Technische Hochschule Augsburg

Fakultät für Informatik

Seminararbeit

Im Rahmen des DVA-Seminars
in Informatik

SQL Analyser-Erweiterungen

erstellt von Lukas Konietzka

Lukas Konietzka

Sebastian-Kneipp-Gasse 6A
86152 Augsburg
T +49 172-2728-376
lukas.konietzka@tha.de

Matrikelnummer:
2122553

Technische Hochschule Augsburg

An der Hochschule 1
D-86161 Augsburg
T +49 821 5586-0
F +49 821 5586-3222
www.tha.de
info@tha.de

Prüfer	Prof. Matthias Kolonko, Ph.D. (ONPU)
Eingereicht am	10. Januar 2025
Geheimhaltungsvereinbarung	Nein

Kurzfassung

Der SQL-Standard wird bald 40 Jahre alt und hat bis heute diverse Erweiterungen erfahren. Eine der wichtigsten sind die Analyse-Erweiterungen. Einige dieser Fähigkeiten haben es im Laufe der Jahre in den SQL-Standard geschafft und sind ab diesem Zeitpunkt ein fester Bestandteil von SQL. Trotz Ihrer doch langen Geschichte, stehen die Analyse-Erweiterungen eher im Schatten der breiten Palette an SQL-Befehlen. Dieser Artikel zeigt die Möglichkeiten und Sprachkonstrukte anhand von zwei Analyse-Erweiterungen. Hierzu wurde eine umfassende Literaturrecherche zum aktuellen Stand der Technik durchgeführt. Den Kern der Arbeit sollen die Beispielimplementierungen bilden, die anhand der *LibraryDB* aufgezeigt werden. Durch die spätere Analyse dieser Beispiele wird klar, dass die meisten Analyse-Erweiterungen viele Möglichkeiten bieten, jedoch auch eine Steigerung der Komplexität mitbringen.

Inhaltsverzeichnis

1. Einleitung

Der SQL-Standard wird bald 40 Jahre alt und hat bis heute diverse Erweiterungen erfahren. Ein der wichtigsten sind die Analyse-Erweiterungen. Einige dieser Analyse-Fähigkeiten haben es im Laufe der Jahre in den SQL-Standard geschafft und sind ab diesem Zeitpunkt ein fester Bestandteil von SQL. Trotz Ihrer doch langen Geschichte, stehen die Analyse-Erweiterungen eher im Schatten der breiten Palette an SQL-Befehlen und konnten nicht sonderlich viel an Bekanntheit gewinnen.

1.1. Ziel der Arbeit

Diese Arbeit soll für Klarheit sorgen und das Geheimnis dahinter offenbaren. Es werden einige dieser Analyse-Erweiterungen anhand einfacher Beispiele aufzeigen, um eine eingehende Erklärung zu bieten. Hierzu sollen die unterschiedlichen Sprachkonstrukte und Methoden demonstriert und die Konzepte dahinter erläutert werden. Es kann so folgende konkrete Problemstellung abgeleitet werden.

"Welche Analyse-Erweiterungen gibt es und welche Möglichkeiten ergeben sich durch einen Einsatz dieser Konstrukte."

Sofern auf die Forschungsfrage eine Antwort gefunden werden kann, besteht am Ende dieser Arbeit ein grober Überblick über die bekanntesten Analyse-Erweiterungen im SQL-Standard und für welche Fälle diese eingesetzt werden. Außerdem soll ein Verständnis bestehen, wie auch komplexere Datenabfragen durch Einsatz der Analyse-Erweiterungen vereinfacht werden. Darüber hinaus soll die Arbeite eine praxisnahe Orientierungshilfe liefern, die als Stütze bei der Datenanalyse mit SQL fungiert. Um diesen Aspekt noch weiter zu verdeutlichen, soll das nächste Kapitel die Relevanz dieser ARbeit noch genauer beleuchten.

1.2. Relevanz der Analyse-Erweiterungen

Datenanalysen gehören heute zu den unverzichtbaren Werkzeugen für Entscheidungen in Unternehmen und Forschung. Da sich viele Unternehmen auf die klassischen relationalen Datenbanken stützen, wird SQL als Standard zur Abfrage von Daten und zur Erstellung von Analysen eingesetzt. Die SQL Analyse-Erweiterungen sollen genau diesen Analyseprozess unterstützen.

Die Analyse-Erweiterungen für SQL finden vorrangig ihren Einsatz im Bereich Data Analysis, so belegen es auch Fotache und Strimbei (2015, Kapitel 3). Es ist in diesem Kontext oft von *analytical* oder *window functions* die Rede (vgl. Fotache und Strimbei 2015, Kapitel 3). Die Begriffe *statistical inspired aggregate functions* und *multiple group by operators* tauchen im Fachbereich Data Analysis ebenfalls immer wieder auf (vgl. Fotache und Strimbei 2015, Kapitel 4.3). Alle diese Fachbegriffe sind auf die Analyse-Erweiterungen in SQL zurückzuführen. Auch in den Punkten Komplexität und Kompaktheit können die Erweiterungen punkten. Mit den Analyse-Erweiterungen ist eine komplexe Datenanalyse durch nur wenige Zeilen Quelltext möglich so Maue (2022, Abstract).

