess V

```
    REPEAT {
(1)   wait(reifen_abgelegt);
(2)   wait(rahmen_abgelegt);
(3)   wait(reifen_abgelegt);
(4)   wait(mutex);
(5)   nimm 1 Rahmen und 2 Reifen aus der Zwischenablage;
(6)   signal(rahmen_frei);
(7)   signal(reifen_frei);
(8)   signal(mutex);
(9)   baue Rahmen und Reifen zusammen;
    }
```

(a) Wie bereits erwähnt, ist diese Lösung fehlerhaft und daher können bei diesen Prozessen Deadlocks entstehen. Skizzieren Sie einen möglichen Ablauf der Prozesse, der zu einem Deadlock führt. Ihre Beschreibung darf dabei wahlweise mit einer vollständig gefüllten Zwischenablage oder mit einer leeren Zwischenablage beginnen. Tragen Sie Ihre Lösung in folgende Tabelle ein. In den Spalten, E_1, E_2 und V sollen dabei die Zeilennummern der Aktionen aus obigen Prozessbeschreibungen eingetragen werden, die die Semaphoren verändern. In den Spalten $mutex$, $reifen\_frei$, $reifen\_abgelegt$, $rahmen\_frei$ und $rahmen\_abgelegt$ sollen die jeweiligen Werte der Semaphoren stehen. In der Spalte $Ready$ ist die Menge der Prozesse einzutragen, die zum jeweiligen Zeitpunkt nicht blockiert sind.

(Hinweis: Sie benötigen nicht alle zur Verfügung stehenden Zeilen für eine richtige Lösung.)

Lösungsvorschlag:

E_1	E_2	V	mutex	reifen_frei	reifen_abgelegt	rahmen_frei	rahmen_abgelegt	Ready
			1	0	2	0	1	V, E_1, E_2
		(1)	1	0	1	0	1	V, E_1, E_2
		(2)	1	0	1	0	0	V, E_1, E_2
		(3)	1	0	0	0	0	V, E_1, E_2
		(4)	0	0	0	0	0	V, E_1, E_2
		(6)	0	0	0	1	0	V, E_1, E_2
		(7)	0	1	0	1	0	V, E_1, E_2
		(8)	1	1	0	1	0	V, E_1, E_2
(2)			1	0	0	1	0	V, E_1, E_2
(3)			0	0	0	1	0	V, E_1, E_2
(5)			1	0	0	1	0	V, E_1, E_2
(6)			1	0	1	1	0	V, E_1, E_2
	(2)		1	0	1	0	0	V, E_1, E_2
	(3)		0	0	1	0	0	V, E_1, E_2
	(5)		0	0	1	0	1	V, E_1, E_2
	(6)		1	0	1	0	1	V, E_1, E_2
(2)			1	0	1	0	1	V, E_2
	(2)		1	0	1	0	1	V
		(1)	1	0	0	0	1	V
		(2)	1	0	0	0	0	V
		(3)	1	0	0	0	0	

Bewertung: Je 1P pro 3 Zeilen der Tabelle.

(b) Schreiben Sie die Prozesse E_1, E_2 und V, soweit notwendig, so um, dass kein Deadlock mehr vorkommen kann und die Prozesse wie beschrieben funktionieren.

Nochmals zur besseren Übersicht die vorgegebenen Prozessbeschreibungen:

```
// dient dem wechselseitigen Ausschluss der Zugriffe auf die Zwischenablage
var mutex: binary_semaphore;
init(mutex, 1);

// zeigt an, ob der Platz für Rahmen bzw. Reifen in der Zwischenablage frei ist
var rahmen_frei, reifen_frei: semaphore;
init(rahmen_frei, 1); init(reifen_frei, 2);

// zeigt an, wieviele Rahmen bzw. Reifen in der Zwischenablage abgelegt wurden
var rahmen_abgelegt, reifen_abgelegt: semaphore;
init(rahmen_abgelegt, 0); init(reifen_abgelegt, 0);
```

Prozess E_1

```
    REPEAT {
(1)   erzeuge Reifen;
(2)   wait(reifen_frei);
(3)   wait(mutex);
(4)   lege Reifen in die Zwischenablage;
(5)   signal(mutex);
(6)   signal(reifen_abgelegt);
    }
```

