Gruppe A

Bitte tragen Sie **sofort** und **leserlich** Namen, Studienkennzahl und Matrikelnummer ein und legen Sie Ihren Studentenausweis bereit.

| PRÜFUNG AUS DATENBANKSYSTEME **VU 184.686** | | | 06. 05. 2015 |
|---|---|---|---|
| Kennnr. | Matrikelnr. | Familienname | Vorname |
| | | | |

Arbeitszeit: 100 Minuten. Aufgaben sind auf den Angabeblättern zu lösen; Zusatzblätter werden nicht gewertet.

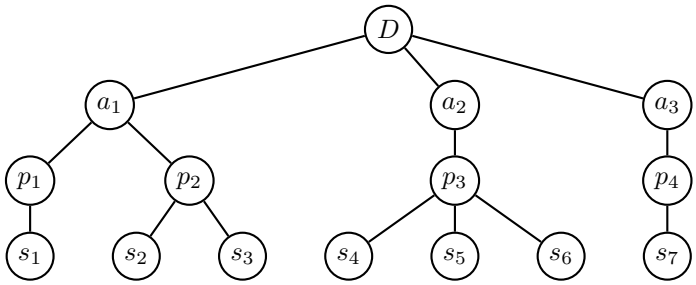
**Aufgabe 1:** (15)
Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind.

1. Die Historie $r_1(A)$, $w_2(A)$, $w_1(B)$, $r_2(B)$, $w_1(C)$, $c_1$, $c_2$ ist serialisierbar aber nicht strikt. wahr ⊗ falsch ◯
2. Die Historie $r_1(A)$, $r_2(B)$, $w_2(B)$, $w_1(B)$, $w_2(A)$, $c_1$, $c_2$ hat folgende Eigenschaften: Sie vermeidet kaskadierendes Rücksetzen, und sie ist nicht serialisierbar. wahr ⊗ falsch ◯
3. Bei den Einstellungen steal/force ist im Falle eines Recovery kein Redo nötig. wahr ⊗ falsch ◯
4. Betrachten Sie drei Relationen $R(AB)$, $S(ABC)$ und $T(BC)$. Dann gilt auf jeden Fall folgende Gleichheit: $(R \bowtie S) \cap (S \bowtie T) = (R \bowtie S) \bowtie T$. wahr ⊗ falsch ◯
5. Betrachten Sie zwei Relationen $U(AB)$ und $V(BC)$. Mit $\sigma_P$ bezeichnen wir die Selektion mit einem beliebigen Selektionsprädikat $P$. Dann gilt auf jeden Fall folgende Gleichheit: $\sigma_P(U \bowtie V) = \sigma_P(U) \bowtie \sigma_P(V)$. wahr ◯ falsch ⊗
6. Nehmen Sie an, dass beim Sortieren ein Datenbank-Puffer der Größe $m$ zur Verfügung steht. Dann werden $m + 1$ Level-0 Runs erzeugt. wahr ◯ falsch ⊗
7. Nehmen Sie an, dass eine relationale Datenbank um die beiden objektrelationalen Features "Objektidentität" und "Referenzen" erweitert wurde. Dann lassen sich $n{:}\,m$ Beziehungen – im Gegensatz zu rein relationalen Datenbanken – ohne Hilfstabelle realisieren. wahr ◯ falsch ⊗
8. Wenn in einem Auswertungsplan alle Block Nested Loop Joins durch Index Nested Loop Joins ersetzt werden, dann kann dies unter Umständen zu einer Verringerung der Anzahl der Tupel in den Zwischenergebnissen führen. wahr ◯ falsch ⊗
9. Wenn beim Zweiphasen-Commit-Protokoll zwei Agenten ausfallen, sind die restlichen Agenten blockiert, d.h.: sie können die Transaktion erst beenden, wenn zumindest einer der ausgefallenen Agenten wieder Nachrichten mit dem Koordinator austauschen kann. wahr ◯ falsch ⊗
10. Nehmen Sie an, dass in einem verteilten DBMS das globale Relationenschema $S(\underline{A}BCDE)$ (d.h.: $A$ ist der Primärschlüssel) in die vertikalen Fragmente $S(AB)$, $S(ACD)$, und $S(ABE)$ unterteilt wurde. Diese Fragmentierung ist vollständig und rekonstruierbar. wahr ⊗ falsch ◯

(Pro korrekter Antwort 1.5 Punkte, **pro inkorrekter Antwort -1.5 Punkte**, pro nicht beantworteter Frage 0 Punkte, für die gesamte Aufgabe mindestens 0 Punkte)

**Aufgabe 2:** (12)

Multi-Granularity Locking. Betrachten Sie folgende Datenbasis-Hierarchie.




