Database Modeling and Database Systems — Unit 5

Dr.-Ing Anna Androvitsanea

IU Internationale Hochschule GmbH

November 27, 2023

TOPIC OUTLINE

Relational Database Basics	1
Database Queries to Exactly One Table	2
Conception and Modeling of Relational Databases	3
Creating Relational Databases	4
Complex Database Queries on Multiple Tables	5

TOPIC OUTLINE

Manipulating Records in Databases	**6**
NoSQL Database System	**7**

UNIT 5

COMPLEX DATABASE QUERIES ON MULTIPLE TABLES

STUDY GOALS

- Use composite quantities or JOINs to query more than one table.
- Use set operations to gather combine the results of more than one query.
- Use views to reuse the results of complex queries.

EXPLAIN SIMPLY

1. What is the main advantage of JOINs?
2. Why are there different types of JOINs?
3. What is a VIEW?

Table of Contents

1. Introduction to SQL Joins
2. Types of SQL Joins
3. Introduction to Set Operations
4. Using Set Operations in Queries
5. What is a VIEW in SQL?
6. Creating and Using VIEWS
7. Introduction to Common Table Expressions (CTEs)
8. Example CTE Query and Results
9. Transfer Tasks and Queries

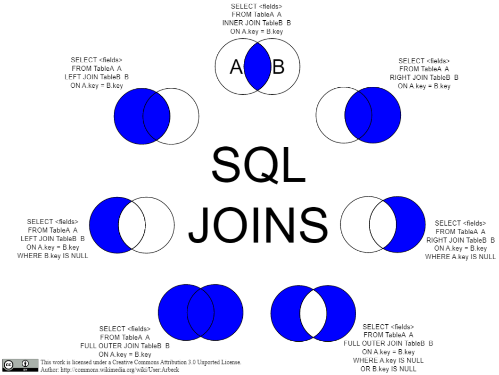
Composite Quantities (JOIN)

- The need for JOINs: combining related data
- Types of JOINs: INNER, LEFT, RIGHT, FULL OUTER
- Example JOIN queries with illustrations

Main Advantage of JOINs

- Efficiency in data retrieval
- Reduced redundancy and improved data consistency

DIFFERENT TYPES OF JOINS



```
SELECT <fields>
FROM TableA A
INNER JOIN TableB B
ON A.key = B.key
```

```
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
```

```
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
```

```
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
WHERE B.key IS NULL
```

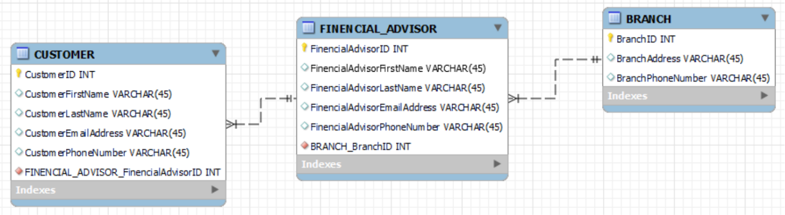
```
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
```

```
SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
```

