DES3UE Datenbanksysteme WS 2018 Übung 4

Abgabetermin: 31.10.2018, 13:30 Uhr
Abgabeform elektronisch

☐ **DES31UE Niklas** **Name** Niklas Vest **Aufwand in h** 5
☐ **DES32UE Niklas**
☒ **DES33UE Traxler** **Punkte** _____ **Kurzzeichen Tutor** _____

Hinweise und Richtlinien:

- Übungsausarbeitungen müssen den im elearning angegebenen Formatierungsrichtlinien entsprechen – Nichtbeachtung dieser Formatierungsrichtlinien führt zu Punkteabzug.
  - Treffen Sie, falls notwendig, sinnvolle Annahmen und dokumentieren Sie diese nachvollziehbar in ihrer Lösung!

Ziel dieser Übung ist die Formulierung von hierarchischen Abfragen in SQL, die Manipulation von Zeilen und Spalten mit den Befehlen PIVOT und UNPIVOT und der Einsatz von analytischen Abfragen.

1. Hierarchische Abfragen – Human Resources (5 Punkte – 1+1+1+1+1)

1. Lex De Haan verlässt das Unternehmen. Sein Nachfolger wünscht Berichte über die Angestellten, die De Haan direkt unterstellt sind. Erstellen Sie eine SQL-Anweisung, um die Angestelltennummer, den Nachnamen, das Einstellungsdatum und das Gehalt anzuzeigen, wobei auf Folgendes eingeschränkt werden soll:
   a. Die Angestellten, die De Haan direkt unterstellt sind
   b. Die Organisationsstruktur unter De Haan (Angestelltennummer: 102)
2. Erstellen Sie eine hierarchische Abfrage, um die Angestelltennummer, die Managernummer und den Nachnamen für alle Angestellten unter De Haan anzuzeigen, die sich genau zwei Ebenen unterhalb dieses Angestellten (De Haan, Angestelltennummer: 102) befinden. Zeigen Sie zudem die Ebene des Angestellten an. *(2 Zeilen)*
3. Der CEO benötigt einen hierarchischen Bericht über alle Angestellten. Er teilt Ihnen die folgenden Anforderungen mit: Erstellen Sie einen hierarchischen Bericht, der die Angestelltennummer, die Managernummer, die Pseudospalte `LEVEL` und den Nachnamen des Angestellten anzeigt. Für jede Zeile der Tabelle `EMPLOYEES` soll eine Baumstruktur ausgegeben werden, die den Angestellten, seinen Manager, dessen Manager usw. zeigt. Verwenden Sie Einrückungen (LPAD) für die Spalte `LAST_NAME`.
4. Erstellen Sie einen Bericht, der alle Angestellten enthält (Vor- und Nachname) und bestimmt, von wie vielen Personen er/sie Vorgesetzte/r ist. Zählen Sie nicht nur die direkt unterstellten, sondern die gesamte Hierarchie darunter (zB Steven King ist der Vorgesetzte von 19 Personen). Sie benötigen für diese Aufgabe eine spezielle Pseudo-Spalte, recherchieren Sie diese!

2. Hierarchische Abfragen – Sakila-Datenbank (6 Punkte)

Anmerkungen: Beachten Sie, dass für diese beiden Abfragen nur Filme mit einer ID <= 13 berücksichtigt werden. Für die Hierarchie soll die Bekanntschaftskette nicht innerhalb eines

Filmes durlaufen werden (zusätzliche Bedingung bei CONNECT BY). Verwenden Sie EXECUTE SYS.KILL_MY_SESSIONS() um hängen gebliebene Session/Abfragen zu beenden.

1. Für Schauspielerinnen wird ein soziales Netzwerk aufgebaut. Initial wird davon ausgegangen, dass sich Schauspielerinnen, die gemeinsam in einem Film spielen bereits kennen. Das System soll nun eine Liste von neuen Kontaktvorschlägen generieren, die darauf beruht, jemanden „über (genau) einen gemeinsamen Bekannten" zu kennen.

   Erstellen Sie eine View partners, die Ihnen Schauspielerinnen und Schauspieler-Kollegen ausgibt (wenn sie in irgendeinem Film miteinander gespielt haben). Achten Sie darauf, dass keine Beziehung der SchauspielerInnen auf sich selbst entsteht. Die View soll beide Actor-IDs und die Film-ID enthalten. Damit das Ergebnis überschaubar bleibt, sollen Sie hierfür nur die ersten 13 Filme (film_id <= 13) der Datenbank heranziehen. *(2 Punkte, 438 Zeilen)*

   Erstellen Sie aufbauend auf der obigen View eine Vorschlagsliste für Schauspieler „NICK WAHLBERG". *(4 Punkte, 10 Zeilen)*