Beantworten Sie, welche der folgenden geplanten Sequenzen von Sperr-Anforderungen (bei zwei Transaktionen $T_1$ und $T_2$) zu Blockierungen bzw. Deadlocks führen. Hier bedeutet $(T_i, x, L)$, dass Transaktion $T_i$ versucht, Knoten $x$ in der Hierarchie mit einer Sperre vom Typ $L$ zu belegen.

**Hinweis**: Unter Umständen werden nicht alle Sperren dieser Sequenzen auch tatsächlich angefordert, d.h.: Im Falle einer Blockierung einer Transaktion werden die weiter hinten liegenden Sperr-Anforderungen dieser Transaktion gar nicht mehr durchgeführt.

1. $(T_1, D, IS)$, $(T_2, D, IX)$, $(T_1, a_1, IS)$, $(T_1, p_1, IS)$, $(T_2, a_1, IX)$, $(T_2, p_2, X)$, $(T_1, s_1, S)$:
   Blockierung: ja ◯ nein ⊗ Deadlock: ja ◯ nein ⊗
2. $(T_1, D, IX)$, $(T_2, D, IX)$, $(T_1, a_3, IX)$, $(T_2, a_2, IX)$, $(T_1, p_4, IX)$, $(T_2, p_3, IX)$, $(T_1, s_7, X)$, $(T_2, s_5, X)$:
   Blockierung: ja ◯ nein ⊗ Deadlock: ja ◯ nein ⊗
3. $(T_1, D, IS)$, $(T_2, D, IX)$, $(T_1, a_1, S)$, $(T_2, a_2, IX)$, $(T_2, p_3, X)$, $(T_1, a_2, S)$, $(T_2, a_1, IX)$, $(T_2, p_1, IX)$, $(T_2, s_1, X)$:
   Blockierung: ja ⊗ nein ◯ Deadlock: ja ⊗ nein ◯
4. $(T_1, D, IX)$, $(T_2, D, IX)$, $(T_1, a_1, IX)$, $(T_2, a_2, X)$, $(T_1, p_2, X)$, $(T_2, a_1, IX)$, $(T_2, p_2, IX)$, $(T_2, s_2, X)$:
   Blockierung: ja ⊗ nein ◯ Deadlock: ja ◯ nein ⊗

(Pro korrekter Antwort 1.5 Punkte, **pro inkorrekter Antwort -1.5 Punkte**, pro nicht beantworteter Frage 0 Punkte, für die gesamte Aufgabe mindestens 0 Punkte)

**Aufgabe 3:** (18)

Eine Datenbank über Musikstücke und Konzerthäuser enthält folgende Relationen:

Werk(<u>WNr</u>, Bezeichnung, Kat) (kurz $w$),
Konzerthaus(<u>KName</u>, Ort, Max) (kurz $k$) und
gespielt(<u>WNr</u>, <u>KName</u>, <u>Datum</u>) (kurz $g$).