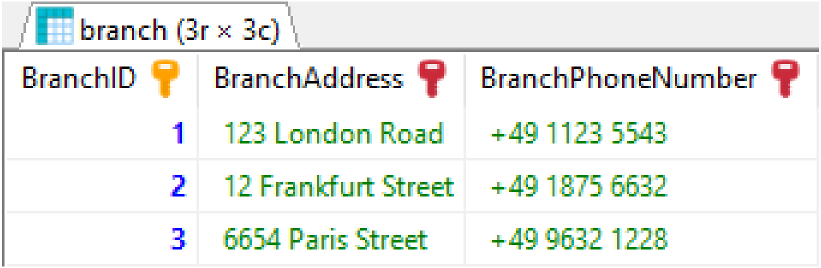
```
SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

Source of the graphic: Arbeck, Wikimedia Commons

COMPOSITE QUANTITIES (JOIN)


CUSTOMER
CustomerID INT
CustomerFirstName VARCHAR(45)
CustomerLastName VARCHAR(45)
CustomerEmailAddress VARCHAR(45)
CustomerPhoneNumber VARCHAR(45)
FINENCIAL_ADVISOR_FinencialAdvisorID INT
Indexes
FINENCIAL_ADVISOR
FinencialAdvisorID INT
FinencialAdvisorFirstName VARCHAR(45)
FinencialAdvisorLastName VARCHAR(45)
FinencialAdvisorEmailAddress VARCHAR(45)
FinencialAdvisorPhoneNumber VARCHAR(45)
BRANCH_BranchID INT
Indexes
BRANCH
BranchID INT
BranchAddress VARCHAR(45)
BranchPhoneNumber VARCHAR(45)
Indexes

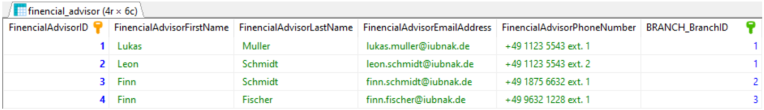
Source of the image: Aljendi, 2022



branch (3r × 3c)

BranchID	BranchAddress	BranchPhoneNumber
1	123 London Road	+49 1123 5543
2	12 Frankfurt Street	+49 1875 6632
3	6654 Paris Street	+49 9632 1228

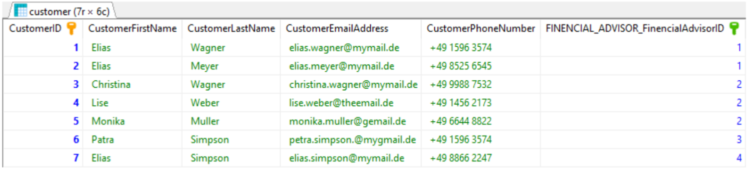
THE FINANCIAL ADVISOR TABLE



FinencialAdvisorID	FinencialAdvisorFirstName	FinencialAdvisorLastName	FinencialAdvisorEmailAddress	FinencialAdvisorPhoneNumber	BRANCH_BranchID
1	Lukas	Muller	lukas.muller@iubnak.de	+49 1123 5543 ext. 1	1
2	Leon	Schmidt	leon.schmidt@iubnak.de	+49 1123 5543 ext. 2	1
3	Finn	Schmidt	finn.schmidt@iubnak.de	+49 1875 6632 ext. 1	2
4	Finn	Fischer	finn.fischer@iubnak.de	+49 9632 1228 ext. 1	3

Source of the image: Aljendi, 2022

THE CUSTOMER TABLE



customer (7r × 6c)

CustomerID	CustomerFirstName	CustomerLastName	CustomerEmailAddress	CustomerPhoneNumber	FINENCIAL_ADVISOR_FinencialAdvisorID
1	Elias	Wagner	elias.wagner@mymail.de	+49 1596 3574	1
2	Elias	Meyer	elias.meyer@mymail.de	+49 8525 6545	1
3	Christina	Wagner	christina.wagner@mymail.de	+49 9988 7532	2
4	Lise	Weber	lise.weber@theemail.de	+49 1456 2173	2
5	Monika	Muller	monika.muller@gemail.de	+49 6644 8822	2
6	Patra	Simpson	petra.simpson.@mygmail.de	+49 1596 3574	3
7	Elias	Simpson	elias.simpson@mymail.de	+49 8866 2247	4

Source of the image: Aljendi, 2022

INNER JOIN

```
SELECT employees.name, departments.name
FROM employees
INNER JOIN departments
ON employees.department_id = departments.id;
```

Explanation: This query retrieves records with matching department IDs from both the employees and departments tables.

LEFT (OUTER) JOIN

```
SELECT employees.name, departments.name
FROM employees
LEFT JOIN departments
ON employees.department_id = departments.id;
```

Explanation: This query returns all employees and their department names. If an employee does not belong to a department, the department name is returned as NULL.

RIGHT (OUTER) JOIN