Es wird also deutlich, dass die Analyse-Erweiterungen nicht nur eine komplexe Analyse zulassen, sondern für das ganze Gebiet der Datenanalyse einen hohen Wert bietet. Sie gewähren eine weitaus detailliertere Einsicht in Datensätze, als es mit herkömmlichen Befehlen möglich ist. Daraus lässt sich auch ableiten, dass das Themer der Analyse-Erweiterungen große Kreise wirft. Um beim lesen dieser Arbeit einen groben Überblick zu schaffen wird die Gliederung der Arbeit im Folgenden Besxchrieben

1.3. Aufbau der Arbeit

Diese Arbeit liefert am Ende einen groben Überblick über die Analyse-Erweiterungen und deren Möglichkeiten. Darüber hinaus sollen einige praxisbezogenen Beispiele diskutieren werden. Hierzu teilt sich das Thema in drei Teile auf. Der erste Teil bildet eine Recherche zum aktuellen Stand der Technik und führt in die Grundlagen der hier fokussierten Erweiterungen ein. Hierzu wird einen Blick auf die Funktionsweise der unterschiedlichen Erweiterungen geworfen. Ergänzend dazu sollen auch zusätzliche Erweiterungen aufgezählt werden. Der zweite Teil bildet den Kern der Arbeit und soll die fokussierten Analyse-Erweiterungen anhand eines passenden Beispiels erläutern. Hierzu wird ein globales Beispiel einer relationalen Datenbank generiert, mit deren Hilfe alle der ausgewählten Erweiterungen demonstriert und analysiert werden. Für die Analyse wird auf Bewertungskriterien zurückgegriffen, die generisch gewählt werden. So

können die sehr unterschiedlichen Erweiterungen kategorisiert werden. Im letzten Teil der Arbeit sollen die Ergebnisse evaluiert werden. Hierzu werden die Beispiele analysiert und mögliche Interpretationen ausgearbeitet.

1.4. Fokus der Arbeit

Es existieren diverse Analyse-Erweiterungen für SQL die für die unterschiedlichsten Anwendungsfälle eingesetzt werden können. Einige dieser Erweiterungen lösen Nischenprobleme, andere bieten eine breite Palette an Funktionen. Dieser Artikel wählt zwei der vielen Analyse-Erweiterungen aus und setzt so einen Fokus.

Für eine konkrete Auswahl stützt sich die Arbeit auf Erweiterungen, die laut Fotache und Strimbei (2015, Kapitel3), für den Fachbereich Data Analysis eine wichtige Rolle spielen. Fotache und Strimbei (2015, Kapitel 3) sprechen in ihrer Arbeit von zwei wichtigen Punkten, die auch hier den Fokus bilden sollen.

Wie bereits zu Anfang der Arbeit hervorgeht, sind zwei dieser möglichen Erweiterungen die folgenden.

- *Window-Functions* als statistical inspired aggregate functions
  (vgl. Fotache und Strimbei 2015, Kapitel 4.3)
- *ROLLUP and CUBE* als multiple group by operators
  (vgl. Fotache und Strimbei 2015, Kapitel4.3)

Im Laufe des Artikels soll Wissen zu diesen beiden Erweiterungen gesammelt und an praxisbezogenen Beispiel angewandt werden. Alle weiteren Analyse-Erweiterungen werden in diesem Artikel nur kurz namentlich erwähnt und erfahren keine weitere Behandlung. Um dies zu gewährleisten sind einige Theoretische Grundlagen nötig, die zum verstehen der Ergebnisse essenziell sind. Das Kapitel 2 führt demnach in alle wichtigen Hintergründe ein.

2. Theoretische Grundlagen

Die SQL Analyse-Erweiterungen sind spezielle Erweiterungen der SQL-Sprache, die es ermöglichen, anspruchsvollere analytische Abfragen und Datenanalysen direkt in SQL durchzuführen. Dieses Kapitel soll die Grundsteine legen und in die Konzepte und Theorien der Analyse-Erweiterungen einführen. Um die Fragestellung dieser Arbeit einordnen zu können, ist unter anderem ein Blick auf die Historie hilfreich. Der Hauptgrund für die Wahl der zwei oben genannten Erweiterungen in Kapitel 1.4 liegt nämlich nicht nur in ihren vielseitigen Möglichkeiten, sondern auch in ihrer Integration in den SQL-Standard, wie aus der Arbeit von Grust (2017, S. 10) hervorgeht. Durch nähere Betrachtung der Arbeit von Grust 2017 wird deutlich, dass im Jahre 1987 die erste Version des SQL-Standards veröffentlicht wurde. Mit SQL3 1999 kamen dann die ersten Analyse-Erweiterungen mit den *recursive queries* hinzu (vgl. Grust 2017, S. 10). Die Erweiterungen `ROLLUP` und `CUBE` wurden ebenfalls im Jahre 1999 in den SQL-Standard aufgenommen (vgl. Melton und Simon 2001, Kapitel 9.12). Grust (2017, S. 10) verdeutlicht, dass die *Window Functions* mit einer späteren Version im Jahr 2003 erschienen sind. Das nachfolgende Kapitel führt in diese eben erwähnten *Window Functions* ein.

2.1. Window Functions

Die *Analytic Functions* oder auch *Window Functions* genannt, sind also schon lange Teil des SQL-Standards und somit auch für die Datenanalyse innerhalb von SQL ein wichtiger Bestandteil. Diese Funktionen sind aus eine komplexe Datenanalyse nicht mehr wegzudenken. Cao u. a. (2012, Abstract) bezeichnen sie für den aktuellen Stand der Technik als repräsentativ und absolut notwendig. Hinzu kommt, dass sie laut Kellenberger u. a. (2019, Kapitel 8) einen Performance Vorteil bietet. Dieser Performanc vorteil ergibt sich jedoch laut Kellenberger u. a. (2019, Kapitel 8) nicht primär durch die bessere Laufzeit, sondern vielmehr dadurch, wie einfach komplexe Abfragen damit werden können. Für eine prägnante und eindeutige Definition der *Analytic Functions* sei auf die Dokumentation der Oracle Corporation (2021) Datenbank verwiesen, die die *Analytic Functions* als aggregierten Wert basierend auf einer Gruppen von Datensätzen beschreibt. Diese Gruppen an Datensätzen werden als Fenster bezeichnet und sind über die *analytic clausel* definiert (vgl. Oracle Corporation 2021).

Wenn man dieser Definition folgt, dann werden darunter also aggregierte Werte verstanden, die mittels dem Schlüsselwort `OVER` auf bestimmte Zeilen in einer Tabelle angewendet werden. So ist es Beispielsweise möglich, auch benachbarte Datensätze mit einzubeziehen (vgl. Oracle Corporation 2021).