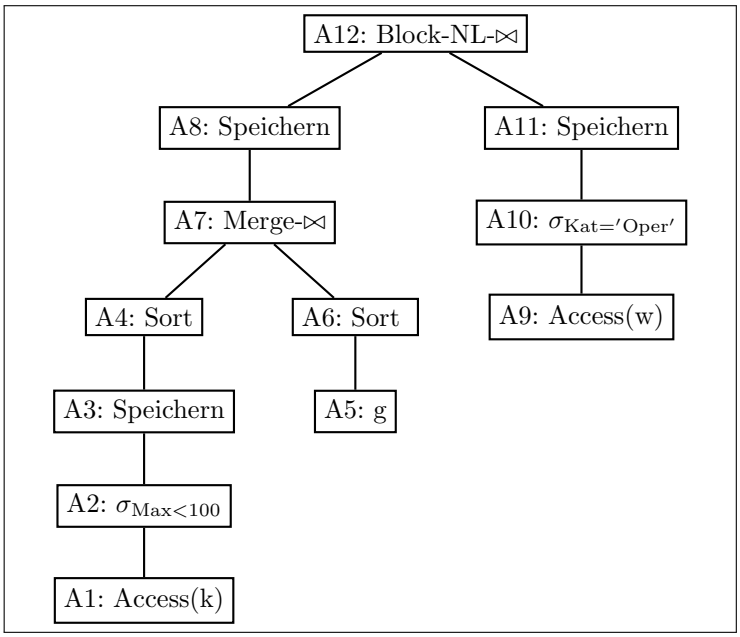
(Werk.Kat enthält die Kategorie eines Werks, und Konzerthaus.Max enthält die maximal zulässige Anzahl der Zuschauer eines Konzerthauses). Nehmen Sie an, dass $|w| = 50000$, $|k| = 20000$, und $|g| = 500000$. Für die durchschnittlichen Tupelgrößen von $w$, $k$ und $g$ sind die Werte 90, 100 und 60 Bytes anzunehmen. Nehmen Sie weiters an, dass pro Seite 2000 Bytes an Nutzinformation gespeichert werden können und dass die Hauptspeicher-Puffergröße 64 Seiten beträgt. Es ist die Anfrage

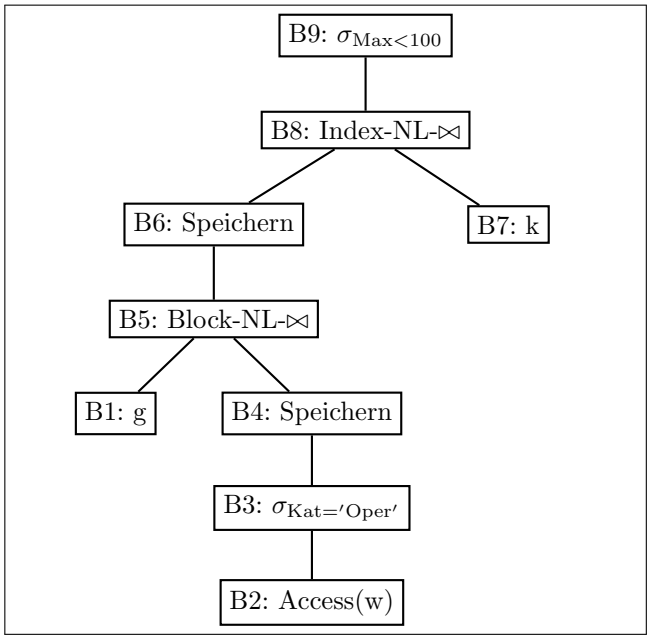
```
select *
from Werk w, Konzerthaus k, gespielt g
where w.WNr = g.WNr
and k.KName = g.KName
and w.Kat = 'Oper'
and k.Max < 100;
```

auszuführen (d.h. gesucht sind Informationen über Opern, die in relativ kleinen Konzerthäusern gespielt wurden).

Es sind folgende Selektivitäten anzunehmen: $Sel_{w/g} = 1/50000 = 0.00002$, $Sel_{k/g} = 1/20000 = 0.00005$, $Sel_{w.Kat='\text{Oper}'} = 0.2$ und $Sel_{k.Max<100} = 0.1$. Für die Primärschlüssel der Relationen w und k sei jeweils ein Hash-Index vorhanden. Nehmen Sie an, dass das Auslesen eines einzelnen Tupels mit einem Hash-Index durchschnittliche Kosten von 1.4 Page I/O erfordert.

Für diese Anfrage sind die Operator-Bäume für 2 Auswertungspläne gegeben: Plan A im linken Kästchen und Plan B im rechten Kästchen. Nehmen Sie bei den Block Nested Loop Joins an, dass $k = 1$ gilt und dass die äußere Relation jeweils links im Baum steht – unabhängig davon, ob dies optimal ist oder nicht. Bei der Berechnung der benötigten Seiten zum Speichern einer Relation dürfen Sie vereinfachend annehmen, dass die Tupel nicht unbedingt vollständig auf einer Seite Platz haben müssen. Außerdem dürfen Sie annehmen, dass die Tupelgröße beim Join von 2 Relationen gleich der Summe der einzelnen Tupelgrößen ist.