```
SELECT employees.name, departments.name
FROM employees
RIGHT JOIN departments
ON employees.department_id = departments.id;
```

Explanation: This query returns all departments and the names of employees in those departments. If there are departments without employees, the employee name is returned as NULL.

FULL (OUTER) JOIN

```
SELECT employees.name, departments.name
FROM employees
FULL OUTER JOIN departments
ON employees.department_id = departments.id;
```

Explanation: This query returns all employees and all departments. Where there is no match, the result is NULL on either side.

CROSS JOIN

```
SELECT employees.name, departments.name
FROM employees
CROSS JOIN departments;
```

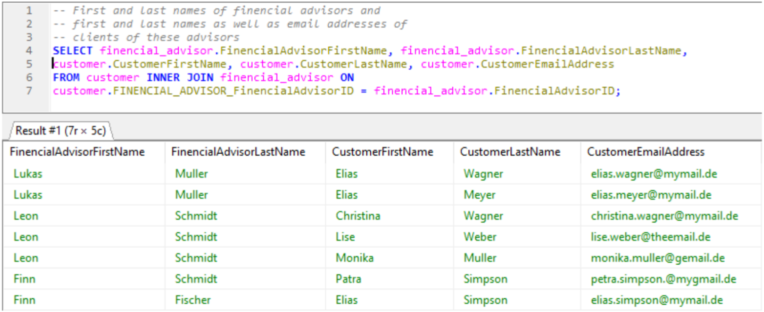
Explanation: This query returns a Cartesian product of employees and departments, combining every employee with every department.

SELF JOIN

```
SELECT A.name AS EmployeeName,
       B.name AS ManagerName
FROM employees A
LEFT JOIN employees B
ON A.manager_id = B.employee_id;
```

Explanation: This query performs a self join to return employees along with their managers from the same employees table.

EXAMPLE OF A COMPOUND SET

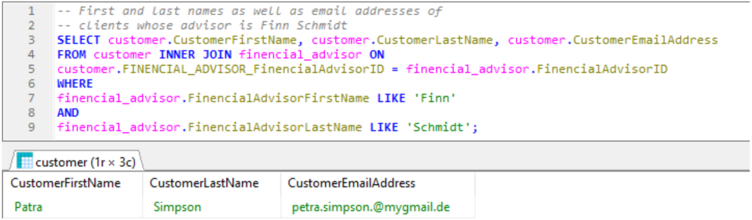


```sql
-- First and last names of finencial advisors and
-- first and last names as well as email addresses of
-- clients of these advisors
SELECT finencial_advisor.FinencialAdvisorFirstName, finencial_advisor.FinencialAdvisorLastName,
customer.CustomerFirstName, customer.CustomerLastName, customer.CustomerEmailAddress
FROM customer INNER JOIN finencial_advisor ON
customer.FINENCIAL_ADVISOR_FinencialAdvisorID = finencial_advisor.FinencialAdvisorID;
```

FinencialAdvisorFirstName	FinencialAdvisorLastName	CustomerFirstName	CustomerLastName	CustomerEmailAddress
Lukas	Muller	Elias	Wagner	elias.wagner@mymail.de
Lukas	Muller	Elias	Meyer	elias.meyer@mymail.de
Leon	Schmidt	Christina	Wagner	christina.wagner@mymail.de
Leon	Schmidt	Lise	Weber	lise.weber@theemail.de
Leon	Schmidt	Monika	Muller	monika.muller@gemail.de
Finn	Schmidt	Patra	Simpson	petra.simpson.@mygmail.de
Finn	Fischer	Elias	Simpson	elias.simpson@mymail.de

Source of the graphics: Aljiendi, 2022

EXAMPLE OF A COMPOUND SET



