DES3UE Datenbanksysteme WS 2024 Übung 2

Abgabetermin: siehe e-Learning, Abgabeform elektronisch

☒ **DES3UEG1: Glock** **Name** Elias Leonhardsberger **Aufwand in h** 8

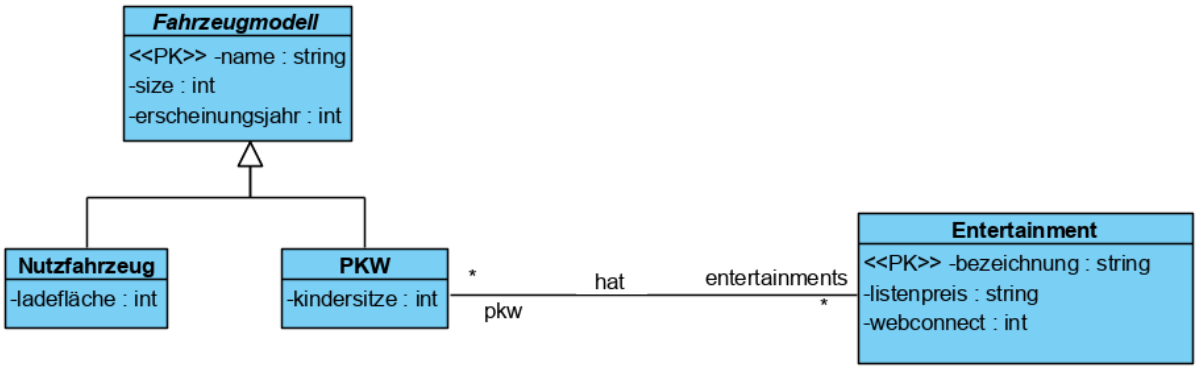
☐ **DES3UEG2: Werth** **Punkte** _____ **Kurzzeichen Tutorin** _____

1. Abbildung Generalisierung **(4 Punkte** – je 2 Pkte.**)**

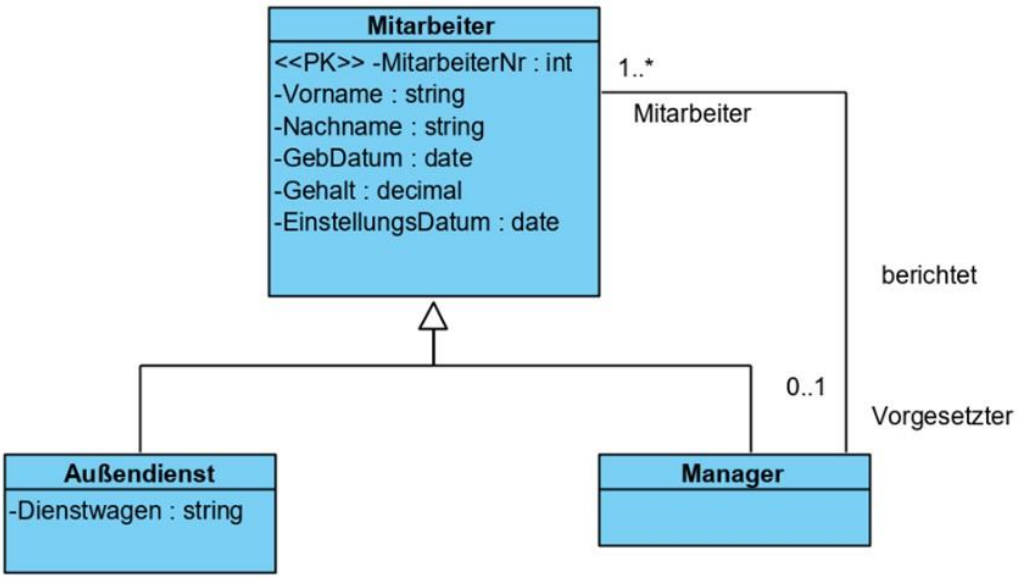
Erstellen Sie für die angegebenen UML-Klassen-Diagramme jeweils ein **Relationenmodell**, um die Attribute in Tabellen in der Datenbank abzubilden, zB Tabelle 1 (PKAttr1, Attr2). Wählen Sie jeweils ein geeignetes Abbildungsmodell für die Generalisierungsbeziehung.