Wie die eben erwähhnte Definition bereits vermuten lässt, sind die analytischen Funktionen stark verwandt mit den weit verbreiteten Aggregatfunktionen. Zur Unterscheidung der Aggregatfunktion und den analytischen Funktionen kann nach Nuijten und Barel (2023, Seite 36) die Ergebnismenge betrachtet werden. Sie erläutern, dass Beispielsweise ein `COUNT(<...>)` als Aggregatfunktion nur eine Zeile zurückliefert, während analytische Funktionen die Ergebnismenge nicht verändern und für jeden Eintrag einen Wert vorsehen (vgl. Nuijten und Barel 2023, Seite 36).

Die Abarbeitung einer solchen Analyse-Funktion lässt sich in drei Stufen einteilen, wobei die Sortierung dieser drei Schritte auch die Reihenfolge der Abarbeitung festlegt (vgl. Nuijten und Barel 2023, Seite 35).

- *1. Stufe* auflösen der `JOIN`-, `WHERE`-, `GROUP BY`- und `HAVING`-Klausel (vgl. Nuijten und Barel 2023, Seite 35)
- *2. Stufe* die analytische Funktion wird auf die Ergebnismenge angewandt (vgl. Nuijten und Barel 2023, Seite 35)
- *3. Stufe* die ORDER BY-Klausel wir auf die Ergebnismenge angewandt (vgl. Nuijten und Barel 2023, Seite 35)

Um nun den Aufbau einer einfachen *Window Function* etwas genauer verstehen zu können, sei an dieser Stelle nochmals auf die Dokumentation von Oracle Corporation (2021) verwiesen. Hier ist ein konkreter Aufbau einer analytischen Funktion zu finden, die hier näher betrachtet werden soll. Der erste Teil beschreibt die konkrete analytische Funktion,

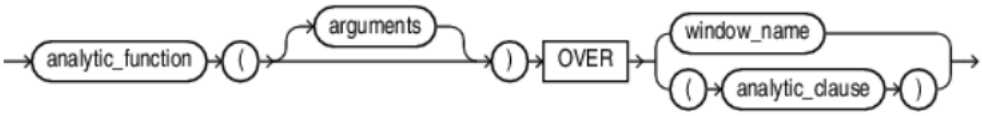

Abbildung 2.1.: Aufbau einer analytischen Funktion (vgl. Oracle Corporation 2021)

.

die auf die Ergebnismenge ausgeführt wird. Hier besteht die größte Parallelität zu den klassischen Aggregatfunktionen. Das Argument einer Analyse-Funktion legt fest, auf welche Spalte diese angewendet werden soll (vgl. Schicker 2017, S. 110). Eine grobe Übersicht über verschiedenen Window-Functions beschreibt Ogunbiyi (2023) in seinen Artikel. Hierzu unterteilt er die möglichen Funktionen übersichtlich in drei verschiedenen Gruppen. Eine weitere gute Informationsquelle bietet wieder die Oracle Corporation (2021) Dokumentation.

Das Schlüsselwort `OVER` wurde bereits zuvor in diesem Artikel erwähnt. Es ist der namensgebende Teil der Window-Functions, da so die Fenster über die Ergebnismenge gelegt werden. Bleibt das Argument der `OVER()` Funktion leer, so wird das Fenster über alle Einträge gespannt. Ein Beispiel liefert Nuijten und Barel (2023, Seite 36).

```
AVG(<..>) OVER()
```

Im Argument der `OVER` Funktion wird die Ausprägung und die Ordnung des Fensters definiert. Dieses Argument ist bekannt als *analytic-clause* (Oracle Corporation 2021, vgl.). Der Aufbau dieser Klausel sei in Abbildung 2.2 beschrieben (Oracle Corporation 2021, vgl.). Aus dem Aufbau der *analytic-clause* gemäß der Oracle Corporation (2021)

Abbildung 2.2.: Aufbau einer analytischen Klausel (vgl. Oracle Corporation 2021)

.

wird klar, dass diese grob in zwei Teile aufgeteilt wird. Ein *partition-clause*, die die einzelnen Fenster festlegt und die *order-by-clause* , welche die Reihenfolge in den einzelnen Fenstern festlegt. Diese beiden Klauseln können mit den Schlüsselwortern `PARTITION BY` und `ORDER BY` realisiert werden. Ein etwas generische Beispiel sieht wie folgt aus (Oracle Corporation 2021, vgl.).

```
PARTITION BY <..> ORDER BY <..>
```

Wenn die gezeigten Aufbauten aus den Abbildungen 2.1 und 2.2 zusammengesetzt werden, entsteht eine Analyse-Funktion, die bereits eine höhere Ausprägung hat. Für eine korrekte Fusion der beiden Klauseln wir die analytische Klausel in das Argument der `OVER` Funktion eingefügt (vgl. Nuijten und Barel 2023, Seite 36).

```
AVG(<..>) OVER (Partition BY <..> ORDER BY <..>)
```

Dieses Sprachkonstrukt kann nun in eine bekannte `SELECT` Klausel eingebaut werden. Sie wird an der gleichen Stelle eingebaut, an der auch eine klassische Aggregatfunktion verwendet wird. Für weitere Details sei auf die Dokumentation von Oracle Corporation (2021) verwiesen, die einen tieferen Einblick in den Aufbau der einzelnen Klauseln gewähren.Für die zweite konkrete analytischen Funktionen führt das Kapitel Theoretische Grundlagen im folgenden in die Multidimensonalen Group BY Klauseln ein, welches alle relevanten Grundlagen zum Thema `ROLLUP` und `CUBE` enthält.