```sql
-- First and last names as well as email addresses of
-- clients whose advisor is Finn Schmidt
SELECT customer.CustomerFirstName, customer.CustomerLastName, customer.CustomerEmailAddress
FROM customer INNER JOIN financial_advisor ON
customer.FINENCIAL_ADVISOR_FinencialAdvisorID = financial_advisor.FinencialAdvisorID
WHERE
financial_advisor.FinencialAdvisorFirstName LIKE 'Finn'
AND
financial_advisor.FinencialAdvisorLastName LIKE 'Schmidt';
```

CustomerFirstName	CustomerLastName	CustomerEmailAddress
Patra	Simpson	petra.simpson.@mygmail.de

Source of the image: Aljendi, 2022

Introduction to Set Operations

Set operations in SQL are used to combine the results of two or more SELECT queries. These operations are analogous to mathematical set operations. The involved SELECT statements must return the same number of columns with compatible data types.

UNION

Example:

```
-- List all unique phone numbers from BRANCH and
-- FINANCIAL_ADVISOR tables
SELECT BranchPhoneNumber AS PhoneNumber
FROM BRANCH
UNION
SELECT FinancialAdvisorPhoneNumber
FROM FINANCIAL_ADVISOR;
```

Explanation: This ‘UNION‘ operation combines phone numbers from both the ‘BRANCH‘ and ‘FINANCIAL_ADVISOR‘ tables, removing any duplicates, to provide a list of unique phone numbers in the organization.

INTERSECT

Example:

```
-- Assuming a hypothetical scenario where customers can als
-- be financial advisors, find people who are both
SELECT CustomerEmailAddress
FROM CUSTOMER
INTERSECT
SELECT FinancialAdvisorEmailAddress
FROM FINANCIAL_ADVISOR;
```

Explanation: The ‘INTERSECT‘ operation retrieves email addresses that are present in both ‘CUSTOMER‘ and ‘FINANCIAL_ADVISOR‘ tables, indicating people who are both customers and financial advisors.

EXCEPT

Example:

```
-- Find all customers who are not financial advisors
SELECT CustomerEmailAddress
FROM CUSTOMER
EXCEPT
SELECT FinancialAdvisorEmailAddress
FROM FINANCIAL_ADVISOR;
```

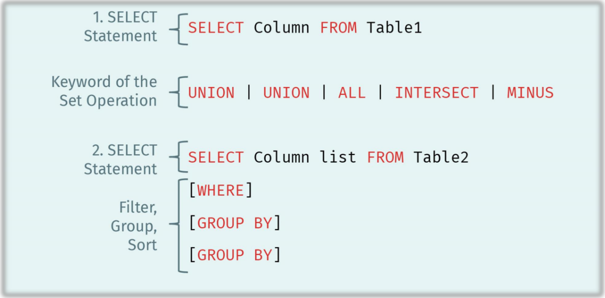
Explanation: This ‘EXCEPT‘ operation finds email addresses that are in the ‘CUSTOMER‘ table but not in the ‘FINANCIAL_ADVISOR‘ table, effectively listing customers who are not financial advisors.

Usage and Considerations

When performing set operations, ensure that the columns' data types are compatible. Set operations are powerful for performing bulk data comparisons or combinations and are useful in data analysis, reporting, and data integration tasks.

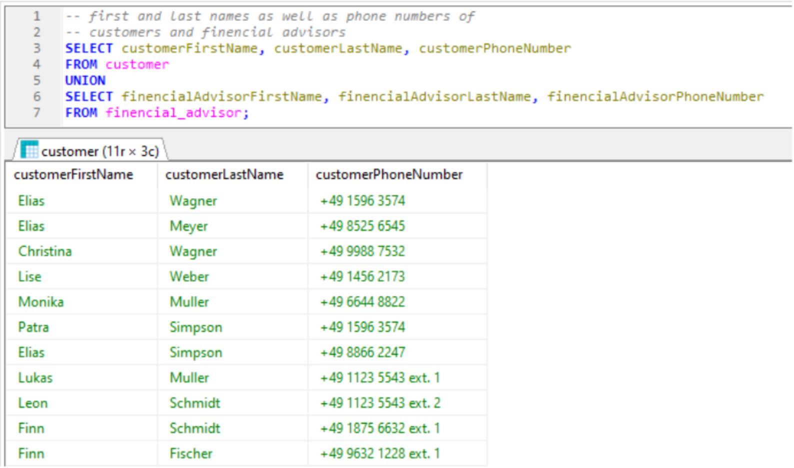
The choice between UNION, INTERSECT, and EXCEPT depends on the specific set manipulation you aim to perform, whether combining datasets, finding common elements, or excluding elements.

SET OPERATIONS