Begründen Sie Ihre Wahl: D.h. nennen Sie das gewählte Abbildungsmodell und geben Sie dessen Vor- und Nachteile an. Nehmen Sie dabei Bezug auf die Eigenschaften der Generalisierungsbeziehung, d.h. ob diese **vollständig/unvollständig** und **überlappend/disjunkt** ist. Geben Sie auch an, warum sich die anderen Modelle nicht bzw. schlechter eignen.

a) Ein Fahrzeughersteller verwaltet die Fahrzeuge von zwei Sparten Nutzfahrzeuge und PKWs. Die Sparten werden getrennt verwaltet, ein Modell wird nicht in beiden Sparten hergestellt.



b) In einem Unternehmen werden Mitarbeiter verwaltet, Mitarbeiter im Außendienst erhalten ein Fahrzeug zur Verfügung gestellt; im Außendienst gibt es ebenso Manager.



2. SQL-Wiederholung (Sakila-DB) (5 Punkte)

1. Geben Sie eine Liste mit Filmtitel aus, deren Namen an vierter Stelle ein 'A' enthält, geben Sie die Titel so aus, dass jeweils der erste Buchstabe eines Wortes mit einem Großbuchstaben beginnt (zB Atlantis Cause). *(67 Zeilen, 0,5 Punkte)*

2. Geben Sie alle Inventar-Ids aus, die noch nie verliehen wurden. Verwenden Sie einen Mengenoperator. *(1 Zeile, 0,5 Punkte)*

3. Geben Sie die Anzahl der Verleihvorgänge zwischen 1.1.2015 und 31.12.2015 aus (Start des Verleihvorgangs, *rental_date*). *(1 Zeile, 1 Punkt)*

4. Geben Sie alle Filialen mit ID (store_id) und Stadt (city) gemeinsam mit den Nachnamen der zugeordneten Angestellten aus (JOIN). Listen Sie auch Filialen auf, die keine:n Angestellte:n zugewiesen haben und verwenden Sie den Platzhalter 'no staff'. *(9 Zeilen, 1 Punkt)*

5. Geben Sie die Namen aller Schauspieler aus, die in Filmen mitspielen, die nach 2006 erschienen sind und in denen auch die Schauspieler mit den IDs 10, 20 oder 30 vorkommen. *(36 Zeilen, 1 Punkt)*

6. Geben Sie für den Kunden mit der ID 250 alle Verleihvorgänge mit dem Start des Verleihvorgangs und dem bezahlten Betrag aus. Geben Sie das Datum in zwei Sprachen aus (**Format** 'Mi, 11. Oktober 2023' bzw. 'Wed, 11st October 2023' aus, verwenden Sie dazu die Funktion *to_char* inkl. NLS-Parameter und recherchieren Sie bei Bedarf in der Oracle-Dokumentation. *(1 Punkt)*

3. Sakila-Statistik (3 Punkte)

Erstellen Sie die folgenden Statistikberichte für die Sakila-Geschäftsführung: Nehmen Sie die Nummer, Vor und Nachname des Managers und die Größe des Inventars (Anzahl der Filme) für jeden Store auf, die folgende Bedingungen erfüllt:

1. Keine oder mehr als 1 Angestellte/r
2. Höchste Anzahl von Filmen
3. Niedrigste Anzahl von Filmen

Hinweis:
Beachten Sie, dass es Stores ohne Mitarbeiter und ohne Inventar geben kann bzw. könnte. Prüfen Sie die Korrektheit Ihrer Abfragen, indem Sie einen neuen Store einfügen (z.B. address_id 223, last_update jetzt, managere_staff_id 1, store_id 7)
Machen Sie durchgeführte Änderungen nach Ausführung der Teststatements wieder rückgängig!