(a) Berechnen Sie für jeden Knoten im Operatorbaum des Auswertungsplans A eine Abschätzung für die Anzahl der Tupel im Resultat, die Tupelgröße, die Anzahl der Seiten im Resultat, und die Kosten (Page I/O). Für das Sortieren und für Joinoperationen ist auch noch die passende Kostenformel anzugeben. Tragen Sie Ihre Berechnungen in folgende Tabelle ein.

| Knoten# $i$ | Anzahl Tupel $T_i$ | Tupel-größe $g_i$ | Anzahl Seiten $b_i$ | Kostenformel | Kosten (Page I/O) |
|---|---|---|---|---|---|
| A1 | 20000 | 100 | 1000 ..... | - | 1000 ..... |
| A2 | 2000 ..... | 100 ..... | 100 ..... | - | 0 ..... |
| A3 | 2000 ..... | 100 ..... | 100 ..... | - | 100 ..... |
| A4 | 2000 ..... | 100 ..... | 100 ..... | $2 * b_3 * (1 + I)$ mit ..... $I = \log_{63}(\lceil b_3/64 \rceil) = 1$ ..... | 400 ..... |
| A5 | 500000 | 60 | 15000 ..... | - | 0 ..... |
| A6 | 500000 .... | 60 ..... | 15000 ..... | $2 * b_5 * (1 + I)$ mit ..... $I = \log_{63}(\lceil b_5/64 \rceil) = 2$ ..... | 90000 ..... |
| A7 | 50000 ..... | 160 ..... | 4000 ..... | $b_4 + b_6$ ..... | 15100 ..... |
| A8 | 50000 ..... | 160 ..... | 4000 ..... | - | 4000 ..... |
| A9 | 50000 | 90 | 2250 ..... | - | 2250 ..... |
| A10 | 10000 ..... | 90 ..... | 450 ..... | - | 0 ..... |
| A11 | 10000 ..... | 90 ..... | 450 ..... | - | 450 ..... |
| A12 | 10000 ..... | 250 ..... | 1250 ..... | $b_8 + 1 + \lceil b_8/62 \rceil * (b_{11} - 1)$ ..... | 33186 ..... |

Kosten insgesamt (Page I/O):

1000 + 100 + 400 + 90000 + 15100 + 4000 + 2250 + 450 + 33186 = 146486 .....

(b) Berechnen Sie für jeden Knoten im Operatorbaum des Auswertungsplans B eine Abschätzung für die Anzahl der Tupel im Resultat, die Tupelgröße, die Anzahl der Seiten im Resultat, und die Kosten (Page I/O). Für die Joinoperationen ist auch noch die passende Kostenformel anzugeben. Tragen Sie Ihre Berechnungen in folgende Tabelle ein.

| Knoten# $i$ | Anzahl Tupel $T_i$ | Tupel-größe $g_i$ | Anzahl Seiten $b_i$ | Kostenformel | Kosten (Page I/O) |
|---|---|---|---|---|---|
| B1 | 500000 | 60 | 15000 ..... | - | 0 ..... |
| B2 | 50000 | 90 | 2250 ..... | - | 2250 ..... |
| B3 | 10000 ..... | 90 ..... | 450 ..... | - | 0 ..... |
| B4 | 10000 ..... | 90 ..... | 450 ..... | - | 450 ..... |
| B5 | 100000 .... | 150 ..... | 7500 ..... | $b_1 + 1 + \lceil b_1/62 \rceil * (b_4 - 1)$ ..... | 123659 .... |
| B6 | 100000 .... | 150 ..... | 7500 ..... | | 7500 ..... |
| B7 | 20000 | 100 | 1000 ..... | - | 0 ..... |
| B8 | 100000 .... | 250 ..... | 12500 ..... | $b_6 + 1.4 * T_6$ ..... | 147500 .... |
| B9 | 10000 ..... | 250 ..... | 1250 ..... | - | 0 ..... |

Kosten insgesamt (Page I/O):

2250 + 450 + 123659 + 7500 + 147500 = 281359 .....