```
SELECT Column FROM Table1
UNION | UNION | ALL | INTERSECT | MINUS
SELECT Column list FROM Table2
[WHERE]
[GROUP BY]
[GROUP BY]
```

Source of the graphic: Course Book DLBCSDMD01, p. 113



```sql
-- first and last names as well as phone numbers of
-- customers and finencial advisors
SELECT customerFirstName, customerLastName, customerPhoneNumber
FROM customer
UNION
SELECT finencialAdvisorFirstName, finencialAdvisorLastName, finencialAdvisorPhoneNumber
FROM finencial_advisor;
```

customerFirstName	customerLastName	customerPhoneNumber
Elias	Wagner	+49 1596 3574
Elias	Meyer	+49 8525 6545
Christina	Wagner	+49 9988 7532
Lise	Weber	+49 1456 2173
Monika	Muller	+49 6644 8822
Patra	Simpson	+49 1596 3574
Elias	Simpson	+49 8866 2247
Lukas	Muller	+49 1123 5543 ext. 1
Leon	Schmidt	+49 1123 5543 ext. 2
Finn	Schmidt	+49 1875 6632 ext. 1
Finn	Fischer	+49 9632 1228 ext. 1

Figure: Example of SET Operation

What is a VIEW?

- A VIEW in SQL is a virtual table resulting from a predefined SQL query.
- It is used as a table that does not physically store data, but provides results dynamically.
- Views are useful for abstracting underlying tables, simplifying complex queries, enhancing security, and providing a level of indirection.

Purpose of a VIEW

- Views can hide the complexity of data by encapsulating joins, filters, and aggregations.
- They help with permission management by providing access to specific data in the underlying tables without giving direct table access.
- Views can present a different representation of the data, such as pivoted, aggregated, or joined from multiple tables.

Creating a VIEW Example

```
CREATE VIEW View_AdvisorDetails AS
SELECT fa.FinancialAdvisorFirstName,
       fa.FinancialAdvisorLastName,
       b.BranchAddress,
       b.BranchPhoneNumber
FROM FINANCIAL_ADVISOR fa
JOIN BRANCH b ON fa.BRANCH_BranchID = b.BranchID;
```

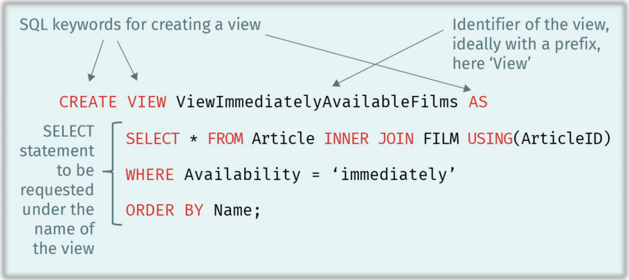
Explanation: This VIEW, ‘View_AdvisorDetails‘, joins the ‘FINANCIAL_ADVISOR‘ and ‘BRANCH‘ tables to provide a combined view of financial advisors and their associated branch details.

Using a VIEW Example