4. Aggregate und Gruppierungen (Sakila-Datenbank) (5 Punkte)

1. Ermitteln Sie die Titel jener Filme, die zwar in der Filmdatenbank existieren, allerdings in keinem Geschäft angeboten werden. (*42 Zeilen, 1 Punkt*)
2. Geben Sie pro Kunden-Name (Vorname, Nachname) die Anzahl geliehener Filme an. Sortieren Sie nach der Anzahl aufsteigend. (*599 Zeilen, 1 Punkt*)
3. Ermitteln Sie alle Filme, die die längsten in ihrer Sprache sind und geben Sie Titel, Dauer (*length*) sowie Sprache (Name) aus. Im Ergebnis sollen nur jene Filme angezeigt werden, bei denen eine Originalsprache eingetragen ist. *(11 Zeilen, 1 Punkt*)
4. Geben Sie in Absteigender Reihenfolge an, welcher Kunde (Vorname, Nachname, ID) bereits vier oder mehr Horror-Filme ausgeliehen hat. (*34 Zeilen, 1 Punkt*)

5. Geben Sie den besten Kunden (gemessen am Umsatz, d.h. wie viel er gesamt bezahlt hat) pro Store an, sowie den Umsatz und die Store-ID des Kunden). Bedenken Sie dabei den Fall, dass Personen gleich heißen könnten. (*6 Zeilen, 1 Punkt*)

5. GROUP BY mit GROUPING SETS / ROLLUP (Sakila-Datenbank) (3 Punkte)

Erstellen Sie eine Filmabfrage und ermitteln Sie die Anzahl der Filme und die Summe der Längen in den Kategorien ‚Comedy' und ‚Music'

- je Rating
- je Kategorie und Rating
- „Comedy' und ‚Music' gesamt

Entwickeln Sie drei Varianten:

(1) einmal mit der Verwendung des GROUP BY ROLLUP Operators (*1 Punkt*) und
(2) zum Vergleich dazu mit GROUPING SETS (*1 Punkt*) und
(3) komplett ohne ROLLUP und SETS (Hinweis: 3 Selects mit UNION) (*1 Punkt*).

Vergleichen Sie die Zugriffspläne und kommentieren Sie diese.

Hinweis: SET AUTOTRACE ON erlaubt in SQL*PLUS die Ausgabe des Zugriffsplan sowie einiger Statistiken eines SQL-Statement und bietet eine einfache Möglichkeit zum Vergleich von Abfragen.

5. GROUP BY mit GROUPING SETS / CUBE (Sakila-Datenbank) (4 Punkte)

1. Erstellen Sie mit `GROUPING SETS` eine Abfrage, um folgende Gruppierungen anzuzeigen: (1 Punkt)
    - `manager_staff_id, store_id, staff_id`
    - `manager_staff_id, store_id`
    - `store_id, staff_id`

   Die Abfrage soll die Summe der Erlöse für jede dieser Gruppen berechnen (s. Lösungsauszug).

MANAGER_STAFF_ID	STORE_ID	STAFF_ID	SUM...
1	1	1	44694,83
2	2	2	…
3	3	3	…
…	…	..	…
5	5	5	6662,75
…	…	…	…
1	1	(NULL)	44694,83
(NULL)	1	1	44694,83
…	…	…	…
…	5	6	6454,64

2. Erstellen Sie einen Bericht über verliehene Filme, der pro Land, Jahr und Kategorie die Summe des Umsatzes und die Anzahl der Bezahlvorgänge (vgl. payment) zusammenfasst. Berücksichtigen Sie nur die Kategorien 'Family', 'Children' und 'Travel'. Erstellen Sie ein SQL-Statement und verwenden Sie GROUPING SETS. Sortieren Sie die Ausgabe nach Ländern. (1,5 Punkte)

JAHR	LAND	KATEGORIE	ANZAHL	SUMME
NULL	Australia	Children	153	1128,88
	...			
2013	Australia	NULL	5	41,16
	...			
NULL	NULL	NULL	2878	21871,51

3. Erstellen Sie eine Abfrage, um für alle Manager, die in den Stores angestellt sind, folgendes anzuzeigen (1 Punkte):
    - Manager-Id
    - Store und Gesamterlös für jeden Manager
    - Gesamterlöse aller Manager
    - Kreuztabellenwerte für die Anzeige des Gesamterlöses für jeden Standort
    - Gesamterlös, unabhängig vom Standort

   (Auszug aus der Lösung; Bitte vervollständigen Sie die Tabelle in Ihrer Abgabe)

MANAGER-ID	STORE	Gesamterlös
NULL	NULL	115657,58
NULL	1	44694,83
…	…	…
…	…	…
3	NULL	6620,07
3	3	6620,07
…	…	…

4. Prüfen Sie die Ausgabe der obigen Aufgabe. Schreiben Sie mit der `GROUPING`-Funktion eine Abfrage, um festzustellen, ob die Nullwerte in den Spalten, die den `GROUP BY` Ausdrücken entsprechen, von der `CUBE`-Operation verursacht werden. (0,5 Punkte)

DES3UE Übung 2

Elias Leonhardsberger

23. November 2024, Hagenberg

Inhaltsverzeichnis

1 Abbildung Generalisierung

Die Groß/Kleinschreibung wurde aus der Angabe übernommen.