2.2. Multidimensonale Group By-Klauseln

Auch die letzte Analyse-Erweiterung die in diese Arbeit behandelt werden soll, macht ihrem Namen alle Ehre und gewährt mit wenigen Zeilen Quelltext, eine viel analytischere Einsicht in die Ergebnismenge, als es mit herkömmlichen SQL-Befehlen möglich ist (vgl. Melton 2002, Kapitel 7.2.3). Melton (2002, Kapitel 7.2.3) beschreibt diese als mehrdimensionale Zusammenfassungen und kontrollierte Unterbrechungen von gruppierten Date. Diese Formulierung lässt erahnen, dass die Konzepte, `ROLLUP` und `CUBE` Erweiterungen der `GROUP BY` Klausel sind. Bevor mit den eigentlichen Erweiterungen begonnen werden kann, muss hier ein Grundkonzept vorangestellt werden. Dieses Führt schrittweise die Schlüsselwörter `ROLLUP` und `CUBE` ein. Hierbei handelt es sich um das Konzept des `GROUPING SETS`.

2.2.1. Grouping Sets

Das Schlüsselwort `GROUPING SETS` ist ebenfalls eine Erweiterung der herkömmlichen `GROUP BY` Klausel wie auch die Oracle Corporation (2016) Dokumentation belegt. Wie aus dem Namen schon hervorgeht, handelt es sich nicht um eine einzelne Gruppierung, sondern um ein ganzes Set an Gruppierungen. Das Set lässt darauf deuten, dass jede Gruppierung nur einmal vorkommt. Das Schlüsselwort `GROUPING SETS` erlaubt mehrere Gruppierungen in einer Abfrage. Jedoch wird nur eine angegebene Gruppe gruppiert und nicht alle möglichen Gruppen (vgl. Oracle Corporation 2016). Für eine Kombination an Gruppierungen sei an dieser Stelle schon auf die Erweiterungen `ROLLUP` und `CUBE` verwiesen. Folgender Ausschnitt zeigt ein generisches Beispiel.

```
GROUP BY GROUPING SETS (A, B)
```

Folgt man der Dokumentation von Oracle Corporation (2016) und dem hier gezeigten Beispiel, so ergeben sich bei einem `GROUPING SETS` folgenden Gruppierungen:

$$\{A\}, \quad \{B\}$$

Die Ergebnismenge dieser Gruppierung ist einmal die Gruppierung nach A und einmal nach B. Es werden keine Kombinationen gruppiert, nur die einzelnen Spalten (vgl. Oracle Corporation 2016). Sollen unterschiedliche Kombinationen der Spaltern Gruppiert werden, so sei auf die nachfolgenden Kapitel `ROLLUP` und `CUBE` verwiesen.

2.2.2. Rollup

Wie aus dem vorherigen Kapitel bereits hervorgeht, ist `ROLLUP` eine Erweiterung der `GROUP BY`- und `GROUPING SETS`-Klausel. Einen Rollup ist als Zwischensumme zu betrachten, die von der kleinsten Ebenen bis hin zur größten "*aufgerollt*"wird, so beschreibt es die Oracle Corporation (1999). Die Reihenfolge der Gruppierung folgt der angegebenen Gruppierungsliste. Die Richtung der Gruppierungsreihenfolge ist von rechts nach links (vgl. Oracle Corporation 2016).

```
GROUP BY ROLLUP (A, B, C);
```

Durch genauere Betrachtung des hier gezeigten Beispiels soll die genau Art und Weise der `ROLLUP` Gruppierung näher erläutert werden. Hierzu werden die unterschiedlichen Gruppierungen wieder als Menge betrachtet.

$$\{\}, \{A\}, \{A, B\}, \{A, B, C\}$$

Mit einem `ROLLUP` werden also in diesem Beispiel vier Mengen geliefert. Die leer Klammer steht für die leere Menge und damit die Gesamtsumme. Es wird hier rechts mit der kleinsten Menge begonnen und links auf die größte aufgerollt (`ROLLUP`).

2.2.3. Cube

Mit einem `ROLLUP` sind nicht alle möglichen Kombinationen an Gruppierungen möglich (vgl. Oracle Corporation 1999). Die Erweiterung `ROLLUP` gibt eine eindeutige Gruppierungsreihenfolge vor, die ohne Ausnahme eingehalten wird. Wenn eine andere Reihenfolge oder mehrere Gruppierungen gewollt sind, muss auf das Konzept der `CUBE` Erweiterung zurückgegriffen werden, welche Oracle Corporation (1999) in ihrer Dokumentation beschreiben. Wird die Methode des `CUBE` verwendet, so ist oft von einem Datenwürfel die Reden. Dieser aggregiert alle möglichen Kombinationen der Gruppierungen, einschließlich der Gesamtsumme und erstellt so Datenwürfel (vgl. Oracle Corporation 1999).

```
GROUP BY CUBE (A, B, C);
```

Es ergeben sich wieder verschiedenen Kombinatinen für die Spalten A, B, C

$$\{A, B, C\}, \{A, B\}, \{A, C\}, \{B, C\}, \{A\}, \{B\}, \{C\}, \{\}$$

Durch Betrachtung dieser Mengen wird klar, dass die `CUBE` Erweiterung bietet demnach noch genaueren Einblick in die vorliegenden Datensätze gewährt.

3. Methodik

Dieses Kapitel soll das Vorgehen und die konkrete Methodik während der Ausarbeitung dieser Arbeit näher erläutern. Begonnen wurde mit einer umfassenden Literaturrecherche zum aktuellen Stand der Technik. Anschließend wird eine Beispielimplementierung eingeführt anhand derer konkrete Beispiele aufgezeigt werden.

3.1. Literaturrecherche

Die Literaturrecherche bildet das Fundament dieser Arbeit und liefert alle nötigen Grundlagen und Hintergründe. Es werden die hier relevanten Konzepte und Methoden zu den Analyse-Erweiterungen eingeführt und erläutert. Darüber hinaus sind Grundkonzepte der einzelnen Erweiterungen zu finden. Die Recherer soll auch den aktuellen Stand der Kunst wiederspieglen und die bekanntesten und größten Erweiterungen fokusieren. Das Ziel der Recherche war es auch die offiziellen Dokumentationen der großen DBMS Hersteller mit einzubeziehen.