Prozess E_2

```
    REPEAT {
(1)   erzeuge Rahmen;
(2)   wait(rahmen_frei);
(3)   wait(mutex);
(4)   lege Rahmen in die Zwischenablage;
(5)   signal(rahmen_abgelegt);
(6)   signal(mutex);
    }
```

Prozess V

```
    REPEAT {
(1)   wait(reifen_abgelegt);
(2)   wait(rahmen_abgelegt);
(3)   wait(reifen_abgelegt);
(4)   wait(mutex);
(5)   nimm 1 Rahmen und 2 Reifen aus der Zwischenablage;
(6)   signal(rahmen_frei);
(7)   signal(reifen_frei);
(8)   signal(mutex);
(9)   baue Rahmen und Reifen zusammen;
    }
```

Lösungsvorschlag:

```
Prozess V
      REPEAT {
(1)       wait(reifen_abgelegt);
(2)       wait(reifen_abgelegt);
(3)       wait(rahmen_abgelegt);
(4)       wait(mutex);
(5)       nimm 1 Rahmen und 2 Reifen aus der Zwischenablage;
(6)       signal(mutex);
(7)       signal(rahmen_frei);
(8)       signal(reifen_frei);
(9)       signal(reifen_frei);
(10)      baue Rahmen und Reifen zusammen;
      }
```

Aufgabe 3 Speicherverwaltung (3+3+3+1 Punkte)

Eine Frei-Liste beschreibt die ungenutzten Bereiche innerhalb eines Hauptspeichers mit dynamischer Partitionierung. Sie wird durch eine Folge von Zahlen dargestellt, welche die Größe der ungenutzten Bereiche in der Reihenfolge ihres Vorkommens im Speicher repräsentiert. Die genaue Position dieser Bereiche im Hauptspeicher wird dabei außer acht gelassen.

Betrachten Sie die Frei-Liste $[13, 16, 4, 14, 10, 5]$.

Nun werden Speicherbereiche folgender Größen in dieser Reichenfolge angefordert: $16, 9, 4, 6, 8, 4$.

Geben Sie die Frei-Listen an, die nach jeder einzelnen dieser Speicheranforderungen bei Verwendung der Speicherbelegungsstrategien First Fit, Next Fit bzw. Best Fit entstehen.

Zur Erinnerung:

- First Fit sucht stets vom Anfang der Frei-Liste aus den ersten Bereich, der groß genug ist, um die Anforderung zu erfüllen.
- Next Fit arbeitet so wie First Fit mit dem Unterschied, dass nicht jedes Mal am Anfang der Frei-Liste begonnen wird, nach dem ersten passenden Platz zu suchen, sondern an der Stelle, an der zuletzt ein freier Platz belegt wurde. Ist das Ende der Frei-Liste erreicht, und kein freier Platz wurde gefunden, so wird die Suche am Anfang fortgesetzt. Bei der ersten Anforderung wird am Anfang der Liste mit der Suche begonnen.
- Best Fit durchsucht stets die gesamte Frei-Liste nach dem kleinsten Bereich, der die Anforderung erfüllt.

(a) First Fit

Lösungsvorschlag:

Anforderung	Frei-Liste
	$[13, 16, 4, 14, 10, 5]$
16	$[13, 4, 14, 10, 5]$
9	$[4, 4, 14, 10, 5]$
4	$[4, 14, 10, 5]$
6	$[4, 8, 10, 5]$
8	$[4, 10, 5]$
4	$[10, 5]$

(b) Next Fit

Lösungsvorschlag:

Anforderung	Frei-Liste
	$[\underline{13}, 16, 4, 14, 10, 5]$
16	$[13, \underline{4}, 14, 10, 5]$
9	$[13, 4, \underline{5}, 10, 5]$
4	$[13, 4, \underline{1}, 10, 5]$
6	$[13, 4, 1, \underline{4}, 5]$
8	$[\underline{5}, 4, 1, 4, 5]$
4	$[\underline{1}, 4, 1, 4, 5]$

(c) Best Fit

Lösungsvorschlag:

Anforderung	Frei-Liste
	$[13, 16, 4, 14, 10, 5]$
16	$[13, 4, 14, 10, 5]$
9	$[13, 4, 14, 1, 5]$
4	$[13, 14, 1, 5]$
6	$[7, 14, 1, 5]$
8	$[7, 6, 1, 5]$
4	$[7, 6, 1, 1]$

(d) Welches Kriterium kann man zum Vergleich der Güte der drei Speicherbelegungsstrategien heranziehen? Welche Strategie schneidet demnach für die gegebenen Speicheranforderungen am besten ab?