1.1 a) Fahrzeuge

Für die Vollständigkeit der Generalisierung kann man nur Annahmen treffen, da die Angabe nicht ausführlich genug ist. Ich nehme daher an das die Generalisierung *vollständig* ist. Aus der Angabe liest man auch heraus das sie *disjunkt* ist.

Da es eine *n to m* Beziehung auf eine der Subklassen gibt entscheide ich mich für das Partitionierungsmodell. Dieses bietet mir die Möglichkeit die Relation nur auf die Subklasse via Foreign Key zu definieren. Weiters ist sie die einzige Abbilding die die Normalformen einhält. Performanzweise könnte man über eine Einrelationenabbildung streiten, da man aber die Beziehung nicht richtig abbilden kann fällt diese weg.

1.1.1 Relationenmodell

$$
\begin{aligned}
Fahrzeugmodell(&\underline{name}, size, erscheinungsjahr)\\
Nutzfahrzeug(&\underline{name} : \mathrm{FK}(Fahrzeugmodell), ladefläche)\\
PKW(&\underline{name} : \mathrm{FK}(Fahrzeugmodell), kindersitze)\\
Entertainment(&\underline{bezeichnung}, listenpreis, webconnect)\\
PKW\_Entertainment(&\underline{bezeichnung} : \mathrm{FK}(Entertainment), \underline{name} : \mathrm{FK}(PKW))
\end{aligned}
$$

1.2 b) Mitarbeiter

Dieses Beispiel ist eindeutig *unvollständig* und *überlappend*.

Da es wieder eine Beziehung auf eine der Subklassen gibt entscheide ich mich wieder für das Partitionierungsmodell. Die Einrelationenabbildung wäre wieder möglich und ist, da ein Manager keine Attribute hat, auch sinnvoll, aber man kann nicht über das Model garantieren, dass nur Manager Vorgesetzte sein können.

1.2.1 Relationenmodell

$$
\begin{aligned}
Mitarbeiter(&\underline{MitarbeiterNr}, Vorname, Nachname,\\
&GebDatum, Gehalt, EinstellungsDatum\\
&Vorgesetzter : \mathrm{FK}(Manager))
\end{aligned}
$$

$$
\begin{aligned}
Manager(&\underline{MitarbeiterNr} : \mathrm{FK}(Mitarbeiter))\\
Aussendienst(&\underline{MitarbeiterNr} : \mathrm{FK}(Mitarbeiter), Dienstwagen)
\end{aligned}
$$

2 SQL-Wiederholdung

```sql
-- 1
SELECT INITCAP(TITLE) AS CapitalizedTitle
FROM FILM
WHERE TITLE LIKE '___A%';

-- 2
SELECT INVENTORY_ID
FROM INVENTORY i
WHERE NOT EXISTS (SELECT r.RENTAL_ID FROM RENTAL r WHERE r.INVENTORY_ID =
 ↪  i.INVENTORY_ID);

-- 3
SELECT COUNT(*)
FROM RENTAL
WHERE RENTAL_DATE >= TO_DATE('2015-01-01', 'YYYY-MM-DD')
  AND RENTAL_DATE <= TO_DATE('2015-12-31', 'YYYY-MM-DD');

-- 4
SELECT s.STORE_ID, C.CITY, COALESCE(m.LAST_NAME, 'no staff') AS
 ↪  ManagerLastName
FROM STORE s
INNER JOIN ADDRESS A
    ON s.ADDRESS_ID = A.ADDRESS_ID
INNER JOIN CITY C
    ON a.CITY_ID = C.CITY_ID
LEFT JOIN STAFF m
    ON s.MANAGER_STAFF_ID = m.STAFF_ID;

-- 5
SELECT DISTINCT a.FIRST_NAME, a.LAST_NAME
FROM ACTOR a
INNER JOIN FILM_ACTOR fa
    ON a.ACTOR_ID = fa.ACTOR_ID
INNER JOIN FILM f
    ON fa.FILM_ID = f.FILM_ID AND
       f.RELEASE_YEAR > 2006
INNER JOIN FILM_ACTOR fa2
    ON f.FILM_ID = fa2.FILM_ID AND fa2.ACTOR_ID IN (10, 20, 30);

-- 6
SELECT p.AMOUNT,
    INITCAP(TO_CHAR(r.RENTAL_DATE, 'DY, dd. MONTH YYYY',
     ↪  'NLS_DATE_LANGUAGE = german')),
    INITCAP(TO_CHAR(r.RENTAL_DATE, 'DY, ddth MONTH YYYY',
     ↪  'NLS_DATE_LANGUAGE = english'))
```

```
42 FROM RENTAL r
43 JOIN PAYMENT p
44     ON r.RENTAL_ID = p.RENTAL_ID
45 WHERE r.CUSTOMER_ID = 250;
```