3.2. Implementierung

Bevor mit dem Erstellen einer Beispieldatenbank begonnen werden kann, muss ein Datenbank-Management-System (DBMS) gewählt werden, mit dem die Beispiele generiert werden können. Aufgrund der einfachen Installation und des wenigen Overheads, wird für die folgenden Beispiele eine MYSQL Datenbank verwendet. Dieses bietet alle Erweiterungen, die in dieser Arbeit näher betrachtet werden sollen. Als praxisnahes Beispiel im Rahmen dieser Arbeit wird auf ein allgemein bekanntes Situation zurückgegriffen, um die Komplexität zu reduzieren und den Fokus mehr auf die Analyse-Erweiterungen zu lenken. Konkret handelt es sich um die Datenbank einer Bibliothe. Dieses trägt fortan den Namen *LibraryDB*. Das Skript für die Erstellung der Datenbank und auch die konkreten Relationen können dem Anhang entnommen werden. Die Implementierung des SQL Skripts soll innerhalb dieser Arbeit unberührt bleiben, bilde aber die Grundlage der Beispiele im nachfolgenden Kapitel, dass nun thematisiert werdern soll.

4. Ergebnis

In diesem Kapitel sollen aufbauend auf der Datenbankimplementierung konkreten Implementierungsbeispiele zu den zwei Analyse-Erweiterungen betrachtet werden. Hierzu pflegt das Kapitel eine schrittweise Einführung in die Erweiterungen. Das Skript für das Erstellen der Datenbank kann dem Anhang entnommen werden.

4.1. Implementierung - Window Function

Dieses Kapitel beschäftigt sich ausschließlich mit der konkreten Anwendung der Window-Functions und deren Verwandtheit zu den herkömmlichen Aggregatfunktionen. Hierzu wird das Beispiel der *LibraryDB* herangezogen. Um das Beispiel näher erläutern zu können sei zunächst auf eine einfache Aggregatfunktion mit `GROUP BY` Klausel verwiesen.

```
1 SELECT
2 genre,
3 AVG(published_year) AS avg_year
4 FROM Books
5 GROUP BY genre;
```

4.1.Listing.: Aggregatfunktion und Group_By

genre	avg_year
Fiction	1945.3333
Dystopian	1940.5000
Adventure	1851.0000
Fantasy	1968.5000
Thriller	2003.0000

Tabelle 4.1.: Ergebnis Group_by

Diese Query liefert genau soviel Records, wie es verschiedenen Einträger in der Spalte *genre* gibt, die Tabelle 4.1 zeigt einen Ausschnitt. Es wird dann für jedes Genre ein durchschnittliches Alter der Bücher angegeben. Für die Nächste Betrachtung soll aus der Aggregatfunktion `AVG(<..>)` eine Analyse-Funktion generiert werden. Hierzu ist hinter die Aggregatfunktion das Schlüsselwort `OVER()` anzuhängen, wie in 4.2 zu sehen ist.

```
SELECT AVG(published_year) OVER()
AS avg_year
FROM Books;
```

4.2.Listing.: Aggregatfunktion mit Over

avg_year
1947.2500
1947.2500
1947.2500
1947.2500

Tabelle 4.2.: Ergebnis Over

Nach Betrachtung der Ergebnismenge fällt sofort auf, dass nun ein viel detaillierter Einblick gewonnen wird. `OVER()` sorgt dafür, dass für jeden ursprünglichen Wert der Menge der bestimmte Durchschnittswert angegeben wird. Anders als bei den klassischen Aggregatfunktionen aus **??** wird die Ergebnismenge nicht zusammengefasst. Im Beispiel aus dem Block 4.2 wird durch das `Over` bereits ein Fenster über die Ergebnismenge gelegt. In diesem Fall umschließt das Fenster alle Einträge, da keine genaueren Angaben im Argument angegeben wurden. Für ein praxisnahes Beispiel mit mehreren Fenstern, soll eine Ausleihhistorie aufgebaut werden, die angibt welches Mitglied wie viele Bücher ausgeliehen hat. Hierzu soll die Analyse-Funktion `ROW_NUMBER` herangezogen werden. Das Ziel ist es über die Ergebnismenge der *Borrowed_Books* ein Fenster zu legen und diese anhand der *member_id* aufzuteilen. `ROW_NUMBER` zählt dann die Zeilen in jedem Fenster. Listing 4.3 zeigt den nötigen Quelltext für dieses Beispiel.

```
SELECT member_id, book_id, borrow_date, return_date,
ROW_NUMBER() OVER(PARTITION BY member_id) AS borrow_sequence
FROM Borrowed_Books;
```

4.3.Listing.: Analyse-Funktion mit mehreren Fenstern

member_id	book_id	borrow_date	return_date	borrow_sequence
1	1	2023-01-01	2023-01-15	1
1	4	2023-04-10	NULL	2
2	2	2023-02-01	2023-02-10	1
2	1	2023-06-15	NULL	2
3	3	2023-03-05	2023-03-20	1
4	5	2023-05-12	2023-06-01	1

Tabelle 4.3.: Ergebnis Analyse-Funktion

Durch das Zählen der Zeilen für jedes Fenster entsteht etwas, das in der Fachliteratur als *running total* beschrieben wird (Nuijten und Barel 2023). Die Gruppierung der Fenster sind anhand der *member_id* zu erkennen.