```
SELECT *
FROM View_AdvisorDetails
WHERE FinancialAdvisorLastName = 'Muller';
```

Explanation: This SELECT statement retrieves all financial advisors with the last name 'Muller' from the ‘View_AdvisorDetails‘. It demonstrates how to use a VIEW just like a regular table in queries.

VIEWS



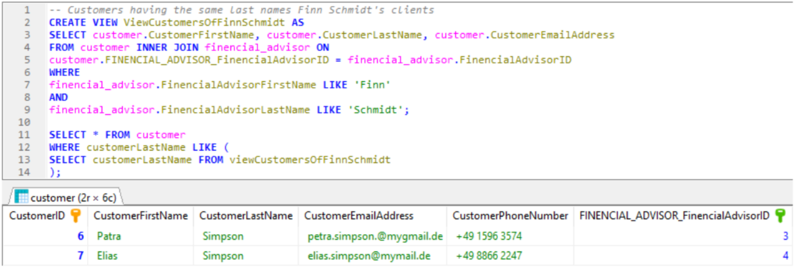
SQL keywords for creating a view

Identifier of the view, ideally with a prefix, here ‘View’

```
CREATE VIEW ViewImmediatelyAvailableFilms AS
SELECT * FROM Article INNER JOIN FILM USING(ArticleID)
WHERE Availability = ‘immediately’
ORDER BY Name;
```

SELECT statement to be requested under the name of the view

Source of the graphic: Course Book DLBCSDMD01, p.140



```sql
-- Customers having the same last names Finn Schmidt's clients
CREATE VIEW ViewCustomersOfFinnSchmidt AS
SELECT customer.CustomerFirstName, customer.CustomerLastName, customer.CustomerEmailAddress
FROM customer INNER JOIN financial_advisor ON
customer.FINENCIAL_ADVISOR_FinencialAdvisorID = financial_advisor.FinencialAdvisorID
WHERE
financial_advisor.FinancialAdvisorFirstName LIKE 'Finn'
AND
financial_advisor.FinancialAdvisorLastName LIKE 'Schmidt';

SELECT * FROM customer
WHERE customerLastName LIKE (
SELECT customerLastName FROM viewCustomersOfFinnSchmidt
);
```

CustomerID	CustomerFirstName	CustomerLastName	CustomerEmailAddress	CustomerPhoneNumber	FINENCIAL_ADVISOR_FinencialAdvisorID
6	Patra	Simpson	petra.simpson.@mygmail.de	+49 1596 3574	3
7	Elias	Simpson	elias.simpson@mymail.de	+49 8866 2247	4

Figure: EXAMPLE OF CREATING AND USING A VIEW

Introduction to Common Table Expressions

- A Common Table Expression (CTE) is a temporary result set which you can reference within another SQL statement.
- CTEs are used to simplify complex queries by breaking them down into simpler blocks.
- They are similar to subqueries but are more readable and can be referenced multiple times within the same query.
- CTEs can be recursive, allowing them to reference themselves, which is useful for hierarchical data queries.

Example CTE Query