Lösungsvorschlag:

Die Länge der resultierenden Frei-Liste.
First Fit schneidet am besten ab, weil die resultierende Frei-Liste am kürzesten ist.

Aufgabe 4 Relationale Algebra (1+2+2+2 Punkte)

R

A	B
1	3
2	1
3	4
4	6

S

A	B	C	D
1	3	2	3
1	5	4	1
2	1	1	2
3	4	2	2
3	4	5	2
2	3	1	2
4	3	2	2
2	5	6	2

T

C	D	E
2	2	3
2	7	1
4	1	2

Geben Sie die Ergebnisrelationen folgender Ausdrücke der relationalen Algebra als Tabellen an.

(a) $\pi_{A,D}(S)$

Lösungsvorschlag:

$\pi_{A,D}(S)$ mit Duplikaten

A	D
1	3
1	1
2	2
3	2
3	2
2	2
4	2
2	2

$\pi_{A,D}(S)$

A	D
1	3
1	1
2	2
3	2
4	2

***Bewertung**: 1 Punkt; 0.5 Punkte Abzug pro Fehler (z.B. falsches/fehlendes Tupel)*

(b) $\pi_A(R) \underset{A=C}{\bowtie} \pi_{C,D}(T)$

Lösungsvorschlag:

$\pi_A(R)$

A
1
2
3
4

$\pi_{C,D}(T)$

C	D
2	2
2	7
4	1

$\pi_A(R) \underset{A=C}{\bowtie} \pi_{C,D}(T)$

A	C	D
2	2	2
2	2	7
4	4	1

***Bewertung**: 2 Punkte*

R

A	B
1	3
2	1
3	4
4	6

S

A	B	C	D
1	3	2	3
1	5	4	1
2	1	1	2
3	4	2	2
3	4	5	2
2	3	1	2
4	3	2	2
2	5	6	2

T

C	D	E
2	2	3
2	7	1
4	1	2

(c) $\sigma_{B>3}(\pi_{A,B}(S) - R)$

Lösungsvorschlag:

$\pi_{A,B}(S)$

A	B
1	3
1	5
2	1
3	4
2	3
4	3
2	5

R

A	B
1	3
2	1
3	4
4	6

$\pi_{A,B}(S) - R$

A	B
1	5
2	3
4	3
2	5

$\sigma_{B>3}(\pi_{A,B}(S) - R)$

A	B
1	5
2	5

***Bewertung**: 2 Punkte*

(d) $(\pi_B(R) \cap \pi_B(S)) \times \sigma_{C<E}(T)$

Lösungsvorschlag:

$\pi_B(R)$

B
3
1
4
6

$\pi_B(S)$

B
3
5
1
4

$\pi_B(R) \cap \pi_B(S)$

B
3
1
4

$\sigma_{C<E}(T)$

C	D	E
2	2	3

$(\pi_B(R) \cap \pi_B(S)) \times \sigma_{C<E}(T)$

B	C	D	E
3	2	2	3
1	2	2	3
4	2	2	3

***Bewertung**: 2 Punkte*

Aufgabe 5 SQL (2+2+1+3+1+1 Punkte)

Gegeben sei das folgende Datenbank-Schema, das für die Speicherung der Daten zur Fußball-EM 2012 entworfen wurde, zusammen mit einem Teil seiner Ausprägung. Die Primärschlüssel-Attribute sind jeweils unterstrichen.

Mannschaft

Land	Trainer
Deutschland	Joachim Löw
Polen	Francisyek Smuda
Italien	Cesare Prandelli
...	...

Spieler

SpielerNr	Name	Mannschaft	Geburtsdatum	Position
21	Marco Reus	Deutschland	31.05.1989	Mittelfeld
21	David Silva	Spanien	08.01.1986	Mittelfeld
9	Mario Balotelli	Italien	12.08.1990	Sturm
...	...	...	...	...