3 Sakila-Statistik

```sql
-- Annahme: Eine der Bedingungen muss gegeben sein, Min und Max würden sonst imme eine leere Zeile ergeben.
SELECT s.STORE_ID, m.FIRST_NAME, m.LAST_NAME, COUNT(DISTINCT i.INVENTORY_ID) AS InventoryCount
FROM STORE s
LEFT JOIN STAFF m
    ON s.MANAGER_STAFF_ID = m.STAFF_ID
LEFT JOIN INVENTORY i
    ON s.STORE_ID = i.STORE_ID
LEFT JOIN STAFF e
    ON e.STORE_ID = s.STORE_ID
GROUP BY s.STORE_ID, m.FIRST_NAME, m.LAST_NAME
HAVING COUNT(DISTINCT e.STAFF_ID) > 1
    OR COUNT(DISTINCT e.STAFF_ID) = 0
    OR COUNT(DISTINCT i.INVENTORY_ID) = MAX(DISTINCT i.INVENTORY_ID)
    OR COUNT(DISTINCT i.INVENTORY_ID) = MIN(DISTINCT i.INVENTORY_ID);

-- INSERT INTO STORE (STORE_ID, MANAGER_STAFF_ID, ADDRESS_ID, LAST_UPDATE)
-- VALUES (7, 1, 223, CURRENT_TIMESTAMP);
--
-- DELETE
-- FROM STORE
-- WHERE STORE_ID = 7;
```