```
WITH AdvisorCustomerCount AS (
    SELECT FINANCIAL_ADVISOR_FinancialAdvisorID,
           COUNT(*) AS NumberOfCustomers
    FROM CUSTOMER
    GROUP BY FINANCIAL_ADVISOR_FinancialAdvisorID
)
SELECT fa.FinancialAdvisorFirstName,
       fa.FinancialAdvisorLastName,
       acc.NumberOfCustomers
FROM FINANCIAL_ADVISOR fa
JOIN AdvisorCustomerCount acc
ON fa.FinancialAdvisorID =
                acc.FINANCIAL_ADVISOR_FinancialAdvisorID;
```

Results of the CTE Query

Advisor First Name	**Advisor Last Name**	**Customer Count**
Elias	Wagner	2
Christina	Wagner	3
Patra	Simpson	1

- The CTE ‘AdvisorCustomerCount‘ calculates the number of customers for each financial advisor.
- The main query then joins this CTE with the ‘FINANCIAL_ADVISOR‘ table to list advisors alongside their customer counts.

Advantages of Using CTEs

- CTEs provide better readability and organization for complex queries, which makes understanding and maintenance easier.
- They allow for recursive queries, which is not directly possible with standard subqueries or joins.
- CTEs can be referenced multiple times within the same query, preventing the need to write the same subquery multiple times.
- They can serve as a building block for larger queries, acting as a named subquery.
- CTEs help in breaking down complex calculations and logic, which can improve performance in certain scenarios.

EXPLAIN SIMPLY

1. What is the main advantage of JOINs?
2. Why are there different types of JOINs?
3. What is a VIEW?

What is the main advantage of JOINs?

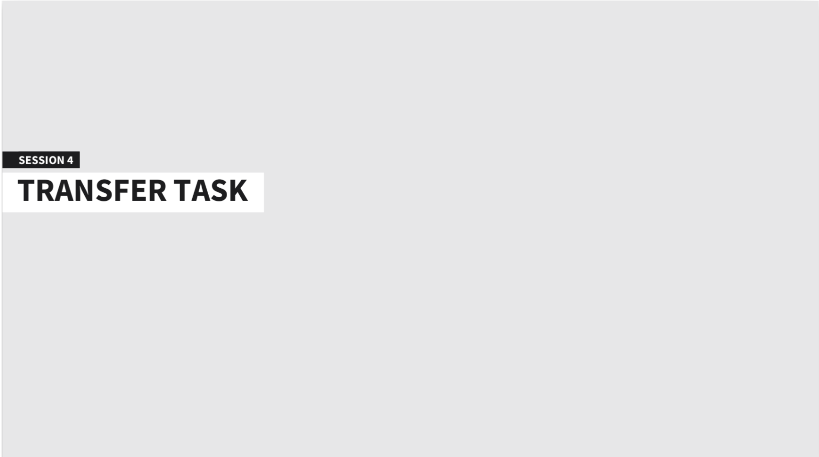
- The main advantage of JOINs in SQL is that they allow you to combine rows from two or more tables based on a related column between them.
- This enables you to create more complex and detailed queries, pulling in diverse data from various parts of the database in a single query.
- It enhances the efficiency and power of database queries, eliminating the need for multiple separate queries and manual data merging.

Why are there different types of JOINs?

- Different types of JOINs exist because they serve different purposes and provide different views of the data, depending on the relationships between the tables.
- INNER JOIN selects records with matching values in both tables.
- LEFT (OUTER) JOIN and RIGHT (OUTER) JOIN include all records from one side, even if there are no matches in the other table.
- FULL (OUTER) JOIN combines LEFT JOIN and RIGHT JOIN, including all records when there is a match in either left or right table.
- These variations give flexibility in how data is combined and presented, allowing for a wide range of queries to be constructed.

What is a VIEW?

- A VIEW in SQL is a virtual table that is based on the result-set of an SQL statement.
- It contains rows and columns, just like a real table, and you can use it as you would use a table.
- VIEWS are created with the CREATE VIEW statement and can comprise data from one or more tables.
- The advantage of a VIEW is that it can simplify complex SQL queries, encapsulate the complexity of data, and provide a layer of security by restricting access to the underlying base tables.
- It can be used to present a subset of data or to simulate a table for users without storing the data separately.



SESSION 4

TRANSFER TASK

Transfer Tasks

Given the same database described previously in this session, create queries that return:

1. The ranch address, the first and last names as well as the phone number of the financial advisors working in the branch that has the phone number +49 1123 5543.
2. A list of the branch id and first and last names of the financial advisors working in the branch that has the phone number +49 1123 5443 as well as their phone numbers along with their clients.
3. The branch ID, the first and last names as well as the phone numbers of financial advisors whose last name is Schmidt along with the first and last names of their clients.
4. The branch ID, the advisor‘s first and last names as well as their phone numbers, the client‘s first and last names as well as their phone numbers for any person (whether client or advisor) whose last name is Muller.

Transfer Tasks (Continued)

5. Using a view, the unique branch numbers of any person (whether an advisor or a client) whose last name is Muller.
6. Number of clients in each branch address.
7. The branch phone number as well as the number of clients in the branch of which the phone number is +49 1123 5543.

Query 1: Advisors in a Specific Branch