Austragungsort

Stadion	Plätze
Miejski-Stadion	45500
Gdansk-Arena	44000
Lviv-Stadion	35000
...	...

Schiedsrichter

SID	Name	Nationalität
1	Jonas Eriksson	Schweden
2	Wolfgang Stark	Deutschland
3	Howard Webb	England
...	...	...

Spiel

SpielID	Datum	MannschaftA	MannschaftB	Austragungsort	Zuschauer
59	18.06.2012	Italien	Irland	Lviv-Stadion	32400
61	22.06.2012	Deutschland	Griechenland	Gdansk-Arena	41000
64	28.06.2012	Deutschland	Italien	Miejski-Stadion	NULL
...	...	...	...	...	...

Tor

TorNr	SpielerNr	Mannschaft	SpielID	Minute
144	21	Spanien	33	25
138	9	Italien	64	20
139	9	Italien	64	36
...	...	...	...	...

leitet

SpielID	SID
23	1
18	3
...	...

Formulieren Sie die folgenden Anfragen bezüglich dieses Datenbankschemas in SQL.

(a) Bestimmen Sie für jedes Spiel den Austragungsort, das Datum, die Anzahl der Zuschauer und den Namen des Schiedsrichters, aufsteigend sortiert nach dem Datum; bei gleichem Datum, nach dem Austragungsort.

Lösungsvorschlag:

```
SELECT        Austragungsort, Datum, Zuschauer, Name
FROM          Spiel  NATURAL JOIN leitet
              NATURAL JOIN Schiedsrichter
ORDER BY Datum, Austragungsort;
```

***Bewertung**: 0.5 Punkte SELECT, 0.5 Punkte FROM, 1 Punkt ORDER BY*

(b) Bestimmen Sie, wieviele Tore jeder Spieler während der EM geschossen hat.

Lösungsvorschlag:

nicht ganz konform zur Angabe:

```
SELECT        Name, count(*)
FROM          Spieler  NATURAL JOIN Tor
GROUP BY Name;
```

konforme Alternative, aber abweichend vom VL-Stoff:

```
SELECT        SpielerNr, Mannschaft, count(*)
FROM          Tor
GROUP BY SpielerNr, Mannschaft;
```

***Bewertung**: 0.5 Punkte SELECT, 0.5 Punkte FROM, 1 Punkt GROUP BY*

(c) Bestimmen Sie, wieviele Zuschauer durchschnittlich bei allen Spielen der italienischen Mannschaft anwesend waren.

Lösungsvorschlag:

```
SELECT avg(Zuschauer) FROM Spiel
WHERE MannschaftA = 'Italien'
        OR MannschaftB = 'Italien';
```

***Bewertung**: 0.5 Punkte SELECT mit FROM, 0.5 Punkte WHERE*

(d) Bestimmen Sie die Anzahl der Spiele, die Deutschland im Laufe der EM gespielt hat, in denen mindestens vier Tore fielen.

Lösungsvorschlag:

```
SELECT count (SpielID)
FROM Spiel
WHERE SpielID IN(
    SELECT SpielID
    FROM Spiel NATURAL JOIN TOR
    WHERE MannschaftA = 'Deutschland'
      OR MannschaftB = 'Deutschland'
    GROUP BY SpielID
    HAVING COUNT(TorNr) >= 4
);
```

***Bewertung**: 0.5 Punkte SELECT mit FROM, 0.5 Punkte WHERE, 1 Punkt GROUP BY, 1 Punkt HAVING*

(e) Ändern Sie die Zuschauerzahl des EM-Halbfinal-Spieles Deutschland gegen Italien (SpielID = 64) in der Relation Spiel **Spiel** auf die volle Anzahl der Plätze des Stadions, in dem das Spiel stattfindet.