4 Aggregate und Gruppierungen

```sql
-- 1
SELECT TITLE
FROM FILM f
WHERE NOT EXISTS (SELECT * FROM INVENTORY i WHERE i.FILM_ID = f.FILM_ID);

-- 2
SELECT C.FIRST_NAME, C.LAST_NAME, COUNT(*) AS RentedFilmCount
FROM CUSTOMER C
INNER JOIN RENTAL r
    ON C.CUSTOMER_ID = r.CUSTOMER_ID
INNER JOIN INVENTORY i
    ON r.INVENTORY_ID = i.INVENTORY_ID
GROUP BY C.FIRST_NAME, C.LAST_NAME
ORDER BY COUNT(*);

-- 3
SELECT OUTPUT.TITLE, OUTPUT.LENGTH, G.LanguageName
FROM FILM OUTPUT
INNER JOIN
(SELECT l.NAME AS LanguageName, l.LANGUAGE_ID, MAX(f.LENGTH) AS MaxLength
FROM FILM f
INNER JOIN LANGUAGE l
    ON f.LANGUAGE_ID = l.LANGUAGE_ID
WHERE f.ORIGINAL_LANGUAGE_ID IS NOT NULL
GROUP BY l.NAME, l.LANGUAGE_ID) G
    ON G.LANGUAGE_ID = OUTPUT.LANGUAGE_ID AND G.MaxLength = OUTPUT.LENGTH
WHERE OUTPUT.ORIGINAL_LANGUAGE_ID IS NOT NULL;

-- 4
SELECT C.FIRST_NAME, C.LAST_NAME, C.CUSTOMER_ID AS RentedFilmCount
FROM CUSTOMER C
INNER JOIN RENTAL r
    ON C.CUSTOMER_ID = r.CUSTOMER_ID
INNER JOIN INVENTORY i
    ON r.INVENTORY_ID = i.INVENTORY_ID
INNER JOIN FILM f
    ON f.FILM_ID = i.FILM_ID
INNER JOIN FILM_CATEGORY fc
    ON f.FILM_ID = fc.FILM_ID
INNER JOIN CATEGORY cat
    ON fc.CATEGORY_ID = cat.CATEGORY_ID
WHERE cat.NAME = 'Horror'
GROUP BY C.FIRST_NAME, C.LAST_NAME, C.CUSTOMER_ID
HAVING COUNT(*) >= 4
ORDER BY COUNT(*) DESC;
```

```sql

-- 5
-- Als einzelne Query ist das recht komplex, RevenuePerCustomer könnte man herausheben um die Query zu optimieren
SELECT cOut.FIRST_NAME, cOut.LAST_NAME, cOut.CUSTOMER_ID, cOut.STORE_ID, RevenuePerCustomer.Revenue
FROM CUSTOMER cOut
INNER JOIN
(SELECT CUSTOMER_ID, SUM(AMOUNT) AS Revenue FROM PAYMENT GROUP BY CUSTOMER_ID) RevenuePerCustomer
    ON RevenuePerCustomer.CUSTOMER_ID = cOut.CUSTOMER_ID
INNER JOIN
(SELECT C.STORE_ID, MAX(RevenuePerCustomer.Revenue) AS MaxRevenue
FROM CUSTOMER C
INNER JOIN
(SELECT CUSTOMER_ID, SUM(AMOUNT) AS Revenue FROM PAYMENT GROUP BY CUSTOMER_ID) RevenuePerCustomer
    ON C.CUSTOMER_ID = RevenuePerCustomer.CUSTOMER_ID
GROUP BY C.STORE_ID) MaxRevenuePerStore
    ON cOut.STORE_ID = MaxRevenuePerStore.STORE_ID AND
     RevenuePerCustomer.Revenue = MaxRevenuePerStore.MaxRevenue;
```

5 GROUP BY mit GROUPING SETS / ROLLUP

```sql
-- rollup
SELECT C.NAME, f.RATING, COUNT(*), SUM(f.LENGTH)
FROM FILM f
INNER JOIN FILM_CATEGORY fc
    ON f.FILM_ID = fc.FILM_ID
INNER JOIN CATEGORY C
    ON fc.CATEGORY_ID = C.CATEGORY_ID
WHERE C.NAME IN ('Comedy', 'Music')
GROUP BY ROLLUP (C.NAME, f.RATING);

-- grouping set
SELECT C.NAME, f.RATING, COUNT(*), SUM(f.LENGTH)
FROM FILM f
INNER JOIN FILM_CATEGORY fc
    ON f.FILM_ID = fc.FILM_ID
INNER JOIN CATEGORY C
    ON fc.CATEGORY_ID = C.CATEGORY_ID
WHERE C.NAME IN ('Comedy', 'Music')
GROUP BY GROUPING SETS ((C.NAME, f.RATING), C.NAME, ());

-- manuell
SELECT C.NAME, f.RATING, COUNT(*), SUM(f.LENGTH)
FROM FILM f
INNER JOIN FILM_CATEGORY fc
    ON f.FILM_ID = fc.FILM_ID
INNER JOIN CATEGORY C
    ON fc.CATEGORY_ID = C.CATEGORY_ID
WHERE C.NAME IN ('Comedy', 'Music')
GROUP BY C.NAME, f.RATING
UNION ALL
SELECT C.NAME, NULL, COUNT(*), SUM(f.LENGTH)
FROM FILM f
INNER JOIN FILM_CATEGORY fc
    ON f.FILM_ID = fc.FILM_ID
INNER JOIN CATEGORY C
    ON fc.CATEGORY_ID = C.CATEGORY_ID
WHERE C.NAME IN ('Comedy', 'Music')
GROUP BY C.NAME
UNION ALL
SELECT NULL, NULL, COUNT(*), SUM(f.LENGTH)
FROM FILM f
INNER JOIN FILM_CATEGORY fc
    ON f.FILM_ID = fc.FILM_ID
INNER JOIN CATEGORY C
    ON fc.CATEGORY_ID = C.CATEGORY_ID
```

```
46 WHERE C.NAME IN ('Comedy', 'Music');
```