(c) Wie sehen im Auswertungsplan B die Berechnungen für den Knoten B5 aus, wenn der Block Nested Loop Join durch einen Hash Join ersetzt wird?

| Knoten# | Anzahl Tupel | Tupelgröße | Anzahl Seiten | Kostenformel | Kosten |
|---|---|---|---|---|---|
| B5 | 100000 .... | 150 ..... | 7500 ..... | $3 * (b_1 + b_4)$ ..... | 46350 ..... |

**Die folgende Datenbankbeschreibung gilt für die Aufgaben 4 – 7:**

Ein Online-Versandhändler speichert seine Produkte in folgendem stark vereinfachten Datenbankschema.

`Produkt`(pid, name, kid: *Kategorie.kid*, preis)
`Kategorie`(kid, name, uebergeordnet: *Kategorie.kid*, angebot: *Produkt.pid*)

**Auf der letzten Seite dieser Prüfung finden Sie eine Beispielinstanz dieses Schemas!**

In der Tabelle `Produkt` werden die verschiedenen Produkte des Händlers gespeichert. Diese werden eindeutig durch die Nummer `pid` identifiziert. Jedes Produkt hat einen Namen `name`, einen `preis` und wird einer Kategorie `kid` zugeordnet. Der Preis muss einen Wert größer als 0 haben.

In der Tabelle `Kategorie` werden die verschiedenen Produktkategorien gespeichert. Diese werden eindeutig durch die Nummer `kid` identifiziert. Jede Kategorie hat einen Namen `name`. Zu jeder Kategorie wird auch dessen übergeordnete Kategorie in `uebergeordnet` gespeichert. Falls eine solche nicht existiert, wird der Wert `NULL` gespeichert (für die oberste Kategorie der Hierarchie). Zu jedem Zeitpunkt ist in jeder Kategorie ein Produkt im Angebot. Die `pid` dieses Produkts wird im Feld `angebot` gespeichert.

Treffen Sie plausible Annahmen bezüglich der Datentypen der Attribute, sofern nicht angegeben.

**Aufgabe 4:** (7)

Geben Sie die CREATE TABLE Statements mit allen nötigen Constraints für die zwei Tabellen an.

```
CREATE TABLE kategorie (
        kid INTEGER PRIMARY KEY,
        name VARCHAR(50) NOT NULL,
        uebergeordnet INTEGER REFERENCES kategorie(kid),
        angebot INTEGER
);

CREATE TABLE produkt (
        pid INTEGER PRIMARY KEY,
        name VARCHAR(50) NOT NULL,
        kid INTEGER REFERENCES kategorie(kid),
        preis NUMERIC(7,2) CHECK (preis > 0)
);

ALTER TABLE kategorie ADD CONSTRAINT fk_angebot FOREIGN KEY (angebot)
      REFERENCES produkt(pid) DEFERRABLE INITIALLY DEFERRED;
```

**Aufgabe 5:** (8)

Evaluieren Sie die folgende Sequenz von SQL-Statements bezüglich der Datenbankinstanz **produkte** (siehe letzte Seite), und geben Sie die Ausgaben der Abfragen an:

```
CREATE VIEW subKat AS
WITH RECURSIVE temp(kid, sub) AS (
     SELECT k1.kid, k2.kid
     FROM kategorie k1 LEFT JOIN kategorie k2 ON k1.kid = k2.uebergeordnet
   UNION
     SELECT t.kid, k.kid
     FROM temp t, kategorie k
     WHERE t.sub = k.uebergeordnet
)
SELECT * FROM temp t ORDER BY kid, sub;

SELECT * FROM subKat;
```

| kid | sub |
|---|---|
| 100 | 110 |
| 100 | 111 |
| 100 | 120 |
| 110 | 111 |
| 111 | |
| 120 | |
| 200 | |

```
CREATE VIEW prodKat AS
   SELECT subKat.kid, pid FROM subKat JOIN produkt ON subKat.kid = produkt.kid
  UNION
   SELECT subKat.kid, pid FROM subKat JOIN produkt ON subKat.sub = produkt.kid;

SELECT * FROM prodKat ORDER BY kid, pid;
```

| kid | pid |
|---|---|
| 100 | 1 |
| 100 | 2 |
| 100 | 3 |
| 100 | 4 |
| 100 | 5 |
| 110 | 1 |
| 110 | 2 |
| 110 | 3 |
| 111 | 2 |
| 120 | 4 |
| 120 | 5 |
| 200 | 6 |
| 200 | 7 |

**Aufgabe 6:** (7)

Erstellen Sie einen PL/pgSQL Trigger `chgAngPreis`, der beim Verändern des Angebots einer Kategorie, den Preis des Produkts wie folgt mit 1.2 multipliziert oder dividiert. Der Trigger soll zumindest bei folgenden (exemplarischen) SQL Befehlen ausgeführt werden und das erklärte Verhalten zeigen.