Lösungsvorschlag:

```
UPDATE Spiel SET Zuschauer = 45500 WHERE SpielID = 64;
```

alternativ, teils berechnet:

```
UPDATE Spiel SET Zuschauer =
    (SELECT Plätze FROM Austragungsort
        WHERE Stadion = 'Miejski-Stadion';)
WHERE SpielID = 64;
```

alternativ, voll berechnet:

```
UPDATE Spiel SET Zuschauer =
    (SELECT Plätze FROM Austragungsort WHERE Stadion =
        (SELECT Austragungsort FROM Spiel
            WHERE SpielID = 64))
WHERE SpielID = 64;
```

***Bewertung**: 1 Punkt UPDATE*

(f) Ändern Sie die Vergangenheit: Fügen Sie ein Tor ein, das Marco Reus in der 87. Minute des Halbfinal-Spieles Deutschland gegen Italien (SpielID = 64) erzielt. Fügen Sie entsprechend ein neues Tupel in die Relation Tor ein. Nehmen Sie dazu an, dass in der Datenbank bereits 150 Tore enthalten sind. Die Tore sind fortlaufend durchnummeriert.

Lösungsvorschlag:

```
INSERT INTO Tor VALUES (151, 21, 'Deutschland', 64, 87);
```

***Bewertung**: 1 Punkt INSERT*

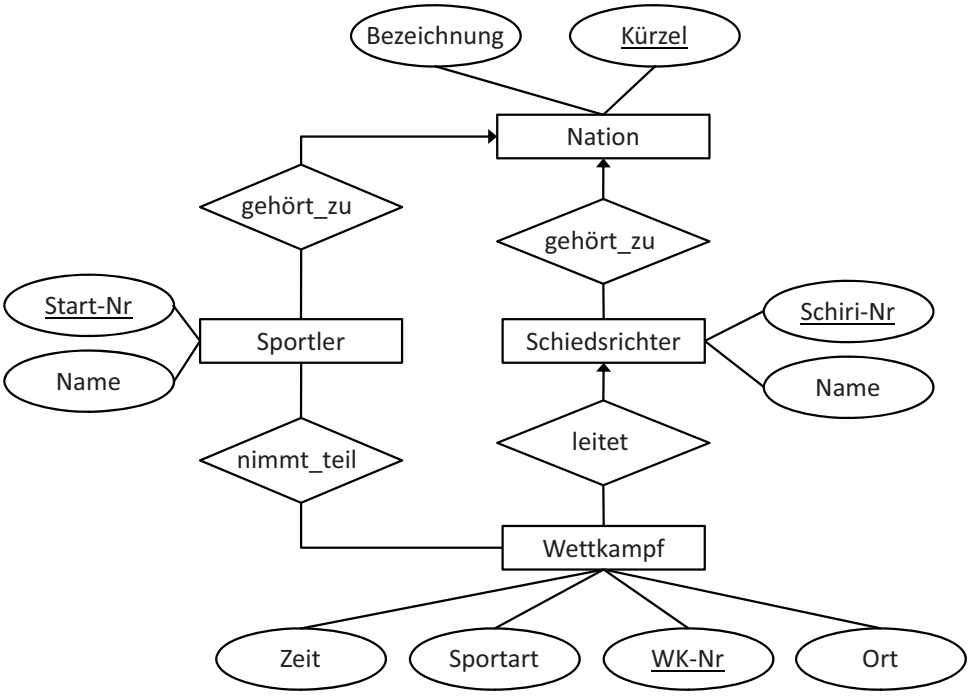
Aufgabe 6 E/R-Modellierung (6+5 Punkte)

In einer relationalen Datenbank sollen Informationen über die Olympischen Spiele gespeichert werden.

Die einzelnen Wettkämpfe besitzen eine eindeutige Wettkampfnummer, den Namen der Sportart, einen Termin und eine Wettkampfstätte. An jedem Wettkampf nehmen beliebig viele Sportler teil, die durch eine Startnummer identifiziert werden. Jeder Wettkampf wird von genau einem Schiedsrichter geleitet, der eine eindeutige Schiedsrichternummer besitzt. Es sollen die Nationen mit eindeutigem Kürzel und dem vollen Namen abgespeichert werden. Jede Person besitzt einen Namen und gehört zu einer Nation.

(a) Entwerfen Sie zu diesem Zweck ein E/R-Modell. Markieren Sie die Funktionalität jeder Beziehung. Schlüsselkandidaten müssen hier nicht markiert werden.

Lösungsvorschlag:



***Bewertung**: 2 Punkte Entities mit Attributen (0.5 Punkte Abzug je Fehler), 1 Punkt je Relationship (0.5 Punkte Abzug je Fehler, z.B. falsche Modellierung, falsche Funktionalität, ...)*

(b) Setzen Sie das vollständige E/R-Diagramm in ein entsprechendes relationales Datenbankschema um. Identifizieren Sie für jede Relation einen Primärschlüssel und unterstreichen Sie diesen. Achten Sie auf eine geeignete Modellierung der Relationships. Sie müssen keine SQL-DDL-Befehle angeben.