5.1 Vergleich der Query Pläne

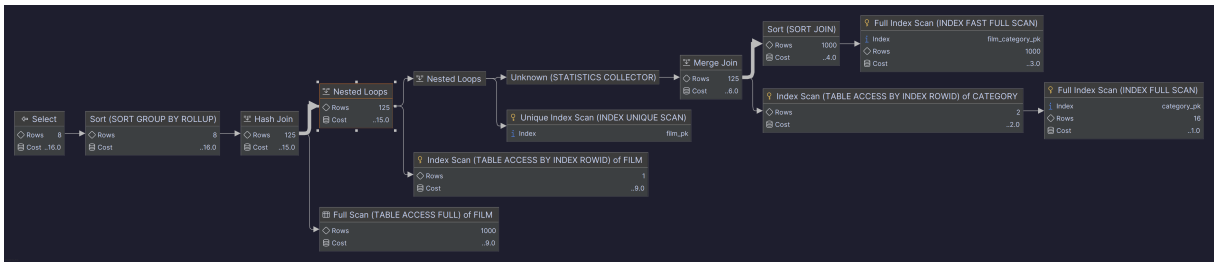


Abbildung 1: Queryplan ROLLUP

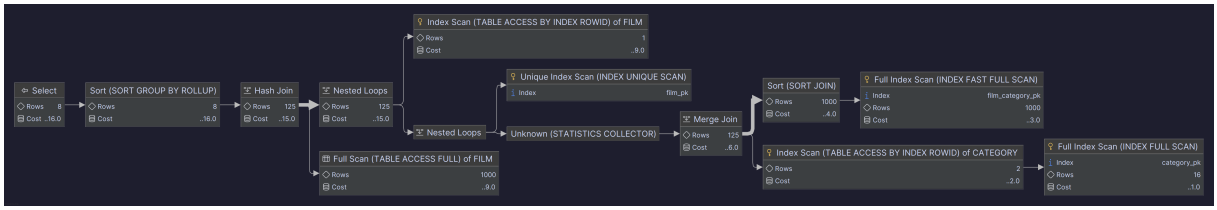


Abbildung 2: Queryplan GROUPING SETS

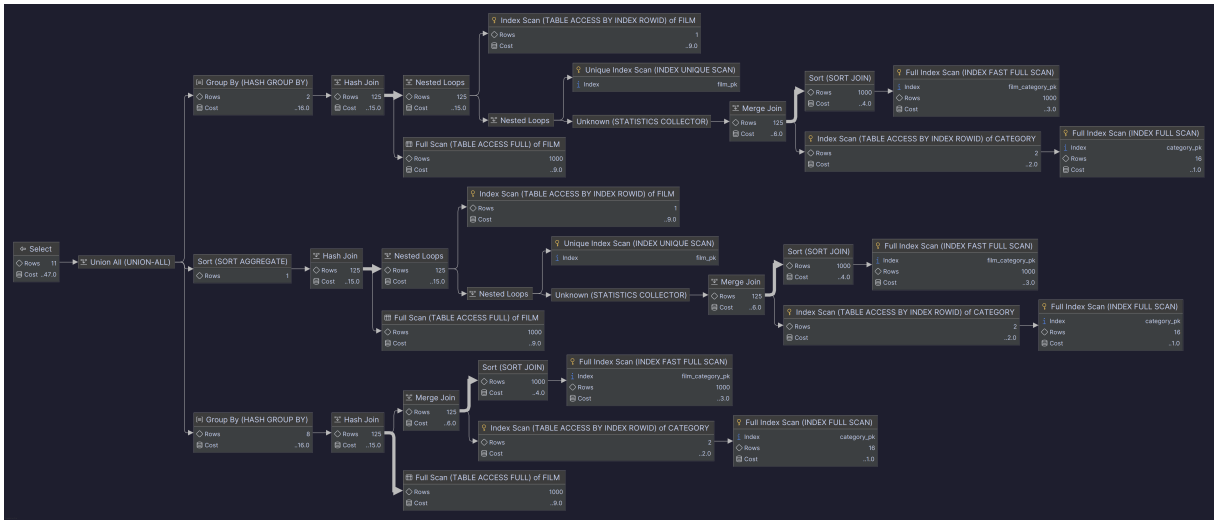


Abbildung 3: Queryplan manuelles GROUP BY

Wie man sieht optimiert die Datenbank die Querys unterschiedlich. Die GROUPING SETS und ROLLUP Querys sind sich sehr ähnlich und haben das selbe *weight*. Die manuelle GROUP BY Query hat ein höheres *weight* und ist daher langsamer, weil durch das UNION mehrfach abgefragt wird.