Mit der `ORDER BY`-Klausel kann noch ein Schritt weiter gegangen werden und die Einträger innerhalb eines Fensters sortiert werden. In dem konkreten Beispiel der

LibraryDB soll jedes Fenster nach der *book_id* sortiert werden. Die konkrete Syntax für diese Query zeigt der folgenden Bock.

```
1 SELECT member_id, book_id, borrow_date, return_date,
2 ROW_NUMBER() OVER(PARTITION BY member_id ORDER BY book_id)
3 AS borrow_sequence
4 FROM Borrowed_Books;
```

4.4.Listing.: Analytische-Funktion und Order By

member_id	book_id	borrow_date	return_date	borrow_sequence
1	1	2023-01-01	2023-01-15	1
1	4	2023-04-10	NULL	2
2	1	2023-06-15	NULL	2
2	2	2023-02-01	2023-02-10	1
3	3	2023-03-05	2023-03-20	1
4	5	2023-05-12	2023-06-01	1

Tabelle 4.4.: Ergebnis Order By

Wie zu erkennen ist, wird die *oder_by_clause* wie in früheren Kapiteln bereits beschrieben einfach an die *partition_by_clause* Klausel angehängt. Sie ist somit ein Teil der *analytic_clause*.

Mit Abschluss dieser Beispielreihe wurde noch lange nicht alle Möglichkeiten der Window-Funktions angesprochen die Dokumentation von Oracle Corporation (2021) beschreibt noch weitere Sprachkonstrukte, die in dieser Arbeit aber unberücksichtigt belieben. Beispielsweise ist es durch das sogenaten *windowing* möglich, ein laufendes Fenster zu erstellen das durch über alle Einträge läuft und so die analytischen Funktionen auf die Einträge anwendet.

4.2. Implementierung - Rollup and Cube

Die Erweiterungen `ROLLUP` und `CUBE` werden als Erweiterung der `GROUP BY` Klausel verstanden. Dieses Kapitel soll nun durch Hilfe der *LibraryDB* die Möglichkeiten und Sprachkonstrukte näher aufschlüsseln. Dazu sollen die Analogien zu der herkömmlichen `GROUP BY` Funktion visualisiert werden. Um mit herkömmlichen Befehlen Einsicht in mehrere Gruppierungen zu erhalten, muss für jede gewünschte Gruppierung ein `SELECT` geschrieben werden. Die Codeblöcke 4.5 und 4.6 zeigen dies.

```
SELECT borrow_date,
FROM Borrowed_Books
GROUP BY(borrow_date);
```

4.5.Listing.: Gruppe Date

```
SELECT member_id,
FROM Borrowed_Books
GROUP BY(member_id);
```

4.6.Listing.: Gruppe ID

Um dieses Vorgehen zu vereinfachen, kann auf das Schlüsselwort `GROUPING SETS` zurückgegriffen werden, dass die Blöcke 4.6 und 4.5 in einer Abfrage vereint. Die Ergebnismenge wird ebenfalls in einer Relation dargestellt. Alle nicht relevanten Felder werden dann mit `NULL` aufgefüllt.

```
SELECT borrow_date, member_id,
FROM Borrowed_Books
GROUP BY GROUPING SETS(borrow_date, member_id);
```

4.7.Listing.: Beispiel eines Grouping sets

Mit `GROUPING SETS` ist es nicht möglich eine Kombination aus den angegebenen Attributen zu gruppieren. Diese Funktion ist den Erweiterungen `ROLLUP` und `CUBE` überlassen. Um aus einem Set an Gruppierungen einnen `ROLLUP` zu erstellen, ist lediglich das Schlüsselwort auszutauschen. Konkret ersetzten wir also das `GROUPING SETS` gegen ein `ROLLUP`. Die Implementierung kann nun so verändert werden, dass ein praktisches Beispiel sichtbar wird. Mit der Aggregatfunktion `COUNT` und der `ROLLUP` Funktion kann eine Query gebaut werden, die in einer Ergebnismenge folgenden Daten liefert.

- *1. Punkt* Wie viele Bücher wurden insgesamt schon ausgeliehen
- *2. Punkt* Wie viel Bücher wurden an einem bestimmten Tag ausgeliehen
- *2. Punkt* Wie viel Bücher wurden von einer Person an einem Tag ausgeliehen

```
SELECT borrow_date, member_id, COUNT(book_id)
        AS total_borrowed
FROM Borrowed_Books
GROUP BY ROLLUP(borrow_date, member_id);
```

4.8.Listing.: Beispiel eines ROLLUP

Tabelle 4.5 zeigt einen Ausschnitt der Ergebnismenge aus der Query 4.8. Bei genauerem Betrachten der Ergebnismenge lassen sich die oben genannten Punkte aus der Aufzählung 4.2 wiederfinden. Der erste Punkt (Gesamtverleih) ist auf die letzte Zeile zurückzuführen, die angibt, dass insgesamt bereits zehn Bücher verliehen wurden. Das Datum oder das Mitglied wird hier also nicht berücksichtigt. Der Punkt zwei weist auf alle Einträge hin, die in der Spalte *member_id* ein `NULL` vorfinden. Beispielsweise wurden am *2023-01-01* insgesamt drei Bücher verliehen (Zeile drei). Die kleinste Betrachtung sind dann

borrow_date	member_id	total_borrowed
2023-01-01	1	2
2023-01-01	2	1
2023-01-01	NULL	3
2023-03-05	3	1
2023-03-05	NULL	1
NULL	NULL	10

Tabelle 4.5.: Ergebnis Rollup

alle Spalten, die kein `NULL` aufweisen. Betrachtet man die erste Zeile, so kann dort herausgelesen werden, dass am *2023-01-01* das Mitglied mit der Nummer *eins*, zwei Bücher ausgeliehen hat. Ohne die Erweiterung `ROLLUP` müsste für jede dieser drei Punkte ein eigenes `SELECT` geschrieben werden. Durch die richtige Anordnung der Attribute innerhalb der Rollup-Funktion kann also ein immer detaillierterer Einblick gewonnen werden. Auch wenn der `ROLLUP` bereits einen sehr tiefen Einblick mit nur wenigen Zeilen Code gewährt, gibt es noch ein Konstrukt, dass noch tiefer geht. Durch ersetzten des Schlüsselwortes `ROLLUP` gegen `CUBE` lassen sich noch mehr Kombinationen der Ergebnismenge anzeigen. Hierzu sei auf das Kapitel 2.2.3 verweisen. Damit lässt sich auch ermitteln, wie viele Bücher ein Mitglied unabhängig vom Tag ausgeliehen hat.

```
1 SELECT borrow_date, member_id, COUNT(book_id)
2         AS total_borrowed
3 FROM Borrowed_Books
4 GROUP BY CUBE(borrow_date, member_id);
```