```
SELECT b.BranchAddress,
       fa.FinancialAdvisorFirstName,
       fa.FinancialAdvisorLastName,
       fa.FinancialAdvisorPhoneNumber
FROM FINANCIAL_ADVISOR fa
JOIN BRANCH b ON fa.BRANCH_BranchID = b.BranchID
WHERE b.BranchPhoneNumber = ’+49 1123 5543’;
```

Query 2: Branch Details and Clients

```
SELECT b.BranchID,
       fa.FinancialAdvisorFirstName,
       fa.FinancialAdvisorLastName,
       fa.FinancialAdvisorPhoneNumber,
       c.CustomerFirstName,
       c.CustomerLastName
FROM BRANCH b
JOIN FINANCIAL_ADVISOR fa ON b.BranchID =
                            fa.BRANCH_BranchID
JOIN CUSTOMER c ON fa.FinancialAdvisorID =
              c.FINANCIAL_ADVISOR_FinancialAdvisorID
WHERE b.BranchPhoneNumber = ’+49 1123 5443’;
```

Query 3: Advisors and Clients with Last Name Schmidt

```
SELECT b.BranchID,
       fa.FinancialAdvisorFirstName,
       fa.FinancialAdvisorLastName,
       fa.FinancialAdvisorPhoneNumber,
       c.CustomerFirstName,
       c.CustomerLastName
FROM FINANCIAL_ADVISOR fa
JOIN BRANCH b ON fa.BRANCH_BranchID = b.BranchID
JOIN CUSTOMER c ON fa.FinancialAdvisorID =
                    c.FINANCIAL_ADVISOR_FinancialAdvisorID
WHERE fa.FinancialAdvisorLastName = 'Schmidt';
```

Query 4: Details for Persons with Last Name Muller

```
SELECT b.BranchID,
       fa.FinancialAdvisorFirstName,
       fa.FinancialAdvisorLastName,
       fa.FinancialAdvisorPhoneNumber,
       c.CustomerFirstName,
       c.CustomerLastName,
       c.CustomerPhoneNumber
FROM FINANCIAL_ADVISOR fa
JOIN BRANCH b ON fa.BRANCH_BranchID = b.BranchID
JOIN CUSTOMER c ON fa.FinancialAdvisorID =
                    c.FINANCIAL_ADVISOR_FinancialAdvisorID
WHERE fa.FinancialAdvisorLastName = ’Muller’
   OR c.CustomerLastName = ’Muller’;
```

Query 5: Unique Branch Numbers for Mullers

```
CREATE VIEW MullersBranches AS
SELECT DISTINCT b.BranchID
FROM BRANCH b
JOIN FINANCIAL_ADVISOR fa ON b.BranchID =
                             fa.BRANCH_BranchID
WHERE fa.FinancialAdvisorLastName = 'Muller'
UNION
SELECT DISTINCT b.BranchID
FROM BRANCH b
JOIN CUSTOMER c ON b.BranchID = c.BRANCH_BranchID
WHERE c.CustomerLastName = 'Muller';

SELECT * FROM MullersBranches;
```

Query 6: Number of Clients per Branch Address

```
SELECT b.BranchAddress,
       COUNT(c.CustomerID) AS NumberOfClients
FROM BRANCH b
JOIN FINANCIAL_ADVISOR fa ON b.BranchID =
                            fa.BRANCH_BranchID
JOIN CUSTOMER c ON fa.FinancialAdvisorID =
                    c.FINANCIAL_ADVISOR_FinancialAdvisorID
GROUP BY b.BranchAddress;
```

Query 7: Client Count for a Specific Branch

```
SELECT b.BranchPhoneNumber,
       COUNT(c.CustomerID) AS NumberOfClients
FROM BRANCH b
JOIN FINANCIAL_ADVISOR fa ON b.BranchID =
                            fa.BRANCH_BranchID
JOIN CUSTOMER c ON fa.FinancialAdvisorID =
                    c.FINANCIAL_ADVISOR_FinancialAdvisorID
WHERE b.BranchPhoneNumber = ’+49 1123 5543’
GROUP BY b.BranchPhoneNumber;
```