6 GROUP BY mit GROUPING SETS / CUBE

```sql
-- 1
SELECT s.MANAGER_STAFF_ID, s.STORE_ID, e.STAFF_ID, SUM(p.AMOUNT)
FROM PAYMENT p
INNER JOIN STAFF e
    ON p.STAFF_ID = e.STAFF_ID
INNER JOIN STORE s
    ON e.STORE_ID = s.STORE_ID
GROUP BY GROUPING SETS ( (s.MANAGER_STAFF_ID, s.STORE_ID, e.STAFF_ID ), (
↪  s.MANAGER_STAFF_ID, s.STORE_ID), ( s.STORE_ID,
    e.STAFF_ID));

-- 2
-- Annahme: Das Land des Geschäftes ist gemeint und alle Kombinationen
↪  von Jahr, Land und Kategorie sollen angezeigt werden
SELECT f.RELEASE_YEAR, co.COUNTRY, C.NAME, COUNT(*), SUM(p.AMOUNT)
FROM FILM f
INNER JOIN FILM_CATEGORY fc
    ON f.FILM_ID = fc.FILM_ID
INNER JOIN CATEGORY C
    ON fc.CATEGORY_ID = C.CATEGORY_ID
INNER JOIN INVENTORY i
    ON f.FILM_ID = i.FILM_ID
INNER JOIN RENTAL r
    ON i.INVENTORY_ID = r.INVENTORY_ID
INNER JOIN PAYMENT p
    ON r.RENTAL_ID = p.RENTAL_ID
INNER JOIN STORE s
    ON i.STORE_ID = s.STORE_ID
INNER JOIN ADDRESS a
    ON s.ADDRESS_ID = a.ADDRESS_ID
INNER JOIN CITY ci
    ON a.CITY_ID = ci.CITY_ID
INNER JOIN COUNTRY co
    ON ci.COUNTRY_ID = co.COUNTRY_ID
WHERE C.NAME IN ('Family', 'Children', 'Travel')
GROUP BY GROUPING SETS ( (f.RELEASE_YEAR, co.COUNTRY, C.NAME), (
↪  f.RELEASE_YEAR, co.COUNTRY),
    ( f.RELEASE_YEAR, C.NAME), ( co.COUNTRY, C.NAME), ( f.RELEASE_YEAR),
    ↪  ( co.COUNTRY), ( C.NAME), ())
ORDER BY co.COUNTRY;

-- 3
SELECT s.MANAGER_STAFF_ID, s.STORE_ID, SUM(p.AMOUNT)
FROM STORE s
INNER JOIN STAFF e
```

```
42      ON s.STORE_ID = e.STORE_ID
43  INNER JOIN PAYMENT p
44      ON e.STAFF_ID = p.STAFF_ID
45  GROUP BY CUBE (s.MANAGER_STAFF_ID, s.STORE_ID);
46
47  -- 4
48  -- laut DataGrip sollte man DECODE statt CASE verwenden, das führt aber
    ↪  meiner Meinung nach zu einem unleserlichen Statement
49  SELECT CASE GROUPING(s.MANAGER_STAFF_ID)
50             WHEN 1 THEN 'All Managers'
51             ELSE TO_CHAR(s.MANAGER_STAFF_ID)
52      END AS MANAGER_STAFF_ID,
53      CASE GROUPING(s.STORE_ID)
54          WHEN 1 THEN 'All Stores'
55          ELSE TO_CHAR(s.STORE_ID)
56          END AS STORE_ID,
57      SUM(p.AMOUNT)
58  FROM STORE s
59  INNER JOIN STAFF e
60      ON s.STORE_ID = e.STORE_ID
61  INNER JOIN PAYMENT p
62      ON e.STAFF_ID = p.STAFF_ID
63  GROUP BY CUBE (s.MANAGER_STAFF_ID, s.STORE_ID);
```