3. PIVOT und UNPIVOT (6 Punkte – 2+2+2)

1. Erstellen Sie eine Pivot-Tabelle die angibt, welcher Verkäufer welche Film-Kategorien am besten verleihen kann. Erstellen Sie eine Spalte pro Verkäufer (berücksichtigen Sie nur Verkäufer 1 und 2) und listen Sie die Film-Kategorien zeilenweise auf. Zählen Sie die Anzahl der jeweils verliehenen Filme, sortieren Sie nach Kategorie. *(16 Zeilen)*

Kategorie	Verk1_Anzahl	Verk2_Anzahl
Action	424	440
Animation	465	457
…	…	…

2. Erstellen Sie eine Pivot-Tabelle für die Kategorien „Family", „Children" und „Animation" (Spalten), welche die Durchschnittslänge der Filme pro Sprache (Zeilen) angibt. *(6 Zeilen, 4 Spalten)*
3. Erstellen Sie eine Anfrage, die für jeden Film die Sprache und die Original-Sprache enthält (verwenden Sie nur Filme aus dem Jahr 1983). Erstellen Sie eine Tabelle, die als Spalten den Film-Namen, die Sprache und die Art der Sprache (L, OL) enthält, sortieren Sie nach Film-Titel. *(42 Zeilen)*

Titel	Art	Sprache
BORN SPINAL	L	Japanese
BORN SPINAL	OL	German
…	…	…

4. Analytische Abfragen (7 Punkte – 2+2+3)

Anmerkung: Verwenden Sie für die Lösung der folgenden Aufgaben analytische Funktionen.

1. Geben Sie Titel und Verleihdatum der zuletzt verliehenen Filme aus (zumindest zehn). Enthält das Verleihdatum gleiche Werte (Ties), geben Sie alle Filme mit diesem Datum aus. Ermitteln Sie dazu den Rang der Filme innerhalb des Verleihdatums. (*31 Zeilen)*
2. Geben Sie zu jeder Filmkategorie drei Filme aus, wählen Sie jene Filme, die neueren Datums sind. Geben Sie den Kategorienamen aus, den Filmtitel und das Erscheinungsjahr. Verwenden Sie ROW_NUMBER().
3. Ermitteln Sie, wie viele Tage ein Kunde durchschnittlich zwischen zwei aufeinanderfolgenden Verleihvorgängen verstreichen lässt, dh. wann er den nächsten Film ausleiht. Die Auswertung ist nur möglich, wenn der Kunde bereits mehr als einen Film ausgeborgt hat (beachten Sie NULL-Werte). *(zB Dana Hart leiht durchschnittlich alle 24 Tage einen Film aus)*

Ausarbeitung Übung 04

Table of Contents

Hierarchische Abfragen (HR)

```
-- 1.1a
SELECT employee_id,
       last_name,
       hire_date,
       salary
FROM employees
    -- basically WHERE manager_id = 102;
START WITH manager_id = 102
CONNECT BY PRIOR employee_id = manager_id
       AND level = 1;
```

	EMPLOYEE_ID	LAST_NAME	HIRE_DATE	SALARY
1	103	Hunold	1990-01-03 00:00:00	9000.00

```
-- 1.1b
SELECT employee_id,
       last_name,
       hire_date,
       salary
FROM employees
START WITH employee_id = 102
CONNECT BY PRIOR employee_id = manager_id;
```

	EMPLOYEE_ID	LAST_NAME	HIRE_DATE	SALARY
1	102	De Haan	1993-01-13 00:00:00	17000.00
2	103	Hunold	1990-01-03 00:00:00	9000.00
3	104	Ernst	1991-05-21 00:00:00	6000.00
4	107	Lorentz	1999-02-07 00:00:00	4200.00

```
-- 1.2
SELECT employee_id,
       last_name,
       hire_date,
       salary
FROM employees
WHERE level = 3
START WITH employee_id = 102
CONNECT BY PRIOR employee_id = manager_id;
```

	EMPLOYEE_ID	LAST_NAME	HIRE_DATE	SALARY
1	104	Ernst	1991-05-21 00:00:00	6000.00
2	107	Lorentz	1999-02-07 00:00:00	4200.00

```
-- 1.3
SELECT employee_id,
       manager_id,
       LPAD(last_name, LENGTH(last_name) + level, '-') AS hierarchy
FROM employees
START WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id;
```