4.9.Listing.: Analytische-Funktion und Order By

4.3. Zusätzliche Erweiterungen

Weitere Analyse-Erweiterungen, die hier aber unberührt bleiben sollen, sind Beispielsweise die rekursiven Abfragen. Eine Rekursion ist laut Benecke und Benecke (1998) die Definition eines Problems, einer Funktion oder eines Verhaltens durch sich selbst. Aufbauend auf dem Konzept der *Common Table Espressions* (CTE) können damit auch in SQL Probleme rekursiv gelöst werden (Bisso 2022). Hinzu kommt das Konzept des *Pivot and unpivot*, das Nuijten und Barel (2023) näher beschreiben.

5. Fazit

Durch die hier behandelten Konstrukte bieten sich eine Vielzahl an Möglichkeiten. Gerade in Bezug auf die Datenanalyse, wie es Fotache und Strimbei (2015) in ihrer Arbeit *SQL and Data Analysis* beschreiben. Die Erweiterungen Rollup, Cube und Window-Funktions bieten grundsätzlich eine gute Lesbarkeit des Quelltextes. Jedoch sind für eine korrekte Implementierung einer Query mit Fenster-Funktion wesentlich mehr Fachkenntnisse nötig. Es müssen viel mehr einzelne Klauseln verstanden und betrachtet werden, die alle eine eigene Syntax haben. Ein `ROLLUP` oder `CUBE` ist hier wesentlich einfacher zu handhaben. Diese können lediglich der `GROUP BY` Klausel vorangestellt werden. Da die Window-Functions das deutlich komplexere Sprachkonstrukt bieten, ergeben sich hierdurch mehr Möglichkeiten und die größere Flexibilität. Die Erweiterungen `ROLLUP` und `CUBE` sind hingegen etwas starrer in ihrer Anwendung. Betrachtet man nur die Ergebnismenge, so fällt auf, dass das Resultat der komplexeren Window-Functions anfangs leichter zu interpretieren ist. Bei den Konzepten `ROLLUP` und `CUBE` gestaltet sich die Interpretation der Ergebnisrelation etwas schwieriger. Es muss erst verstanden werden, wie die `NULL` Werte zu interpretieren sind. Auch sind die verschiedenen Eben der Gruppierungen nicht sofort ersichtlich.

Mit Blich auf die Forschungsfrage, lässt sich zusammenfassend feststellen, dass die Integration dieser Analyse-Erweiterungen einen erheblichen Mehrwert für die Möglichkeiten in der Datenanalyse bietet, insbesondere in Hinblick darauf, wie einfach komplexe Abfragen geworden sind. Jedoch wurde auch klar, dass für mehr Möglichkeiten immer komplexer werdende Sprachkonstrukte nötig sind. Für die Zukunft wäre eine weitergehende Untersuchung der spezifischen Anwendungsfälle dieser Erweiterungen sinnvoll, um ihre Potenziale vollständig auszuschöpfen. Auch eine weiterführende Recherche zu dem Thema rekursive Abfragen würde die Arbeit hervorragend ergänzen.

A. Anhang

LibraryDB

Hinweis: Dieses SQL-Skript bildet die Grundlage für die in der Arbeit gezeigten Beispiele.

```sql
-- Create the LibraryDB database
CREATE DATABASE LibraryDB;

-- Select the LibraryDB database
USE LibraryDB;
```

A.1.Listing.: SQL Skript LibraryDB create DB

```sql
-- Creates the table books
CREATE TABLE Books (
    book_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    author VARCHAR(255) NOT NULL,
    genre VARCHAR(50),
    published_year INT CHECK (published_year > 0)
);
```

A.2.Listing.: SQL Skript LibraryDB Books

```sql
-- Creats the table Members
CREATE TABLE Members (
    member_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    membership_start_date DATE,
    membership_type ENUM('Standard', 'Premium')
        DEFAULT 'Standard',
    age_group ENUM('Child', 'Teen', 'Adult')
        NOT NULL
);
```

A.3.Listing.: SQL Skript LibraryDB Members

```sql
-- Creats the table Borrowed_Books
CREATE TABLE Borrowed_Books (
    borrow_id INT AUTO_INCREMENT PRIMARY KEY,
    member_id INT,
    book_id INT,
    borrow_date DATE,
    return_date DATE,
    FOREIGN KEY (member_id)
        REFERENCES Members(member_id) ON DELETE CASCADE,
    FOREIGN KEY (book_id)
        REFERENCES Books(book_id) ON DELETE CASCADE
);
```

A.4.Listing.: SQL Skript LibraryDB Borrowed_Books

Beispieldaten LibraryDB

Hinweis: Dieser Datensatz dient lediglich als Beispiel und bildet nicht die vollständig Grundlage für die in der Arbeit gezeigten Beispiele.

```
-- Sample data for table Books
INSERT INTO Books (title, author, genre, published_year)
VALUES
('To Kill a Mockingbird', 'Harper Lee', 'Fiction', 1960),
('The Great Gatsby', 'F. Scott Fitzgerald', 'Fiction', 1925),
('The Catcher in the Rye', 'J.D. Salinger', 'Fiction', 1951),
('Moby-Dick', 'Herman Melville', 'Adventure', 1851),
('Pride and Prejudice', 'Jane Austen', 'Romance', 1813),
('War and Peace', 'Leo Tolstoy', 'Historical', 1869),
('The Hobbit', 'J.R.R. Tolkien', 'Fantasy', 1937),
('The Da Vinci Code', 'Dan Brown', 'Thriller', 2003),
('The Catch-22', 'Joseph Heller', 'Satire', 1961),
('The Shining', 'Stephen King', 'Horror', 1977);
```