Lösungsvorschlag:

Relation	**Attribute (Schlüsselattribute unterstrichen)**
Sportler	Name, Start-Nr, Nationkürzel
Wettkampf	WK-Nr, Sportart, Ort, Termin, Schiri-Nr
Nation	Bezeichnung, Kürzel
Schiedsrichter	Name, Schiri-Nr, Nationkürzel
nimmt_teil	Start-Nr, WK-Nr

***Bewertung**: 1 Punkt je Relation*
alternativ:

Relation	**Attribute (Schlüsselattribute unterstrichen)**
Sportler	Name, Start-Nr
Wettkampf	WK-Nr, Sportart, Ort, Termin
Nation	Bezeichnung, Kürzel
Schiedsrichter	Name, Schiri-Nr
nimmt_teil	Start-Nr, WK-Nr
sp_gehört_zu	Start-Nr, Nationkürzel
sch_gehört_zu	Schiri-Nr, Nationkürzel
leitet	Schiri-Nr, WK-Nr (Schiri-Nr, WK-Nr)

***Bewertung**: 1 Punkt für die Relationen gehört_zu (wg. Schlüsselzuordnung), 0.5 Punkte für alle anderen*

Aufgabe 7 Normalisierung (3+3+3 Punkte)

1 Normalisierung

Es wurde eine Datenbank `klausurDB` erzeugt:
Kunde(kund_nr,kund_name,adresse,ort)
Personal(persnr,nachname,vorname,einsatz,plz_p,vorgesetzt,gehalt)
Verkauf(auftr_nr,art_nr,bestelldat,persnr,kund_nr)
Ausgang(auftr_nr,art_nr,menge)
Inventar(art_nr, plz_i,art_bez,lagerbest,lagerort)

Es gelten, über triviale funktionale Abhängigkeiten und die funktionalen Abhängigkeiten hinaus, die auf Schlüsseln basieren, die folgenden funktionalen Abhängigkeiten:

`plz_p` $\longrightarrow$ `einsatz` d.h., `plz_p` ist die Postleitzahl des `einsatz`-Ortes
`plz_i` $\longrightarrow$ `lagerort` d.h., `plz_i` ist die Postleitzahl von `lagerort`
`art_nr` $\longrightarrow$ `art_bez`

Es sei zudem angenommen, daß alle Relationen der Datenbank `klausurDB` bereits in erster Normalform sind.

1.1 Teilaufgabe

Bestimmen Sie für jede Relation der Datenbank `klausurDB`, ob sie in zweiter Normalform (2NF) ist und wenn ja, ob sie auch schon in dritter Normalform (3NF) ist.
Begründen Sie kurz Ihre Entscheidung.

Lösungsvorschlag:

Tabelle	2NF	3NF
kunde	JA	JA
personal	JA	NEIN
verkauf	JA	JA
ausgang	JA	JA
inventar	NEIN	NEIN

1.2 Teilaufgabe

Falls eine Relation nicht in zweiter oder in dritter Normalform ist, dann sind dafür Relationen in dritter Normalform anzugeben.
Geben Sie für die neu erstellten Tabellen jeweils einen Schlüssel an.

Lösungsvorschlag:

- – `personal*` : Schlüssel: `persnr`

```
create table personal*
   ( persnr int, nachname char(20), vorname char(12),
 plz_p int, vorgesetzt int, gehalt int );
```

– `plz_einsatz` : Schlüssel: `plz_p`

```
create table plz_einsatz
( einsatz char(12), plz_p int );
```

- – `inventar*` : Schlüssel: `art_nr, plz_i`

```
create table inventar*
   ( art_nr int, lagerbest int, plz_i int );
```

– `plz_lagerort` : Schlüssel: `plz_i`

```
create table plz_lagerort
   ( plz_i int,  lagerort char(12) ) ;
```

– `nr_bez` : Schlüssel: `art_nr`

```
create table nr_bez
   ( art_nr int, art_bez char(30) ) ;
```