	EMPLOYEE_ID	MANAGER_ID	HIERARCHY
1	100	<null>	-King
2	101	100	--Kochhar
3	200	101	---Whalen
4	205	101	---Higgins
5	206	205	----Gietz
6	102	100	--De Haan
7	103	102	---Hunold
8	104	103	----Ernst
9	107	103	----Lorentz
10	124	100	--Mourgos
11	141	124	---Rajs
12	142	124	---Davies
13	143	124	---Matos
14	144	124	---Vargas
15	149	100	--Zlotkey
16	174	149	---Abel
17	176	149	---Taylor
18	178	149	---Grant
19	201	100	--Hartstein
20	202	201	---Fay

```
-- 1.4
WITH superiors AS (SELECT CONNECT_BY_ROOT first_name AS first_name,
                          CONNECT_BY_ROOT last_name AS last_name
                   FROM employees
                   WHERE level > 1
                   CONNECT BY PRIOR employee_id = manager_id)
SELECT first_name,
```

```
       last_name,
       COUNT(*)
FROM superiors
GROUP BY first_name, last_name;
```

	FIRST_NAME	LAST_NAME	COUNT(*)
1	Eleni	Zlotkey	3
2	Lex	De Haan	3
3	Neena	Kochhar	3
4	Steven	King	19
5	Kevin	Mourgos	4
6	Shelley	Higgins	1
7	Alexander	Hunold	2
8	Michael	Hartstein	1

Hierarchische Abfragen (Sakila)

```
-- 2
CREATE OR REPLACE VIEW partners AS
  SELECT a1.actor_id AS actor_id,
         a2.actor_id AS partner_id,
         a1.film_id AS film_id
  FROM film_actor a1
         INNER JOIN film_actor a2 ON a1.actor_id != a2.actor_id AND
                                     a1.film_id = a2.film_id
  WHERE a1.film_id <= 13;
-- Nick Wahlberg is has actor_id 2!
SELECT DISTINCT partner_id,
                last_name,
                first_name
FROM partners p
  INNER JOIN actor a ON p.partner_id = a.actor_id
WHERE partner_id NOT IN
      (SELECT partner_id
       FROM partners
       WHERE actor_id = 2)
START WITH p.actor_id = 2
CONNECT BY NOCYCLE PRIOR partner_id = p.actor_id
       AND partner_id != 2
       AND level = 2;
```

	PARTNER_ID	LAST_NAME	FIRST_NAME
1	85	ZELLWEGER	MINNIE
2	35	DEAN	JUDY
3	117	TRACY	RENEE
4	90	GUINESS	SEAN
5	29	WAYNE	ALEC
6	37	BOLGER	VAL
7	142	RYDER	JADA
8	188	DUKAKIS	ROCK
9	160	DEPP	CHRIS
10	157	MALDEN	GRETA

PIVOT und UNPIVOT

```
-- 3.1
SELECT *
FROM
     (SELECT staff_id,
             name AS category
      FROM rental
             INNER JOIN inventory USING (inventory_id)
             INNER JOIN film USING (film_id)
             INNER JOIN film_category USING (film_id)
             INNER JOIN category USING (category_id))
        PIVOT
        (COUNT(*) AS anzahl
        FOR staff_id
        IN (1 AS verk1, 2 AS verk2))
ORDER BY category ASC;
```

	CATEGORY	VERK1_ANZAHL	VERK2_ANZAHL
1	Action	424	440
2	Animation	465	457
3	Children	377	350
4	Classics	379	355
5	Comedy	363	360
6	Documentary	401	403
7	Drama	401	400
8	Family	443	387
9	Foreign	373	414
10	Games	366	375
11	Horror	334	323
12	Music	332	323
13	New	377	352
14	Sci-Fi	420	433
15	Sports	456	475
16	Travel	305	331

```
-- 3.2
SELECT name,
       ROUND(family, 2),
       ROUND(children, 2),
       ROUND(animation, 2)
FROM
     (SELECT lang.name,
             cat.name AS category,
             length
      FROM category cat
             INNER JOIN film_category USING (category_id)
             INNER JOIN film USING (film_id)
             INNER JOIN language lang USING (language_id))
        PIVOT
        (AVG(length)
        FOR category
        IN ('Family' AS family, 'Children' AS children, 'Animation' AS animation));
```

	NAME	ROUND(FAMILY,2)	ROUND(CHILDREN,2)	ROUND(ANIMATION,2)
1	Japanese	116.76	120	113.7
2	Mandarin	108.08	129.8	105.67
3	French	102.71	120.06	99.1
4	Italian	115.92	97.18	93.67
5	German	107	111	112.87
6	English	145.88	80.3	128.92

```
-- 3.3
SELECT *
FROM
     (SELECT title,
             l.name AS lang,
             ol.name AS original
      FROM film f
             INNER JOIN language l ON f.language_id = l.language_id
             INNER JOIN language ol ON f.original_language_id = ol.language_id
      WHERE release_year = 1983)
        UNPIVOT
        (language
        FOR kind
        IN (lang AS 'L', original AS 'OL'))
ORDER BY title ASC;
```