A.5.Listing.: Beispieldaten für LibraryDB Books

```
-- Sample data for table Members
INSERT INTO Members (name, membership_start_date,
    membership_type, age_group)
VALUES
('Alice Smith', '2020-01-15', 'Premium', 'Adult'),
('Bob Johnson', '2021-06-10', 'Standard', 'Teen'),
('Charlie Brown', '2019-04-05', 'Standard', 'Child'),
('Diana Prince', '2021-09-25', 'Premium', 'Adult'),
('Eve White', '2020-11-17', 'Standard', 'Teen'),
('Frank Miller', '2022-02-20', 'Standard', 'Adult'),
('Grace Hopper', '2021-03-15', 'Premium', 'Adult'),
('Irene Adler', '2020-12-01', 'Standard', 'Adult'),
('James Moriarty', '2019-07-25', 'Premium', 'Teen'),
('Lucy Gray', '2021-08-30', 'Standard', 'Teen'),
('Michael Scott', '2022-06-10', 'Premium', 'Adult');
```

A.6.Listing.: Beispieldaten für LibraryDB Member

```sql
-- Sample data table Borrowd_Books
-- Null means not yet returned
INSERT INTO Borrowed_Books
(member_id, book_id, borrow_date, return_date)
VALUES
(1, 1, '2023-01-01', '2023-01-15'),
(2, 2, '2023-02-01', '2023-02-10'),
(3, 3, '2023-03-05', '2023-03-20'),
(1, 4, '2023-04-10', NULL),
(4, 5, '2023-05-12', '2023-06-01'),
(2, 1, '2023-06-15', NULL);
```

A.7.Listing.: Beispieldaten für LibraryDB:Borrowd Books

Erklärung zur Abschlussarbeit

Hiermit versichere ich, die eingereichte Abschlussarbeit selbständig verfasst und keine andere als die von mir angegebenen Quellen und Hilfsmittel benutzt zu haben. Wörtlich oder inhaltlich verwendete Quellen wurden entsprechend den anerkannten Regeln wissenschaftlichen Arbeitens zitiert. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht anderweitig als Abschlussarbeit eingereicht wurde. Das Merkblatt zum Täuschungsverbot im Prüfungsverfahren der Hochschule Augsburg habe ich gelesen und zur Kenntnis genommen. Ich versichere, dass die von mir abgegebene Arbeit keinerlei Plagiate, Texte oder Bilder umfasst, die durch von mir beauftragte Dritte erstellt wurden.

Augsburg, den 2. Januar 2025

_____

Lukas Konietzka

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

Benecke, Klaus und Klaus Benecke (1998). „Rekursion“. In: *Strukturierte Tabellen: Ein neues Paradigma für Datenbank-und Programmiersprachen*, S. 151–155 (siehe S. 15).

Bisso, Ignacio L. (Jan. 2022). *What is a common Table Expression in SQL*. URL: https://learnsql.com/blog/what-is-common-table-expression/ (besucht am 17. 10. 2024) (siehe S. 15).

Cao, Yu, Chee-Yong Chan, Jie Li und Kian-Lee Tan (2012). „Optimization of analytic window functions“. In: *Proceedings of the VLDB Endowment* 5.11, S. 1244–1255 (siehe S. 5).

Fotache, Marin und Catalin Strimbei (2015). „SQL and Data Analysis. Some Implications for Data Analysits and Higher Education“. In: *Procedia Economics and Finance* 20. Globalization and Higher Education in Economics and Business Administration - GEBA 2013, S. 243–251. DOI: https://doi.org/10.1016/S2212-5671(15)00071-4 (siehe S. 3, 4, 16).

Grust, Torsten (2017). *Advanced SQL* (siehe S. 5).

Kellenberger, Kathi, Clayton Groom und Ed Pollack (2019). *Expert T-SQL Window Functions in SQL Server 2019*. Springer (siehe S. 5).

Maue, Laurent (Nov. 2022). *SQL Analytical Functions: A Comprehensive Guide*. URL: https://restapp.io/blog/sql-analytical-functions/ (besucht am 17. 10. 2024) (siehe S. 3).

Melton, Jim (2002). *Advanced SQL: 1999: Understanding object-relational and other advanced features*. Elsevier (siehe S. 8).

Melton, Jim und Alan R Simon (2001). *SQL: 1999: understanding relational language components*. Elsevier (siehe S. 5).

Nuijten, Alex und Patrick Barel (2023). „Analytic Functions and (un)Pivoting“. In: *Modern Oracle Database Programming: Level Up Your Skill Set to Oracle's Latest and Most Powerful Features in SQL, PL/SQL, and JSON*. Berkeley, CA: Apress, S. 35–61. DOI: 10.1007/978-1-4842-9166-5_2 (siehe S. 6, 7, 12, 15).

Ogunbiyi, Ibrahim Abayomi (Feb. 2023). *Window Functions in SQL Explained with Examples*. Zugriff am 3. November 2024. freeCodeCamp. URL: https://www.freecodecamp.org/news/window-functions-in-sql/ (siehe S. 6).

Oracle Corporation (1999). *ROLLUP and CUBE: Oracle8i Concepts*. Zugriff am: 08. November 2024 (siehe S. 9).

Oracle Corporation (2016). *GROUPING SETS-Ausdruck im Oracle Endeca Server EQL-Handbuch*. Zugriff am: 05 November 2024 (siehe S. 8, 9).

Oracle Corporation (2021). *Oracle Database SQL Language Reference: Analytic Functions*. Zugriff am 3. November 2024 (siehe S. 5–7, 13, 22).

Schicker, Edwin (2017). *Datenbanken und SQL*. Springer (siehe S. 6).