- `INSERT INTO kategorie VALUES (112,'14-Zoll',110,1);`
  Der Preis von Produkt '1' durch 1.2 dividiert.
- `UPDATE kategorie SET angebot = 5 WHERE kid = 120;`
  Der Preis von Produkt '4' (aktueller Wert in der Instanz `produkte`) mit 1.2 multipliziert und der Preis von Produkt '5' durch 1.2 dividiert.

*Hinweis*: In der Variable `TG_OP` ist der durchgeführte Statement Typ gespeichert (`'INSERT'`, `'UPDATE'` oder `'DELETE'`). Verwenden Sie diese Variable um insgesamt nur eine Trigger Funktion zu benötigen.

```
CREATE OR REPLACE FUNCTION chgAngPreis() RETURNS TRIGGER AS $$
BEGIN
   IF TG_OP = 'UPDATE' THEN
      UPDATE produkt SET preis = preis * 1.2 WHERE pid = OLD.angebot;
   END IF;
   IF TG_OP = 'UPDATE' OR TG_OP = 'INSERT' THEN
      UPDATE produkt SET preis = preis / 1.2 WHERE pid = NEW.angebot;
   END IF;

   RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER t_chgAngPreis BEFORE UPDATE OR INSERT
       ON kategorie FOR EACH ROW EXECUTE PROCEDURE chgAngPreis();
```

**Aufgabe 7:** (8)

Vervollständigen Sie die folgende Java Funktion `updateProdukt`, sodass für jede übergebene `pid` im Array `pids` der Preis um 10% erhöht wird (neuer Preis = alter Preis * 1.1). Verwenden Sie dazu ein **PreparedStatement**, dass beim Durchlaufen des Arrays `pids` jedes Mal nur für die aktuelle `pid` den Preis erhöht.

Kümmern Sie sich auch um den Verbindungsaufbau zur Datenbank. Verwenden Sie dazu den Verbindungsstring `jdbc:postgresql://localhost/dbs` mit Benutzernamen `user` und Passwort `pass`.

Sorgen Sie dafür dass alle Ressourcen ordnungsgemäß geschlossen werden. Exceptions müssen Sie nicht behandeln.

```
public static void updateProdukt(int[] pids) throws Exception{
    //Treiber initialisieren
    Class.forName("org.postgresql.Driver");
    //Verbindung zur Datenbank herstellen
    Connection c = DriverManager.getConnection(
                   "jdbc:postgresql://localhost/dbs","user","pass");

    PreparedStatement psUpdateProd = c.prepareStatement(
                "UPDATE produkt SET preis = preis * 1.1 WHERE pid = ?");

    for (int i = 0; i<pids.length;i++) {
        psUpdateProd.setInt(1, pids[i]);
        psUpdateProd.executeUpdate();
    }

    psUpdateProd.close();
    c.close();
}
```

Gesamtpunkte: 75

**Sie können diese Seite abtrennen und brauchen sie nicht abzugeben!**

Datenbankinstanz **produkte**:

| Kategorie | | | |
|---|---|---|---|
| kid | name | uebergeordnet | angebot |
| 100 | Hardware | | 5 |
| 110 | Notebooks | 100 | 3 |
| 111 | 15-Zoll | 110 | 2 |
| 120 | Tablets | 100 | 4 |
| 200 | Software | | 6 |

| Produkt | | | |
|---|---|---|---|
| pid | name | kid | preis |
| 1 | Business NB 1 | 110 | 1500 |
| 2 | Business NB 2 | 111 | 1250 |
| 3 | Gamer NB 1 | 110 | 1500 |
| 4 | Ultra Smart Tablet | 120 | 400 |
| 5 | Apfel-Brett | 120 | 800 |
| 6 | Real OS | 200 | 100 |
| 7 | Yet Another Spreadsheet Clone | 200 | 350 |