	TITLE	KIND	LANGUAGE
1	BORN SPINAL	L	Japanese
2	BORN SPINAL	OL	German
3	BOWFINGER GABLES	OL	German
4	BOWFINGER GABLES	L	Italian
5	BUNCH MINDS	L	English
6	BUNCH MINDS	OL	Mandarin
7	CHITTY LOCK	L	Mandarin
8	CHITTY LOCK	OL	German
9	CIDER DESIRE	OL	German
10	CIDER DESIRE	L	Italian
11	CLOSER BANG	L	Japanese
12	CLOSER BANG	OL	Mandarin
13	DIVIDE MONSTER	OL	German
14	DIVIDE MONSTER	L	Mandarin
15	DRUMS DYNAMITE	L	German

Analytische Abfragen

```
-- 4.1
SELECT title,
       rental_date,
       RANK() OVER (PARTITION BY title ORDER BY rental_date) AS rank
FROM rental
       INNER JOIN inventory USING (inventory_id)
       INNER JOIN film USING (film_id)
ORDER BY rental_date DESC
FETCH FIRST 10 ROWS WITH TIES;
```

	TITLE	RENTAL_DATE	RANK
1	ANONYMOUS HUMAN	2015-11-04 00:00:00	13
2	YENTL IDAHO	2015-11-04 00:00:00	23
3	VIRTUAL SPOILERS	2015-11-04 00:00:00	14
4	VELVET TERMINATOR	2015-11-04 00:00:00	26
5	SWEETHEARTS SUSPECTS	2015-11-04 00:00:00	29
6	SOMETHING DUCK	2015-11-04 00:00:00	19
7	GOODFELLAS SALUTE	2015-11-04 00:00:00	31
8	ENCOUNTERS CURTAIN	2015-11-04 00:00:00	19
9	ATLANTIS CAUSE	2015-11-04 00:00:00	24
10	ADAPTATION HOLES	2015-11-03 00:00:00	12
11	ZORRO ARK	2015-11-03 00:00:00	31
12	UPTOWN YOUNG	2015-11-03 00:00:00	12
13	TRAMP OTHERS	2015-11-03 00:00:00	20
14	TARZAN VIDEOTAPE	2015-11-03 00:00:00	8
15	SWEETHEARTS SUSPECTS	2015-11-03 00:00:00	28

```
-- 4.2
SELECT name,
       title,
       release_year
FROM
     (SELECT name,
             title,
             release_year,
             ROW_NUMBER() OVER (PARTITION BY category_id ORDER BY release_year) AS rn
      FROM film_category
             INNER JOIN film USING (film_id)
             INNER JOIN category USING (category_id))
WHERE rn < 4;
```

	NAME	TITLE	RELEASE_YEAR
1	Action	DRAGON SQUAD	1983
2	Action	WOMEN DORADO	1983
3	Action	BULL SHAWSHANK	1984
4	Animation	NASH CHOCOLAT	1983
5	Animation	WATCH TRACY	1983
6	Animation	THEORY MERMAID	1983
7	Children	TIES HUNGER	1983
8	Children	UPTOWN YOUNG	1983
9	Children	CIRCUS YOUTH	1984
10	Classics	LEAGUE HELLFIGHTERS	1984
11	Classics	WASTELAND DIVINE	1984
12	Classics	TOWERS HURRICANE	1984
13	Comedy	CLOSER BANG	1983
14	Comedy	PINOCCHIO SIMON	1983
15	Comedy	BRINGING HYSTERICAL	1984

```
-- 4.3
-- I tried to partition by customer id
-- and then let a window of size 2 compare
-- 2 adjacent dates recursively but calculating
-- the date difference turned out to be quite
-- tricky so I used regular joins. :)
WITH dates AS (SELECT customer_id,
                      rental_date,
                      ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY rental_date ASC) AS rn
               FROM rental)
SELECT d1.customer_id,
       c.last_name,
       ROUND(AVG(d2.rental_date - d1.rental_date)) AS average_rental_interval
FROM dates d1
       INNER JOIN dates d2 ON d1.customer_id = d2.customer_id AND
                              d1.rn + 1 = d2.rn -- CONNECT BY (...) AND PRIOR rn + 1 = rn
       INNER JOIN customer c ON (d1.customer_id = c.customer_id)
```

```
GROUP BY d1.customer_id, c.last_name;
```

	CUSTOMER_ID	LAST_NAME	AVERAGE_RENTAL_INTERVAL
1	1	SMITH	21
2	10	TAYLOR	28
3	24	LEE	27
4	38	GONZALEZ	19
5	50	COLLINS	18
6	51	STEWART	22
7	55	REED	32
8	66	WARD	19
9	68	PETERSON	31
10	92	SIMMONS	25
11	98	GRIFFIN	29
12	103	HAMILTON	21
13	120	ORTIZ	22
14	137	KENNEDY	17
15	142	BURNS	27